# Raymarcher

## Official General Documentation

v3.0.0

| Official Website | APi Documentation | Contact Support | Discord Server |
|---|---|---|---|

It is recommended to open the documentation online to stay up-to-date.

*Latest package version preview-teaser*


*Video documentation series*

# Content

# Raymarcher Essentials

Introduction, features, rendering pipeline, and core structure.
Everything you need to know about Raymarcher.

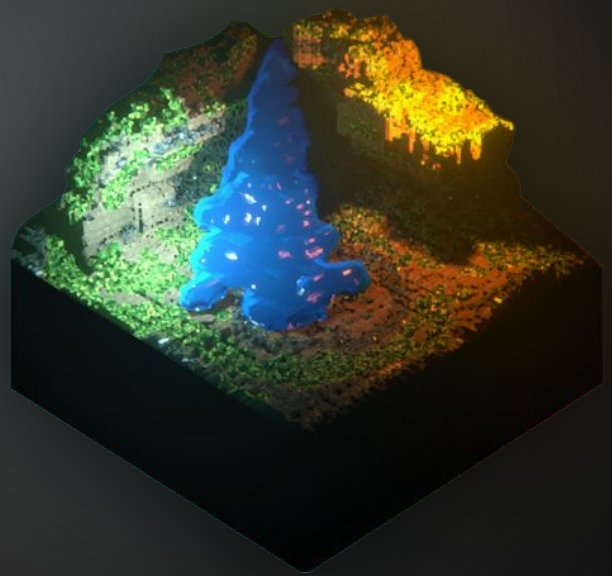
# Introduction

Raymarcher is a universal, cross-platform renderer that features a complete framework for manipulating signed distance functions and volumetric data. The renderer includes a fully customizable material pipeline and comes with an out-of-the-box toolkit library, encompassing a simple voxel system, particle tracking and a series of mesh-to-volume data converters.

**Key Features:**

- One-click setup
- Simple and user-friendly interface
- Collection of SDF primitives, fractals, and volumes
- Collection of well-known SDF modifiers and boolean operations
- Built-in standard material library
- Built-in rendering filters
- Complete toolkit for simple voxel manipulation
- Mesh-to-3D volume converter tools
- Cross-platform support, including VR, mobile & WebGL
- Compatibility with all Unity render pipelines (Built-in, URP, HDRP)
- Complete video documentation series
- Real-time support and access to unlimited updates
- Original example content

Ready for Unity LTS 2021, 2022 and 2023.

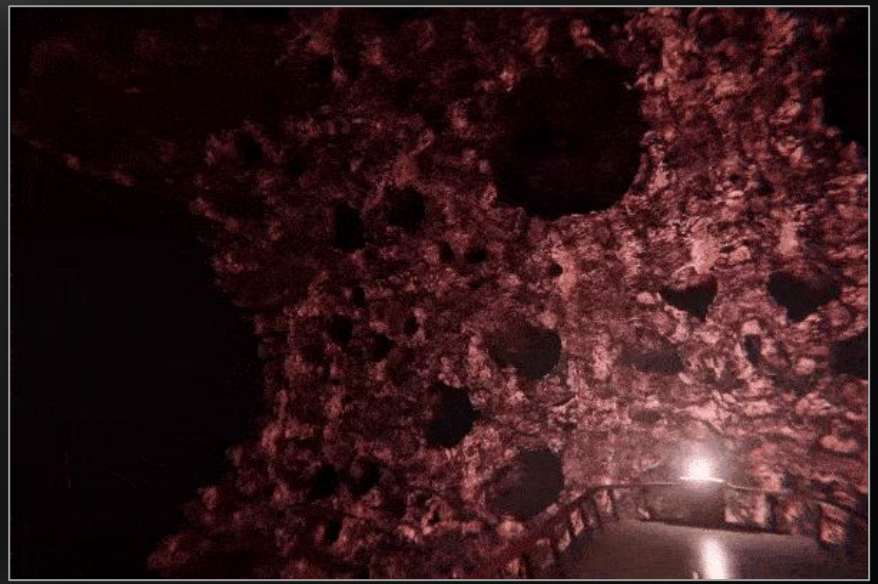


# *Unleash your imagination...*

Raymarcher is an "out-of-the-box" package featuring a core renderer based on the Raymarching rendering technique, a toolkit that ranges from working with simple voxels to volume converters, and more. Once you become familiar with the framework and its pipeline, you can create virtually anything. Take a look at what can be achieved using Raymarcher below:

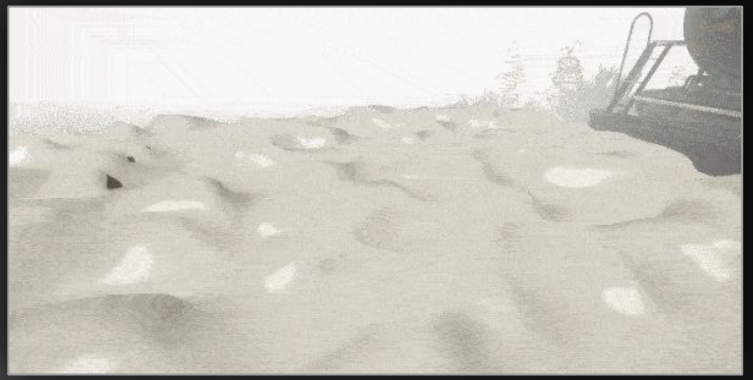
*Manipulating Volume Data - Destructing a Voxel Cave Object*

*3D Fractals - Exploring Fractals from a Different Perspective*



*Particle Tracker - Creating Special Effects Using Raymarcher Materials and Volumes*

*Voxel Modification - Modifying a Volume Surface*



*Clouds - Editing a Simple Cloud Volume Box*

*Creature Editor - Editing Expressions & Shapes of Creatures Using SDFs*

*... From simple 'blobs' and infinite fractals to cloud volumes, unique visualizations, and tiny voxel worlds – all in real-time, directly within* ***Unity Engine****.*

# Key Features

### Simple & friendly

No coding, no dependencies, or required assets. Just choose a session name, target platform, render type, and begin your journey.

### Collection of SDFs

Major collection of primitive, fractal, and volume SDFs, each with its own unique behavior and purpose. Extend with your own formulas.

### Out of the box Toolkit

Complete out-of-the-box toolkit including a simple voxel system, volume character controller, particle tracking, and more. Just drag and drop to play right away.

### Cross platform support

Deploy to any platform, including PC-VR, standalone VR, mobile, mobile cardboard, and WebGL. Anywhere, anytime.

### Universal renderer

Cross-platform & modular renderer based on Raymarching rendering technique. Includes a full API for creating custom render features, SDFs, and materials. Plug in and expand.

### Collection of modifiers

Collection of well-known SDF modifiers, including target operations, deformations, rotations, and more. Modify SDF objects beyond conventional constraints.

### Series of Mesh-To-Volume converters

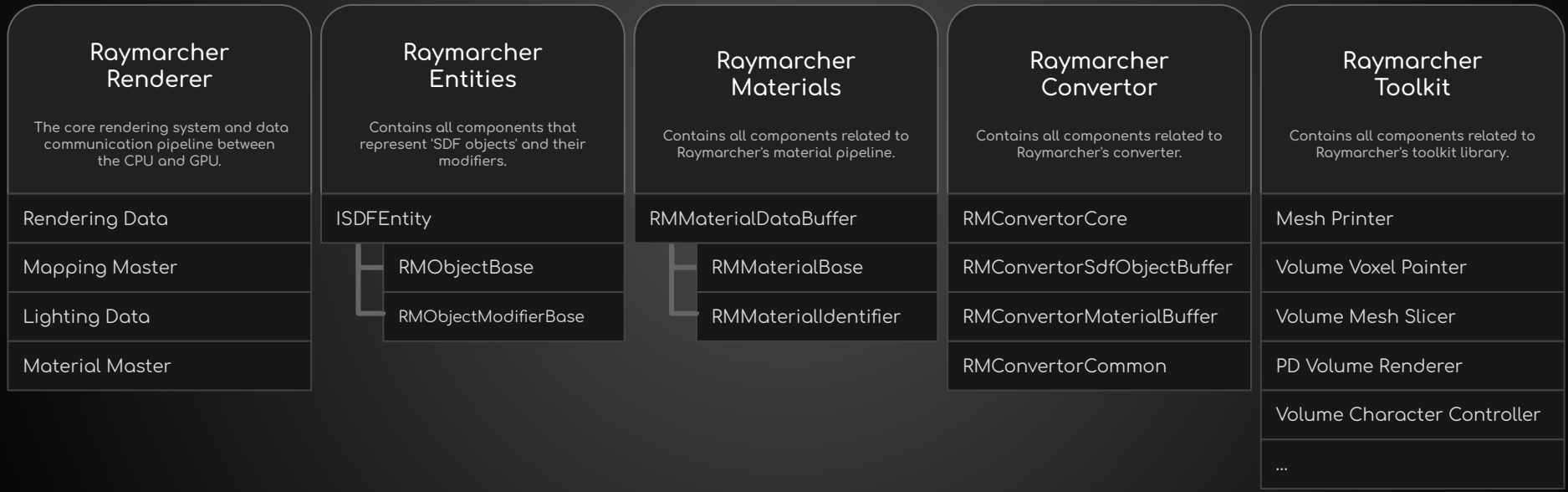
Convert a regular mesh to a volume box/texture 3D using three different techniques: direct mesh print vertex by vertex, perspective-driven volume renderer, and mesh slice renderer.

