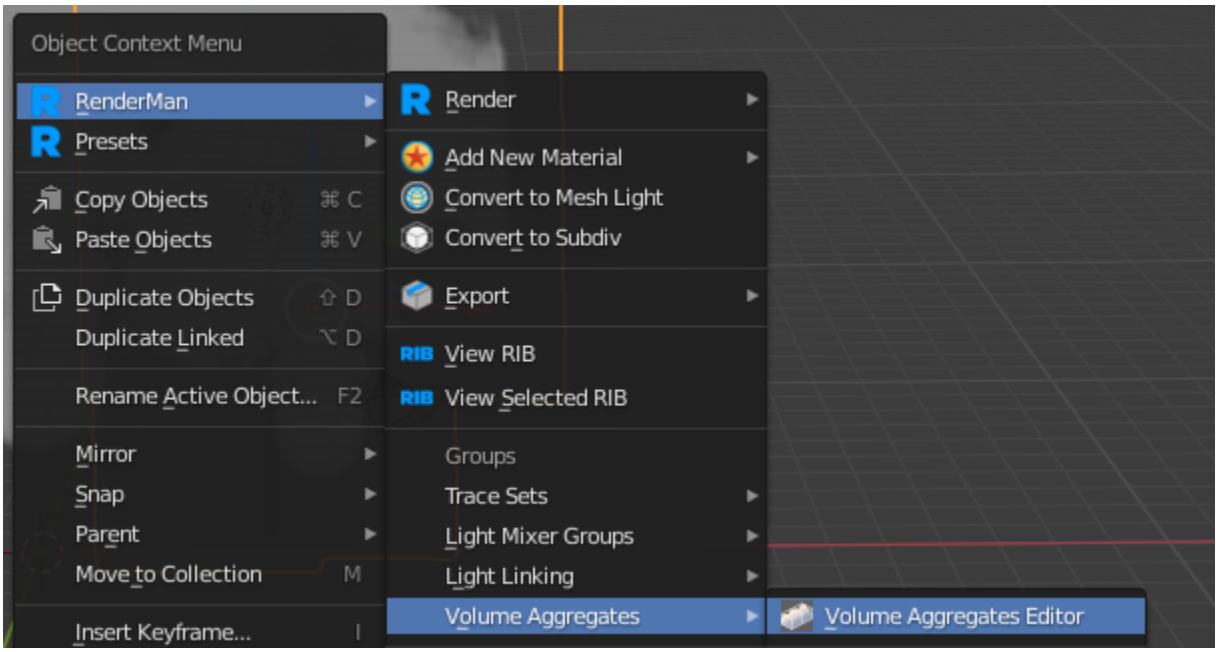
Aggregate Volumes in Blender

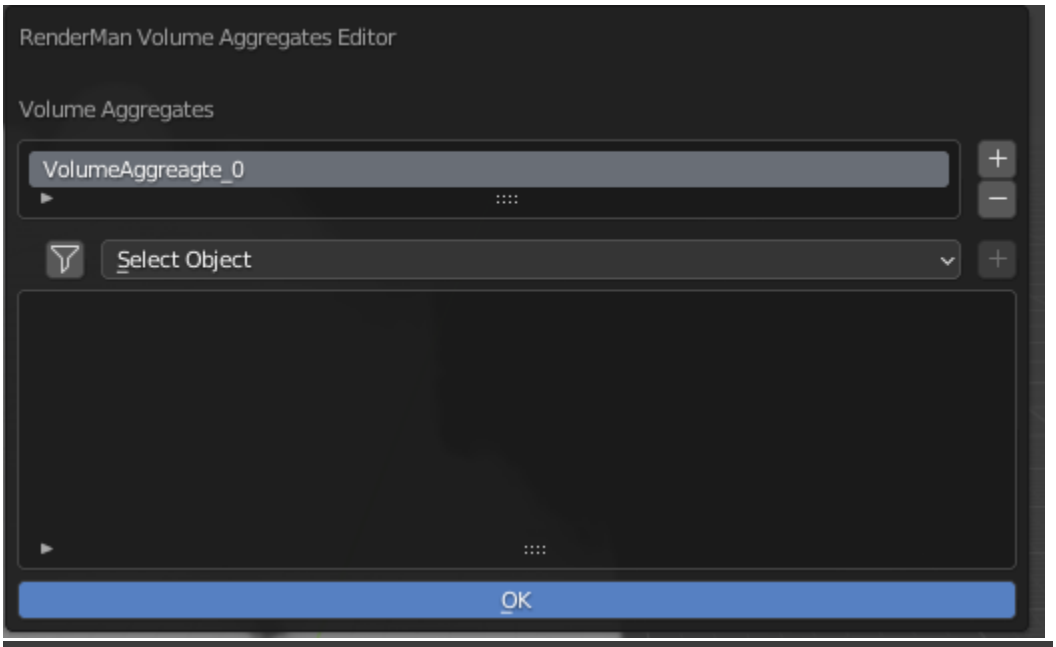
Using Volume Aggregates in Blender

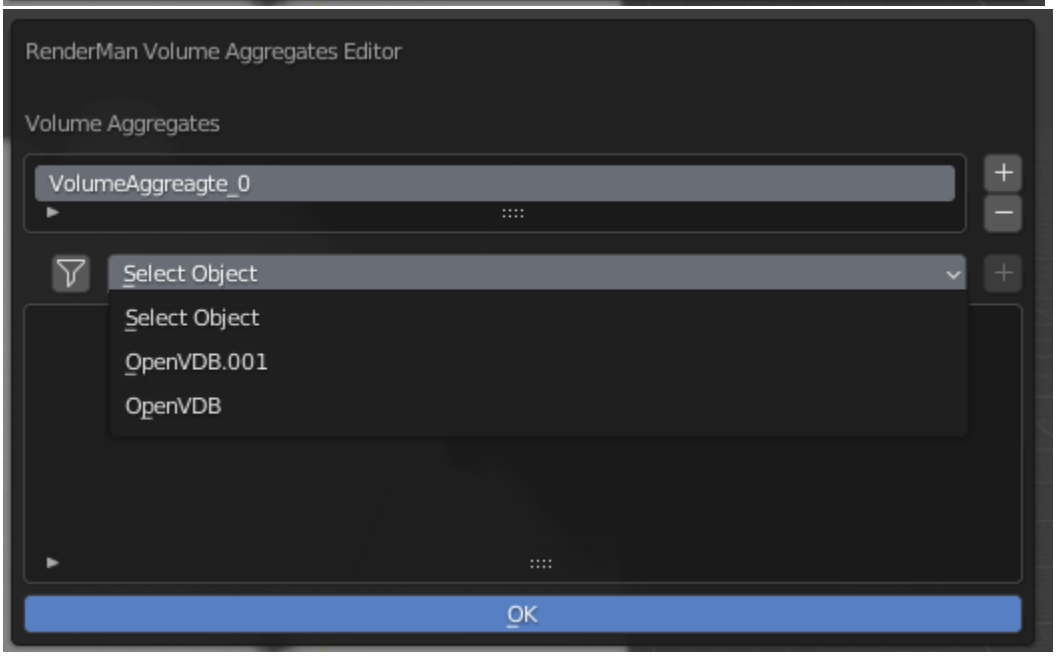
Volume Aggregates Editor

To create a volume aggregate, you can use the Volume Aggregate Editor, which can be accessed from the right click menu in the viewport:

