



Rain VFX · SDF Collision & Surface Physics System

...

Support & Feedback

Email: care@skulmovski.studio • Discord: [\[Join our community\]](#)

If you encounter any issues, please contact me via Discord or Email before submitting a review.

...

Table of Contents

⚠ IMPORTANT: Known Constraints & By-Design Behavior.....	2
⚠ IMPORTANT: First Time Setup (Read Before Starting).....	9
Project Requirements.....	9
Fast Integration (Quick Start / Demo Scene).....	10
Using Setup Assistant.....	11
Setting Up Rain in a Custom Scene.....	13
Rain SDF Masking System Architecture (Under the Hood).....	14
Aquarium SDF Controller.....	17
Rain VFX Follow Target.....	29
Roof, Water & Ground Mask Controllers.....	34
Roof Mask Controller.....	34
Water Mask Controller.....	36
Ground Mask Controller.....	38
Mask Precision.....	40
Ground Layer Configuration.....	40
Creating Custom Occluders (Proxy Workflow).....	42
Creating Water Occluders (Proxy Workflow).....	44
SDF Editor Tools.....	47
SDF Source Marker.....	47
SDF Source Baker.....	52
SDF Scene Overview.....	58
SDF Source Cleaner.....	60
Recommended SDF Authoring Workflow.....	63
SDF Proxy Geometry Authoring.....	67
SDF Geometry, Animated Objects & Large Structures.....	72
Spawn Settings & Particle Optimization.....	78
⚠ IMPORTANT: Optimization for Production.....	80
Troubleshooting & Bonus Tips.....	83
Technical Reference: Parameter Glossary.....	92
Thank You!.....	98



IMPORTANT: Known Constraints & By-Design Behavior

Before using the asset, please review the following system behaviors and limitations.

Some of these are intentional architectural decisions used to keep real-time SDF collision practical and performant. Others are inherent limitations of voxel-based geometry representation, discrete particle simulation, or screen-space rendering.

Where applicable, the asset provides workflows that help work around or minimize these limitations.

However, they are important to understand in advance because some projects may require additional SDF authoring, scene preparation, or project-specific integration.

Each topic is covered in more detail later in this documentation.

Some Geometry May Require Dedicated SDF Proxies

SDF collision is voxel-based, so not every visual mesh will produce an equally reliable SDF when baked directly.

Thin surfaces, angled geometry, small details, complex shapes, or geometry placed inside an inefficiently large Source volume may require a different bake resolution, a tighter Source layout, or separate simplified geometry created specifically for SDF collision.

The rendered mesh and the SDF geometry do not need to be identical. If the original visual mesh does not produce a sufficiently reliable SDF, a dedicated proxy can be used while the visible model remains unchanged.

There is no universal minimum geometry or proxy thickness. The required representation depends on the shape, Source dimensions, bake resolution, surface orientation, and final voxelized result. Important Sources should therefore be inspected in the SDF Preview and verified in Play Mode.

Large Structures May Require Multiple SDF Sources

Each SDF Source is baked into a cubic [Texture3D](#). Because of this, very large, long, flat, or irregular structures can waste a significant amount of voxel space when baked as one oversized Source.

This can force the use of a higher bake resolution simply to preserve useful detail in a relatively small part of the volume. SDF texture cost also grows very quickly with resolution: doubling the resolution on every axis produces **8×** as many voxels.

For large buildings and environments, it is therefore often more efficient to divide the geometry into smaller logical Sources such as floors, wings, rooms, building sections, or other spatially compact parts.

Depending on bake resolution, an individual SDF texture can range from only kilobytes to tens of megabytes, so several tightly bounded lower-resolution Sources can be considerably more memory-efficient than one very large high-resolution bake.



Splitting should not be excessive either. Every active Source still contributes to the capacity of its corresponding runtime tier, so Source size, resolution, memory usage, and Source count should be balanced together.

SDF Tier Capacity Is Limited

Baked Sources are grouped into five resolution tiers: 16^3 , 32^3 , 64^3 , 128^3 , and 256^3 .

Each tier has separate limits for unique baked SDF textures and runtime Source instances.

These limits are intentional. Each resolution tier uses its own runtime texture atlas, and that atlas occupies persistent GPU memory while the SDF system is active. More tiers or arbitrarily large capacities could be provided, but doing so would reserve additional GPU memory even when that capacity is not needed.

For this reason, the supplied tier capacities are intentionally bounded and can be increased only within supported limits. Increasing a tier's texture capacity also increases the persistent memory required by its runtime atlas.

If more unique textures of one tier are relevant than the tier can currently hold, the runtime prioritizes the more relevant nearby textures. A farther Source may therefore temporarily be omitted even while it intersects the Aquarium.

Runtime Source instances use a separate capacity from unique textures. Reusing a baked texture reduces unique-texture usage, but every placed Source still consumes its own runtime instance entry.

Projects containing unusually large numbers of unique SDF objects or active Source instances should plan their tier usage accordingly.

Full SDF Collision Is Local, Not World-Sized

The Aquarium is intentionally a local SDF volume centered around a moving target rather than a permanently generated SDF for the entire game world.

The target is typically the player or camera, but another moving Transform can be used depending on the project setup. The Aquarium size and culling configuration should be adjusted appropriately for how that target is used.

This local architecture is a deliberate performance optimization. Full three-dimensional SDF collision is most useful near the active gameplay target, where rain interaction can actually be seen and matters most. Continuously evaluating distant SDF geometry would add substantial GPU work and memory cost with little practical benefit.

Geometry outside the Aquarium can still participate through the larger-area Roof Mask and screen-space Depth Buffer collision systems where applicable, but full three-dimensional SDF collision exists only inside the current Aquarium volume.



Aquarium physical size is determined by: **Resolution × Voxel Size**

A practical production starting point is: **256³ resolution × 10 cm voxels = 25.6 × 25.6 × 25.6 m**

This provides approximately **12.8 m** of full SDF coverage from the Aquarium center in each direction.

The maximum supported **512³** resolution at the same **10 cm** voxel size increases the volume to:

51.2 × 51.2 × 51.2 m

or approximately **25.6 m from the center in each direction.**

However, increasing resolution from **256³** to **512³** creates **8×** as many voxels. The runtime Aquarium texture alone increases from approximately **32 MiB** at **256³** to approximately **256 MiB** at **512³**, in addition to the substantially larger compute workload. Higher Aquarium resolutions should therefore be used only when the additional coverage or precision is actually required.

Elevated and Orbit Cameras May Require a Different Aquarium Tracking Setup

The Aquarium uses one assigned **Camera Transform** as both its positional tracking reference and the directional reference for Camera Culling.

In setups where the rendering camera or camera rig remains close to the active gameplay area, the same Transform can often provide both values naturally.

In top-down, isometric, or orbit-camera games, the rendering camera may be positioned far away from the player. Assigning that camera directly also places the Aquarium around the camera position, which may move the local SDF volume away from the gameplay area.

For these setups, use an existing camera rig or pivot Transform located near the player when that Transform also follows the camera's horizontal rotation.

If no suitable combined tracking Transform exists, assign a Transform near the gameplay area and use **360° Camera Culling**. This keeps the Aquarium correctly positioned without depending on a horizontal forward direction, at the cost of disabling directional Aquarium culling.

Projects that require the Aquarium to follow one Transform for position while independently using another Transform for directional culling require custom integration.

Collision at Extreme Rain Speeds

Rain drops use physically driven motion rather than travelling at a permanently fixed velocity.

Gravity continuously accelerates falling drops, while optional wind is applied as an additional force during the VFX simulation. Strong wind can therefore affect both the trajectory and resulting velocity of the rain.



At unusually high particle velocities, a drop can travel a significant distance during a single simulation step.

As with other discrete collision systems, sufficiently fast particles can eventually tunnel through collision geometry or have their collision detected deeper inside the geometry than intended.

This is a general limitation of discrete particle collision rather than a failure specific to the asset's SDF system.

Under typical rain conditions this is generally not an issue. For extreme velocity or wind configurations, collision should be verified in Play Mode.

A thicker or otherwise more suitable SDF representation can provide additional collision margin in some cases, but geometry thickness cannot guarantee collision at arbitrarily high particle speeds.

Scale Changes Require Rebaking

Scale is part of the geometry represented by a baked SDF.

If an SDF Source, one of its parents, or otherwise relevant hierarchy scale changes after baking, the existing **Texture3D** no longer represents the correct shape and the Source must be baked again.

This follows directly from the pre-baked SDF architecture rather than from a runtime Transform limitation.

Runtime position and rotation are different: a rigid Source can move and rotate while continuing to use the same baked SDF because the existing representation can simply be transformed into its new position and orientation.

If the same object is required at multiple substantially different scales, prepare separate baked scale variants instead of relying on arbitrary runtime scaling.

Runtime SDF Rebaking Is Not Supported

SDF baking is an editor-time authoring process. The asset does not generate new baked Source **Texture3D** data during runtime.

This is an intentional performance decision. Runtime SDF generation is considerably more expensive than transforming and combining already baked Sources, and continuously regenerating volumetric SDF data during gameplay would add significant processing cost.

The runtime system therefore works with pre-baked Source textures. It can register, remove, move, rotate, select, transform, and combine those Sources into the local Aquarium, but it does not rebake their underlying mesh geometry.

If a project specifically requires runtime generation or rebaking of SDF textures, that functionality must be implemented as a custom system.



Skinned and Deforming Geometry Uses a Static Baked SDF

`SkinnedMeshRenderer` geometry is supported by the SDF Baker, **but its current deformed pose is captured as a static mesh snapshot during baking.**

The resulting SDF does not continuously follow:

- Skeletal animation
- Bone deformation
- Blend shapes
- Runtime vertex deformation
- Other changes to the actual mesh shape

Moving or rotating the complete baked Source is supported, but later deformation of the mesh does not modify the existing SDF.

This is a consequence of the intentional pre-baked architecture. Continuously rebuilding volumetric SDF data for deforming meshes would introduce substantial runtime cost.

For continuously animated characters or similar objects, simpler rigid SDF proxy Sources can be attached to important bones when approximate rain collision is sufficient. *For example, separate torso, head, or limb proxies can follow their corresponding bone Transforms without requiring the complete deforming mesh to be rebuilt every frame.*

Projects that require an exact continuously deforming SDF need a custom runtime solution.

Only One Active Aquarium Controller Is Supported

The standard runtime architecture supports one active `AquariumSDFController`.

This is intentional. The Aquarium controller acts as the single owner of the local runtime SDF volume, Source processing, tier data, compute dispatch, and shared Rain VFX bindings. If another active Aquarium controller is present, the additional controller reports the conflict and disables itself.

This keeps ownership and GPU resource management predictable and avoids multiple independent Aquarium systems competing for the same standard runtime setup.

Projects that require several completely independent simultaneous Aquarium volumes — for example, certain multi-camera or split-screen configurations — require a custom integration rather than several standard Aquarium controllers operating independently.

Water Mask Does Not Reproduce GPU-Only Water Deformation

Water interaction outside the full SDF workflow is based on a top-down height mask generated from geometry rendered on the `VFXWaterOccluders` layer.



This is intentionally a much simpler representation than full three-dimensional water collision. The system only needs the surface height at which rain should interact with water, allowing a large area to be represented efficiently with a 2D height texture.

Because this mask captures proxy mesh geometry, GPU-only visual deformation is not reproduced automatically. Shader-driven vertex displacement, tessellation, Shader Graph waves, and similar effects may visually move the water surface above or below the height represented by the Water Mask.

The standard Water Proxy workflow also uses `MeshFilter` geometry and does not support `SkinnedMeshRenderer` geometry.

Normal root Transform movement, rotation, and scale are inherited by the Water Proxy. Independently animated child transforms, however, are not synchronized automatically.

If the visual water surface changes significantly and rain must follow those changes, the Water Proxy must approximate the desired physical interaction height or be updated through a project-specific solution.

Weather Proxies Must Be Recreated After Structural Changes

A Weather Occluder Proxy is generated as a structural copy of the source object's rendering hierarchy.

This allows the proxy system to remove unnecessary gameplay components and retain only the geometry required by the Roof Mask, keeping the technical masking pass lightweight.

Normal root movement and supported child animation can be synchronized, but the generated proxy does not continuously rebuild itself when the original hierarchy structure changes.

Recreate the Weather Proxy after significant structural changes such as:

- Replacing a `MeshFilter` or `SkinnedMeshRenderer` mesh
- Adding or removing hierarchy children
- Changing the rig or skeleton
- Replacing major parts of the original hierarchy

This is intentional: continuously rebuilding technical proxy hierarchies would add unnecessary runtime complexity and cost.

Custom Streaming Can Require Project-Specific Integration

Normal scenes can use automatic SDF Source registration and usually require no custom runtime registration code.

For manually managed or streamed content, the runtime exposes a small set of public registration methods: `PushToAquarium()`, `RemoveFromAquarium()` and their batch equivalents.

These methods provide the basic hooks required to explicitly add and remove Sources from the Aquarium when automatic registration is not appropriate.



This functionality is intentionally kept minimal rather than attempting to provide a complete world-streaming framework. Straightforward additive-scene or chunk-based workflows can usually call these methods directly.

More complex Addressables, pooling, world-streaming, or custom resource-management architectures may require additional project-specific integration because their loading, lifetime, ownership, and pooling rules differ substantially between projects.

Strongly Angled Rain May Require Spawn-Area Adjustment

Strong wind can move falling particles significantly sideways before they reach the ground.

This horizontal movement is a natural consequence of the force-based rain simulation. The spawn region itself does not automatically shift or expand to predict arbitrary future particle drift.

As a result, very strongly angled rain may reduce effective coverage around the target even when the collision systems themselves are functioning correctly.

For stronger wind configurations, it may be necessary to:

- Increase the spawn area
- Reduce the spawn height
- Offset the Spawn Box Center
- Implement project-specific spawn compensation

The asset intentionally leaves this control explicit rather than continuously resizing or moving the spawn volume automatically, since the desired coverage depends on the project's camera, gameplay area, rain height, and wind configuration.

This is separate from high-speed particle tunneling: spawn coverage determines **where particles travel**, while tunneling concerns **collision reliability at extreme particle velocity**.

Depth Buffer Collision Depends on Camera Depth

Depth Buffer Collision is a screen-space collision method. It can only interact with geometry that is represented in the active camera's usable depth buffer.

Standard opaque geometry normally participates automatically. Transparent materials and custom shaders that do not write appropriate depth may be visible on screen while still providing no Depth Buffer collision.

This is inherent to screen-space depth collision rather than an asset-specific limitation.

Depth Buffer Collision is therefore used as a supplementary collision layer and should not be treated as a replacement for SDF or the other masking systems when geometry must interact reliably regardless of its camera-depth representation.



IMPORTANT: First Time Setup (Read Before Starting)

If you are importing this asset into your project for the first time, we **highly recommend** running the Setup Assistant before opening the demo scene or adding prefabs to your custom scenes.

If the Assistant window didn't open automatically upon import, you can launch it manually via the top menu:

Tools > Skulmovski Studio > Rain VFX · SignedDistanceField Masking · [URP or HDRP] > VFX Setup Assistant

Project Requirements

Ensure your project meets these core requirements before using the asset.

- **Visual Effect Graph:** Must be installed through the Unity Package Manager.
- **Compute Shader Support:** The Aquarium SDF system uses GPU Compute Shaders and requires a target platform and graphics API that support them.
- **Pipeline Match (CRITICAL):** The Asset Store package uses **URP** by default. If your project uses **HDRP**, import the separate HDRP **.unitypackage** included with the asset. **Do not mix URP and HDRP versions of the asset.**

URP (Universal Render Pipeline)

- **Depth Texture:** Must be enabled in the active URP Asset.
- **Opaque Texture:** Must be enabled in the active URP Asset.
- The **Setup Assistant** checks the required URP configuration and highlights missing settings automatically.

HDRP (High Definition Render Pipeline)

- Import the included **HDRP .unitypackage** instead of using the default URP files.
- The Roof, Water, and Ground masking systems use the provided **HDRP Custom Pass** setup.
- The Setup Assistant checks the required HDRP scene configuration and can apply the supported fixes automatically.

General Settings

- **Weather Occluder Layer:** The project requires a layer named exactly **VFXWeatherOccluders**.



- **Water Occluder Layer:** The project requires a layer named exactly `VFXWaterOccluders`.

The Setup Assistant can create and configure these layers automatically.

Your original gameplay objects should normally **remain on their existing layers**. Use the included Weather and Water Proxy tools when geometry needs to participate in these masking systems.

- **Camera Depth:** Depth Buffer Collision requires geometry to be represented in the active camera depth buffer. Standard opaque materials normally work correctly, while transparent materials or shaders that do not write depth may not participate in Depth Buffer Collision.

Fast Integration (Quick Start / Demo Scene)

To get the most out of the Demo Scene without encountering any rendering errors, please follow this exact sequence:

Step 1: Run Assistant (Project-Wide Setup)

Upon importing the asset, the **Setup Assistant** window should open automatically. If it doesn't, launch it manually via `Tools > Skulmovski Studio > Rain VFX · SignedDistanceField Masking · [URP or HDRP] > VFX Setup Assistant`. **Do not open the demo scene yet!** First, click **Refresh** and resolve all required **Project-Wide Settings**, using the provided actions where available.

Step 2: Open the Demo Scene

Once the project-wide settings are resolved, navigate to the asset's `Demo` folder and open the provided demo scene.

Step 3: Run Assistant Again (Scene Setup)

With the demo scene now open, return to the **Setup Assistant** window and click **Refresh** again. Resolve all required **Scene Settings**, using automatic actions where available and completing manual reference assignments when requested.

Step 4: Play & Explore!

Enter **Play Mode**. If your project has an active Input System configured, you can run around the scene with the included character.



Using Setup Assistant

Before manually configuring the project or troubleshooting the masking system, we highly recommend running the built-in **VFX Setup Assistant**.

Separate Setup Assistants are provided for the **URP** and **HDRP** versions of the asset. Both follow the same general workflow, but some pipeline-specific checks and automatic fixes are different.

The Assistant verifies the required project and scene configuration, provides automatic fixes where possible, and guides you to components that still require manual reference assignment.

After making any manual changes, return to the Assistant and click **Refresh** to update the checklist.

Project-Wide Settings

These checks apply to the project rather than the currently open scene.

Both versions verify that the correct Render Pipeline is active and that the required technical proxy layers exist.

The **URP Assistant** additionally checks and configures:

- Depth Texture and Opaque Texture.
- Visual Effect Graph package availability.
- Registration of the supplied Roof, Water, and Ground Occlusion Renderer Data assets in the active URP Asset's Renderer List.

Where supported, the Assistant provides an action to apply or open the required configuration.