### Complete Material Library

Collection of standard material libraries, including PBR, unlit, and lit types. Mobile and WebGL-friendly materials are included, with the possibility to create custom materials using Raymarcher's API.

# Core Structure

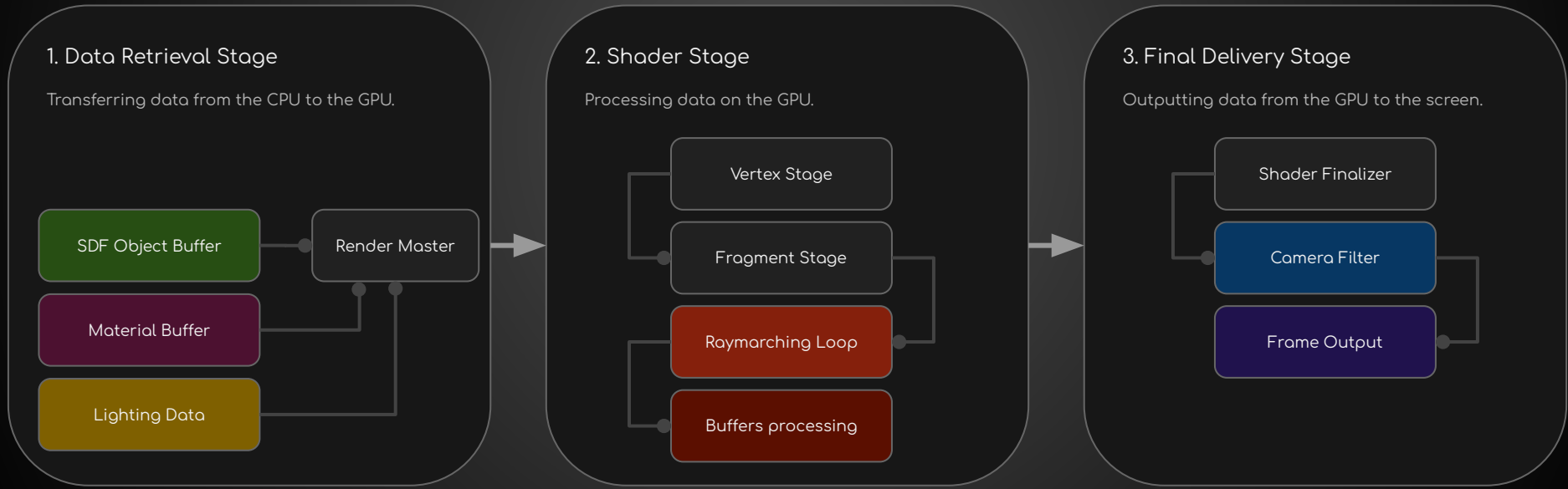
Raymarcher comprises a group of components, each handling its specific area. There are five major groups:




Read more about core structure in the [Technical APi Documentation](#).

# Rendering Pipeline

Raymarcher is a part of Unity's default rendering pipeline. However, Raymarcher expands the pipeline with additional steps necessary to properly process the final frame and deliver it to the screen.




Read more about rendering pipeline in the Technical APi Documentation.

# Compatibility

Raymarcher is compatible with all Unity RPs and uses camera filters to display the currently processed frame by the Raymarcher. However, some render pipelines have certain features limited, depending on the Raymarcher's render type.

Raymarcher features with three different **render types**:

**- Quality**
*Suitable for high-end devices only.*
**- Standard**
*Suitable for all devices, mostly mid-end.*
**- Performant**
*Suitable for mobiles and low-end devices.*

Each render type has its own advantages and disadvantages in terms of memory allocation and performance, and features with different limitations and possibilities of using the Raymarcher components.

| Render Type: | Performant | Standard | Quality |
|---|---|---|---|
| Rendering precision | Low | High | Highest |
| Per-object allocation | 8 bytes | 16 bytes | 32 bytes |
| Object transform synchronisation | Position Only | Full | Full |
| Per-instance material | Global Only | Per Inst + Global | Per Inst + Global |
| Per-instance color | Hue Shift only | Hue Shift only | Full RGB |
| Mobile platform | Supported | Supported | Unsupported |
| WebGL platform | Supported | Supported | Unsupported |
| VR platform | Supported | Supported | Experimental |
| Toolkit library | Partial | Full | Full |

Read more about render types in the Technical APi Documentation.

# Implementation & Setup

Introduction to Raymarcher Implementation and Raymarcher Session.
Your first steps start here.

# First Setup

The setup for Raymarcher is simple and straightforward. Just import the package into your Unity project, extract the necessary Camera Filter dependency (if needed), and create a Raymarcher Session using the Setup Wizard. Everything in three steps.

You can have a look at the complete video documentation/tutorial series.

| | Built In RP | URP | HDRP |
|---|---|---|---|
| 1. | Import the correct package to your Unity project. | Import the correct package to your Unity project. | Import the correct package to your Unity project. |
| 2. | Go to **Window/Raymarcher/Setup Wizard**, and follow the steps. | Go to your **URPAsset** and add a renderer feature with the **Raymarcher Camera URP Filter.** | Go to **Window/Raymarcher/Setup Wizard**, and follow the steps. |
| 3. | | Go to **Window/Raymarcher/Setup Wizard**, and follow the steps. | Go to your **scene,** create a **Custom Pass** gameObject, add the **Raymarcher Camera HDRP Filter** and assign the existing **Raymarcher Session material.** |
| | | If you can't see the **RMCamFilterURP** in renderer features, go to directory **Raymarcher/CameraFilterDependencies** and extract the correct camera filter dependency. | If you can't see the **RMCamFilterHDRP** in custom-pass features, go to directory **Raymarcher/CameraFilterDependencies** and extract the correct camera filter dependency. |

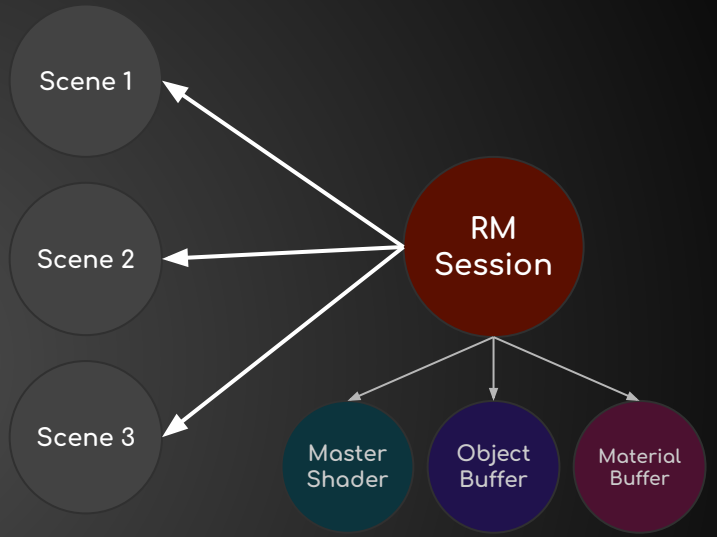
# Raymarcher Session

A Raymarcher Session represents a unique identity for an existing Raymarcher renderer, holding all the necessary references to generated shaders and dependencies. Typically, each Raymarcher Session is bound to a specific Unity scene.

A Raymarcher Session can be removed by either directly deleting the Raymarcher renderer gameObject from the scene or using the 'Remove Raymarcher Session' button on the Render Master. Additionally, a Raymarcher Session can be cloned to create a new, unique copy with all the current data of the specific Raymarcher Session.

Certain actions processed by the user in Raymarcher may require manual recompilation.