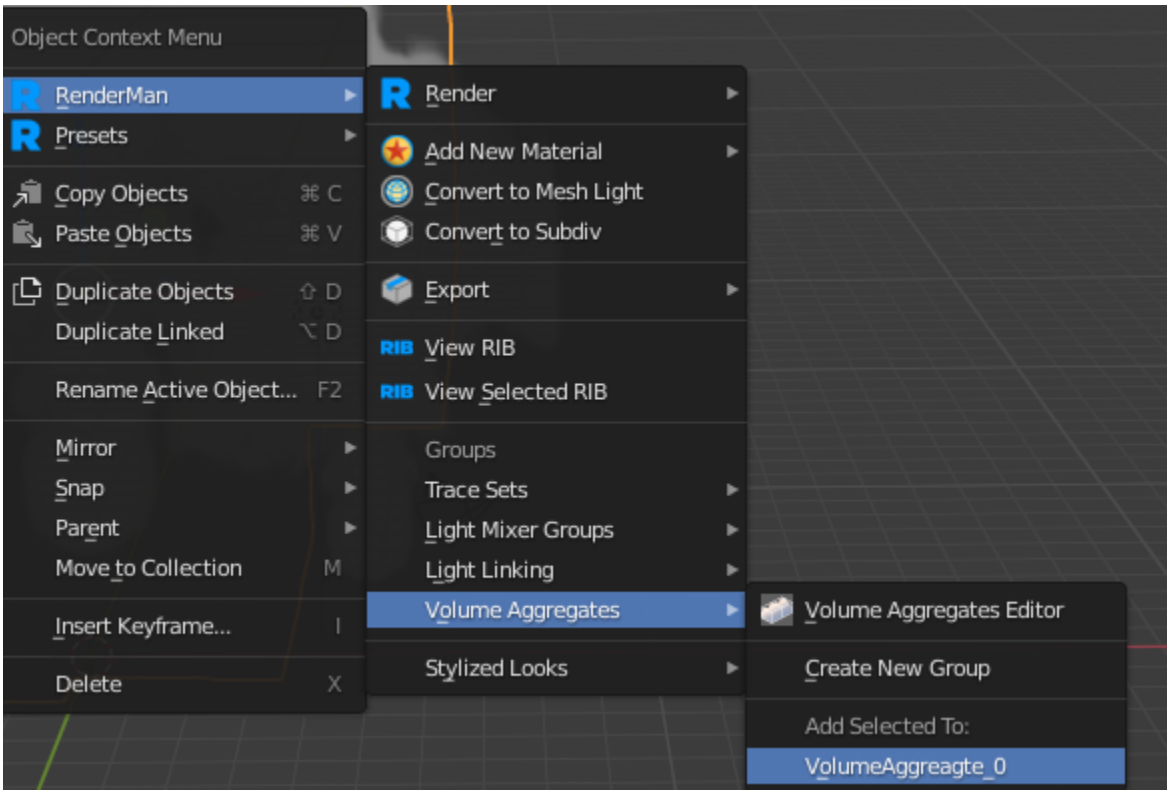


Hit the + button, to create a new aggregate. You can then start adding volumes that are in the scene to the aggregate:



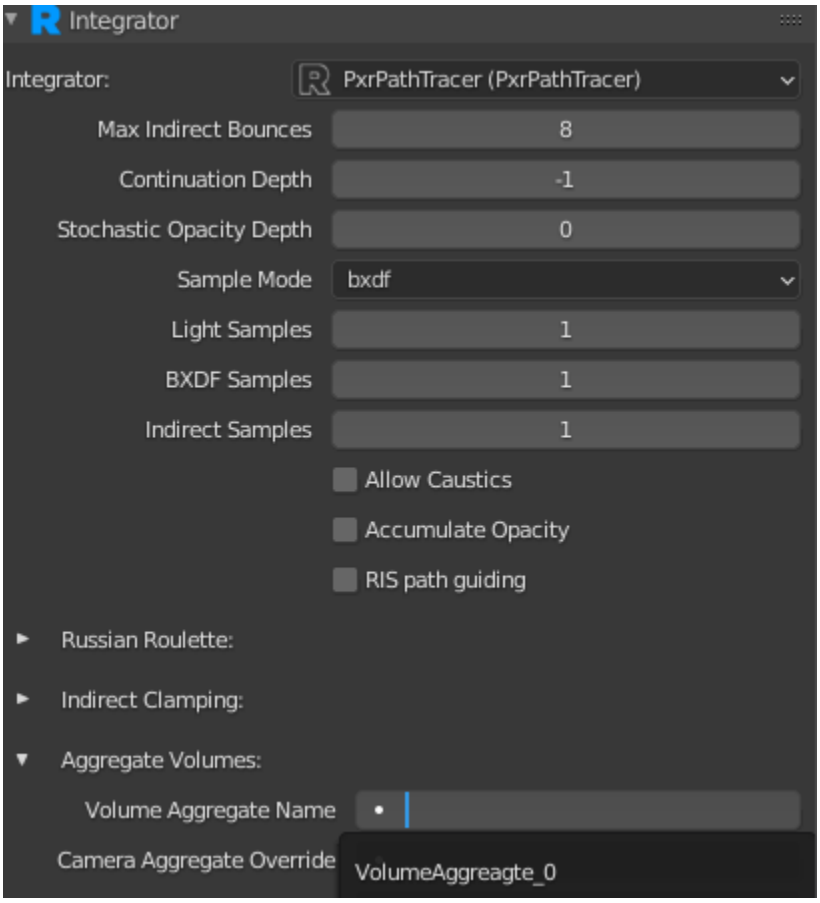


You can also select the volume you want from the viewport and then add it to the group, from the right click menu:



Integrator

To specify the global volume aggregate to the integrator, make sure an integrator node tree has been created (see Integrators in Blender), and then select the volume aggregate name in Volume Aggregate Name drop down list.