The **HDRP Assistant** does not use those URP Renderer Data assets. Its project-wide setup mainly verifies the active HDRP pipeline and the required Weather and Water proxy layers.

Where an automatic action is available, the Assistant can apply the required change directly.

Scene Settings

Scene Settings evaluate the currently open scene.

In both URP and HDRP, the Assistant can:

- Add the supplied **Aquarium SDF Controller** if it is missing.
- Add the supplied **Rain Follow Target** if it is missing.
- Check that only one Aquarium SDF Controller exists.
- Verify that the player camera is tagged **MainCamera**.
- Verify the Aquarium tracking Transform and Rain VFX references.
- Verify the Rain Follow Target and Rain Root references.



- Verify that the Aquarium and Rain Follow reference the same Rain VFX hierarchy.
- Check that Ground layers have been selected in the Ground Mask Controller.
- Exclude the Weather and Water proxy layers from the Main Camera.

Some references are intentionally assigned manually. When a required reference is missing, the Assistant selects the corresponding component and tells you what needs to be assigned before clicking **Refresh** again.

In the standard setup, the **Main Camera** can be used as both the Aquarium tracking Transform and the Rain Follow Target.

These references do not have to be identical in custom setups. Top-down, isometric, orbit-camera, or other camera-rig configurations may use different Transforms when the Aquarium and Rain VFX need different positional or directional references.

URP Mask Configuration

In URP, the Assistant can configure the Roof and Water masking pipelines using the supplied Renderer Data assets.

For the **Ground Mask**, you first choose the required ground layers manually in the **Ground Mask Controller**. You can select a single layer or multiple layers, depending on how your project is organized.

After that, the Assistant can apply the selected **Ground Layer Mask** to the Ground occlusion camera, assign the supplied Ground Occlusion Renderer, and update the Ground Render Objects Layer Mask to match.

HDRP Mask Configuration

HDRP uses a masking setup based on the supplied **height shader, HDRP camera data, and Custom Passes** rather than URP Renderer Data assets.

For the **Roof** and **Water** masks, the Assistant can configure the required proxy layers, assign the package height shader, ensure the required HDRP camera data exists, and rebuild or synchronize the corresponding Custom Pass setup.

For the **Ground Mask**, you first choose the required ground layers manually in the **Ground Mask Controller**. You can select one or multiple layers. The Assistant then applies that selected **Ground Layer Mask** to the Ground occlusion camera and configures the required HDRP height-shader and Custom Pass setup.

The Assistant also verifies that the Roof, Water, and Ground mask cameras use **Custom Frame Settings** with Post-processing explicitly disabled, so HDRP does not modify the technical height data.



Useful Tools

Both Assistants provide quick access to:

- **Recompile VFX Graphs**
- **SDF Source Marker**
- **SDF Source Baker**
- **SDF Source Cleaner**
- **SDF Scene Overview**

The HDRP Assistant additionally provides **Repair All HDRP Mask Cameras**, which can re-apply the package height shader, layer masks, HDRP camera data, Custom Pass setup, injection point, and required Frame Settings to all three mask cameras.

Additional Notes

The Setup Assistant covers the **essential project and scene setup**, but it is not intended to manage every possible custom project architecture.

Advanced workflows, custom renderer setups, proxy behavior, SDF authoring, optimization, and known limitations are covered in the corresponding sections of this documentation.

Setting Up Rain in a Custom Scene

 **Quick Setup Videos:** Short visual examples of the correct setup workflow for both the Demo Scene and a custom scene are available in the [Quick Setup Videos playlist](#).

For custom scenes, the recommended workflow is to use the **VFX Setup Assistant** rather than configuring the complete masking pipeline manually.

Because the Rain SDF system depends on several connected project-wide and scene-specific settings, the Assistant helps ensure that the required renderers, layers, cameras, controllers, and references are configured correctly.

Recommended Workflow

1. Open the **VFX Setup Assistant** immediately after importing the asset.
2. Complete all required **Project-Wide Settings** before opening the Demo Scene or setting up a custom scene.
3. Open or create your custom scene.
4. Add the Rain VFX to the scene.
5. Position the Rain VFX approximately **10–20 m above the ground**.
6. Open the **VFX Setup Assistant** again and click **Refresh**.



7. Complete all required **Scene Settings**.
8. Assign the requested **tracking Transform**, **Rain Follow Target**, and **Rain VFX** references. In the standard setup, the Main Camera can be used for both tracking references.
9. Select the appropriate Ground layers when requested.
10. Continue using **Refresh** until all required Scene Settings are resolved.
11. Enter **Play Mode** and verify the result.

Rain SDF Masking System Architecture (Under the Hood)

The asset uses a hybrid GPU-driven masking and collision system instead of relying on a single collision technique for the entire rain effect.

At the center of the system is the **Aquarium** — a local runtime SDF volume stored as a **Texture3D**.

The Aquarium moves through the world together with its assigned tracking Transform, typically following the player or camera. Nearby pre-baked SDF Sources are dynamically selected and combined into this local volume on the GPU.

The system intentionally keeps full three-dimensional SDF collision local. Accurate SDF interaction is most useful near the active gameplay target, while continuously evaluating distant geometry would add substantial GPU and memory cost with little practical benefit.

The complete system combines several specialized layers:

- **Aquarium SDF** — Local three-dimensional SDF collision around the active tracking area.
- **Roof Mask** — A large-area top-down height mask used to block rain below roofs and other overhead surfaces outside the Aquarium.
- **Water Mask** — A top-down height mask used to detect the water interaction surface, spawn Ripples, and remove Splashes entering water.
- **Ground Mask** — An optional Aquarium optimization that allows unnecessary SDF processing below suitable ground surfaces to be skipped.
- **Depth Buffer Collision** — Additional screen-space collision against geometry represented in the active camera depth buffer.

Each system is used where its representation provides the best balance between collision detail, coverage, and runtime cost.

Inside the Aquarium

Inside the Aquarium, nearby registered SDF Sources are selected, transformed, and combined into the runtime SDF **Texture3D** on the GPU.

The Rain VFX samples this volume to determine signed distance to nearby geometry, allowing walls, ceilings, interiors, overhangs, vehicles, props, and other spatial geometry to participate in three-dimensional rain collision.



Inside the Aquarium, rain uses: **SDF + Depth Buffer**

Roof Occlusion is intentionally not used inside the Aquarium because the local SDF already provides the more complete three-dimensional representation required for nearby geometry.

Outside the Aquarium

The Aquarium only represents a finite local region around its tracking position.

Outside this volume, full SDF collision is intentionally not evaluated. Instead, the system uses the much cheaper large-area Roof Mask while Depth Buffer Collision remains available.

Outside the Aquarium, rain uses: **Roof Occlusion + Depth Buffer**

This hybrid architecture avoids the cost of maintaining a massive world-sized SDF while still allowing rain to remain correctly blocked by major overhead structures farther from the active SDF region.

Roof Mask

The Roof Mask is a top-down height mask used primarily for blocking rain below roofs, canopies, bridges, and other overhead surfaces.

Weather Occluder geometry on the **VFXWeatherOccluders** layer is rendered from above. The resulting world-space height information is stored in a signed floating-point mask texture and passed to the Rain VFX.

Because the Roof Mask stores only top-down height instead of complete three-dimensional geometry, it can cover a much larger physical area at significantly lower cost than extending full SDF collision across the same region.

The Roof Mask is therefore primarily responsible for large-area rain occlusion outside the Aquarium.

Water Mask

The Water Mask is also a top-down height representation, but it serves a different purpose from Roof Occlusion.

Geometry on the **VFXWaterOccluders** layer defines the surface height that the Rain VFX should treat as water.

The Water Mask is used to:

- Spawn Ripples at the detected water surface.
- Remove Splashes that enter water so they do not continue normal SDF sliding below the surface.



Because this is a height-mask system rather than full three-dimensional water collision, the mask represents the rendered proxy geometry rather than GPU-only shader deformation. Detailed Water Proxy limitations and workflows are covered later in this documentation.

Ground Mask

The Ground Mask is not another rain-blocking system.

It is an optional Aquarium optimization.

The system captures top-down ground height and supplies it to the Aquarium compute process. Voxels below suitable ground surfaces can then be skipped before more expensive SDF evaluation is performed.

This is useful for terrain, streets, and other solid surfaces where no relevant SDF geometry needs to remain active below the captured ground height.

Ground Mask geometry should therefore only include surfaces where everything underneath can safely be ignored by the Aquarium.

Depth Buffer Collision

Depth Buffer Collision provides an additional screen-space collision layer using geometry represented in the active camera depth buffer.

It remains available both inside and outside the Aquarium:

- **Inside the Aquarium:** SDF + Depth Buffer
- **Outside the Aquarium:** Roof Occlusion + Depth Buffer

Depth Buffer Collision is useful for visible geometry that does not require its own SDF Source.

Because it is screen-space, only geometry represented in the camera depth buffer can participate. Transparent materials or custom shaders that do not provide usable camera depth may therefore require another collision or masking method.

Physically Driven Rain Motion

Rain drops are not moved using a permanently fixed velocity.

Gravity continuously affects falling particles, while the optional Wind Force is applied separately during simulation. Strong wind can therefore change both the trajectory and resulting velocity of the rain.



This physical particle motion is independent from the masking architecture described above. At unusually high particle velocities, the normal tunneling limitations of discrete particle collision can eventually become relevant, as described in **Known Constraints & By-Design Behavior**.

Why the System Uses Multiple Collision Layers

A single collision technique would either be too expensive to apply everywhere or would provide insufficient information for some types of interaction.

The asset therefore intentionally combines specialized systems:

- **Aquarium SDF** provides detailed local three-dimensional collision.
- **Roof Mask** provides inexpensive large-area overhead occlusion.
- **Water Mask** provides a lightweight water-surface height representation.
- **Ground Mask** reduces unnecessary Aquarium compute work.
- **Depth Buffer Collision** adds screen-space collision against visible scene geometry.

The most expensive processing remains concentrated around the active gameplay area, while cheaper representations handle geometry where full local SDF collision would provide little additional benefit.

URP and HDRP

The overall masking architecture is the same in both render pipelines, but the top-down height masks are rendered differently.

In URP, the mask cameras use the supplied URP rendering setup to write world-space height information into their mask textures.

In HDRP, Roof, Water, and Ground masking use dedicated Custom Pass implementations that write signed world-space height data into the corresponding textures.

Empty areas use a large negative sentinel value so the VFX and compute systems can distinguish valid captured surfaces from empty mask regions.

Aquarium SDF Controller

The `AquariumSDFController` is the central runtime component of the local SDF collision system.

It creates a local three-dimensional SDF volume — the **Aquarium** — and rebuilds that runtime volume every frame from nearby **pre-baked** `AquariumSource` objects.



The resulting runtime **Texture3D** is passed directly to the Rain VFX and provides detailed three-dimensional collision within the active Aquarium region.

The Aquarium is intentionally local rather than world-sized. It moves through the world with its assigned tracking Transform so that expensive SDF processing remains concentrated around the area where detailed rain interaction is currently useful.

Resolution

Resolution controls the voxel dimensions of the runtime Aquarium cube.

The supported resolutions are: **32³**, **64³**, **128³**, **256³** and **512³**.

The physical size of the Aquarium is calculated from both Resolution and Voxel Size:

Aquarium Size = Resolution × Voxel Size

For example:

- **128³** at **10 cm** per voxel = **12.8 × 12.8 × 12.8 m**
- **256³** at **10 cm** per voxel = **25.6 × 25.6 × 25.6 m**
- **256³** at **5 cm** per voxel = **12.8 × 12.8 × 12.8 m**
- **512³** at **10 cm** per voxel = **51.2 × 51.2 × 51.2 m**

A useful production starting point is:

- **Resolution:** **256³**
- **Voxel Size:** **10 cm**
- **Calculated Size:** **25.6 × 25.6 × 25.6 m**

This provides approximately **12.8 m** of SDF coverage from the Aquarium center in each direction.

512³ at the same **10 cm** voxel size increases this to approximately **25.6 m** in each direction, but should not be treated as a free coverage increase. Doubling resolution on every axis creates **8×** as many voxels and substantially increases both runtime texture memory and GPU compute workload.

Voxel Size

Voxel Size controls how much world space is represented by one Aquarium voxel.

Smaller voxels provide finer spatial detail but reduce the physical area covered by a given Aquarium resolution.

Larger voxels provide greater physical coverage but reduce spatial precision.

*For example, a **256³** Aquarium provides:*



- 25.6 m per axis at 10 cm voxel size.
- 12.8 m per axis at 5 cm voxel size.

Resolution and Voxel Size should therefore be tuned together rather than independently.

Increasing resolution while keeping the same Voxel Size increases physical coverage while also increasing GPU and memory cost.

Reducing Voxel Size while keeping the same resolution improves spatial precision but reduces the physical Aquarium area.

Runtime SDF Texture

The Aquarium uses a three-dimensional runtime Render Texture in RHalf format.

Its width, height, and depth are equal to the selected Aquarium resolution.

Approximate raw texture sizes are:

- 128^3 RHalf ≈ 4 MiB
- 256^3 RHalf ≈ 32 MiB
- 512^3 RHalf ≈ 256 MiB

Because this is a full three-dimensional texture, memory and compute requirements grow rapidly with resolution.

Doubling the resolution on every axis produces **8×** as many voxels. *For example, moving from 256^3 to 512^3 increases the Aquarium voxel count from approximately 16.8 million to approximately 134.2 million.*

Use higher Aquarium resolutions only when the additional physical coverage or precision is actually required.

Local Tracking Volume

The Aquarium is centered on the assigned **Camera Transform** reference.

Despite the field name, this is a Transform reference used to determine the Aquarium's runtime position. In the standard setup it is normally the player camera, but the tracking arrangement can be configured according to the project.

Every frame, the controller calculates the Aquarium cube around the current tracking position using the selected Resolution and Voxel Size.

The same tracking Transform is also used by Camera Culling as its directional reference, so custom tracking arrangements should configure the Aquarium size and culling settings accordingly.

After generating the runtime SDF, the controller sends the following data to the Rain VFX:



- Runtime SDF **Texture3D**
- Aquarium world transform
- Aquarium center
- Aquarium size
- Aquarium resolution

Geometry outside the current Aquarium does not participate in full local SDF collision until it enters the active volume. Roof Occlusion and Depth Buffer Collision can continue to handle applicable geometry outside it.

Elevated, Top-Down & Orbit Cameras

The assigned **Camera Transform** acts as both the Aquarium tracking reference and the directional reference used by Camera Culling.

Its position defines the center of the local Aquarium volume, while its horizontal forward direction is used by the directional culling system.

For elevated, top-down, isometric, or orbit-camera setups, the physical rendering camera may be too far from the gameplay area to use directly as the Aquarium reference.

For example, an orbit camera positioned far from the player would also place the Aquarium around that distant camera position if assigned directly.

When available, use an existing camera rig or pivot Transform located near the player that also follows the camera's horizontal rotation. This keeps the Aquarium near the active gameplay area while preserving meaningful directional culling.

If no suitable combined Transform exists, assign a Transform near the gameplay area and set **Culling Angle** to **360°**.

With **360°** culling, the Aquarium remains centered on the assigned Transform while its forward direction no longer restricts the active horizontal SDF region.

If a project requires one Transform for Aquarium position and a different Transform for culling direction, that setup requires custom integration.

Source Registration

Auto Register Sources controls how **AquariumSource** objects enter the runtime SDF registry.

Auto Register Sources — Enabled

This is the normal plug-and-play workflow.



Enabled **AquariumSource** components automatically register themselves with the Aquarium system.

Once an object has been correctly marked and baked, no additional runtime registration code is normally required.

For most projects, leave **Auto Register Sources** enabled.

Auto Register Sources — Disabled

Disable automatic registration when another project system should explicitly control which SDF Sources are active.

Typical examples include:

- World streaming
- Additive scene loading
- Chunk-based environments
- Custom pooling systems
- Project-specific runtime content management

In this mode, Sources do not automatically register themselves when enabled.

Manual registration is available through a small set of public methods:

- **PushToAquarium()**
- **RemoveFromAquarium()**
- Batch equivalents where appropriate

These methods are intended as simple integration points rather than a complete world-streaming framework. More complex loading, pooling, Addressables, or streaming architectures may require additional project-specific integration.

Scene References

The Aquarium acts as the main owner of the shared scene references used by the SDF masking system.

The controller contains references to:

- Camera Transform
- Target VFX
- Roof Mask Controller
- Water Mask Controller
- Compute Shaders Data

Despite its field name, **Camera Transform** is the Aquarium tracking and culling Transform and does not require a **Camera** component.



The Aquarium propagates the required tracking and Rain VFX references to the Roof and Water Mask controllers.

If Ground Culling is configured, the Ground Mask Controller also receives the required tracking reference.

In the standard setup, the Main Camera can be assigned as the Aquarium **Camera Transform**. Custom camera configurations may instead use another Transform as described in **Elevated, Top-Down & Orbit Cameras**.

The Setup Assistant verifies that the required top-level references are assigned, but does not require the Aquarium tracking Transform and **RainVFXFollowTarget.Target** to be the same Transform.

SDF Source Resolution Tiers

Baked SDF Sources are separated into five bake-resolution tiers: **16³**, **32³**, **64³**, **128³** and **256³**.

These Source tiers are separate from the **Aquarium Resolution** itself. The Aquarium may use up to **512³**, while individual baked Source textures use the five Source resolutions listed above.

Each Source tier has its own runtime atlas and capacity settings.

This allows small or simple Sources and larger or more detailed Sources to use different bake resolutions while still contributing to the same final Aquarium.

Each frame, relevant Sources from all five tiers are prepared and bound to the Aquarium compute shader before the local SDF volume is generated.

Unique Texture Slots

Each Source resolution tier has a configurable maximum number of **unique baked SDF textures** that can be available to that runtime tier.

The supplied limits are:

Source Resolution	Def Unique Tex Slots	Maximum
16³	64	128
32³	32	64
64³	16	32
128³	8	16
256³	4	8



These limits are intentional memory controls.

Each tier owns a persistent runtime texture atlas. Increasing a tier's capacity allows more unique SDF textures to be available simultaneously, but also increases the GPU memory reserved by that atlas.

For this reason, the system does not allocate arbitrarily large tier capacities by default. Increase a tier only when actual project content requires additional capacity.

Reusing the Same Baked SDF

Multiple [AquariumSource](#) objects can reference the same baked SDF [Texture3D](#).

For example: 50 identical objects using one baked SDF = 1 unique texture slot + 50 runtime instances

To reuse an SDF efficiently:

1. Prepare the original object.
2. Mark it as an [AquariumSource](#).
3. Bake and commit its SDF.
4. Verify collision quality.
5. Save the prepared object as a Prefab.
6. Create additional instances from that Prefab.