There are three types of recompilation requests: **Session Recompilation, SDF Object Buffer Recompilation,** and **Material Buffer Recompilation**. Raymarcher automatically notifies the user to perform the required recompilation based on the specific request.


| Session recompilation is required once the user: | Sdf Object Buffer recompilation is required once the user: | Material Buffer recompilation is required once the user: |
|---|---|---|
| - Changed the target platform<br>- Changed the render type<br>- Modified the list of additional lights | - Added/ removed a sdf object<br>- Added/ removed a sdf modifier | - Added/ removed a material instance on object<br>- Added/ removed a material in global list<br>- Changed a texture on a material instance |

# Render Master Component

Once the Raymarcher Session is created, it is stored in a dedicated component called Render Master. **Render Master** is a core component responsible for managing all Raymarcher objects, materials, and renderer features. This component can be located in the hierarchy with a custom editor style.

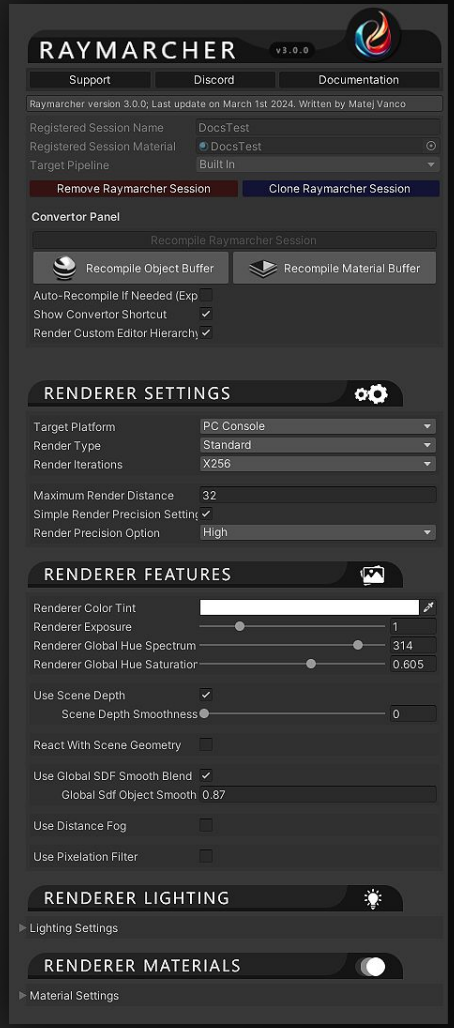
It is not possible to have two or more Render Master components in a scene; only one object with Render Master is allowed. If you use the same Render Master object with an identical Raymarcher Session in multiple scenes, **conflicts may occur** as Raymarcher is **compile-sensitive**. This means that every created SDF object, modifier, or material is constantly written into shader code. In conclusion, use multiple Render Master instances in multiple scenes when you will not change the Raymarcher's hierarchy and will not recompile the object buffer or material buffer.

As mentioned in the Raymarcher Session slide, you can **clone the Raymarcher Session** to keep all the modified data and work with them in a different scene.

The RM Render Master is divided into 5 sections:

- Main session panel
- Renderer settings
- Renderer features
- Renderer lighting
- Renderer materials

Read more about the Render Master in the [Technical APi Documentation](Technical APi Documentation).



# Raymarcher Objects

SDF Primitives, Fractals, and Volumes.
A simple guide to Raymarcher Objects.

# Primitive SDFs

Raymarcher contains a collection of primitive signed distance functions. Each SDF object can generate a different mixture of primitive shapes, allowing for an expanded collection of shapes.

Create a primitive SDF object via GameObject/Raymarcher/Primitives.

The more SDF objects the user creates, the more memory and performance it consumes.

Currently, there are 9 unique primitive SDFs:

- Sphere
- Capsule
- Cone
- Cube
- Rounded Cube
- Cube Frame
- Line
- Metaballs
- Torus

# Fractal SDFs

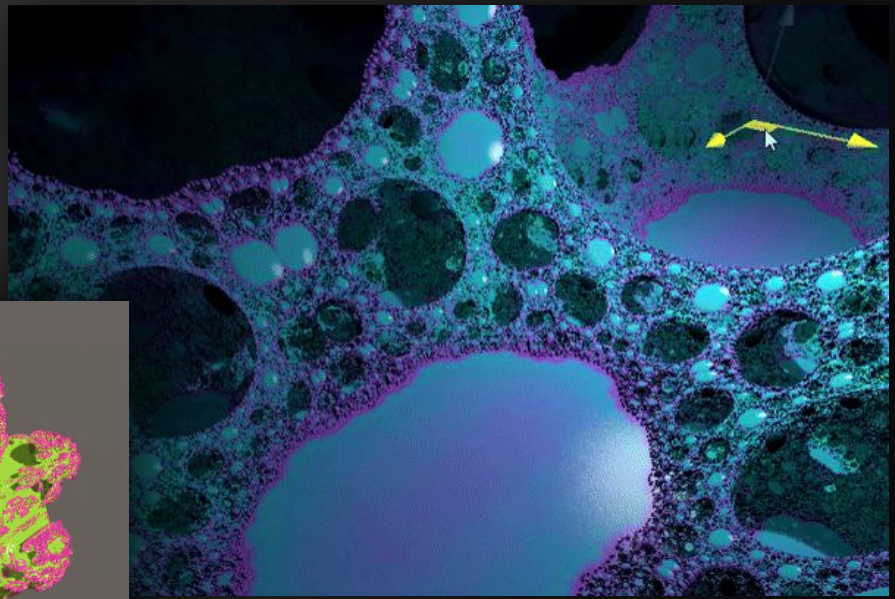
Raymarcher contains a collection of well-known fractal signed distance functions.
Fractals are complex shapes made of recursive formulas.

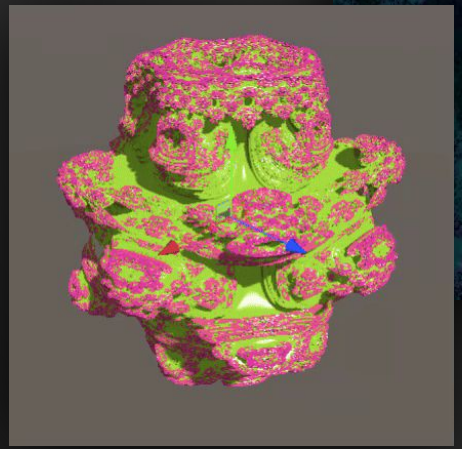
Create a fractal SDF object via GameObject/Raymarcher/Fractals.

It is highly recommended to create just one fractal at a time!

Currently, there are 4 unique 3D fractal SDFs:

- Apollonian
- Kleinian
- Mandelbulb 3D
- Tetrahedron 3D





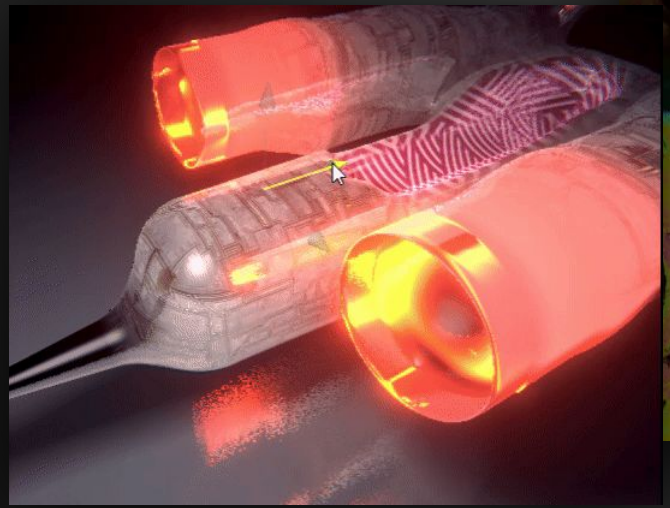
# SDF Modifiers

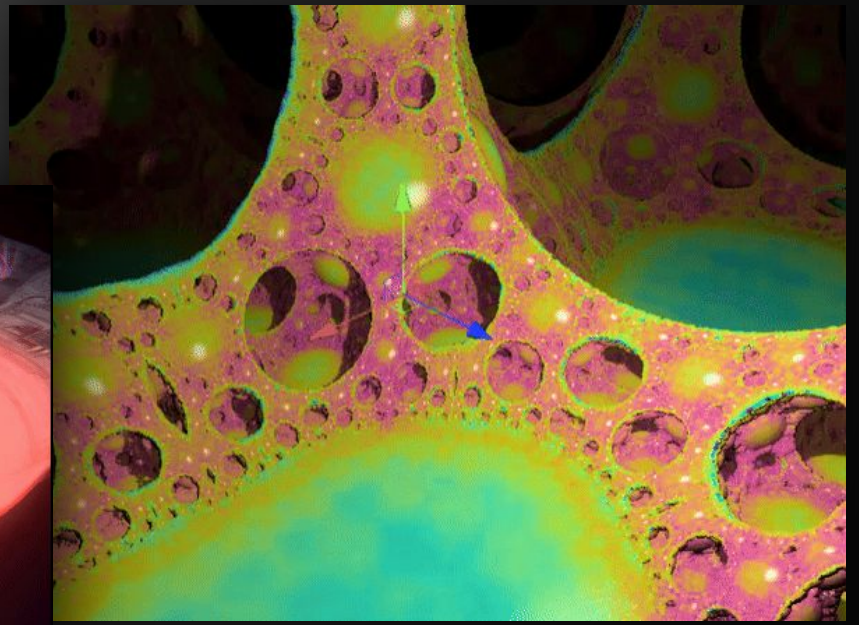
Raymarcher contains a collection of modifiers that modify specific signed distance functions.