If a volume aggregate name has not been specified to the integrator, any volumes that are part of a volume aggregate will not render

Aggregate volume composition of cloudscape vs final render - Luca © Disney/Pixar

Why Aggregate Volumes?

Aggregate volumes are designed for workflows that involve many individual volumes which do not have any boundary behavior. The knowledge that rays will not reflect or refract at the boundary of a volume allows the renderer to perform significant optimizations over the set of many such volumes.

In the Luca cloud scene shown above, several dozen individual cloud elements have been layered together into a unified cloudscape. The aggregate volume workflow allow these cloud elements to be rendered efficiently without having to combine them into a single volume, which may be an expensive process external to the renderer.

In addition, aggregate volumes allow for a named set of volumes that can be used in special circumstances, such as binding a heterogeneous interior to the interior of dielectric surface.

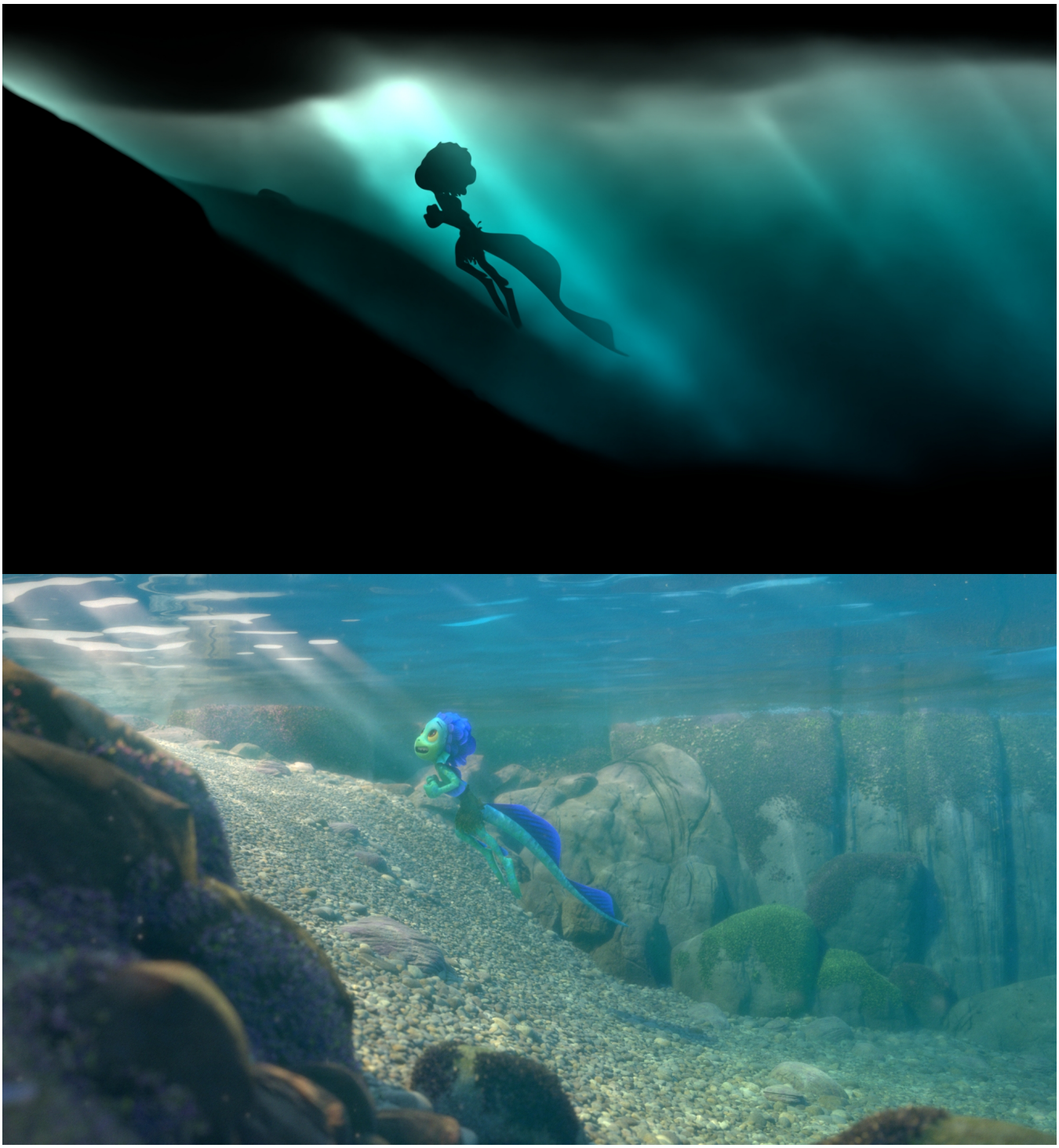
Workflow

The usual workflow for aggregate volumes is to create a named aggregate volume by attaching the `Attribute "volume" "string aggregate" ["name"]` to one or more RiVolumes primitives, where `"name"` is the desired name of the aggregate volume. These volume primitives should use the PxrVolume shader to control their volumetric appearance. Once several volume primitives have been tagged with this attribute with the same name, the named aggregate volume can be supplied as the `"volumeAggregate"` parameter to the PxrPathTracer integrator. This signifies that the *global volume aggregate* for the scene is the named aggregate volume. The named volume aggregate will now appear in renders. Control over how the volume is integrated and sampled is via parameters on the PxrPathTracer integrator.

If the data source for the volumes are OpenVDB files, the density signals comes directly from the file, and is not increased by shading, then it is highly suggested that the new `Attribute "volume" "int dsominmax"` be set to `[1]`. Doing so will allow the renderer to build an acceleration structure from the volume much faster using minimum and maximum values of the density, either by reading prebaked data directly from the VDB file or generating it on the fly as the volume is loaded. This results in much faster time to first pixel as the renderer no longer needs to run density shading in a preprocess upfront before shooting rays. If the volume density is derived procedurally in a shader, then `dsominmax` should be set to 0. An exception exists for cases when a quick density manipulation is desired: `impl_openvdb` supports new controls that allow for global scaling and shaping of the density while still supporting the `dsominmax 1` optimization.