Each instance keeps its own runtime Transform data while sharing the same committed SDF texture.

If different instances require different scales, they should not reuse the original bake as though their geometry were unchanged. Scale is part of the baked SDF representation and requires an appropriate pre-baked scale variant or rebake.

Instance Limit

Every Source resolution tier also has a separate maximum number of runtime Source instances.

The supplied default is **512 instances per tier** and the configurable maximum is **2048 instances per tier**.

This limit is independent from unique texture capacity.

A large number of repeated objects may therefore consume very little unique-texture capacity while still using many runtime instance entries.



What Happens When a Tier Is Full

If more unique baked textures of one Source tier are relevant than that tier can currently hold, not every texture can be available simultaneously.

The runtime prioritizes the more relevant / closer textures. A farther Source may therefore temporarily be omitted even while it still intersects the Aquarium.

Increasing tier capacity allows more unique textures to coexist, but also increases persistent runtime atlas memory.

For this reason:

- Reuse baked SDF textures where appropriate.
- Use a Source resolution appropriate for the Source size and required geometric detail.
- Avoid unnecessary unique SDF variants.
- Increase tier capacities only when actual content requires them.

Camera Culling

The Aquarium does not need to perform the same amount of expensive SDF processing in every horizontal direction.

Camera Culling restricts the useful processing region relative to the assigned **Camera Transform** and its horizontal forward direction.

The main controls are:

- Culling Angle
- Max Cull Back Offset
- Always Include SDF Zone Size

Tracking Transform and Camera Direction

Camera Culling uses the horizontal forward direction of the same Transform assigned as the Aquarium **Camera Transform**.

This is important when the rendering camera rotates independently from the Transform used to position the Aquarium.

For example, assigning the player Transform keeps the Aquarium centered on the player, but Camera Culling then follows the player's horizontal forward direction rather than the direction of an independently rotating orbit camera.

If the project already provides a camera rig or pivot near the player that rotates together with the camera, that Transform can normally be used as the Aquarium reference.



If no suitable shared position-and-direction Transform exists, use **360° Culling Angle** to disable directional Aquarium culling while keeping the Aquarium centered on the required gameplay position.

Using **360°** preserves full horizontal Aquarium coverage but does not provide the performance savings of narrower directional culling.

Culling Angle

Culling Angle defines the horizontal angular region used by the Aquarium culling system.

Supported range: **90° – 360°**

Conceptually:

- **360°** keeps every horizontal direction available.
- **180°** represents approximately the forward-facing half.
- **90°** produces a much narrower forward region.

The calculation uses the assigned **Camera Transform**'s flattened horizontal forward direction, so the angle is evaluated in X/Z rather than as a full three-dimensional view cone.

Example: 180° Culling + Ground Culling

Imagine the player above a large flat ground surface:

- Ground Culling is enabled.
- The surface belongs to the Ground Mask.
- The Aquarium follows the player area.
- Culling Angle is **180°**.

Angular culling can remove approximately the rear half of the useful Aquarium region.

At the same time, Ground Culling can skip voxels below the captured ground surface.

In an idealized case where the ground roughly divides the Aquarium vertically in half, the two optimizations together may reduce the expensive SDF region to roughly one quarter of the complete Aquarium volume.

This is an illustrative example rather than a guaranteed percentage. The exact result depends on camera position, ground height, Aquarium placement, Always Include zone, Source placement, and scene geometry.

Runtime Culling Angle Control

Culling Angle is evaluated every frame and can be changed at runtime through the **AquariumSDFController**.



This allows different gameplay situations to use different balances between coverage and performance.

For example:

- *Use a narrower angle during normal forward movement.*
- *Increase the angle during rapid camera rotation.*
- *Temporarily use 360° when SDF interaction outside the current forward culling region is important.*

Changing Culling Angle does not require Source rebaking, texture regeneration, or Aquarium reinitialization.

Max Cull Back Offset

Max Cull Back Offset shifts the culling origin backward along the assigned **Camera Transform's** flattened forward direction.

The current supplied value is: **3 m**

This makes angular culling less aggressive immediately around and slightly behind the Aquarium tracking position.

Always Include SDF Zone Size

Always Include SDF Zone Size defines a small cubic region around the Aquarium tracking position where SDF processing remains available regardless of angular culling.

The supplied value is: **1 m**

This protects nearby geometry from aggressive view-direction culling. A wall directly beside or slightly behind the player may still require local SDF collision even when it lies outside a narrow forward-facing Culling Angle.

Ground Culling

Ground Culling is an optional Aquarium optimization.

The Ground Mask supplies a top-down height representation of suitable ground surfaces. The Aquarium compute process can then skip voxels below that captured height rather than evaluating SDF data where no useful rain interaction can occur.

Only surfaces below which SDF evaluation can safely be discarded should participate.

Do not include:



- Roofs
- Bridges
- Upper floors
- Elevated platforms
- Other surfaces with valid playable or SDF space underneath

Treating these surfaces as ground may incorrectly remove valid SDF processing beneath them.

Ground Culling is an optimization only and is not a replacement for Roof Occlusion or SDF collision.

Runtime Build Process

The **runtime Aquarium SDF** is rebuilt on the GPU every frame from already pre-baked Source data.

This should not be confused with rebaking the individual Source textures themselves. Source SDF baking remains an editor-time process.

At a high level, each frame:

1. The current Aquarium bounds are calculated.
2. Camera culling parameters are calculated.
3. Ground Culling data is applied if enabled.
4. Registered Sources are processed by their bake-resolution tier.
5. Relevant unique SDF textures and Source instance data are prepared.
6. The five Source resolution tiers are bound to the Aquarium compute shader.
7. The final local SDF is generated into the runtime **RHalf Texture3D**.
8. The generated Texture3D and its world-space data are passed to the Rain VFX.

Because the runtime combines already baked Source representations, it can respond to:

- Tracking movement
- Runtime Source position changes
- Runtime Source rotation changes
- Sources entering or leaving the Aquarium
- Source registration and removal

Scale changes are different because scale changes the geometry represented by the baked Source SDF and therefore require rebaking.

One Aquarium Controller Per Scene

Only one active **AquariumSDFController** is supported by the standard runtime setup.

If another active controller already exists, the additional controller reports an error and disables itself.



This is intentional: the Aquarium acts as the single runtime owner of the local SDF volume, Source tier data, compute processing, and shared Rain VFX bindings.

Projects requiring multiple independent simultaneous Aquarium volumes need a custom runtime integration.

Runtime Monitoring

The Debug Settings section can display the number of currently registered Sources in each Source resolution tier: **16³**, **32³**, **64³**, **128³**, and **256³**.

When **Update Counts** is enabled, these values refresh during Play Mode.

They are useful for diagnosing:

- Unexpected Source registration
- Large active Source populations
- Tier usage
- Sources remaining registered unexpectedly
- Streaming or pooling integration

These values represent registered Source instances, not unique baked **Texture3D** counts.

Gizmos & Helpers

Show Helpers enables Scene View visualization of the Aquarium and its culling configuration.

The controller currently visualizes:

- The complete Aquarium cube
- The Always Include SDF zone
- Camera forward direction
- Culling-angle limits

These helpers are useful when tuning Aquarium size, culling coverage, Source placement, and the relationship between the tracking position and camera-direction-based optimization.

Recommended Starting Configuration

For a general-purpose production starting point:

- **Resolution:** **256³**
- **Voxel Size:** **10 cm**
- **Calculated Size:** **25.6 × 25.6 × 25.6 m**
- **Auto Register Sources:** Enabled



- **Culling Angle:** Start with the supplied value and adjust when the project requires a different coverage pattern.
- **Tier Limits:** Keep the supplied values until actual scene content requires more capacity.
- **Ground Culling:** Enable only when suitable ground surfaces are available.
- **Debug Helpers:** Keep them available during setup and disable unnecessary runtime monitoring for production when appropriate.

The main Aquarium trade-off is:

Coverage ↔ Precision ↔ GPU / Memory Cost

Increasing physical coverage, reducing voxel size, and increasing resolution all affect different parts of this balance. In particular, 512^3 should be selected deliberately: compared with 256^3 , it contains $8\times$ as many voxels and increases the raw runtime Aquarium texture from approximately 32 MiB to 256 MiB.

Start with 256^3 / 10 cm, test representative gameplay in Play Mode, and increase Aquarium cost only when the additional coverage or precision provides a visible benefit.

Rain VFX Follow Target

The `RainVFXFollowTarget` component keeps the active Rain VFX positioned around a moving target.

In the standard setup, the target is usually the player's Main Camera, but it can be another Transform representing the gameplay position and horizontal direction around which the rain should remain active.

Its purpose is simple: instead of placing one enormous rain volume over the entire level, the Rain VFX follows the selected target and keeps the active spawn area only where rain is currently needed.

This is especially useful for large levels and open environments. A relatively compact moving rain volume can create the visual impression of continuous rainfall without requiring a world-sized particle system.

The component updates the Rain Root in `LateUpdate()`, after normal object movement for the frame has already taken place.

In Short

`RainVFXFollowTarget` keeps the active Rain VFX around the selected Target and can:

- Follow the Target position.



- Maintain a configurable vertical offset.
- Follow the Target's horizontal yaw while ignoring pitch and roll.

This allows the active particle volume to travel through large scenes without requiring rain particles to be simulated across the entire world.

Why the Rain Follows a Moving Target

Rain only needs to be actively simulated around the gameplay area where it can currently be seen or interacted with.

Without a follow system, a large level would require either:

- A very large Rain VFX spawn volume covering the playable area.
- Custom logic for manually repositioning the effect.
- Multiple independent rain systems distributed throughout the world.

These approaches can introduce unnecessary particle simulation and rendering cost.

`RainVFXFollowTarget` instead keeps one Rain VFX centered around the selected gameplay target as it moves through the scene.

The Rain VFX movement is independent from the collision and masking geometry. World geometry, SDF Sources, Roof Mask, Water Mask, and Depth Buffer Collision continue to determine how the moving rain particles interact with the environment.

Target

`Target` is the Transform that the Rain VFX follows.

In the standard setup, this should normally be the active player camera or another Transform representing the player's current position and viewing direction.

Every frame, the script reads:

- The Target position.
- The Target forward direction.

and uses them to reposition and orient the Rain Root.

For top-down or orbit-camera setups, remember that the Rain Root follows the Target's actual X/Z position. If the physical rendering camera is positioned far from the gameplay center, assigning it directly will shift the rain volume by the same horizontal offset.

When available, use a camera rig or pivot Transform located near the active gameplay area that also provides the desired horizontal direction.



Rain Root

Rain Root is the Transform containing the Rain VFX that should follow the Target.

The script does not move the **RainVFXFollowTarget** GameObject itself. It specifically moves the assigned **Rain Root**.

This keeps the follow logic independent from the hierarchy object containing the controller.

The references can also be assigned through:

```
SetReferences(Transform followTarget, Transform targetRainRoot)
```

which sets both the Target and Rain Root references.

Height Offset

Height Offset adds a vertical offset above the Target position.

The resulting Rain Root position is: **Target Position + Height Offset on Y**

The script keeps the same X and Z coordinates as the Target and adds only the configured vertical distance.

This allows the Rain VFX spawn volume to remain above the player instead of being centered directly at camera height.

For example, if the player walks through a large level, the Rain Root follows horizontally while remaining at the configured height above them.

Yaw-Only Rotation

The Rain Root follows the Target's **horizontal yaw direction only**.

Pitch and roll are intentionally ignored.

Internally, the Target's forward direction is projected onto the horizontal ground plane:

```
Vector3.ProjectOnPlane(target.forward, Vector3.up)
```

The Rain Root is then rotated to face that horizontal direction.

This prevents vertical or rolled Target orientation from tilting the complete Rain VFX hierarchy.

With yaw-only following:

- Horizontal Target rotation changes the Rain Root yaw.
- Upward or downward Target orientation does not tilt the rain backward or forward.
- Target roll does not rotate the weather volume sideways.



The rain therefore remains vertically aligned with the world while still following the Target's horizontal direction.

Why Follow Yaw at All?

Following position alone would keep the Rain VFX around the Target, but it would not allow a directional spawn volume to follow the Target's horizontal orientation.

Yaw following is useful when the rain spawn area is intentionally biased toward a particular viewing or movement direction rather than distributed equally in every horizontal direction.

For example, a rectangular spawn area can extend farther in front of the Target than behind it. As the Target rotates, the Rain Root rotates with it and keeps that spawn area aligned with the intended direction.

In the standard camera-follow setup, this keeps the useful spawn area oriented toward the visible scene. Custom setups can instead use another suitable Transform as the Target.

Why Pitch and Roll Are Ignored

A normal gameplay camera can rotate independently from the physical orientation of the world.

For example, the player may:

- Look almost straight upward.
- Look toward the ground.
- Use camera shake.
- Apply recoil.
- Use a camera system with temporary roll.

These rotations should not normally change the world's rain direction or tilt the complete rain spawn volume.

For this reason, `RainVFXFollowTarget` extracts only the horizontal direction from the Target and deliberately discards vertical pitch and roll.

The actual visual direction of falling rain can still be controlled separately by the Rain VFX settings, including wind.

Update Order

The Rain Root is updated in `LateUpdate()` rather than `Update()`.

This is useful for camera-follow behavior because character movement and most normal gameplay updates have already occurred before the Rain Root is repositioned.



The Rain VFX therefore follows the Target's latest transform for the current frame rather than using an earlier position before movement was completed.

Vertical Target Directions

When the Target points almost perfectly upward or downward, its forward vector may have almost no horizontal component.

In that case, the script cannot calculate a meaningful yaw direction. Instead of forcing an unstable rotation, it keeps the existing Rain Root rotation and updates only its position.

This prevents sudden rotation changes when the Target has little or no usable horizontal forward direction.

Relationship With the Aquarium

`RainVFXFollowTarget` and `AquariumSDFController` are separate systems with different responsibilities.

- `RainVFXFollowTarget` positions and horizontally rotates the Rain VFX spawn hierarchy around its Target.
- `AquariumSDFController` maintains the local runtime SDF collision volume using its own assigned tracking/camera reference.

The follow component does not create the Aquarium, register SDF Sources, generate SDF data, or bind SDF textures to the VFX. Its only responsibility is maintaining the Rain Root Transform.

In the standard setup, both components can use the Main Camera as their reference.

They are not required to use the same Transform. The Aquarium reference determines the position of the local SDF volume and the direction used by Aquarium Camera Culling, while the Rain Follow Target determines the position and horizontal orientation of the Rain Root.

Custom top-down, isometric, orbit-camera, or camera-rig setups may therefore use different references when those responsibilities need to be controlled independently.

Runtime Reference Setup

The component exposes its current references as read-only properties:

- Target
- RainRoot
- HeightOffset

and provides:



`SetReferences(Transform followTarget, Transform targetRainRoot)`

for assigning the two Transform references from setup or runtime code.

`Height Offset` is currently serialized and readable through the public property, but the supplied script does not expose a runtime setter for it.

Roof, Water & Ground Mask Controllers

The Rain VFX uses three top-down height-mask controllers with different responsibilities:

- **Roof Mask Controller** — provides large-area overhead rain blocking outside the local Aquarium.
- **Water Mask Controller** — provides the water-surface height used for Ripples and Splash handling.
- **Ground Mask Controller** — provides ground height used to reduce unnecessary Aquarium SDF processing below suitable terrain.

All three systems use top-down orthographic cameras and 2D height-mask Render Textures rather than full three-dimensional SDF geometry.

The Weather and Water **Occluder Proxies** described in these mask chapters are technical geometry used only by the corresponding top-down mask cameras. They are separate from custom **SDF proxy geometry**, which is baked into `Texture3D` Sources and is described later in the SDF authoring sections.

In a normal setup, the required cameras, layers, renderers, and references should be configured through the VFX Setup Assistant rather than assembled manually.

Roof Mask Controller

The **Roof Mask Controller** renders a top-down height representation of weather-blocking geometry and passes that mask to the Rain VFX.

It is primarily responsible for rain occlusion **outside the Aquarium**, where a full local SDF is no longer available.

The supplied setup uses geometry from the dedicated `VFXWeatherOccluders` layer. Your original gameplay geometry should normally remain on its existing layer; use the **Create Weather Occluder Proxy** workflow when an object needs to participate in Roof Occlusion.

Occlusion Camera

`Occlusion Camera` references the dedicated top-down camera used to render the Roof Mask.

The controller automatically configures it as an orthographic camera facing downward and keeps it centered on the assigned target.



This reference is already prepared in the supplied Aquarium setup and normally should not be replaced manually.

Roof Layer Mask

Roof Layer Mask defines which layers contribute geometry to the Roof Mask.

For the standard setup, this should use: **VFXWeatherOccluders**

The Setup Assistant configures the required layer and corresponding rendering setup automatically.

Do not move your original visible gameplay object directly onto this layer. Use a generated Weather Occluder Proxy instead.

Camera Height Offset

Camera Height Offset controls how far above the target the Roof Mask camera is positioned.

The supplied value is **20 m**.

In most projects, the supplied value can be left unchanged. Increase it only if the mask camera needs to capture relevant roof geometry located higher above the target.

Far Plane

Far Plane defines the vertical capture range of the top-down camera.

The supplied value is **100 m**.

This determines how far downward the mask camera can see from its current position. Very large vertical scenes may require a larger value, but there is normally no reason to increase it unnecessarily.

Area Size

Area Size controls the horizontal world-space coverage of the Roof Mask.

The supplied value is **50 m**.

The Roof Mask uses independent coverage and does **not** need to match the Aquarium size.

A larger Area Size allows roofs farther from the player to continue blocking rain, but the same Render Texture resolution is then spread across a larger area: **Larger Area Size = lower mask precision.**

This makes **Area Size** the main Roof Mask quality-versus-coverage control.



For example, if you only need nearby buildings to block distant rain, a smaller Roof area can provide sharper mask edges. Large outdoor scenes may benefit from greater coverage at the cost of some precision.

Runtime Behavior

The controller follows the target in X/Z and updates the Roof Mask bounds sent to the Rain VFX as the target moves.

The supplied Aquarium setup propagates the shared tracking target and Rain VFX references automatically, so these references are not normally edited directly.

Water Mask Controller

The Water Mask Controller captures a top-down height representation of water geometry and passes it to the Rain VFX.

Unlike the Aquarium SDF, this is not a volumetric collision system. For each covered position, the Water Mask provides the surface height that the Rain VFX should treat as water.

This height is used for water-specific particle behavior such as:

- Spawning Ripples at the detected water surface.
- Removing Splashes that enter water so they do not continue normal SDF sliding below the surface.

The standard setup captures geometry from the dedicated **VFXWaterOccluders** layer. Original water objects should remain on their normal rendering layers; use the Water Occluder Proxy workflow for geometry that should contribute to the Water Mask.