Every SDF modifier ***requires a target Raymarcher SDF object***. Add an SDF modifier directly to a target SDF object by clicking on **'Add Modifier'** or search for an SDF modifier component in the **'Add Component'**.

Currently, there are 6 major categories of modifiers:

- Deformations
- Displacement
- Material
- Repeat
- Rotations
- Target Operations





# Volume Box

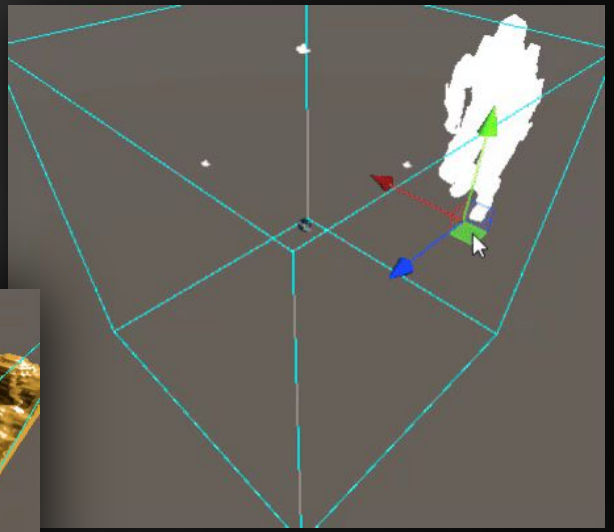
Raymarcher features the processing of volume data and 3D textures. There is a special category of SDFs called 'Volume Box'.

A Volume Box allows users to display a 3D texture with further controls such as pixel amplification, precision, and uniform volume size.

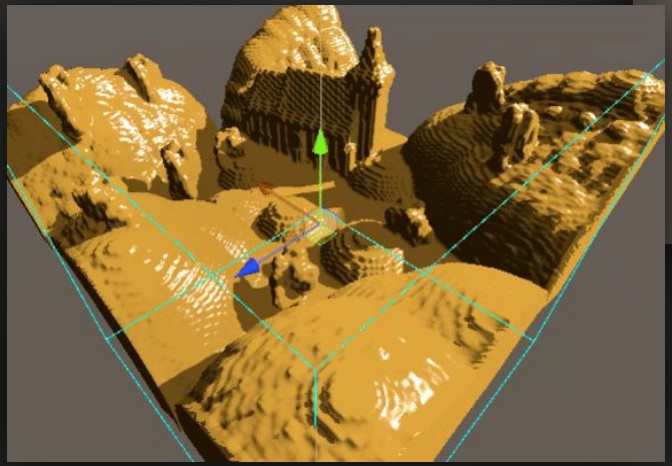
Create a volume SDF object via GameObject/Raymarcher/Volumes.
The more SDF objects the user creates, the more memory and performance it takes.

Currently, there are 2 unique volume sdfs:

- Perspective-driven (PD) volume box
- Texture 3D volume box



*Animated character in PD-Volume Box*



*Switching 3D volumes in Texture 3D Volume Box*

# Raymarcher Materials

Raymarcher materials and their structure.
Everything you need to know about materials in Raymarcher.

# Materials Structure

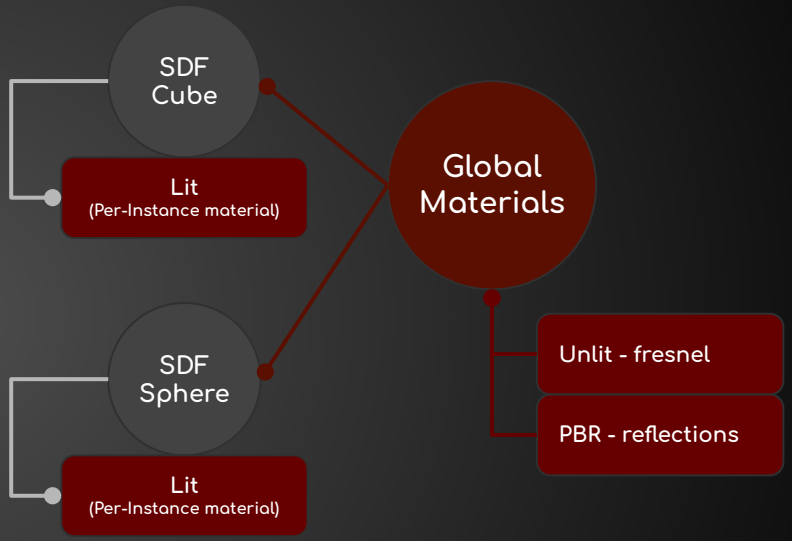
Raymarcher has its own material pipeline that works and handles data in a specific way. Materials in Raymarcher inherit from ScriptableObject, allowing you to create as many material instances as you wish.

The Raymarcher's material pipeline is divided into two major categories:

- Global materials *(applied on all SDF objects globally and equally)*

- Per-instance materials *(applied on individual SDF objects independently)*

All material instances will pass their data to the shader. This is internally done through compute buffers. If your target platform is mobile or WebGL, Raymarcher uses a special type of material types that 'unpack' their data into individual arrays without the use of compute buffers. That's also the reason why materials for mobile/WebGL are simple and contain just a few properties.




Read more about materials structure in the Technical APi Documentation.

# Standard Material Library

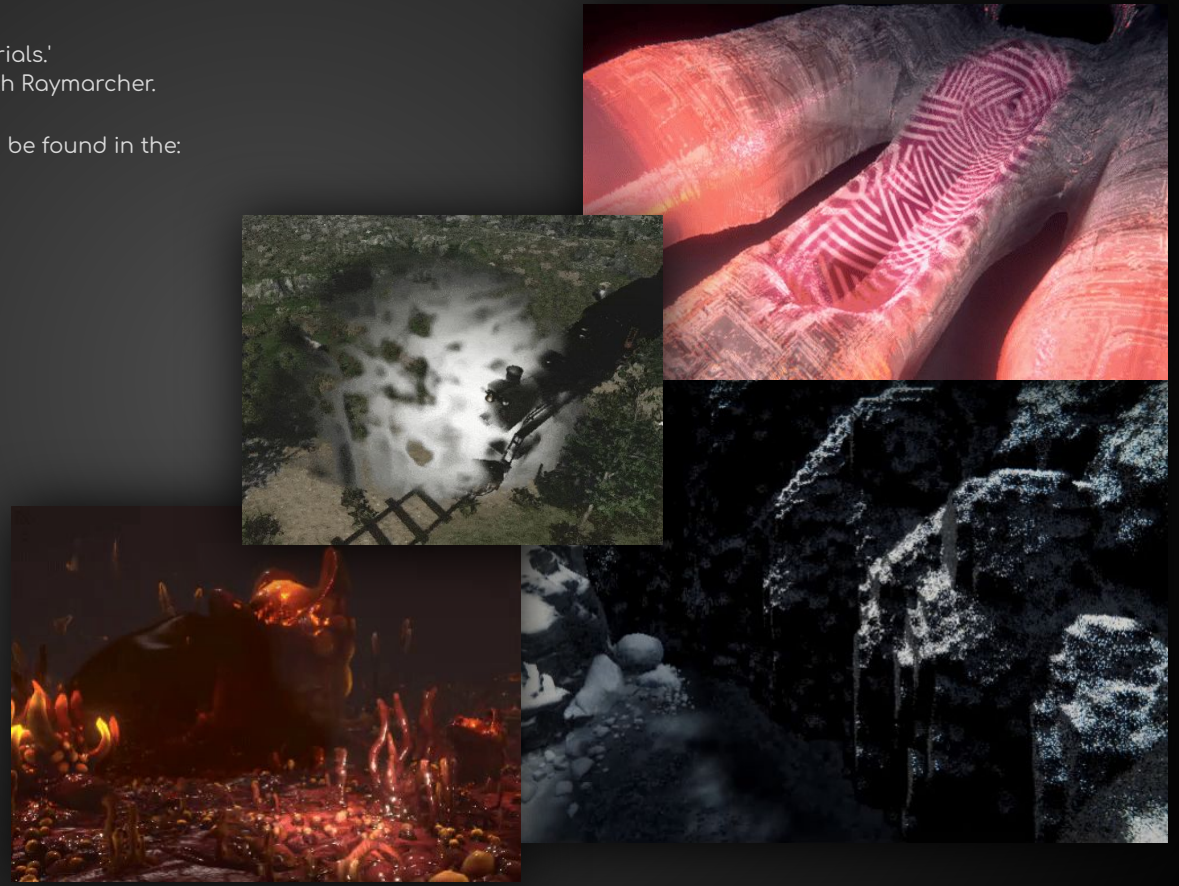
Raymarcher contains its own library of 'standard materials.'
This material library is designed specifically for use with Raymarcher.