In addition, if the OpenVDB files are heavily detailed and it is anticipated that the files may be overly dense for what is required, the VDB files can optionally be preprocessed with `vdbmake` to generate mipmaps. Doing so *may* result in lower peak memory requirements as only lower levels of the mipmap pyramid will need to be loaded by the renderer. On the other hand, it may often be the case that loading two fine levels of the mipmap pyramid may use more memory than simply loading the base level of the pyramid; the OpenVDB file format in particular can make the advantages of mipmapping less obvious than would be seen in other potential volumetric file formats.

While typically the PxrPathTracer integrator can only render one global volume aggregate, any number of aggregates can be created. These aggregates can be bound as the interior to a closed dielectric surface that uses the PxrSurface material. Unlike regular volumes, aggregate volumes are fully supported as a *heterogeneous* interior for PxrSurface. This is the best way for creating complex effects such as occlusions inside gemstones, or a swirling ball of smoke inside a crystal ball... or Luca swimming in the ocean with crepuscular light rays.



Fog beam layer with aggregate volumes bound to ocean interior, vs final composite - Luca © Disney/Pixar

Capabilities and Advantages

Aggregate volumes are fully supported by the PxrPathTracer and PxrUnified integrators. As noted above, aggregate volumes are designed to be more efficient than the previous volume workflow in situations with tens or even hundreds of overlapping volumes, and impose no limitations on the number of overlapping volumes. Their design also means that aggregate volumes are far less prone to issues that can arise when dealing with coincident geometry (such as coincident volume boundaries, or volumes coincident with surfaces).

Aggregate volumes also offer several advantages that are not available to non-aggregate volumes. In situations where the density of the volume comes directly from a VDB file and is not increased by shading, aggregate volumes will usually offer greatly reduced time to first judgement for complex renders. Aggregate volumes also support *mipmapped* volumes, which can potentially reduce memory consumption for overly detailed volumes. Aggregate volumes may also have faster convergence in complex light transport scenarios, as the renderer is able to make better global decisions about sampling with access to information about more volumes in the scene.