Because the mask captures proxy mesh geometry, GPU-only shader displacement is not represented automatically. This limitation and the corresponding proxy workflow are described in more detail later.

Occlusion Camera

Occlusion Camera is the dedicated top-down camera used to render the Water Mask.

The controller keeps it orthographic, facing downward, and centered on the assigned target.

The supplied prefab already contains the required camera reference, so it normally does not need to be changed manually.

Water Layer Mask

Water Layer Mask defines which layers are rendered into the Water Mask.



The standard setup uses:

VFXWaterOccluders

The **VFX Setup Assistant** configures the required technical layer and rendering setup.

Do not move the original visible water object directly to this layer. Use a **Water Occluder Proxy** instead.

Height Offset

Height Offset controls how far above the target the Water Mask camera is positioned.

The supplied value is **20 m**.

It should be high enough for the camera to remain above the water surfaces that need to be captured. In most scenes, the supplied value can be left unchanged.

Far Plane

Far Plane controls the vertical capture range of the Water Mask camera.

The supplied value is **100 m**.

Increase it only if relevant water surfaces fall outside the current vertical capture range.

Area Size

Area Size controls the horizontal world-space coverage of the Water Mask.

The supplied value is **50 m**.

The Water Mask has its **own independent coverage area** and is not tied to the Aquarium size. This is useful because visible water interactions can remain relevant farther from the player than the local SDF volume.

Increasing **Area Size** allows Ripples and other water-related effects to work across a wider area, but the same Render Texture resolution is then spread across more world space.

Larger Area Size = wider coverage, lower spatial precision.

Use an area large enough to cover the visible water surfaces where rain interaction should occur, while keeping enough precision for the required detail.

Runtime Behavior



During runtime, the Water Mask follows the assigned target in X/Z while maintaining its configured **Area Size**.

The controller continuously updates the Water Mask data and passes the current mask bounds and texture information to the Rain VFX.

In the normal supplied setup, the mask camera and internal controller references are already prepared by the prefab. The Aquarium propagates the assigned tracking and Rain VFX references, while the Setup Assistant configures the required technical layer and rendering setup.

Manual changes are usually only necessary when adjusting **Area Size**, **Height Offset**, or **Far Plane** for a particular scene.

Ground Mask Controller

The Ground Mask Controller is an optional Aquarium optimization. It does **not** provide rain collision or rain occlusion.

It captures a top-down ground-height mask that allows the Aquarium compute process to skip voxels below suitable solid ground surfaces.

This can significantly reduce unnecessary SDF work when part of the Aquarium extends below terrain, streets, or other surfaces where no useful SDF collision needs to remain active.

Occlusion Camera

Occlusion Camera references the dedicated top-down camera used to capture ground height.

The controller keeps the camera centered on the assigned target and automatically uses the Aquarium-controlled area size.

The supplied Aquarium prefab already contains this camera reference.

Ground Layer Mask

Ground Layer Mask is the most important user-facing setting in this controller.

It defines which project layers should be interpreted as **solid ground**.

Typical suitable geometry includes:

- Terrain.
- Streets.
- Large outdoor ground meshes.
- Solid surfaces where nothing below them needs SDF collision.



Do **not** include surfaces such as:

- Roofs.
- Bridges.
- Upper floors.
- Elevated platforms.
- Any surface with valid playable or SDF geometry underneath it.

Once a surface is represented by the Ground Mask, the Aquarium may skip SDF evaluation below its captured height. Using an upper floor as ground can therefore remove valid SDF collision from the space underneath it.

The Ground Mask is an optimization only; it is not a replacement for Roof Occlusion or normal SDF collision.

Camera Height Offset

Camera Height Offset controls how far above the target the Ground Mask camera is placed.

The supplied value is **30 m**.

The camera needs to remain high enough to capture the relevant ground surfaces below it.

Far Plane

Far Plane defines the Ground Mask camera's vertical capture range.

The supplied value is **100 m**.

Normally the supplied value should be sufficient unless the scene contains very large vertical elevation differences.

Area Size

Ground Mask **Area Size** is controlled automatically by the **AquariumSDFController**.

It is read-only in the normal workflow and should not be configured independently.

This ensures that the ground-height data covers the same horizontal region currently being processed by the Aquarium.

Changing Aquarium Resolution or Voxel Size therefore changes Ground Mask coverage as well.



Mask Precision

The Roof, Water, and Ground systems all use top-down height masks stored in Render Textures.

Their horizontal spatial precision depends on the relationship between the physical coverage area and the Render Texture resolution:

Larger Area Size + same Render Texture resolution = lower pixel density and lower spatial precision.

Each mask controller exposes the same read-only **Mask Precision** information:

- **Texture Resolution** — the width and height of the current mask texture.
- **Pixels Per Meter** — how many texture pixels represent one meter of world space.
- **Pixel Size** — the approximate world-space size represented by a single mask pixel.

For example, a 2000 × 2000 mask covering a 30 m area provides approximately 66.67 pixels per meter, meaning one pixel represents about 1.5 cm of world space.

Higher pixel density provides more precise mask boundaries. Lower pixel density may become noticeable around thin geometry, small water surfaces, roof edges, and other fine details.

The way **Area Size** is controlled differs between the three masks:

- **Roof Mask** — **Area Size** is configured independently in the Roof Mask Controller.
- **Water Mask** — **Area Size** is configured independently in the Water Mask Controller.
- **Ground Mask** — **Area Size** is controlled automatically by the **AquariumSDFController** and is not intended to be configured independently.

This means Roof and Water coverage can be tuned separately from the Aquarium, while Ground Mask coverage follows the Aquarium region it is intended to optimize.

Use the Inspector's **Pixels Per Meter** and **Pixel Size** values to judge whether the current mask resolution is appropriate instead of increasing Render Texture resolution unnecessarily.

The main trade-off is: **Coverage ↔ Precision ↔ Render Texture Cost.**

Ground Layer Configuration

The **Ground Layer Mask** is configured manually in the **Ground Mask Controller**. You can select a single layer or multiple layers, depending on how ground-related geometry is organized in your project.

The way this configuration is applied differs between **URP** and **HDRP**.



URP

In the default URP setup, the Ground Mask uses the supplied **Ground Occlusion Renderer Data**, which is shared across scenes.

After you select the required layers in the **Ground Mask Controller**, the Setup Assistant can apply that existing **Ground Layer Mask** to the Ground occlusion camera, assign the supplied Ground Occlusion Renderer, and update the Ground Render Objects Layer Mask to match. The Assistant does not decide which Ground layers should be used — that selection is always made by you in the controller.

Because the supplied Ground Occlusion Renderer Data is a shared project asset, its Render Objects Layer Mask is also shared. For this reason, URP projects should normally use a consistent set of ground-related layers across scenes.

If a particular scene requires a completely different Ground configuration, you can duplicate the supplied Ground Occlusion Renderer Data, but this becomes an **advanced manual workflow**.

The duplicated Renderer Data must be added manually to the active URP Asset's **Renderer List**, and that scene's **Ground Occlusion Camera** must be configured to use the corresponding renderer.

The **VFX Setup Assistant only manages the supplied default Ground Occlusion Renderer Data**. Its validation and automatic configuration use the predefined path to:

`Renderer_GroundOcclusionMask.asset`

Because of this, a scene intentionally using a duplicated custom Ground Occlusion Renderer may still show the corresponding Ground setup step in the Assistant as **incomplete or unresolved**, even when the custom configuration is working correctly. This is expected for this advanced workflow and does not by itself indicate an incorrect scene setup.

HDRP

HDRP does not use the shared URP Ground Occlusion Renderer Data asset.

The **Ground Layer Mask** is still selected manually in the **Ground Mask Controller**, and you can select one or multiple layers. The Assistant checks that a Ground Layer Mask has been selected and then applies that selected mask to the Ground masking setup for the current scene.

When applying the Ground configuration, the HDRP Assistant uses the selected **Ground Layer Mask**, configures the Ground occlusion camera, assigns the supplied HDRP height shader, ensures the required HDRP camera data exists, and synchronizes the Ground Custom Pass setup.

Unlike the default URP setup, HDRP does not rely on a shared Ground Renderer Data asset for this configuration. Different HDRP scenes can therefore use different **Ground Layer Mask** selections on their own Ground Mask Controllers.



Creating Custom Occluders (Proxy Workflow)

Turning your custom 3D models into rain shelters is straightforward, but it requires a specific workflow to avoid breaking your game's logic and rendering.

⚠ Terminology Note: Mask Proxies vs. SDF Proxies

The **Weather Occluder Proxy** described in this chapter belongs to the top-down Roof Mask system. It is rendered by the dedicated mask camera through the `VFXWeatherOccluders` layer and is **not** baked into an SDF `Texture3D`.

This is separate from custom **SDF proxy geometry**, which is geometry prepared specifically for SDF baking and used as an `AquariumSource`.

Both workflows may use simplified or hidden geometry, but they serve different systems and should not be confused.

The Problem with Technical Layers

Roof masking only reads geometry from the dedicated `VFXWeatherOccluders` layer. The Main Camera normally excludes this layer so the technical proxy geometry remains invisible to the player.

You should therefore **not move your original gameplay objects to `VFXWeatherOccluders`** because:

1. **It can break game logic:** Changing an object's layer may affect Physics Collision Matrix rules, raycasts, custom layer masks, triggers, or scripts that rely on the object's original layer.
2. **It can break rendering:** Because the Main Camera excludes the technical occluder layer, moving the original object there may make it disappear from the normal scene render.

Using the "Create Weather Occluder Proxy" Tool

To solve this, the asset includes an Editor tool that creates a separate render-only copy of your object. Only the generated proxy is placed on `VFXWeatherOccluders`, while the original object remains on its existing layer with its original materials, scripts, colliders, and gameplay behavior.

How to use it:

1. Select your target object or **multiple objects** in the Hierarchy. The tool only works with scene objects or objects opened in Prefab Mode; Project assets cannot be processed directly.
2. Right-click and navigate to: `GameObject > Skulmovski Studio > Proxy > Create Weather Occluder Proxy`.



3. The generated proxy is created as a child of the original object and its local position, rotation, and scale are reset so it overlaps the source hierarchy. If an existing `<ObjectName>_WeatherOccluderProxy` is found, it is deleted and replaced.
4. The entire operation is registered as a single Undo group.

 **Prefab Workflow Note:** If you run this tool on a Prefab instance in a scene, the proxy becomes an override on that specific instance. If every instance should contain the same proxy, open the Prefab in **Prefab Mode** and create the Weather Occluder Proxy there instead.

How it Works Under the Hood (Safety & Optimization)

The Weather Proxy preserves the structural geometry required by the Roof Mask while removing components that have no purpose in the technical masking pass.

- **Render-Only Hierarchy:** The complete hierarchy is cloned, but only `Transform`, `MeshFilter`, `MeshRenderer`, and `SkinnedMeshRenderer` components are intentionally retained.
- **Component Stripping:** Scripts, colliders, rigidbodies, audio components, particle systems, and other unrelated components are removed from the proxy.
- **Dependency Safety:** `[RequireComponent]` dependencies are checked while components are removed. If a circular dependency prevents safe removal, the remaining component is preserved and a warning is logged instead of forcing an unsafe delete.
- **Technical Layer Assignment:** Every object in the generated proxy hierarchy is moved to `VFXWeatherOccluders`.
- **Render Cost Optimization:** Shadow casting, shadow receiving, light probes, motion vectors, dynamic occlusion, and `Contribute GI` are disabled because they provide no value for height-mask rendering.
- **Technical Material:** The proxy renderers use the provided occluder material instead of the original gameplay materials.

The original object remains responsible for normal gameplay and rendering. The proxy exists only to provide geometry to the Roof Mask.

How Animated Objects Work (SkinnedMeshRenderer)

Weather Occluders can also represent animated characters and other hierarchies containing moving child transforms.

- **Root Movement:** The generated proxy is parented directly under the original object, so movement, rotation, and scale of the root are inherited automatically through the normal Unity hierarchy.
- **Child Transform Syncing:** If the original contains child transforms, the tool adds a lightweight `RoofOccluderProxyBoneFollower` component to the proxy root. It synchronizes the matching child transforms so animation can be mirrored onto the technical proxy hierarchy.
- **Simple Objects Stay Simple:** If the selected object contains no child transforms, the follower is not added.
- **Hierarchy Validation:** If the cloned hierarchy no longer matches the original transform count, pose syncing is not configured and the tool logs a warning telling you to recreate the proxy.



⚠ **CRITICAL: When to Recreate the Weather Proxy**

The generated proxy should be treated as a **structural snapshot** of the source hierarchy.

Recreate it if you:

- **Change the mesh** used by a [MeshFilter](#) or [SkinnedMeshRenderer](#).
- **Add or remove children** from the hierarchy.
- **Change the rig or skeleton structure**.
- **Replace major pieces of the source hierarchy** after the proxy was generated.

Simply run **Create Weather Occluder Proxy** again. The existing proxy is replaced automatically.

Creating Water Occluders (Proxy Workflow)

Water masking uses a similar technical proxy workflow, but its purpose is different from both Weather Occluder Proxies and SDF proxy geometry.

The Water Proxy exists only to provide mesh surface height to the top-down Water Mask. It is not an SDF Source and does not provide volumetric water collision.

The captured height is used to:

- Spawn Ripples at the detected water surface.
- Remove Splashes that enter water so they do not continue normal SDF sliding below the surface.

The Water Proxy is intentionally simpler than the Weather Proxy because the Water Mask only requires a suitable geometric surface height rather than a synchronized general-purpose occlusion hierarchy.

Why Water Uses Proxy Geometry

Gameplay water often uses transparency, refraction, complex shading, custom rendering, or GPU-driven surface deformation.

The Water Mask does not need the final visual appearance of the water. It needs mesh geometry from which a reliable top-down interaction height can be captured.

For this reason, the original water object should remain on its normal gameplay/rendering layer while a dedicated technical proxy is placed on [VFXWaterOccluders](#).

Visual water and Water Mask geometry can therefore remain independent when necessary.



Using the "Create Water Occluder Proxy" Tool

The Water Proxy tool creates a separate render-only copy of the selected water geometry while leaving the original water object unchanged.

How to use it:

1. Select your water object or **multiple objects** in the Hierarchy. Project assets are skipped; use a scene object or open the Prefab in Prefab Mode.
2. Right-click and navigate to: `GameObject > Skulmovski Studio > Proxy > Create Water Occluder Proxy`.
3. The tool creates `<ObjectName>_WaterOccluderProxy` as a child of the original object and resets its local transform so it matches the source geometry. If a Water Proxy already exists, it is replaced automatically.
4. The complete operation is registered as a single Undo group.

💡 **Prefab Workflow Note:** As with Weather Proxies, create the proxy in **Prefab Mode** if every instance of a reusable water prefab should inherit it. Creating it on a scene instance affects only that instance.

How it Works Under the Hood

The Water Proxy is deliberately more restrictive than the Weather Proxy.

- **MeshFilter Geometry Only:** After stripping, the supported geometry consists of `Transform`, `MeshFilter`, and `MeshRenderer`. `SkinnedMeshRenderer` is not supported by the Water Proxy workflow.
- **Supported Geometry Check:** After generation, the tool verifies that at least one valid `MeshFilter + MeshRenderer` pair with a mesh remains. If not, it warns that the proxy will not contribute to the Water Mask.
- **Technical Layer Assignment:** The complete generated hierarchy is placed on `VFXWaterOccluders`.
- **Component Stripping:** Scripts, colliders, rigidbodies, particle systems, and other unrelated components are removed while `[RequireComponent]` dependencies are handled safely.
- **Render Cost Optimization:** Shadow casting, shadow receiving, light probes, motion vectors, dynamic occlusion, and `Contribute GI` are disabled.
- **Technical Material:** Original visual water materials are replaced with the supplied occluder material because the proxy exists only to provide geometry to the masking system.

Moving Water Objects

The Water Proxy is parented directly under the original object.

This means **root-level movement, rotation, and scaling are inherited automatically** through standard Unity parenting.



However, unlike the Weather Proxy, the Water Proxy does **not** create a child-transform or bone synchronization component. Animated or independently moving child transforms are therefore not synchronized after the proxy is created.

For rigid water objects or surfaces that move as a single hierarchy root, no additional synchronization is required.

⚠ **IMPORTANT: Shader-Deformed Water Surfaces**

The Water Mask captures the Water Proxy's mesh height, not the final GPU-deformed visual water surface.

Vertex displacement, tessellation, Shader Graph deformation, and other shader-only wave effects are therefore not reproduced automatically.

If the Rain VFX must react to large changes in the visible water height, use proxy geometry that approximates the physical interaction surface the rain should follow.

This limitation is a consequence of using an inexpensive top-down geometric height mask rather than reconstructing the final GPU-deformed water surface.

Why Water Uses a Separate Proxy

Weather and Water proxies use separate technical layers because they have different responsibilities.

- **VFXWeatherOccluders** represents geometry that should **block falling rain** through Roof Occlusion.
- **VFXWaterOccluders** represents geometry that should be treated as a **water surface** for Ripples and Splash handling.

Keeping these systems separate allows the Rain VFX to distinguish between geometry that blocks rain and geometry that represents water.

Both of these are **mask proxies** and are separate from the custom **SDF proxy geometry** used by the Aquarium SDF system.

⚠ **CRITICAL: When to Recreate the Water Proxy**

The Water Proxy should also be treated as a **structural snapshot** of the source geometry.

Recreate it if you:

- **Change `MeshFilter.sharedMesh`.**
- **Add or remove child mesh objects.**
- **Replace the water geometry with a different mesh or hierarchy.**

You do **not** need to recreate it simply because the original object's root Transform moves, rotates, or scales — the child proxy follows that transform automatically.



If the visual water changes only through shader deformation, recreating the proxy will not capture that deformation because the Water Proxy is based on the underlying mesh geometry.

SDF Editor Tools

The asset includes four dedicated Editor tools for preparing, baking, reviewing, and maintaining SDF Sources.

Together they form the normal SDF authoring workflow:

SDF Source Marker → SDF Source Baker → SDF Scene Overview → SDF Source Cleaner

Each tool has a different responsibility:

- **SDF Source Marker** — defines which hierarchy objects should become **AquariumSource** components and how they should be baked.
- **SDF Source Baker** — creates, previews, and commits the actual baked SDF Texture3D assets.
- **SDF Scene Overview** — provides a scene-wide overview of existing Sources and their bake state.
- **SDF Source Cleaner** — safely removes Sources and cleans up baked Texture3D assets that are no longer used.

For normal authoring, most objects are first prepared in the Marker and then baked in the Baker.

SDF Source Marker

The **SDF Source Marker** is the main tool for defining which objects in a hierarchy should participate in the Aquarium SDF system.