All the materials belonging to the standard library can be found in the:

Project window/Create/Raymarcher/Materials

The library consists of 6 unique material types:

- Unlit
- Lit
- PBR
- Slope Lit
- Mobile Simple Lit
- Volume Noise (Clouds)



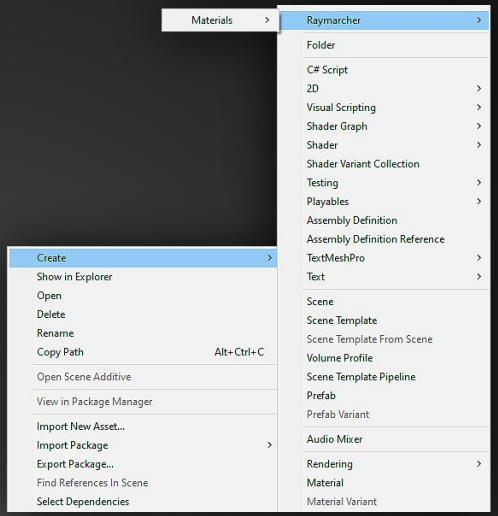
Read more about standard material library in the Technical APi Documentation.

# Creating a Material

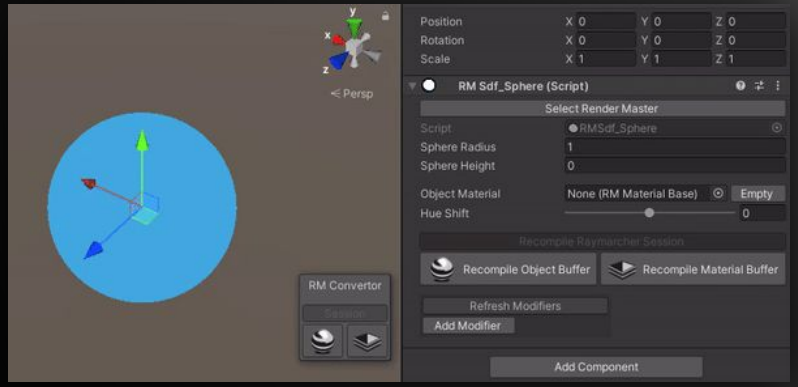
Creating a Raymarcher material is simple. Just go to the Project window, right-click, and choose **Create/Raymarcher/Materials**.

If your render type is set to **Standard** or **Quality**, you can assign the created material instance to individual sdf objects. If your render type is set to **Performant**, you can only use global materials.

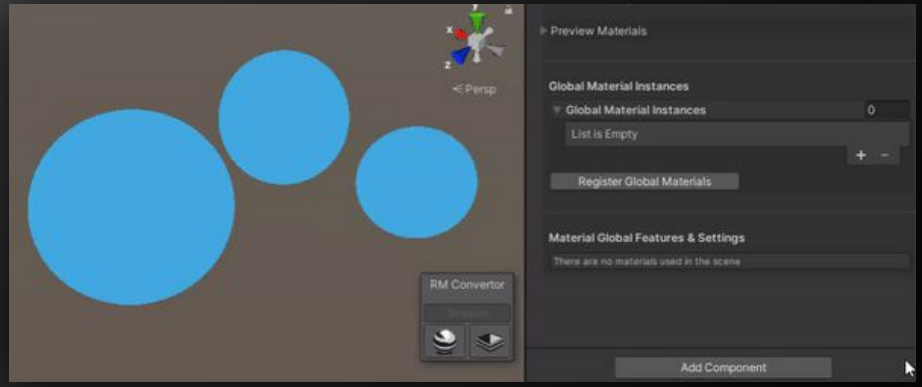
Create an array element in global materials on RM Render Master, assign your created material instance, and register the global instances array. The registered material instances will affect all the sdf objects in the session.




*Assign a material to individual objects if the render type is set to **Standard** or **Quality***




*Assign a material to Global Material instances array if the render type is set to **Performant***



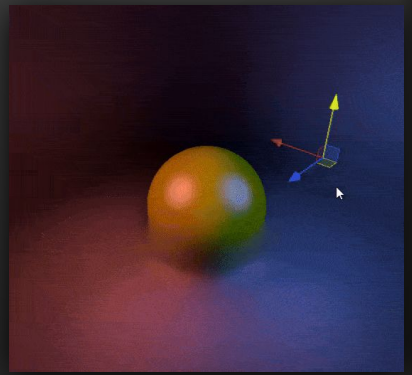
# Lighting

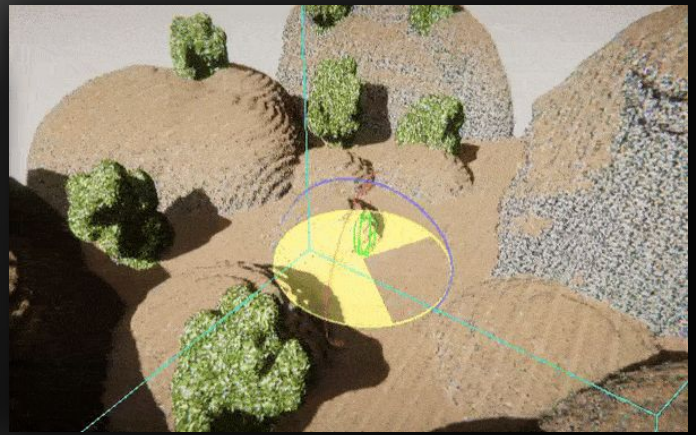
Raymarcher in the current version supports just one directional light and an unlimited number of additional point lights. All the light references can be found on the main RM Render Master component.


Lighting is highly dependent on the material type that you are using. If the material type does not support lighting, the provided light references on the RM Render Master component would not be used. It is highly recommended to keep the additional point lights count as low as possible, as the Lit, PBR, Slope, and Noise material types in the Standard Material Library calculate lighting per-pixel, meaning that a loop of additional lights is processed every fragment shader invocation.

List of material types from Standard Library that support lighting:

- Mobile Simple Lit (directional only)
- Lit (directional and add lights)
- Slope Lit (directional and add lights)
- Noise (add lights only)
- PBR (directional and add lights only)





# Raymarcher Toolkit

Voxels, volume character controller, mesh to volume converters, fluids, and liquids.
The extended toolkit of Raymarcher unlocks the true power of real-time processing.

# The Toolkit

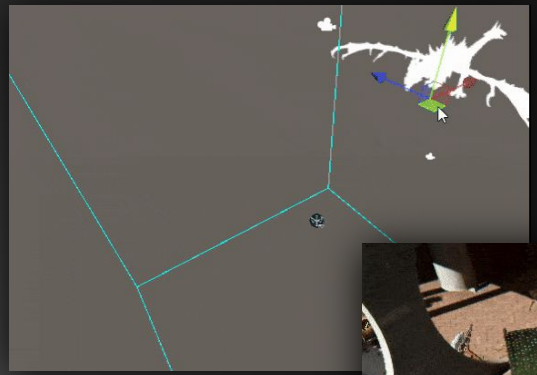
Raymarcher features an out-of-the-box toolkit library consisting of various volume modifications, voxel painting, mesh-to-volume renderers, and more. The toolkit is entirely written in C# and Compute Shaders, resulting in limited platform compatibility (most toolkit components are not supported on mobiles and WebGL). Raymarcher is a universal-purpose package, enabling you to create any type of volumetric-generated content within certain limitations.