It is anticipated that aggregate volumes will be the primary supported volume workflow in future versions of the XPU renderer.

Limitations

The main limitation of aggregate volumes centers on the fact that a single acceleration structure is built over multiple volumes, and the acceleration structure cannot store many properties of the individual volumes. This is for performance reasons; having to built an acceleration structure that is aware of all those individual properties would slow iteration over those volumes immensely. Properties of a single volume that would affect ray traversal would not work with aggregate volumes unless specially handled, and that special handling would require the renderer to perform extra checks with every step in the volume. As an example of this, grouping the volumes into trace memberships and using ray subsets *do work*, but may not be as efficient as in the normal surface rendering case.

The main individual volume properties that do not work include:

- visibility: aggregate volumes do not respect the standard visibility flags for camera, transmission, and indirect visibility that you might normally expect to set on the individual volumes. To get around this limitation, PxrPathTracer allows the specification of *separate volume aggregates* that are used when dealing with rays of a particular type: these are the `volumeAggregateCamera`, `volumeAggregateTransmission`, and `volumeAggregateIndirect` parameters. These controls are not as flexible as the normal visibility flags - you cannot mix the visibility flags on the individual volumes of the aggregate - you can only operate on either the visibility of all aggregates, or specify which aggregates to use for certain types of rays.
- volume sampling flags: the parameters on PxrVolume that control sampling techniques (e.g. `minSamples, maxSamples, equiangularWeight`) do not apply when using aggregate volumes. Instead, the controls for volume sampling are set in the main PxrPathTracer integrator.

Several limitations are also related to renderer optimizations that can work across many disjoint volumes. Aggregate volumes do not support single scattering; multiple scattering is the only choice, and the `multiScatter` parameter to `PxrVolume` will be ignored. Aggregate volumes are also not optimized for strictly homogeneous volumes. While the rendering algorithms are optimized for volumes that are close to homogeneous, they still assume that the aggregate volume as a whole is heterogeneous even if an individual element is homogeneous and cannot perfectly importance sample the volumes accordingly.

Aggregate volumes only support the use of the `RiVolume` geometry primitive.

High speed transformation motion blur of individual volumes will not be as efficient in an aggregate volume workflow compared to non-aggregate volumes.

While aggregate volumes can be bound as interiors to PxrSurface, aggregate volumes are not recommended for complex nested dielectric scenarios (e.g. tinted murky water inside a tinted glass inside a foggy room).

Aggregate volumes are not currently supported in PxrVCM. They are fully supported in both PxrPathTracer and PxrUnified.

Multi-scattering optimizations are fully supported, but the relevant controls on PxrVolume are ignored on aggregate volumes. Instead, in the aggregate volume workflow, those controls are now on the light sources themselves.

In the initial 24.2 release, matte aggregate volumes may not correctly hold out against non-matte volumes. We anticipate that this will be addressed in an upcoming dot release.

Mipmapping

Mipmapping is a standard technique used to improve texture aliasing and rendering performance by automatically choosing an appropriate texture resolution level from a pyramid based on viewing size and angle of the texture on screen. This technique can *optionally* be applied to volumetric data comprised of voxels: in much the same way that a 2D mipmap pyramid level contains texels that are averages of four finer texels, a 3D mipmap pyramid level contains voxels that are each averaged from eight finer voxels. Mipmapping a volume asset will increase its disk space significantly, and may increase memory during rendering somewhat if the finest voxels are appropriately sized to the level of detail required for the camera; however, if the voxels are much finer in detail than the camera settings require, mipmapping can save significant RAM and render time because the renderer will only load coarser averaged voxels instead. Aggregate volumes using the `impl_openvdb` plugin as a data source for the volume primitives can enable mipmapping by setting the `filterWidth` parameter to a value greater than 0; this parameter is a multiplier on the standard filterwidth computed by the renderer.

`vdbmake` is a new utility shipping with the RenderMan distribution to aid in the creation of these optimized files in OpenVDB format. `vdbmake` takes as input an OpenVDB file with a set of grids, and creates OpenVDB files that have auxiliary mipmapped grids and prebaked acceleration information for those grids.

Multi-scattering approximation

The techniques used to approximate the look of deeply multiscattered volumes rely on altering the density properties of the volume based on the depth of the camera ray. For volume aggregates, the controls for this have been moved to the lights. For more information, please consult rendering clouds with aggregate volumes.
Link