Open it through:

Tools > Skulmovski Studio > SDF Source Marker

or from a selected hierarchy object:

GameObject > Skulmovski Studio > Open SDF Source Marker

When opened from the Hierarchy, the selected object becomes the initial root of the displayed tree.

What the Marker Does

The Marker does not generate SDF textures itself.



Its purpose is to define the Source layout and authoring settings that will later be used by the SDF Source Baker.

For each object, you can decide:

- Whether it should become an **AquariumSource**.
- Which bake resolution it should use.
- Whether it should be treated as Stationary.
- Whether its renderer should be hidden after a successful bake is committed.
- How nested Sources divide geometry ownership.

The geometry being marked may be the original rendered model or separate geometry prepared specifically for SDF collision. The Marker treats both workflows in the same way.

This makes the tool suitable for normal scene geometry, dedicated SDF proxies, complex prefabs, buildings, vehicles, and other hierarchies that need to be divided into multiple Sources.

Root Object

The **selected Root** defines which hierarchy the Marker displays.

Only the Root and its descendants are shown in the tree.

Selecting another object in the Unity Hierarchy normally changes the Marker Root automatically.

Use the **Root Lock** button when you want to keep working with the current hierarchy while selecting objects inside it. Once locked, normal Hierarchy selection no longer replaces the Marker Root.

Hierarchy Tree

The Marker reproduces the selected Transform hierarchy as an expandable tree.

Each row represents one object and provides the controls required to configure its final SDF Source state.

The tree also preserves useful hierarchy relationships so nested Sources can be configured directly in context.

Marking an SDF Source

Enable the Source toggle on an object to mark it as an **AquariumSource**.

The Marker stores these changes temporarily until **Apply** is pressed. This allows several additions, removals, or setting changes to be prepared together before modifying the actual hierarchy.

The Apply button displays a summary of pending changes, including how many Sources will be:



- Added.
- Removed.
- Updated.

Bake Resolution

Every Source has its own bake resolution.

Available SDF Source resolutions are: 16^3 , 32^3 , 64^3 , 128^3 and 256^3 .

The Marker uses 64^3 as the initial resolution for newly marked Sources.

Choose the Source resolution according to the physical size of the Source volume and the geometric detail that needs to be preserved.

Larger Sources generally require higher bake resolutions to maintain comparable voxel density, while smaller Sources can often use lower tiers efficiently.

Resolution should therefore not be reduced simply because a lower tier still produces a recognizable shape. The baked SDF must preserve the collision-relevant surfaces with sufficient voxel density.

Use the Source size as a starting point, then verify the result in the **SDF Source Baker Preview** before committing the bake.

Source Size as a Starting Point

When targeting approximately 10 cm per voxel, the following Source sizes can be used as practical starting points:

- Up to approximately 1.6 m on the largest axis — 16^3
- Up to approximately 3.2 m — 32^3
- Up to approximately 6.4 m — 64^3
- Up to approximately 12.8 m — 128^3
- Up to approximately 25.6 m — 256^3

These values are starting points rather than fixed rules. Thin surfaces, small openings, angled geometry, or other important details may require greater voxel density.

Always verify the final result in the **SDF Preview**.

Stationary

The Stationary option should be enabled for Sources that are not expected to move or rotate during normal gameplay.



Stationary Sources can use cached transform information as a runtime optimization.

If a Stationary Source is moved or rotated manually at runtime, its cached matrix must be invalidated before the Aquarium can use the new transform correctly.

Do not enable Stationary simply because an object starts the scene without moving — use it only when the object is genuinely intended to remain stationary during normal runtime operation.

Hide Renderer After Bake

Hide Renderer After Bake is intended primarily for geometry that exists only to provide an SDF representation and should not remain visible in the final scene.

When enabled, the renderer is disabled after a successful SDF bake is committed.

The underlying mesh data is kept intact so the Source can still be previewed, adjusted, and baked again later. The option does not delete the **MeshFilter** or remove the authoring geometry.

This is especially useful for dedicated SDF proxy geometry, where the collision representation may intentionally be thicker, simpler, or otherwise different from the visible model.

Disabling **Hide Renderer After Bake** does not automatically re-enable a renderer that was previously hidden. Re-enable the renderer manually when it needs to become visible again.

Nested Source Ownership

Nested Sources are an important part of the SDF workflow.

Geometry belongs to the **nearest AquariumSource above it in the hierarchy**.

When a child object becomes its own Source, the geometry owned by that child is no longer included in the parent Source bake.

For example:

- *A building can use separate Sources for different floors.*
- *A vehicle can use one Source for the body and separate Sources for movable parts.*
- *A large modular structure can be divided into smaller, tighter SDF volumes.*

The Marker validates these ownership changes before applying them and prevents configurations that would leave a Source without any bakeable geometry.

NO GEO

NO GEO means that a Source does not currently own any bakeable geometry.



This can happen when:

- The object contains no supported mesh geometry.
- All mesh geometry below it belongs to nested **AquariumSource** objects.
- The Source is placed on an empty organizational object.

A Source with **NO GEO** cannot produce a meaningful SDF bake.

Either remove the unnecessary Source or reorganize the hierarchy so it owns at least one supported renderer.

BLOCKED

BLOCKED means that a requested Source change would create an invalid ownership configuration.

For example, marking a nested object as a new Source may attempt to take the last remaining bakeable geometry away from an existing parent Source.

The Marker prevents this operation instead of allowing an invalid Source layout to be created.

Review the hierarchy and decide which Source should own the affected geometry before continuing.

Search

The search field filters the hierarchy tree by object name.

Matching objects remain visible and their parent hierarchy is automatically expanded so the result can be located in context.

When the search field is cleared, the previous expanded/collapsed state is restored.

Apply

Use **Apply** when the Source layout is ready.

The Marker then:

- Adds newly requested **AquariumSource** components.
- Updates Source authoring settings.
- Removes Sources that were unmarked.
- Recalculates final Source geometry bounds.

Renderer visibility is not changed simply by marking a Source. **Hide Renderer After Bake** takes effect only after the corresponding SDF bake has been successfully committed.



Apply & Open Baker

Use **Apply & Open Baker** when the Source layout is ready and you want to continue directly into the baking workflow.

The Marker applies the pending hierarchy changes and opens the **SDF Source Baker** using the same hierarchy root.

For Sources whose geometry is already prepared:

Mark Sources → Configure Settings → Apply & Open Baker

SDF Source Baker

The **SDF Source Baker** creates the actual Texture3D data used by [AquariumSource](#).

Open it through:

`Tools > Skulmovski Studio > SDF Source Baker`

or:

`GameObject > Skulmovski Studio > Open SDF Source Baker`

When opened from the Hierarchy or directly from the Marker, it uses the selected hierarchy as its Root.

Its main workflow is deliberately separated into three stages:

Test Bake → Preview → Apply

This lets you inspect the generated SDF before replacing or creating the committed bake asset.

Root Lock

Like the Marker, the Baker follows the current hierarchy selection unless its Root is locked.

Use **Root Lock** when you need to select and inspect individual Source objects without replacing the hierarchy currently shown in the Baker.

Baker Settings

The Baker includes additional settings for advanced control of the bake process.



Sign Passes

Controls the number of sign-processing passes used by the bake.

The supported range is 1–4, with 4 used as the default.

For normal use, keep the default unless you have a specific reason to change it.

Box Padding

Box Padding adds empty space around the Source geometry before the final cubic SDF bake volume is generated.

The default value is: 0.1 m

Padding is not only a bounds margin. It can also have a significant effect on the quality of the generated SDF.

If the Preview shows unexpected stair-stepping, stripe-like or grid-like gaps, clipped areas, broken edges, or other artifacts that do not match the source mesh, try increasing **Box Padding** and run another Test Bake before assuming that the geometry or bake resolution is the problem.

In some cases, a Source that produces a visibly malformed SDF with a small Padding value can bake correctly after substantially more space is added around the geometry, without changing the mesh or resolution.

Increase Padding progressively and inspect the result after each Test Bake.

Do not assume that the default value is optimal for every Source.

However, Padding also increases the physical size of the cubic bake volume. The same number of voxels is then distributed across a larger area, reducing effective voxel density. Excessive Padding can therefore make small details less precise.

The practical goal is to use enough Padding for a clean and stable bake while keeping the bake volume reasonably tight.

Open Preview After Test Bake

When enabled, a successful Test Bake automatically opens the SDF Preview.

This is recommended during normal authoring because every important Source should be visually checked before committing.



Batch Bake Limits

The Baker also contains safeguards for very large batch operations, including:

- A configurable maximum temporary batch-cache size.
- A warning threshold for unusually large Source counts.

These settings are intended to prevent accidental large bake operations from consuming excessive temporary disk space or Editor resources.

Why Test Bake First

A Test Bake generates a temporary SDF for inspection without replacing the committed **Texture3D** currently used by the Source.

This is important because SDF quality depends on the complete representation being baked, including:

- Geometry shape and topology.
- Source dimensions.
- Surface thickness.
- Surface orientation.
- Bake resolution.
- Box Padding.
- How tightly the cubic bake volume fits the useful geometry.

A higher bake resolution is not a universal solution.

If the geometry itself is poorly suited to voxel SDF representation, a tighter Source layout, simplified geometry, or a dedicated SDF proxy may provide a better result with less texture memory.

Test Bake therefore allows the geometry, Source layout, and bake settings to be evaluated together before any final asset is committed.

Test Bake

Use **Test Bake** to generate a temporary SDF for a Source.

The temporary result is stored in the project's Library cache rather than as the final committed asset. It is therefore working data used only by the Baker workflow and is not intended to be distributed with the project.

After a successful Test Bake, the Source enters the **PREVIEW READY** state.

If **Open Preview After Test Bake** is enabled, the preview opens automatically after the bake completes.



Preview

The Preview allows the generated SDF **Texture3D** to be inspected before it becomes the committed runtime asset.

Do not use the Preview only to judge whether the selected resolution is high enough. Use it to verify whether the chosen geometry itself produces a clean and useful SDF representation.

For detailed inspection, switch to the **SDF** tab in the Preview window and increase the Preview **Scale** when necessary. A larger visualization scale makes the baked surface easier to inspect and can reveal defects that are difficult to notice at the default display size.

The Preview **Scale** affects only the visualization in the Preview window. It does not modify the baked **Texture3D**, Source Transform, voxel size, or runtime collision.

Pay particular attention to:

- Thin and angled surfaces.
- Roof and wall edges.
- Corners.
- Small openings.
- Narrow gaps.
- Seams and connections between separate pieces.
- Unexpected holes or missing sections.
- Stair-stepping, stripes, grid-like gaps, or other malformed regions.
- Large empty areas inside the cubic bake volume.

If the generated SDF looks incorrect:

1. Inspect it closely in the **SDF** tab and increase Preview **Scale** if necessary.
2. Try increasing **Box Padding** and run another Test Bake. Padding can sometimes resolve severe stair-stepping, strip-like gaps, or other malformed bake patterns without changing the mesh or resolution.
3. Check whether the Source bounds are unnecessarily large.
4. Check whether the geometry itself is suitable for SDF baking.
5. Split large geometry into smaller, tighter Sources where appropriate.
6. Use dedicated SDF proxy geometry when the visual mesh does not produce a reliable result.
7. Increase bake resolution when additional voxel density is actually required.

A higher resolution is not a universal solution. If the SDF is already visibly malformed in the Preview, first investigate Padding, Source layout, and geometry representation instead of simply increasing resolution.

A visibly incorrect Preview should not be expected to become correct at runtime. The Aquarium uses the committed baked SDF data; it does not repair missing or malformed regions in the Source texture.

Evaluating SDF Geometry, Not Visual Appearance

The geometry used for SDF baking does not need to match the visible mesh in every detail.



The goal of the bake is to create a reliable collision representation, not to reproduce visual topology unnecessarily.

If the visual mesh produces a clean SDF directly, use it. If it does not, prepare a simpler or more suitable representation and bake that instead.

Dedicated SDF geometry may intentionally use different thickness, simplified shapes, extended boundaries, or overlapping sections when those changes produce a more reliable baked field.

⚠ Detailed proxy-authoring patterns are covered in SDF Proxy Geometry Authoring.

Apply

Once the preview is satisfactory, use **Apply** to commit the result.

The temporary bake is converted into the committed Texture3D asset and assigned to the corresponding [AquariumSource](#).

The committed texture is the version used by the runtime Aquarium.

A Test Bake therefore does not become runtime data until it has been applied.

Source Statuses

The Baker displays the current state of each Source using status badges.

NOT BAKED

The Source has no committed SDF Texture3D yet.

Run a Test Bake to generate one.

PREVIEW READY

A successful temporary bake exists and can be inspected.

Use Preview to check the result, then Apply when satisfied.

BAKED

The Source has a committed SDF Texture3D and its current bake state is valid.

NEEDS REBAKE

The current committed bake no longer matches the Source configuration.

A common reason is that the Source or one of its parents changed scale after baking.



Generate a new Test Bake and apply it again.

INVALID

The Source cannot currently be baked correctly.

One common example is a Source with no geometry that belongs to it.

The Baker provides guidance for **NO GEO** cases and can open the Cleaner when an invalid Source should simply be removed.

FAILED

The most recent bake attempt failed.

Correct the reported issue and use Retry / Test Bake again.

The current Baker exposes these six states directly in its UI.

SkinnedMeshRenderer Sources

The Baker supports Sources containing **SkinnedMeshRenderer** geometry.

However, the current deformed mesh is captured as a **static SDF snapshot** during baking.

Bone animation, blend shapes, and later runtime vertex deformation do not continuously rebuild the baked SDF.

Moving or rotating the complete Source is supported, but changing its actual deformed geometry requires a new bake.

Changing Bake Resolution

The resolution of each Source can be adjusted directly in the Baker before creating a new Test Bake.

Use higher resolutions only when the additional voxel density is actually necessary.

Source resolution applies to the entire cubic bake volume, not to one specific surface or mesh feature. Large or elongated Sources can therefore waste a large number of voxels even at high resolution.

Increasing resolution also grows **Texture3D** storage very quickly. Doubling resolution on every axis produces approximately **8×** as many voxels.

If a Source requires an unusually high resolution only to preserve a small portion of its geometry, first consider:

- Tighter Source bounds.
- Splitting the object into smaller Sources.



- Simplifying the SDF geometry.
- Using dedicated SDF proxy geometry.

Increasing resolution should generally be the solution when the representation is already appropriate but simply requires more voxel density.

Recommended Baker Workflow

For each important Source:

1. Confirm that the geometry and Source layout are appropriate for SDF collision.
2. Select the recommended bake resolution.
3. Run **Test Bake**.
4. Inspect the generated SDF in **Preview**.
5. Check important surfaces, edges, openings, corners, and seams.
6. If the result is poor, adjust the geometry representation, Source layout, padding, or resolution as appropriate.
7. Test Bake again.
8. Use **Apply** only when the generated SDF is satisfactory.
9. Verify the final rain interaction in Play Mode.

A green **BAKED** status means that a valid committed texture exists. It does not guarantee that the chosen geometry representation or resolution is visually sufficient for every collision case.

SDF Scene Overview

The **SDF Scene Overview** provides a single scene-wide view of all **AquariumSource** components in the currently active scene.

Open it through:

Tools > Skulmovski Studio > SDF Scene Overview

Unlike the Marker, Baker, and Cleaner, the Scene Overview does not work from one selected hierarchy Root. It scans the **entire active scene**.

Its purpose is inspection rather than authoring.

Use it when you want to quickly answer questions such as:

- How many SDF Sources exist in this scene?
- Which Sources have already been baked?
- Which Sources are still missing a bake?
- Are any committed Texture3D references invalid?
- How much disk space do the committed SDF textures referenced by this scene occupy?



Scene Summary

The top summary displays:

- **Sources** — total number of SDF Sources found in the active scene.
- **Baked** — Sources with a valid committed Texture3D.
- **Not Baked** — Sources without a committed Texture3D.
- **Invalid** — Sources whose committed texture state is invalid.
- **Disk** — total disk size of the committed Texture3D files referenced by the displayed Sources.

This provides a quick health check for the scene before entering Play Mode or preparing a build.

Source List

Every Source is shown as a separate row.

The row includes information such as:

- Object name.
- SDF resolution.
- Current status.
- Committed Texture3D name.
- Texture asset size.
- Quick actions.

The Overview displays the committed resolution for baked Sources and the desired resolution for Sources that are not yet baked.

Status Filter

Sources can be filtered by:

- All
- Baked
- Not Baked
- Invalid

This makes it easy to find unfinished or broken Sources in a large scene.

Search

The search field filters Sources using both the Source object name and the committed texture name.



This is useful in large environments where many Sources may be present.

Quick Actions

Each row provides direct access to useful operations.

You can:

- Ping the committed Texture3D in the Project window.
- Open the Source directly in the **SDF Source Baker**.

This makes the Overview useful as a central inspection window rather than just a passive report.

Automatic Refresh

The Overview automatically rescans when:

- The active scene changes.
- The Hierarchy changes.

A manual **Refresh** button is also available when an explicit rescan is needed.

When to Use the Overview

The Scene Overview is especially useful:

- Before final scene testing.
- After marking a large number of Sources.
- After reorganizing scene hierarchy.
- When checking whether any Sources were forgotten before a build.
- When investigating unexpectedly large baked SDF storage.
- When locating **NOT BAKED** or **INVALID** Sources without searching the Hierarchy manually.

Think of it as the scene-level **status dashboard** for the SDF authoring system.

SDF Source Cleaner

The **SDF Source Cleaner** is the maintenance tool used to safely remove **AquariumSource** components and clean up committed SDF Texture3D assets that are no longer needed.

Open it through:



[Tools](#) > [Skulmovski Studio](#) > [SDF Source Cleaner](#)

or:

[GameObject](#) > [Skulmovski Studio](#) > [Open SDF Source Cleaner](#)

The Cleaner deliberately separates two different operations:

Removing a Source component and **deleting its baked texture asset** are not the same thing.

This separation is important because multiple objects or Prefab instances may share the same committed SDF Texture3D.

Root Object

The Cleaner displays SDF Sources contained under the selected Root hierarchy.

Like the Marker and Baker, the Root normally follows the current Hierarchy selection.

Use **Root Lock** to keep the current Cleaner hierarchy while selecting other objects in the scene.

Selecting Sources

The Cleaner lists existing [AquariumSource](#) components under the current Root and allows multiple Sources to be selected for removal.

The list also provides information about their committed baked textures and approximate texture memory.

This makes it possible to review what is being removed before performing a destructive operation.

Clear Selected

Clear removes the selected [AquariumSource](#) components.

It does **not** immediately delete their committed Texture3D assets.

Before continuing, the Cleaner displays:

- Number of Sources being removed.
- Approximate Texture3D data referenced by those Sources.
- A warning that the committed Texture3D assets will remain on disk.