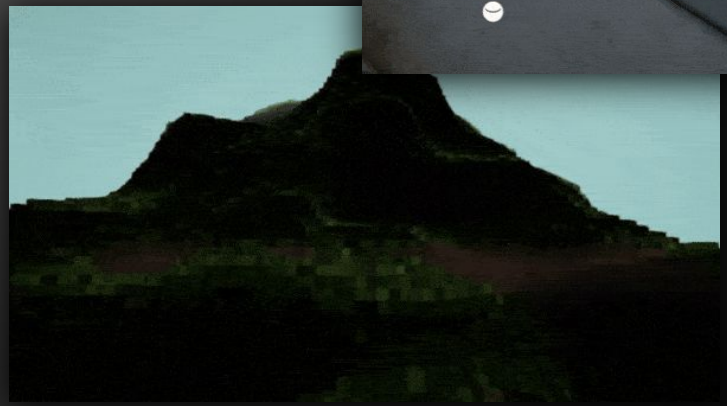
Raymarcher's toolkit library is organized into three subfolders:

1. Volume modification
- Volume character controller
- Voxel painting (with materials override feature)
- Volume depth sampler
- Volume particle tracker

2. Volume Convertors
- Volume mesh printer
- Volume mesh slide renderer
- Volume perspective-driven renderer

3. Experimental - experimental components (not production ready)
- Volume point collision sampler





Read more about Raymarcher toolkit library in the Technical APi Documentation.

# Voxels

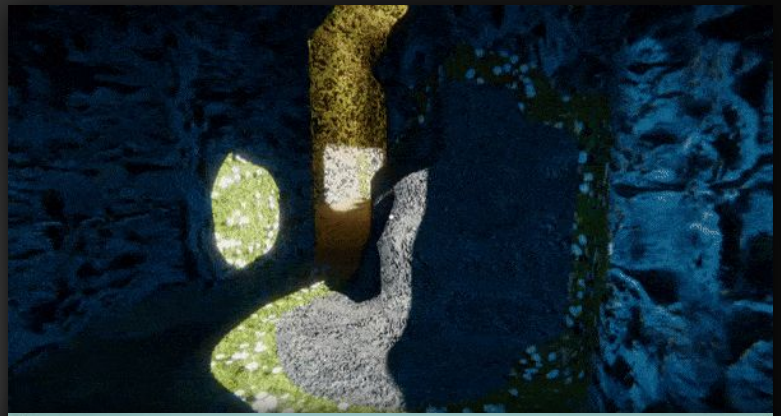
**Raymarcher is not a voxel engine,** but it offers the possibility of creating and handling voxels on a smaller scale. In Raymarcher, voxels are defined in specific 3D render textures created with a common resolution.

There are six common volume resolutions in Raymarcher:
**x16, x32, x64, x128,** and **x256.**

It's important to note that Raymarcher does not provide a solution for creating an infinite world of voxels with dynamic rescaling. Additionally, the voxels in Raymarcher do not work with the octree algorithm; instead, all voxels are defined in each 3D texture pixel and a volume box.

There are various ways to manipulate voxels in Raymarcher:

- Convert regular mesh geometry to voxels
*(You can't convert sdf to regular mesh geometry)*
- Manually paint/erase voxels
- Sample the depth of voxel volume
- Track particles in the target particle system and create voxels on each particle
- Save voxels (RT3D) to the assets and reuse the 3D texture for a different purpose



Read more about Raymarcher voxels in the Technical APi Documentation.

# Voxel Painting

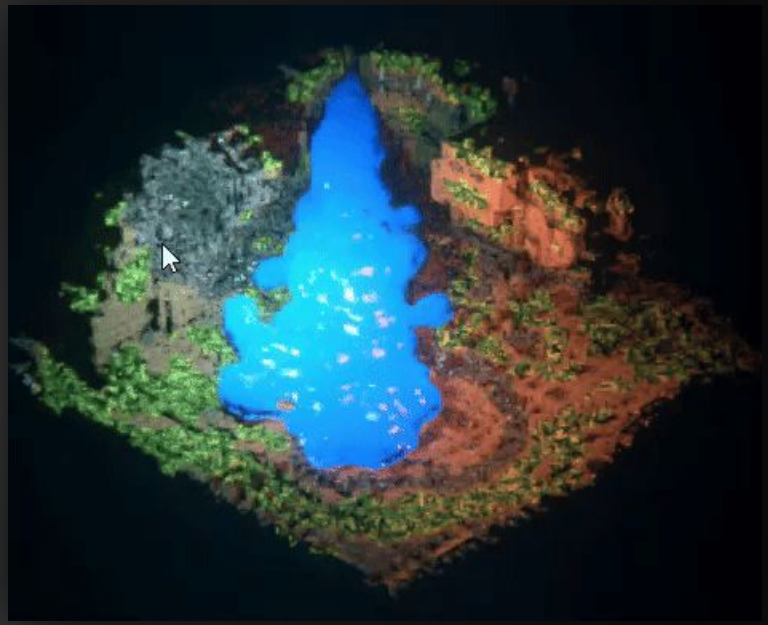
Raymarcher already provides a straightforward solution for voxel painting (or reading). There are two ways you can implement such a feature into your project:

- Use the **Volume Voxel Painter** API featured by Raymarcher and write the functionality yourself.

- Utilize example scripts, such as **RMSample_VolumeVoxelPainterFPS** from the example content.

Voxel painting in Raymarcher is calculated in compute shaders, where pixels are modified by a specific 3D render texture used to display the volume data within a particular volume box.

For a detailed, step-by-step tutorial on setting up voxel painting in a specific volume box, you can refer to the video tutorial or read the corresponding section in the technical documentation.



Read more about voxel painting in the Technical APi Documentation.

# Volume Character Controller

The toolkit library includes an out-of-the-box character controller for volume boxes. The Volume Character Controller enables you to move around the target volume box with a simple collision system.

It's important to note that this character controller specifically works with Raymarcher's Texture 3D Volume Box, which contains tex3D data. Currently, it doesn't support a flying volume character controller; only a gravity-based one is available.

To create a volume character, navigate to GameObject/Raymarcher/Toolkit/VolumeCharacterController.

For a detailed, step-by-step tutorial on setting up volume character controller in a specific volume box, you can refer to the video tutorial or read the corresponding section in the technical documentation.

Read more about Volume Character Controller in the Technical APi Documentation.

# Particle Effects

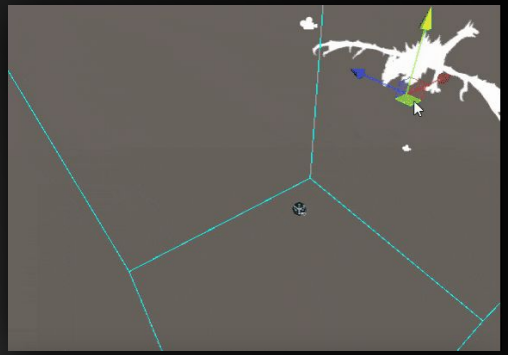
Special particle effects in the Raymarcher are essentially projected particles from the regular Unity Particle System. However, it's important to note that Raymarcher doesn't currently include advanced liquid/fluid simulations using Navier Stokes equations or other solutions.

The Raymarcher Toolkit features a component called **Particle Tracker**, which converts each living particle to 'volume space' and generates a point on a working 3D render texture. This texture is then used to display the 3D volume content on the target volume box. **The maximum number of living particles allowed in one simulation is 1024**.

Similar to other Raymarcher sdf objects, these effects can have various materials with different visuals, allowing for unique and creative results.

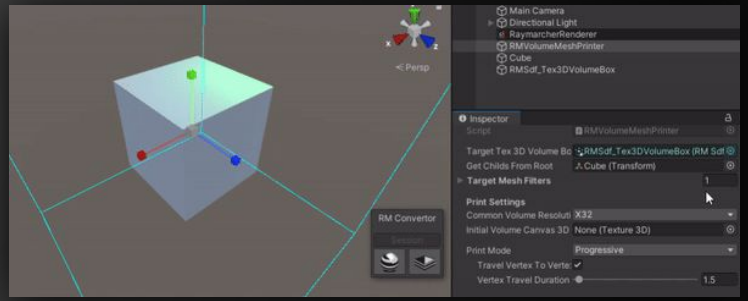
Read more about Particle Tracker in the Technical APi Documentation.

# Mesh To Volume



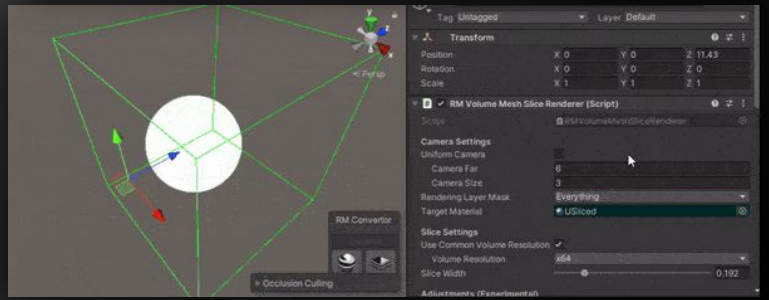
Raymarcher provides three different methods for converting/rendering regular meshes into volume data (specifically into a 3D texture):

1. Mesh Printer
- Prints the vertices of a target mesh instantly or progressively into a volume box.
- Advantage: Prints high-poly meshes.
- Disadvantage: May be noticeable on low-poly meshes. Not compatible with mobiles/webGL due to GPU resource consumption.
- Settings include target tex3DVolumeBox, volume resolution, initial volume canvas, print modes (Instant or Progressive), and brush parameters.




2. Mesh Slice Renderer
- Renders target meshes with the 'slice shader' and packs the rendered slices into a 3D texture.
- Advantage: Independent of vertex count.
- Disadvantage: Less precise as it renders the 'white' pixels of the current slice progress. Meshes with faces aligned perpendicularly towards the camera won't be captured. Works in the Unity Editor only.
- Settings include target 'slice' material, slice width, and common volume resolution.

3. Perspective Driven Volume Renderer
- Renders targets from three different perspectives.
- Advantage: Fast and suitable for prototyping volumetric objects.
- Disadvantage: Not precise, as it approximates 3D data based on three 2D textures.
- Settings include a volume renderer tool assigned to a perspective-driven volume box.




Each option has its own set of advantages and disadvantages in terms of precision and data storage. These converters can be accessed via GameObject/Raymarcher/Toolkit.

Read more about volume convertors in the Technical APi Documentation.

# Mesh Printer

The Mesh Printer is one of the most effective methods for converting regular meshes into volumes within Raymarcher. It accomplishes this by directly printing the mesh vertices into a 3D texture.

Two print modes are available:

Instant

- Prints the entire mesh instantly.

Progressive

- Prints the mesh progressively every frame.
- Option to travel from vertex-to-vertex over a specific duration.

The Mesh Printer offers flexibility to print any mesh into a volume, allowing for modification of its content at runtime. This dynamic approach provides versatility in creating and manipulating volumetric data within the Raymarcher framework.

# Technical Details

Limitations, Warnings, and Testing Benchmark.
Essential pre-reads for Raymarcher Renderer usage.

# Limitations

## General

The latest version of Raymarcher includes support for URP and HDRP. Both render pipelines function well, but it's important to note that the VR mode is still in an experimental version. If you are targeting VR, be aware that there may be issues as it is still in development.

Raymarcher is not compatible with orthographic cameras or 2D mode; it exclusively supports perspective cameras and operates in 3D mode only.

Raymarcher partially supports prefabs, but they must be appropriately compiled in a specific Raymarcher Session and cannot be instantiated or removed at runtime.

Raymarcher for VR does not work with single-pass rendering; it exclusively supports multi-pass rendering mode.

## Visual

Raymarcher does not support transparent meshes. Transparent meshes will either be drawn in front of or behind the Raymarcher objects, depending on the renderer feature drawing queue.

Raymarcher supports point and directional lights only. Spotlights and area lights are not supported.

Regular meshes do not receive shadows from Raymarcher objects, and Raymarcher objects do not receive shadows from regular meshes.

## Specific

Raymarcher is computationally intensive, and even with the Performant render type, it imposes additional performance demands. As a result, many Raymarcher toolkit components are not fully supported on mobile and WebGL platforms, particularly features like voxel painting and particle tracking.

Raymarcher is a compile-time renderer, which implies that users cannot create sdf objects at runtime. To achieve dynamic modifications, it is recommended to make necessary adjustments in advance and create a pool of objects accordingly.

Raymarcher does not have the capability to convert signed distance function (SDF) objects into meshes. It can only convert regular meshes into volumes (tex3D).

Raymarcher does not support point cloud data; it can only work with volumetric data represented through 3D textures and signed distance functions.

# Warnings & Recommendations

## Unity Editor

HDRP is the least tested render pipeline for Raymarcher. If you plan to use Raymarcher in commercial or production projects, it is recommended to use URP or Built-In render pipelines.

It is recommended to always use the latest Unity LTS versions. The minimum supported Unity version is Unity 2021.

Some post-processing effects don't work with Raymarcher, particularly those that use depth texture or depth normals, such as Depth of Field (DoF) or Motion Blur.

## Raymarcher

It is recommended to create as few SDF objects as possible. The more SDF objects you create, the higher the memory allocation and performance impact. Please carefully consider the creation of SDF objects.

Most of the time, Raymarcher runs primarily on the GPU. However, as every GPU is different, it is not guaranteed that all features will work equally well on every GPU.

Raymarcher is a universal-purpose renderer, meaning that it doesn't have a specific focus on any of the listed features. Mixing multiple volume SDFs with complex data and fractal SDFs might lead to extreme FPS drops and performance issues.

It is not recommended to use Global SDF Blend while using SDF volume boxes. Volume boxes utilize approximated formulas and are bounded by AABB boundaries. Disable Global SDF Blend when working with volume boxes.

# Testing Benchmark - PC Standalone

| OS | GPU | CPU | RAM | Display | Results |
|---|---|---|---|---|---|
| Windows 10 (PC) | NVIDIA GeForce RTX 4060 Ti | Intel i7~3.6GHz (8 CPUs) | 16 GB | 2560x1440 (144Hz) | Stable (144 FPS) *Full Settings* |
| Windows 10 (PC) | NVIDIA GeForce RTX 2070 S | Intel i9~3.7GHz (20 CPUs) | 64 GB | 1920x1080 (60Hz) | Stable (60 FPS) *High Settings* |
| Windows 10 (PC) | NVIDIA GeForce GTX 1070 | Intel i7~3.3GHz (8 CPUs) | 16 GB | 1920x1080 (60Hz) | Stable (60 FPS) *Medium Settings* |
| Windows 10 (PC) | NVIDIA GeForce GTX 950 | Intel i7~3.3GHz (12 CPUs) | 48 GB | 1920x1080 (60Hz) | Stable (60 FPS) *Low Settings* |
| OSX Sonoma 14.1 (Laptop) | M1 Built-in | M1 (8 cores) | 16 GB | 3024x1964 (ProMotion) | Stable (60~75 FPS) *Low Settings* |

Subject - Fractal Sailor *(Built via URP)*

Fractal Sailor is an experimental, atmospheric horror tech demo where you take the role of a lone sailor navigating a hovercraft through a dark and mystic fractal environment. The scene contains two point lights with soft shadows, one Apollonian fractal that changes its shape in a random timespan.

# Testing Benchmark - PC Standalone

| OS | GPU | CPU | RAM | Display | Results |
|---|---|---|---|---|---|
| Windows 10 (PC) | NVIDIA GeForce RTX 4060 Ti | Intel i7~3.6GHz (8 CPUs) | 16 GB | 2560x1440 (144Hz) | Stable (144 FPS) |
| Windows 10 (PC) | NVIDIA GeForce RTX 2070 S | Intel i9~3.7GHz (20 CPUs) | 64 GB | 1920x1080 (60Hz) | Stable (60 FPS) |
| Windows 10 (PC) | NVIDIA GeForce GTX 1070 | Intel i7~3.3GHz (8 CPUs) | 16 GB | 1920x1080 (60Hz) | Stable (60 FPS) |
| Windows 10 (PC) | NVIDIA GeForce GTX 950 | Intel i7~3.3GHz (12 CPUs) | 48 GB | 1920x1080 (60Hz) | Stable (60 FPS) |
| OSX Sonoma 14.1 (Laptop) | M1 Built-in | M1 (8 cores) | 16 GB | 3024x1964 (ProMotion) | Partially Stable (40~60 FPS) |

Subject - Tiny Voxels *(Built via URP)*

Tiny Voxels is a short tech demo featuring voxel manipulation. There is no specific goal - you can freely explore, experiment with the sandbox control panel (adjusting voxel details, brush intensity, brush size, etc.), build or destroy voxels, print a house, and control lights. The scene contains four point lights and one directional light with soft shadows, one volume character controller with medium settings, one voxel world with 'mirrored-repeat' modifiers on Z and X axes.