The Source removal itself is registered with Unity Undo as a batch operation.

This behavior is intentional.



For example, imagine ten Prefab instances all sharing the same baked Texture3D.

Removing one of those Sources must not automatically delete the texture still required by the other nine instances.

Why Baked Textures Are Kept

A committed SDF Texture3D may be referenced by:

- Another Source in the scene.
- Another scene.
- A Prefab.
- Multiple Prefab instances.
- Another saved project asset.

For this reason, removing a Source component alone is not enough information to safely decide that its Texture3D should also be deleted.

The Cleaner leaves the baked asset untouched until a separate project-wide unused-texture scan is performed.

Delete Unused Textures

Use **Delete Unused Textures** to scan the project for committed SDF Texture3D assets that are no longer referenced.

The Cleaner performs the reference scan first.

If unused textures are found, it displays:

- Number of unused assets.
- Approximate total Texture3D data.
- A preview of the affected asset paths.

Only after confirmation are those assets deleted.

Important: Asset Deletion Cannot Be Undone

Deleting unused Texture3D assets through **Delete Unused Textures** is an Asset Database deletion and cannot be restored through normal Unity Undo.

The Cleaner therefore asks for confirmation before deleting anything.

Use this operation after you are confident that old baked SDF assets are no longer required.



Search & Refresh

The Cleaner includes search functionality for working with large Source hierarchies.

The **Refresh** button rescans the current Root and updates the displayed Source and committed Texture3D state.

Recommended Cleanup Workflow

When removing SDF texture from objects:

1. Open the **SDF Source Cleaner**.
2. Select the Sources that should no longer exist.
3. Use **Clear** to remove the **AquariumSource** components.
4. Continue working normally if the corresponding baked textures may still be shared elsewhere.
5. When project cleanup is required, use **Delete Unused Textures**.
6. Review the detected unused assets carefully.
7. Confirm deletion only after the project-wide reference scan has completed successfully.

This workflow keeps Source removal safe while still allowing orphaned bake assets to be cleaned up when they are genuinely no longer used.

Recommended SDF Authoring Workflow

SDF authoring begins with choosing the geometry that should represent collision, not with choosing a bake resolution.

The visible model and the SDF collision representation may be identical, but they do not have to be.

There are three normal starting workflows:

1. Bake the Existing Geometry Directly

Use the original scene or Prefab geometry when it already produces a clean SDF representation at a practical Source size and bake resolution.

This is the simplest workflow and requires no separate proxy geometry.

2. Create SDF Geometry in Unity



When the visual mesh is unsuitable for direct SDF baking, create simplified geometry directly in Unity.

This can be useful for roofs, walls, glass, large architectural sections, or other surfaces where the collision representation should be simpler, thicker, larger, or differently segmented than the rendered model.

The dedicated SDF geometry can remain part of the authoring hierarchy and can be hidden automatically after the final bake using **Hide Renderer After Bake**.

3. Create SDF Geometry in a DCC Application

More complex SDF geometry can also be prepared in Blender, Maya, 3ds Max, or another modelling application and imported into Unity.

This is useful when the desired collision representation is easier to build or edit using normal modelling tools.

Once imported, the authoring process is identical to geometry created directly in Unity.

Universal SDF Workflow

Regardless of how the geometry was created, the remaining workflow is the same:

Choose / Prepare SDF Geometry



SDF Source Marker



SDF Source Baker — Test Bake



SDF Preview



Adjust Geometry / Source Layout / Resolution if Required



Apply the Final Bake



Play Mode Validation

The additional Editor tools are then used when needed:

- **SDF Scene Overview** — review the complete active scene for missing, invalid, or unexpectedly large Sources.
- **SDF Source Cleaner** — remove unwanted Sources and safely clean up unused committed **Texture3D** assets.



Step 1: Choose the SDF Representation

First decide whether the original rendered geometry is already suitable for SDF collision.

If it produces a clean result without excessive resolution or inefficient Source bounds, bake it directly.

If it does not, prepare separate SDF geometry instead of modifying the visible model purely to satisfy the collision system.

The purpose of SDF authoring is to represent the surfaces that matter for rain interaction with an efficient and reliable voxel volume.

Step 2: Define Sources in the Marker

Open the **SDF Source Marker** and define which hierarchy objects should become **AquariumSource** components.

Configure:

- Source resolution.
- Stationary state where appropriate.
- Hide Renderer After Bake for dedicated SDF geometry.
- Nested Source boundaries where a hierarchy should be divided into independent SDF regions.

Use **Apply & Open Baker** when the Source layout is ready.

Step 3: Test Bake

In the **SDF Source Baker**, generate a temporary Test Bake before committing the Source.

Start with the recommended resolution for the current Source based on its largest axis.

The purpose of the Test Bake is to evaluate the complete representation — geometry, bounds, padding, and voxel density — without replacing the current committed asset.

Step 4: Inspect the SDF Preview

Inspect the generated field rather than assuming a successful bake is automatically a good bake.

Check important surfaces, edges, corners, openings, slopes, gaps, and connections between separate pieces.

If the representation is poor, determine whether the better solution is:

- Tighter Source bounds.



- Different Source segmentation.
- Different SDF geometry.
- A dedicated proxy.
- More bake resolution.

Do not rely on resolution alone to compensate for an unsuitable geometry representation.

Step 5: Apply the Final Bake

When the Preview is satisfactory, use **Apply** to commit the temporary result as the Source's final SDF **Texture3D**.

Only the committed texture is used by the runtime Aquarium.

If **Hide Renderer After Bake** is enabled, the dedicated authoring renderer is hidden after the successful bake is committed while the mesh data remains available for future preview and rebaking.

Step 6: Validate in Play Mode

Enter Play Mode and verify the actual rain interaction.

The SDF Preview confirms the baked geometry representation, while Play Mode validates the complete runtime result including:

- Aquarium composition.
- Source transforms.
- Particle motion.
- Collision behavior.
- Other active masking systems.

If a Source behaves incorrectly, return to the Baker and determine whether the issue comes from the baked representation, Source layout, or the runtime rain conditions.

Step 7: Review and Maintain

Use **SDF Scene Overview** periodically to inspect scene-wide Source status and committed texture storage.

Use **SDF Source Cleaner** when Sources are removed or old baked assets are no longer required.

These tools are maintenance steps rather than mandatory parts of every individual bake.



The next section, **SDF Proxy Geometry Authoring**, explains how to design dedicated SDF geometry when the original visual mesh is not the best representation for reliable and efficient rain collision.

SDF Proxy Geometry Authoring

Dedicated SDF proxy geometry can be used when the visible mesh is not the best representation for reliable or efficient SDF collision.

The rendered model and the SDF collision geometry serve different purposes:

- **Visual geometry** is designed for appearance.
- **SDF geometry** is designed to produce a useful voxelized collision field.

They may be identical when the original mesh bakes well, but there is no requirement for them to match exactly.

A dedicated SDF proxy may be simpler, thicker, larger in selected directions, divided differently, or constructed from overlapping pieces when those changes produce a more reliable SDF.

The goal is not to reproduce every visual detail. The goal is to preserve the surfaces that matter for rain interaction while using the available SDF voxels efficiently.

When to Use a Dedicated SDF Proxy

Consider dedicated SDF geometry when the original mesh produces problems such as:

- Thin or unreliable surfaces.
- Broken or irregular regions in the SDF Preview.
- Important angled surfaces that voxelize poorly.
- Small details that require an unnecessarily high resolution.
- Large empty regions inside the cubic Source volume.
- Complex visual topology that provides little useful collision information.
- Seams or boundaries where independently baked pieces need more reliable coverage.

Before rebuilding geometry, also test **Box Padding**. Some malformed SDF results can change significantly when additional space is provided around the geometry during baking.

Proxy Geometry Does Not Need to Match the Visual Mesh

A proxy should represent the physical interaction surface that rain needs rather than the exact rendered topology.

This means a proxy can intentionally:



- Remove small decorative details.
- Simplify complex topology.
- Fill irrelevant holes.
- Use fewer separate parts.
- Increase thickness where useful.
- Extend beyond visual boundaries.
- Overlap neighbouring proxy pieces.
- Divide one large structure into smaller reusable sections.

These differences are normal as long as the resulting collision surface remains appropriate for the visible object.

Preserve the Important Collision Surface

When proxy geometry is made thicker or larger, keep the surface that the rain visibly interacts with aligned with the rendered model whenever possible.

For example:

Roofs

For a roof, keep the upper rain-facing surface aligned with the visible roof.

If additional thickness is required, extend most of that thickness downward into the building rather than raising the collision surface above the visible roof.

This keeps rain impacts visually close to the rendered surface.

Exterior Walls

For an exterior wall, keep the outside face aligned with the visible wall.

Additional thickness can normally extend toward the building interior.

This avoids creating a visible gap where rain appears to collide before reaching the wall.

Thin Glass and Similar Surfaces

Thin visual surfaces do not need to remain equally thin in the SDF representation.

A simplified closed proxy can provide a more reliable volume while the original glass or other visual mesh remains unchanged.

Avoid moving the important collision-facing surface away from the visible object simply to make the proxy thicker.



Thickness Is Not a Fixed Requirement

There is no universal minimum SDF proxy thickness.

Required thickness depends on:

- Source dimensions.
- Bake resolution.
- Surface angle.
- Shape.
- Effective voxel density.
- Box Padding.
- Particle motion in the final effect.

A thin proxy can work correctly in one Source while a thicker surface may still produce a poor result in another configuration.

For this reason, do not treat values such as **10 cm**, **20 cm**, or any other fixed measurement as a global rule.

Instead:

1. Create a reasonable representation.
2. Test Bake it.
3. Inspect the SDF Preview.
4. Adjust geometry, Padding, Source segmentation, or resolution where necessary.
5. Verify the final collision in Play Mode.

Geometry thickness can also provide additional collision margin for fast particles, but thickness cannot guarantee collision at arbitrarily high particle velocities.

Extend Proxy Boundaries Beyond Important Edges

SDF voxelization can soften or shift the effective position of boundaries.

For important rain-blocking surfaces, it can therefore be useful for the SDF proxy to extend slightly beyond the visible geometry instead of ending exactly at the same boundary.

This is particularly useful around:

- Roof edges.
- Wall ends.
- Canopies.
- Modular sections.
- Thin architectural pieces.

The extension does not need to be visually rendered because the dedicated proxy can remain hidden after baking.



Use only as much additional coverage as necessary. Excessively oversized proxies can create collision where the player would reasonably expect open space.

Overlapping Adjacent Proxy Pieces

When several separate SDF proxy pieces are intended to form one continuous rain-blocking surface, avoid relying on perfectly matching edge-to-edge seams.

Small gaps introduced by modelling tolerances, voxelization, bake boundaries, or Source placement can become visible as rain leaks.

Where appropriate, allow neighbouring proxy pieces to overlap slightly.

For example, modular roof sections can extend into each other instead of terminating at exactly the same world-space edge.

The overlap should remain small enough that it does not meaningfully change the intended collision shape.

Thickness, Boundary Extension, and Overlap Are Different Tools

These techniques solve different problems:

- **Thickness** improves the volume represented through the surface.
- **Boundary extension** provides additional coverage near the outer edge of a surface.
- **Overlap** prevents gaps between neighbouring pieces.

Do not treat them as interchangeable.

A roof may have sufficient thickness but still leak at the boundary because its proxy ends too precisely at the visible edge.

Two individually good proxy pieces may still create a seam if they meet exactly without overlap.

Keep Proxy Sources Spatially Efficient

SDF Sources are baked into cubic **Texture3D** volumes.

A proxy should therefore not only be simple — it should also fit efficiently inside its Source volume.

For long or irregular structures, several compact proxy Sources can provide better useful voxel density than one enormous Source containing large empty regions.

Useful divisions include:

- Building floors.



- Wings.
- Rooms.
- Roof sections.
- Wall modules.
- Repeated architectural pieces.

Avoid excessive splitting as well. Every placed Source still consumes a runtime Source instance, so Source count and voxel efficiency should be balanced together.

Modular SDF Proxy Geometry

Repeated structures are especially suitable for modular SDF proxies.

For example, a repeated wall section can use:

1. One simplified SDF proxy module.
2. One committed baked *Texture3D*.
3. Multiple *AquariumSource* instances referencing the same bake.

Each placed instance keeps its own runtime Transform while sharing the same unique SDF texture.

This allows a reusable collision representation to be authored once and reused throughout a modular environment.

If substantially different scales are required, prepare separate baked scale variants rather than scaling one committed SDF representation arbitrarily.

Creating Proxy Geometry in Unity

Simple proxies can be authored directly in Unity using ordinary mesh geometry.

Typical examples include:

- Boxes for walls.
- Simplified roof volumes.
- Combined architectural sections.
- Helper meshes created specifically for SDF baking.

Once the geometry is ready:

1. Mark it using **SDF Source Marker**.
2. Enable **Hide Renderer After Bake** when the geometry should not remain visible.
3. Test Bake it in **SDF Source Baker**.
4. Inspect the result in Preview.
5. Commit the bake when satisfactory.

The proxy mesh remains available for future adjustment and rebaking even when its renderer is hidden.



Creating Proxy Geometry in a DCC Application

More complex proxies can be authored in Blender, Maya, 3ds Max, or another modelling package.

The same principles apply:

- Preserve important collision-facing surfaces.
- Simplify unnecessary detail.
- Use meaningful thickness where required.
- Extend important boundaries where useful.
- Overlap neighbouring modules when necessary.
- Keep individual Source volumes reasonably compact.

After importing the geometry into Unity, it uses the same Marker → Baker → Preview workflow as any other SDF Source.

Always Validate the Baked Result

The proxy mesh itself is not the final collision representation.

The **baked SDF** is.

After every important Test Bake:

- Open the Preview.
- Switch to the **SDF** tab.
- Increase Preview **Scale** when necessary.
- Inspect surfaces, edges, slopes, seams, and openings.
- Look for missing sections, stair-stepping, stripes, grid-like gaps, or other malformed regions.

If the result looks incorrect, try **Box Padding** before immediately increasing resolution or rebuilding the proxy.

Only commit the bake when the generated SDF itself represents the intended collision geometry reliably.

SDF Geometry, Animated Objects & Large Structures

This section covers runtime and structural behavior of baked SDF Sources after the geometry representation itself has been prepared.

For detailed guidance on creating dedicated collision geometry, see **SDF Proxy Geometry Authoring**.



Voxel Density Depends on the Complete Source Volume

SDF bake resolution applies to the complete cubic Source volume.

The useful detail available to an individual wall, roof, opening, or other feature therefore depends not only on the selected resolution but also on how large the complete Source volume is.

Two Sources using the same 64^3 bake resolution can provide very different effective geometric precision if one covers a small prop and the other covers an entire building.

For geometry that appears unnecessarily coarse:

- Inspect the SDF Preview.
- Check whether the Source volume contains large unused regions.
- Consider splitting large geometry into smaller Sources.
- Adjust the geometry representation where appropriate.
- Increase bake resolution when additional voxel density is genuinely required.

Do not use one fixed geometry-thickness rule as a substitute for evaluating the actual baked SDF.

Scale Changes Require Rebaking

Source scale is part of the geometry represented by the baked SDF.

If an **AquariumSource**, one of its parent Transforms, or another relevant hierarchy scale changes after baking, the existing SDF no longer represents the correct shape and the Source must be baked again.

The SDF authoring tools detect relevant scale changes and can report the Source as:

NEEDS REBAKE

Runtime position and rotation are different.

A rigid Source can move and rotate while continuing to use the same baked SDF because the runtime system transforms the existing representation into its new position and orientation.

If the same reusable object is required at substantially different scales, prepare separate pre-baked scale variants.

Moving Rigid Sources

SDF Sources do not need to remain static scene geometry.

A baked rigid Source can move and rotate during runtime while continuing to participate in the Aquarium.

Typical examples include:

- Vehicles.



- Doors.
- Moving platforms.
- Rigid props.
- Mechanical objects.
- Other objects whose actual mesh shape does not deform.

The **Stationary** option should only be enabled for Sources that are genuinely expected to remain stationary during normal gameplay.

Stationary Sources cache their Transform data as an optimization.

If a Stationary Source is intentionally moved or rotated through custom runtime code, call:

```
InvalidateMatrixCache()
```

after changing its Transform so the Aquarium can use the updated matrix.

Do not mark an object Stationary simply because it happens to start the scene without moving.

Runtime SDF Rebaking Is Not Supported

Individual Source SDF textures are pre-baked during authoring.

The runtime Aquarium can:

- Register Sources.
- Remove Sources.
- Move them.
- Rotate them.
- Transform them into the local Aquarium.
- Combine their existing baked SDF data.

It does not regenerate the underlying Source **Texture3D** from changed mesh geometry.

Projects that require runtime generation of new volumetric SDF textures need a custom runtime solution.

Skinned Meshes & Deforming Geometry

SkinnedMeshRenderer geometry is supported by the SDF Baker, but it is baked as a **static snapshot of its current deformed pose**.

Each Test Bake captures the current skinned mesh shape and generates the SDF from that pose.

The committed SDF does not continuously follow later deformation.

After baking:

- Moving the complete Source is supported.



- Rotating the complete Source is supported.
- Changing scale requires rebaking.
- Bone animation does not modify the baked SDF.
- Blend shapes do not modify the baked SDF.
- Runtime vertex deformation does not modify the baked SDF.
- Other changes to the actual mesh shape do not modify the baked SDF.

A skinned object can still be useful as a Source when its baked pose is sufficiently representative of the required collision.

Bone-Attached Rigid SDF Proxies

For continuously animated characters or other strongly deforming objects, approximate collision can be created using several simple rigid SDF proxies attached to important bones.

For example:

- *Torso proxy → torso/chest bone.*
- *Head proxy → head bone.*
- *Arm proxy → arm bone.*
- *Other important body regions → corresponding rigid transforms.*

Each proxy uses a normal pre-baked SDF and follows its assigned bone through Transform movement and rotation.

This allows approximate collision to follow the animated character without rebuilding the complete deforming mesh as a volumetric SDF every frame.

The package does **not** automatically generate these proxies. They are an optional authoring strategy for projects that require this type of interaction.

Large & Elongated Structures

Every SDF Source is baked into a **cubic** `Texture3D`.

Because of this, very large, long, flat, or irregular structures can spend a large portion of their available voxel volume on empty space.

A long building baked as one Source may therefore require a much higher resolution simply to preserve useful detail in one relatively small section.

When this happens, divide the structure into spatially compact logical Sources such as:

- Floors.
- Wings.
- Rooms.
- Building sections.



- Roof sections.
- Modular walls.
- Independent moving parts.

Smaller and tighter Source volumes often provide better useful voxel density than simply increasing the resolution of one oversized bake.