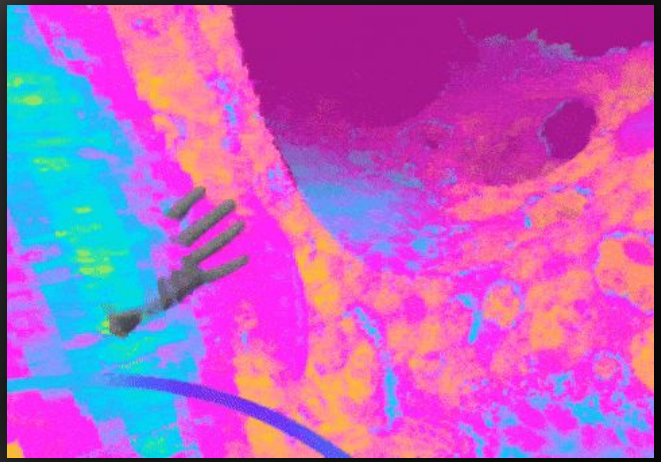
# Testing Benchmark - VR Standalone

| OS | GPU | CPU | RAM | Display | Results |
|---|---|---|---|---|---|
| MetaOS 3 (Android) | Built-in Meta | Built-in Meta Qualcomm Snapdragon XR3 chipset | 8 GB | 3840x2160 (120Hz) per eye | Stable (120 FPS) |
| | | | | | |

Subject - Fractal Trip *(Built via Built-InRP)*

The player floats in a vast fractal environment, moving automatically forward. The scene contains no lights, only simple shading.

The demo is running on **Meta Quest 3**.



# Extras

Example content, products that already use Raymarcher, and video documentation series.
Discover the creative possibilities and projects developed with Raymarcher.

# Example Content

Feel free to test your GPU capabilities with example content created using Raymarcher.
*Just click the image.*


**PC**
Executable
Win-only



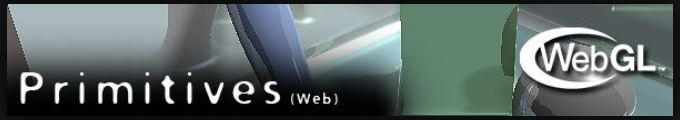
**Standalone VR**
Meta Quest



**PC-VR**
Executable
Win-only
Unity Open XR

**WebGL**
Play in
Browser



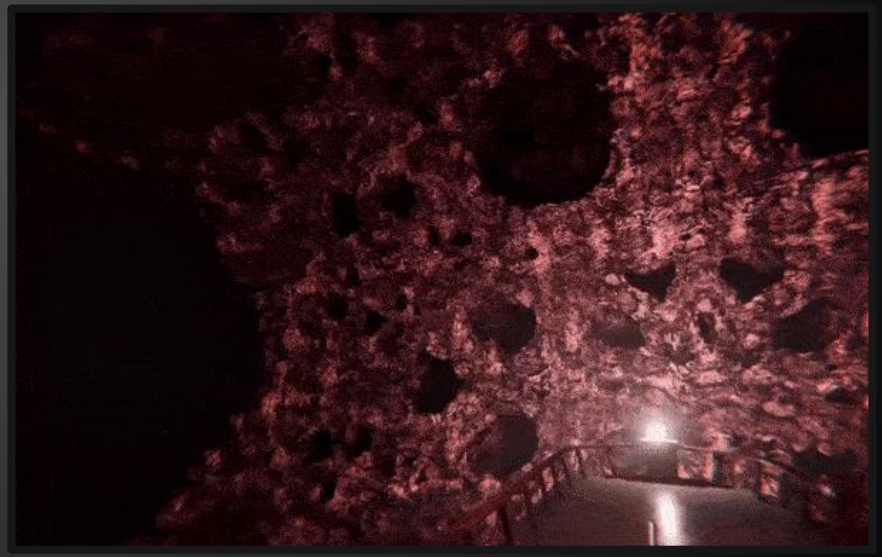



# Technical Demos

Feel free to play these complete tech demos created using Raymarcher.
*Just click the gif.*

### Tiny Voxels
PC Windows - MacOSX

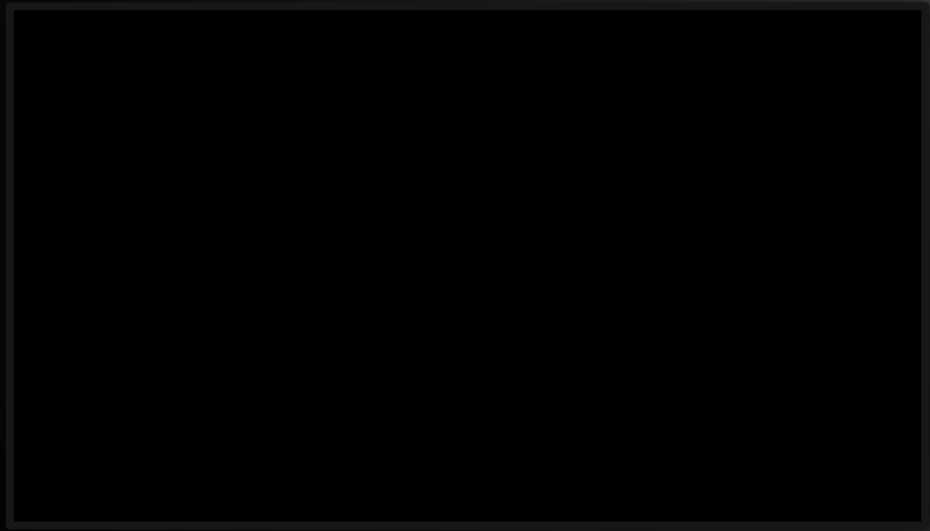
### Fractal Sailor
PC Windows - MacOSX



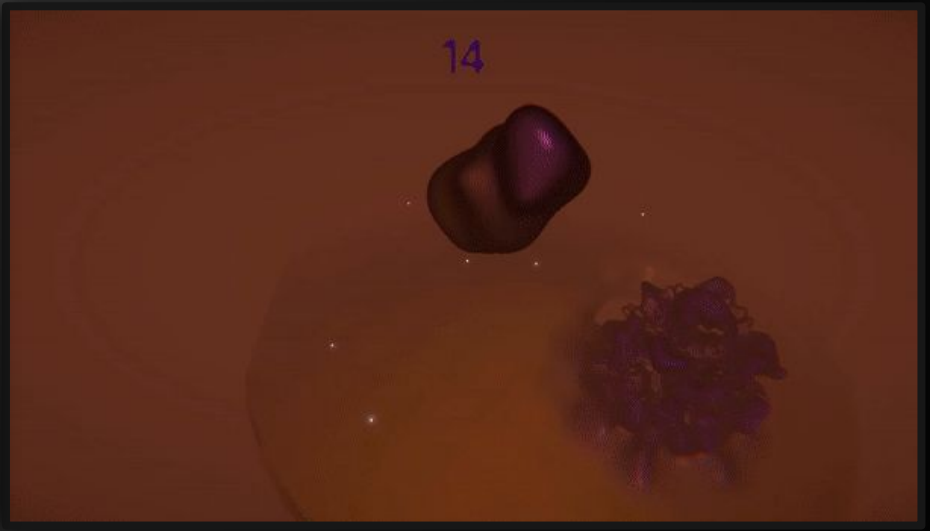
# Commercial Products

Discover products that use the capabilities of the Raymarcher renderer.
Explore and see what creations have been made using Raymarcher
in a commercial industry. *Just click the gif*

Cellings (PC - 2022)



Refractor Dilemma (iOS - 2021)



# Video Doc Series

Are you tired of reading through the entire documentation? Sit back, grab some popcorn, and relax. Raymarcher comes with a complete series of video documentations.

# FAQ

How does the Raymarcher behave on iOS or Android?

Most of the time Raymarcher runs purely on GPU, meaning that if you are using multiple render features (soft shadows, reflections) it becomes pretty powerful and takes a lot of memory. However if the user chooses a proper render type (Performant), a mobile-friendly material and keep the sdf object count as low as possible, Raymarcher is pretty safe to use on mobile devices. It also all depends on what mobile model you use. The newer GPU, the better.

Does the Raymarcher work on Apple Vision Pro and Meta Quest 3+?

Yes, Raymarcher was tested on Meta Quest 3 with realtime liquids, and the performance remained stable, averaging 50 frames per second or more. However, performance can vary depending on the number of render features the user employs and the chosen target render type.

Does the Raymarcher work with Unity fog?

No, Raymarcher doesn't work with Unity fog. Raymarcher has its own fog feature which is called Distance Fog and can be found under the Renderer Features.

How liquids and fluids work in the Raymarcher?

'Liquids' in Raymarcher are basically regular Unity particles, projected into a volume box using a 3D texture. The Raymarcher calculates each particle, generating a 'volume point' that blends with other surrounding points according to predefined parameters such as precision and blending. There is no Navier Stoke equations involved, however I would like to expand the toolkit with more advanced techniques as is this one!

+If you are using a Particle Tracker, your target particle system will be limited to a maximum of 1024 particles.

Does the Raymarcher work with Forward rendering path?

Yes, Raymarcher works with both Deferred & Forward.

# Thank You

Thank you for your attention! If you have any questions, suggestions or issues, do not hesitate and reach out to me on my official Discord server. *(just click the image below)*


If you don't like Discord, write me a direct message on my email address.