Inspect the SDF Preview when deciding where to split. Large unused regions combined with coarse important geometry are a strong indication that the Source layout is inefficient.

Balance Source Size and Source Count

Splitting large structures is useful, but excessive fragmentation is not free.

Every active Source contributes to the runtime instance capacity of its corresponding resolution tier.

The goal is therefore not to create the smallest possible Source for every individual mesh.

Instead, balance:

Useful voxel density ↔ Source count ↔ Bake resolution ↔ Runtime tier capacity

Group geometry that fits efficiently together and separate geometry when one Source would otherwise waste substantial voxel volume.

Nested Sources & Geometry Ownership

Nested **AquariumSource** components can divide a hierarchy into several independent SDF regions.

The ownership rule is:

Each bakeable renderer belongs to the nearest **AquariumSource above it in the hierarchy.**

A nested Source creates a new ownership boundary.

Geometry owned by that child Source is excluded from the parent Source bake.

For example:

- **Building [AquariumSource]**
 - **MainStructure [Mesh]**
 - **RoofSection [AquariumSource + Mesh]**

MainStructure belongs to the Building Source.

RoofSection belongs to its own Source and is not baked again into the parent Building Source.

This makes nested Sources useful for:



- Large structures.
- Independent building sections.
- Movable parts.
- Different bake resolutions inside one hierarchy.

An **AquariumSource** must still own at least one supported piece of bakeable geometry after nested Sources have been excluded.

The **SDF Source Marker** validates these ownership rules and reports invalid no-geometry configurations.

Repeated Geometry & Reusable Prefabs

Identical objects do not require a separate unique baked SDF texture for every placed instance.

Multiple **AquariumSource** objects can reference the same committed **Texture3D**.

For example: 50 identical objects using one bake = 1 unique SDF texture + 50 runtime Source instances

A recommended reusable workflow is:

1. Prepare the original object or SDF proxy.
2. Configure its **AquariumSource**.
3. Test Bake and inspect the SDF.
4. Commit the final bake.
5. Verify collision in Play Mode.
6. Save the prepared object as a Prefab.
7. Create additional instances from that Prefab.

This is particularly useful for:

- Modular SDF proxies.
- Walls.
- Barriers.
- Pillars.
- Repeated roof sections.
- Furniture.
- Props.
- Other reusable environment modules.

Remember that **unique textures and runtime instances are separate capacities**.

Reusing one baked texture reduces unique-texture usage, but each placed Source still consumes an independent runtime Source instance.

If different instances require substantially different scales, prepare appropriate separately baked scale variants.



Spawn Settings & Particle Optimization

The Rain VFX can cover a large area and produce a high number of particles, but larger spawn volumes, higher densities, and unnecessarily long particle lifetimes can all increase GPU cost.

The goal is to keep the effect large and dense enough for the visible scene without simulating or rendering significantly more data than necessary.

Spawn Area

The physical size of the rain spawn area directly affects how many particles may be active at the same time. A larger spawn area usually requires more particles to maintain the same visual density.

For best performance:

- Keep the spawn area only as large as necessary to cover the visible rain around the camera.
- Avoid using an excessively large area simply to guarantee coverage.
- Increase rain density only when the additional particles produce a visible improvement.

Large spawn areas combined with high density can become expensive quickly because more particles must be simulated and rendered.

Spawn Height & Particle Lifetime

Spawn height also affects how many raindrops may remain alive simultaneously.

If particles are spawned much higher than necessary, they remain in the simulation longer before reaching scene geometry. This means that the same spawn rate can result in a larger number of active particles.

Keep the rain spawn height high enough to cover the visible scene and expected camera movement, but avoid unnecessary vertical distance.

Strong Wind & Spawn Coverage

Wind can move falling particles significantly sideways between the spawn area and the ground.

This is expected behavior of the force-driven rain simulation. The spawn area itself does not automatically move or expand to predict future particle drift.

With strong wind, rain can therefore be carried away from part of the intended coverage area before reaching the scene.

If strongly angled rain no longer provides sufficient coverage around the target, consider:



- Increasing the horizontal Spawn Area.
- Reducing Spawn Height so particles have less time to drift.
- Offsetting the Spawn Box Center toward the direction from which additional coverage is required.
- Implementing project-specific spawn compensation when the required behavior depends on dynamic wind.

Avoid making the spawn area much larger than necessary simply to cover every theoretical wind configuration. Larger spawn volumes normally require more particles to maintain the same visible density.

This is separate from high-speed collision tunneling:

- **Spawn coverage** determines whether rain particles travel through the required visible region.
- **Collision tunneling** determines whether extremely fast particles can reliably detect collision geometry while travelling through that region.

Particle Capacity

Each VFX Graph particle subsystem has a fixed maximum **Capacity**.

If a subsystem reaches its Capacity, increasing spawn settings further may no longer produce the expected increase in visible particle density.

Possible symptoms include:

- Rain appearing uneven or pulsing in visible waves.
- Rain density no longer increasing as expected.
- Secondary particle effects becoming less consistent during extremely dense rainfall.

Before increasing Capacity, first try reducing:

- Spawn area.
- Spawn height.
- Rain density.
- Particle lifetime where appropriate.

If a genuinely larger particle budget is required, open the **.vfx** Graph and increase the **Capacity** of the affected subsystem.

⚠ Warning: Increasing Capacity allows more particles to exist simultaneously and can increase GPU memory usage, simulation cost, rendering cost, and overdraw. Increase it only when necessary and profile the result on your target hardware.



⚠ **IMPORTANT: Optimization for Production**

Out of the box, this effect is configured for a **plug-and-play Quick Start**, so you can immediately test the rain without additional optimization steps. Before releasing your final game, however, we highly recommend reviewing the following settings.

Aquarium Resolution & Voxel Size

Start with the recommended production configuration:

- **Resolution:** 256³
- **Voxel Size:** 10 cm
- **Physical Size:** 25.6 × 25.6 × 25.6 m

Increase Aquarium cost only when representative gameplay shows that additional coverage or precision is required.

If more coverage is required, consider whether a larger **Voxel Size** is acceptable before increasing Aquarium Resolution.

If more precision is required, consider whether the Aquarium can cover a smaller physical area before increasing resolution.

512³ should be treated as a deliberate high-cost configuration rather than a normal default.

The practical trade-off is: Coverage ↔ Precision ↔ GPU / Memory Cost

Choose an Appropriate Source Bake Resolution

Source bake resolution should match the physical size of the Source volume and the geometric detail that needs to be preserved.

Use the **Source Size as a Starting Point** guidance in the SDF Source Marker section to choose an initial tier, then run a **Test Bake** and inspect the **SDF Preview**.

Increase resolution when important collision geometry is not represented with sufficient detail.

Before increasing resolution significantly, also check whether the Source contains large unused regions. Tighter bounds, better Source segmentation, or a more efficient SDF representation can often improve useful voxel density without moving to a higher bake tier.



Keep Source Volumes Spatially Efficient

Every SDF Source is represented by a cubic volume.

Large empty regions inside that cube consume voxel resolution without representing useful geometry.

For large or elongated structures, use several reasonably compact Sources instead of one enormous high-resolution Source when this significantly improves useful voxel density.

Do not split geometry excessively either.

Every placed Source still consumes a runtime instance entry, so production authoring should balance:

Voxel efficiency ↔ Source count ↔ Bake resolution ↔ Tier capacity

Do Not Increase Tier Capacity Without a Reason

Each Source resolution tier owns a persistent runtime texture atlas.

Increasing the number of unique texture slots available to a tier therefore increases GPU memory reserved by that atlas.

Keep the supplied capacities until actual content demonstrates that a tier requires more unique textures simultaneously.

Before increasing capacity:

- Reuse identical baked textures.
- Remove unnecessary unique variants.
- Check whether Sources really need the selected resolution tier.
- Inspect runtime tier usage.

Increasing capacity should solve a measured content requirement, not be used as a general precaution.

Use Camera Culling Where Appropriate

Camera Culling can reduce unnecessary Aquarium processing outside the useful horizontal direction defined by the assigned **Camera Transform**.

Use a Culling Angle appropriate for the gameplay setup rather than automatically keeping **360°** coverage when full horizontal SDF coverage is not required.

In custom tracking setups, remember that the culling direction comes from the assigned Aquarium tracking Transform and does not necessarily match the physical rendering camera.



The **Always Include SDF Zone** preserves a small local area around the tracking position even when directional culling is aggressive.

Test representative movement, rotation, nearby geometry, and camera behavior before choosing the final value.

Use Ground Culling Only for True Ground

Ground Culling can remove a substantial amount of unnecessary Aquarium work below terrain and other solid ground surfaces.

It is useful when nothing underneath the captured surface requires SDF collision.

Do not include:

- Bridges.
- Roofs.
- Upper floors.
- Elevated platforms.
- Surfaces with playable or collision-relevant space underneath them.

Incorrect Ground Mask content can improve apparent performance while silently removing valid SDF collision.

Optimize Rain Coverage Before Increasing Particle Count

A larger Spawn Area, greater Spawn Height, and longer particle lifetime can all increase the number of simultaneously active raindrops.

Before increasing particle Capacity:

- Keep the Spawn Area close to the actually visible region.
- Avoid unnecessary Spawn Height.
- Reduce unnecessary particle lifetime.
- Adjust Spawn Box Center when strong wind shifts visible rain coverage.
- Use spawn probability controls where reduced density is acceptable.

Increasing particle Capacity should be a final response to a genuine particle-budget limitation rather than the first optimization adjustment.

Automatic Bounds

By default, the VFX Graph uses **Automatic Bounds**. This helps prevent the rain effect from being unexpectedly culled when the camera moves or rotates, but it also requires Unity to update the effect bounds at runtime.



How to optimize: Once you have finalized the size and position of your rain spawn area, select the Rain VFX object in the Hierarchy, open the Unity **VFX Control** window, and click **Record bounds**.

Tip: Let the rain run for a few seconds before stopping the recording so the captured bounds include the full particle travel distance from the spawn area to the ground.

Then open the `.vfx` Graph asset and change the **Bounds** setting in all relevant Initialize contexts from Automatic to Recorded or Manual.

Using fixed bounds avoids unnecessary runtime bounds updates and is recommended for production once the final effect area is known.

Profile the Final Configuration

Performance should be measured using representative gameplay on the actual target hardware.

When profiling, test scenes that contain realistic combinations of:

- Rain density.
- Aquarium Resolution and Voxel Size.
- Active SDF Sources.
- Source resolution tiers.
- Roof, Water, and Ground masks.
- Strong wind where applicable.
- Splash, Ripple, and Mist effects.
- Typical camera movement.

Avoid optimizing individual settings in isolation if the final game combines several expensive features at the same time.

Troubleshooting & Bonus Tips

Even with a correct project setup, some scene layouts, custom geometry, or masking configurations can produce behavior that initially looks like a bug.

The following cases cover the most common visual artifacts and configuration mistakes.

Z-Fighting or Proxy Geometry Is Visible

Symptom: After creating a Weather or Water Occluder Proxy, duplicate geometry becomes visible, flickers against the original object, or appears in the final game view.

Cause: The Main Camera is rendering one of the technical proxy layers.



Solution: Open the Main Camera **Culling Mask** and make sure the corresponding technical layers are excluded:

- **VFXWeatherOccluders**
- **VFXWaterOccluders**

The original gameplay object should remain on its normal rendering layer. Only the generated proxy should use the technical layer.

If the Setup Assistant reports an incorrect camera mask, apply the suggested Scene Settings fix.

Rain Leaks Through a Roof, Wall, or Other SDF Surface

Symptom: Rain occasionally passes through geometry even though the Source has a valid committed SDF.

Cause: The baked SDF may not represent the required collision surface reliably. This can happen because of insufficient voxel density, unsuitable geometry representation, Source size, surface angle, bake artifacts, exposed edges, or particle velocity.

Solution:

1. Open the Source in the **SDF Source Baker** and run a Test Bake.
2. Inspect the result in the **SDF** Preview tab and increase Preview **Scale** when necessary.
3. Check whether the surface itself is continuous and correctly represented in the baked SDF.
4. If the Preview contains stair-stepping, stripes, grid-like gaps, clipped areas, or other malformed regions, try increasing **Box Padding** and Test Bake again.
5. Check whether the Source volume is unnecessarily large.
6. Use a dedicated SDF proxy when the visual mesh is not a reliable collision representation.
7. Increase Source resolution when the geometry is appropriate but genuinely needs additional voxel density.
8. Verify the final result in Play Mode.

There is no universal minimum wall or roof thickness. The required representation depends on the complete Source and final baked result.

For very high rain velocity, also remember that normal discrete collision tunneling can eventually occur even when the SDF itself is valid.

SDF Preview Shows Steps, Stripes, Grid-Like Gaps, or Broken Areas

Symptom: The source mesh looks correct, but the Test Bake contains unexpected stair-stepping, bands, grid-like holes, clipped areas, or malformed regions.

Cause: The current bake configuration is producing an unstable or unsuitable SDF representation. Geometry positioned too tightly within the bake volume can be one contributing factor.

Solution:



First inspect the result in the Preview's **SDF** tab and increase Preview **Scale** so the generated field is clearly visible.

Then increase **Box Padding** and run another Test Bake.

Padding can sometimes substantially change the generated SDF even when the mesh and resolution remain unchanged.

If the artifact remains:

- Check Source bounds.
- Check the geometry representation.
- Try a dedicated SDF proxy.
- Split an inefficiently large Source.
- Increase bake resolution when additional voxel density is genuinely required.

Do not assume that a malformed Preview will become correct at runtime.

Rain Leaks Around the Edge of an SDF Proxy

Symptom: The main surface blocks rain correctly, but occasional drops pass around roof edges, wall ends, or other outer boundaries.

Cause: The proxy may terminate too precisely at the visible boundary, while the voxelized SDF may not preserve that collision edge exactly.

Solution: Extend the SDF proxy slightly beyond the important visible boundary where doing so does not create unwanted collision.

Keep the rain-facing surface aligned with the visible model while adding only the additional edge coverage required.

Inspect the final edge in the SDF Preview and verify it in Play Mode.

Rain Leaks Between Adjacent SDF Proxy Pieces

Symptom: Two individually correct proxy sections create a thin rain leak where they meet.

Cause: Exact edge-to-edge alignment does not always produce a perfectly continuous voxelized SDF boundary.

Solution: Allow neighbouring SDF proxy pieces to overlap slightly where they are intended to behave as one continuous collision surface.

Do not confuse this with geometry thickness:

- Thickness controls the volume through a surface.
- Overlap closes seams between separate pieces.



Rebake and inspect the seam in the SDF Preview.

Rain Passes Through Geometry at Extreme Speed

Symptom: Collision works under normal rain settings but some particles pass through geometry when gravity, wind, or other motion settings produce unusually high velocity.

Cause: Particle collision is evaluated during discrete simulation steps. A sufficiently fast particle can travel through a thin collision region between evaluations.

Solution: Verify that the baked SDF itself is correct first.

If the problem occurs only at extreme velocities:

- Reduce unnecessary particle velocity where appropriate.
- Use a more suitable SDF representation with additional collision margin where visually acceptable.
- Reduce Spawn Height if the excessive velocity results from unnecessarily long acceleration time.
- Test the final configuration in representative Play Mode conditions.

A thicker collision representation can increase margin, but it cannot guarantee collision at arbitrarily high particle speeds.

Original Object Disappears After Changing Its Layer

Symptom: A roof, building, water surface, or other gameplay object disappears from the Main Camera after being configured for masking.

Cause: The original object itself was moved to a technical proxy layer.

Solution: Restore the original object's normal gameplay/rendering layer and create a Weather or Water Occluder Proxy instead.

Do not manually move the original gameplay object to `VFXWeatherOccluders` or `VFXWaterOccluders`.

The technical layer should belong to the generated proxy, while the original object keeps its existing rendering, physics, raycast, and gameplay configuration.

Rain Passes Through Roofs Outside the Aquarium

Symptom: Rain passes through a roof, canopy, bridge, or other overhead structure when that geometry is outside the local Aquarium volume.

Cause: Outside the Aquarium, overhead rain blocking relies on the Roof Mask rather than the local SDF.

Solution: Check that:



- A Weather Occluder Proxy exists for the required geometry.
- The proxy is on [VFXWeatherOccluders](#).
- The geometry is inside the Roof Mask coverage area.
- The Roof Mask has sufficient spatial precision.
- The required Aquarium, Roof Mask, tracking, and Rain VFX references are assigned correctly.

[AquariumSDFController](#) owns the shared scene references and propagates the tracking and Rain VFX references to the Roof and Water Mask controllers.

Rain Passes Through Geometry Inside the Aquarium

Symptom: Nearby rain passes through walls, ceilings, vehicles, props, or other geometry that should use accurate local collision.

Cause: The geometry is not represented correctly in the Aquarium SDF.

Solution: Check that the object:

- Has an [AquariumSource](#).
- Has a valid baked SDF.
- Is currently inside the Aquarium volume.
- Has not been scaled after baking.
- Uses enough SDF resolution for its geometry.

If the Source exists and is active but only parts of its geometry fail, inspect the actual baked result in the **SDF Source Baker Preview**. Check Padding, geometry representation, Source bounds, and voxel density before assuming that the runtime Aquarium is the problem.

SDF Source Exists but Does Not Affect the Rain

Symptom: An object has an [AquariumSource](#) and a baked Texture3D, but it appears to have no effect on the local Aquarium.

Solution: Check that:

- The Source component is enabled.
- The Source is registered and available to the runtime system.
- The Source is close enough to intersect the current Aquarium volume.
- The baked Texture3D is still assigned and valid.
- The object has not been scaled since the bake.

Remember that the Aquarium is a **local** SDF volume. A valid Source elsewhere in the scene does not contribute until it becomes relevant to the current Aquarium area.

If all of the above looks correct, also check whether the Source's resolution tier has reached one of its runtime limits.



Tier Capacity Reached

- **Symptom:** A valid baked Source intersects the Aquarium but still does not contribute, especially in scenes containing many SDF Sources.
- **Cause:** The Source's resolution tier may have reached its limit for unique baked textures or runtime instances.
- **Solution:** Check the current usage of the affected resolution tier and reduce the number of unique SDF textures or active Source instances where necessary.

Repeated objects can share the same baked Texture3D. These objects use a single unique-texture slot while still existing as separate runtime instances.

If the unique-texture capacity of a tier is full, not every nearby unique SDF texture can be loaded into that tier at the same time.

SDF Source Shows NO GEO

Symptom: An SDF Source is marked, but the editor reports **NO GEO** and there is no geometry available for that Source to bake.

Cause: The Source does not currently own any supported bakeable geometry.

Solution: Check that the Source hierarchy contains valid mesh geometry and that this geometry is not owned by a nested **AquariumSource**.

This can also happen when an **AquariumSource** is used only as an organizational root while all of its actual geometry belongs to nested Sources.

SDF Source Shows BLOCKED

Symptom: The editor prevents a Source marking change and reports **BLOCKED**.

Cause: The requested hierarchy change would create an invalid Source ownership configuration, such as leaving another existing Source without geometry that belongs to it.

Solution: Review the nested **AquariumSource** hierarchy and decide which Source should own the affected geometry. Reorganize or remove conflicting Source markers before trying the operation again.

BLOCKED is therefore a protection against creating an invalid nested Source configuration rather than a bake failure.



Roof, Water, or Ground Mask Looks Too Coarse

Symptom: Rain clips around roof edges, small water surfaces are detected inaccurately, or Ground Mask boundaries look noticeably coarse.

Cause: The mask does not have enough pixel density for the physical area represented by its Render Texture.

Solution: Check the **Mask Precision** values in the corresponding controller:

- Texture Resolution
- Pixels Per Meter
- Pixel Size

For all three top-down masks:

Larger physical coverage + same Render Texture resolution = lower spatial precision.

The Roof and Water Masks have their own independent **Area Size**, while Ground coverage follows the Aquarium area.

If the mask is too coarse, reduce the physical coverage where possible or increase the Render Texture resolution.

The "Invisible Wall" Effect Under Large Overhangs

Symptom: Strong sideways wind pushes raindrops beneath a large roof or overhang, and the particles suddenly disappear in mid-air instead of continuing underneath it.

Cause: This is an inherent limitation of top-down height masking.

The Roof Mask stores a height value for each horizontal X/Z position. It does not represent the complete three-dimensional shape of the roof.

If wind pushes a particle sideways into a position where the stored roof height is above that particle, the particle is treated as being underneath the blocking surface even if it entered from the side.

This behavior primarily matters outside the Aquarium, where Roof Occlusion provides the large-area overhead masking.

 **Tip:** The effect is most noticeable with unusually large overhangs combined with strong horizontal wind.

Water Interaction Does Not Match the Visible Waves

Symptom: Ripples or Splash removal occur at a different height from the visible shader-generated water surface.



Cause: The Water Mask represents the Water Proxy's mesh geometry, not GPU-only shader deformation.

Solution: Make sure the Water Proxy represents the physical surface height that should be used by the Rain VFX.

Shader Graph vertex displacement, tessellation, and other GPU-only deformation are not automatically reproduced by the proxy.

For strongly displaced water, use proxy geometry that approximates the intended physical interaction surface.

Weather Proxy Animation No Longer Matches the Original

Symptom: An animated Weather Proxy freezes, becomes offset, or stops matching the original object after its hierarchy was modified.

Cause: The generated proxy no longer matches the source hierarchy it was created from.

Solution: Run **Create Weather Occluder Proxy** again.

Recreate the proxy after:

- Changing the mesh.
- Adding or removing children.
- Changing the rig or skeleton.
- Replacing major parts of the source hierarchy.

The existing generated proxy is replaced automatically.

Water Proxy Moves Correctly but Animated Children Do Not

Symptom: Moving, rotating, or scaling the original water object moves the proxy correctly, but independently animated child objects do not follow.

Cause: This is expected behavior.

The Water Proxy is parented under the original object, so normal root Transform movement is inherited automatically. However, unlike the Weather Proxy, it does not use a child/bone synchronization component.

For independently animated water geometry, the proxy must be updated through a project-specific workflow.



Ground Culling Removes Valid SDF Geometry

Symptom: SDF collision unexpectedly disappears below a bridge, upper floor, elevated platform, or similar surface after Ground Culling is enabled.

Cause: A surface that is not true ground is being included in the Ground Mask.

Solution: Restrict the Ground Mask to surfaces below which SDF evaluation can safely be skipped.

The Ground Mask provides a ground height that allows the SDF compute process to skip voxels below that surface.

Do not include roofs, bridges, upper floors, or elevated architecture if valid SDF geometry must remain active below them.

Depth Buffer Collision Ignores an Object

Symptom: A visible surface does not produce the expected Depth Buffer collision.

Cause: The geometry may not be represented in the active camera depth buffer.

Solution: Check the object's material and rendering configuration.

Opaque geometry normally participates in camera depth, while transparent materials and custom shaders that do not write usable depth may not.

Depth Buffer collision is supplementary to the Aquarium SDF and Roof Mask systems rather than a replacement for them.

The Demo Works but a Custom Scene Does Not

Symptom: The supplied Demo Scene behaves correctly, but masking or collision breaks after moving the effect into another scene.

Solution: Open the problematic scene and run the **Setup Assistant** again.

Click **Refresh** and review both:

- **Project-Wide Settings**
- **Scene Settings**

Scene-specific checks are particularly useful for finding incorrect camera configuration, technical layers, or culling masks. The Assistant is designed to highlight these missing or incorrect settings.



Technical Reference: Parameter Glossary

This section describes the exposed parameters available in the Rain VFX Graph Inspector.

Parameters are grouped in the same categories used by the VFX Graph. Unless otherwise noted, the values shown below are the supplied defaults.

Terminology Used in This Documentation

Several similarly named systems use different geometry representations:

- **Aquarium** — the local runtime three-dimensional SDF volume generated around the active tracking area.
- **AquariumSource / SDF Source** — a scene object that references a pre-baked SDF [Texture3D](#) and can contribute that representation to the Aquarium.
- **SDF Proxy Geometry** — custom geometry prepared specifically for SDF baking. It is not a Roof or Water mask proxy.
- **Weather Occluder Proxy** — technical geometry rendered into the top-down Roof Mask through [VFXWeatherOccluders](#).
- **Water Occluder Proxy** — technical geometry rendered into the top-down Water Mask through [VFXWaterOccluders](#).
- **Ground Mask** — a top-down optimization mask used to skip unnecessary Aquarium processing below suitable ground; it is not a collision proxy.

Custom Particle Textures

The supplied textures are only defaults. You can assign your own textures to the exposed texture parameters for:

- Raindrops
- Splashes
- Ripples
- Impact Mist

Main

Main controls for the complete rain effect and its secondary particle systems.

- [GlobalScale](#) — Global multiplier applied to particle sizes throughout the effect. **Range: 0–3. Default: 1.**
- [Drop_VisualEnabled](#) — Enables or disables visible raindrop rendering. **Default: On.**
- [Splash_Enabled](#) — Enables or disables Splash particles. **Default: On.**



- **Ripple_Enabled** — Enables or disables water Ripple particles. **Default: On.**
- **Mist_Enabled** — Enables or disables impact Mist particles. **Default: On.**
- **WindForce_Enabled** — Enables or disables the simple Wind Force applied to falling raindrops. **Default: Off.**

Optimization

These parameters reduce particle or SDF-processing cost without requiring changes to the VFX Graph itself.

Lower values generally reduce work at the cost of visual density or collision quality.

- **Splash_SlideSmoothness_EXPENSIVE** — Number of simulation substeps used by SDF Splash sliding. Higher values provide smoother and more stable movement over SDF surfaces but require additional SDF sampling. **Range: 1–8. Default: 8.**
- **Ripple_SpawnChance** — Percentage chance that an eligible impact produces a Ripple. **Range: 0–100. Default: 100.**
- **Splash_SpawnChance** — Percentage chance that an eligible impact produces a Splash. **Range: 0–100. Default: 100.**
- **Mist_SpawnChance** — Percentage chance that an eligible impact produces Mist. **Range: 1–100. Default: 100.**
- **Drop_SpawnChance_OutsideAquarium** — Controls the percentage of raindrops that are allowed to spawn **outside the Aquarium's horizontal coverage area**. Inside the Aquarium, rain keeps its full spawn density; outside it, the density can be reduced to lower rendering and downstream impact-particle cost. Setting this value to **0** completely disables rain spawning outside the Aquarium's horizontal area. Only the horizontal X/Z relationship to the Aquarium is considered — Y is ignored. Raindrops spawning above the Aquarium are therefore still treated as being inside its coverage. **Range: 0–100. Default: 100.**

0. Spawn (S)

Controls the size, position, orientation, and generation rate of the main rain volume.

- **S_ByDropRate** — Selects between area-density spawning and the fixed spawn-rate mode. **Default: On.**
- **S_AreaWidthX** — Width of the rain spawn area along local X. **Range: 1–300. Default: 40.**
- **S_AreaDepthZ** — Depth of the rain spawn area along local Z. **Range: 1–300. Default: 40.**
- **S_DropRatePerUnit** — Drop density used by the area-based spawn mode. Increasing the spawn area therefore increases the total spawn rate while maintaining approximately consistent density. **Range: 0–500. Default: 5.**
- **S_FixedDropRate** — Fixed raindrop spawn rate used by the fixed-rate spawn mode. **Default: 200.**
- **S_BoxAngles** — Local rotation of the spawn box. **Default: (0, 0, 0).**
- **S_BoxCenter** — Local positional offset of the spawn box center. **Default: (0, 0, 0).**

For strong angled rain, **S_BoxCenter** can also be adjusted manually to compensate for visible horizontal particle drift. The asset does not automatically reposition the spawn area based on wind.



1. Drop Init (DI)

Initial properties assigned when a raindrop is created.

- **DI_Texture** — Texture used to render the raindrop.
- **DI_Lifetime** — Maximum lifetime of a falling raindrop before it is removed automatically. **Default: 2.5 s.**
- **DI_SizeFromTo** — Random initial drop-size range. **Default: 0.015–0.03.**
- **DI_SpawnAngle** — Initial sprite orientation of newly spawned raindrops. **Default: (180, 0, 0).**

2. Drop Behavior (DB)

Controls the appearance, gravity, and Depth Buffer collision behavior of falling raindrops.

- **DB_StretchMultiplier** — Controls velocity-based visual stretching of raindrops. **Default: 1.**
- **DB_ColorOverLife** — Color and alpha gradient applied over the raindrop lifetime.
- **DB_CollisionRadius** — Collision radius used by the raindrop Depth Buffer collision. **Default: 0.1.**
- **DB_CollisionSurfaceThickness** — Surface-thickness tolerance used by Depth Buffer collision. **Default: 0.1.**
- **DB_CollisionOffsetY** — Vertical offset applied to the drop collision test. **Default: -0.3.**
- **DB_Gravity** — Gravity vector applied to falling raindrops. **Default: (0, -9.81, 0).**

3. Splash Init (SI)

Controls Splash particles created from eligible rain impacts.

- **SI_Texture** — Texture used to render Splash particles.
- **SI_Lifetime** — Splash particle lifetime. **Default: 4 s.**
- **SI_SizeFromTo** — Random initial Splash-size range. **Default: 0.01–0.04.**
- **SI_RandomCountFromTo** — Random number of Splash particles generated by an eligible impact. **Default: 5–8.**
- **SI_SpawnOffsetY** — Vertical offset applied when spawning a Splash on a normal collision surface. **Default: 0.05.**
- **SI_SpawnOffsetYOnWater** — Vertical Splash spawn offset used for water-surface interaction. **Default: 0.05.**

4. Splash Behavior (SB)

Controls Splash appearance, ricochet behavior, Depth Buffer collision, and SDF surface sliding.

SDF Sliding



Splash particles can continue moving across nearby geometry using the Aquarium SDF rather than stopping immediately at the first collision.

- **SB_Slipperiness** — Controls how much Splash velocity is retained while sliding across an SDF surface. **Range: 0.01–0.999. Default: 0.999.**
- **SB_MaxSlideSpeed** — Maximum Splash velocity allowed while the SDF slide solver is active. **Default: 0.5.**
- **SB_ContactOffset** — Contact distance used to keep the sliding Splash relative to the sampled SDF surface. **Default: -0.01.**

The accuracy and smoothness of this process are also affected by **Splash_SlideSmoothness_EXPENSIVE** in the **Optimization** category.

Appearance

- **SB_ColorOverLife** — Splash color and alpha over particle lifetime.
- **SB_SizeOverLife** — Splash size curve over particle lifetime.
- **SB_Scale** — Additional output scale applied to Splash particles. **Default: (1, 1, 1).**
- **SB_Smoothness** — Material smoothness of the Lit Splash output. **Range: 0–1. Default: 0.**

Depth Buffer Collision

- **SB_GroundBounce** — Bounce response used by Splash Depth Buffer collision. **Default: 0.**
- **SB_GroundFriction** — Friction applied during Splash Depth Buffer collision. **Default: 5.**
- **SB_GroundLifetimeLoss** — Fraction of remaining particle lifetime removed when a ground collision occurs. **Range: 0–1. Default: 0.**
- **SB_GroundRoughness** — Adds variation to the Splash collision response. **Range: 0–1. Default: 0.13.**
- **SB_GroundSurfaceThickness** — Surface-thickness tolerance used by Splash Depth Buffer collision. **Default: 0.03.**

Ricochet & Motion

- **SB_RicochetSpread** — Controls directional spread of newly launched Splash particles. **Default: 0.13.**
- **SB_RicochetSpeedMin** — Minimum initial Splash launch speed. **Default: 0.7.**
- **SB_RicochetSpeedMax** — Maximum initial Splash launch speed. **Default: 1.6.**
- **SB_Gravity** — Gravity vector applied to Splash particles. **Default: (0, -9.81, 0).**

Height Mask Offset

- **SB_HeightMask_OcclusionOffsetY** — Vertical offset used when evaluating Splash particles against the top-down height masking system. **Default: -0.3.**

5. Ripple Init (RI)

Controls Ripples created when rain interacts with a valid water surface.

- **RI_Texture** — Texture used to render Ripples.
- **RI_Lifetime** — Random Ripple lifetime range. **Default: 0.8–1.5 s.**



- **RI_SizeFromTo** — Random initial Ripple-size range. **Default: 0.1–0.2.**
- **RI_SpawnAngle** — Initial Ripple sprite orientation. **Default: (90, 0, 0).**
- **RI_SpawnOffsetY** — Vertical offset above the detected water surface. **Default: 0.001.**
- **RI_AlphaThreshold** — Alpha clipping threshold used by the Ripple output. **Range: 0–1. Default: 0.005.**

6. Ripple Behavior (RB)

Controls which impacts can create Ripples and how Ripples evolve visually.

- **RB_ImpactAngleThreshold** — Surface-orientation threshold used when determining whether an impact is suitable for a Ripple. **Range: 0–1. Default: 0.95.**
- **RB_ColorOverLife** — Ripple color and alpha over particle lifetime.
- **RB_SizeOverLife** — Ripple size curve over particle lifetime.

7. Mist Drop Init (MI)

Controls the local Mist particles generated from eligible drop impacts.

- **MI_Texture** — Texture used to render local Mist.
- **MI_SizeFromTo** — Random initial Mist-size range. **Default: 0.10–0.16.**
- **MI_LifetimeFromTo** — Random Mist lifetime range. **Default: 1.3–2.2 s.**
- **MI_SpawnOffsetY** — Vertical offset applied when Mist is created. **Default: 0.01.**
- **MI_SpawnVelocityFrom** — Minimum random initial Mist velocity. **Default: (-0.1, -0.1, -0.1).**
- **MI_SpawnVelocityTo** — Maximum random initial Mist velocity. **Default: (0.1, -0.02, 0.1).**
- **MI_RandomCountFromTo** — Random number of Mist particles created by an eligible impact. **Default: 3–6.**

8. Mist Drop Behavior (MB)

Controls the appearance and movement of local impact Mist.

- **MB_SoftParticleFadeDistance** — Distance over which Mist uses soft-particle fading near scene geometry. **Range: 0–100. Default: 0.3.**
- **MB_TurbulenceIntensity** — Strength of turbulent motion applied to Mist. **Default: 0.01.**
- **MB_ColorOverLife** — Mist color and alpha over particle lifetime.
- **MB_SizeOverLife** — Mist size curve over particle lifetime.
- **MB_Smoothness** — Material smoothness of the Lit Mist output. **Range: 0–1. Default: 0.**

9. Wind Force (Simple)

Controls the optional wind force applied to falling raindrops.



- **WF_Direction** — Direction of the applied Wind Force. **Default direction: (1, 0.1, 1).**
- **WF_Strength** — Magnitude of the applied Wind Force. **Default: 3.**

Wind is applied to raindrops as a continuous **Force** during the VFX Graph Update stage.

The force vector is calculated as:

Wind Force = Normalize(WF_Direction) × WF_Strength

The Force block uses **Absolute** mode rather than Relative mode, so the supplied force is applied independently from the particle's existing velocity.

Gravity and Linear Drag are applied separately by the VFX Graph.

Because Wind Force is continuously applied during simulation rather than setting a fixed rain velocity, stronger wind can change both particle trajectory and resulting velocity over time.

The system does not automatically offset or enlarge the spawn area to compensate for horizontal rain drift. For strong wind configurations, adjust the spawn area or **S_BoxCenter** manually when additional coverage is required.

10. Common

Parameters shared by particle-motion calculations.

- **DragCoeff** — Linear drag coefficient applied to particle motion. **Default: 0.3.**

11. Auto (Scripted)

Do not configure these parameters manually during normal use.

These properties exist so the runtime controllers can pass Aquarium, Roof Mask, and Water Mask data into the VFX Graph. Their values are updated automatically and the values visible in the graph asset are not meaningful runtime configuration defaults.

Aquarium

- **Auto_Aquarium_SDFTexture** — Runtime Aquarium SDF **Texture3D**.
- **Auto_Aquarium_SDFTransform** — Transform used to map world-space positions into the Aquarium SDF.
- **Auto_Aquarium_Size** — Current physical Aquarium size.
- **Auto_Aquarium_Center** — Current world-space Aquarium center.
- **Auto_Aquarium_Resolution** — Current Aquarium SDF resolution.

Roof Height Mask

- **Auto_HeightMask_Texture** — Current Roof Mask height texture.



- [Auto_HeightMask_BoundsMin](#) — Minimum world-space bounds of the Roof Mask.
- [Auto_HeightMask_BoundsSize](#) — Physical world-space size represented by the Roof Mask.

Water Mask

- [Auto_WaterMask_Texture](#) — Current Water Mask texture.
- [Auto_WaterMask_BoundsMin](#) — Minimum world-space bounds of the Water Mask.
- [Auto_WaterMask_BoundsSize](#) — Physical world-space size represented by the Water Mask.

The Aquarium and mask controllers update these values during runtime. Changing them manually can desynchronize the VFX Graph from the actual masking volumes and textures.

Thank You!

I hope this asset saves you time, helps you avoid some of the usual headaches of building rain interaction from scratch, and becomes genuinely useful in your project.

I've tried to make both the system and its documentation as practical and transparent as possible. If you have any questions, problems, or feedback, feel free to contact me.

Thank you for your support, and good luck with your project!



**SKULMOVSKI
STUDIO**