ecrlabs

DENDRO

Voxel plug-in for Grasshopper

Documentation

v0.01.00

TABLE OF CONTENTS

OVERVIEW

Dendro is a volumetric modeling plug-in for Grasshopper built on top of the OpenVDB library. It provides multiple ways to wrap points, curves, and meshes as a volumetric data type, allowing you to then perform various operations on those volumes. Dendro includes components for boolean, smoothing, offsets, and morphing operations.

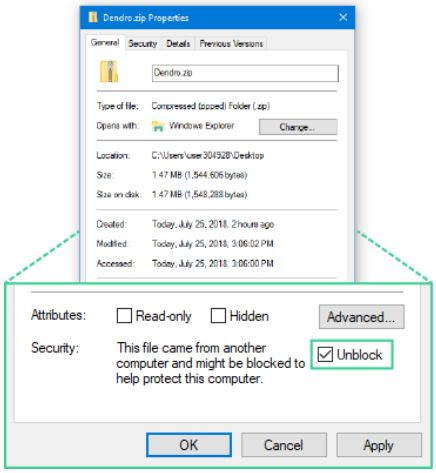
When working with meshes or Breps, these types of operations are often computationally heavy, prone to failures, or cannot handle complex geometry. OpenVDB's volume data structures allow for quicker computation with higher repeatability, enabling you to leverage more complex operations within Grasshopper.

The goal was to make Dendro integrate into Grasshopper as seamlessly as possible. Whereas many voxel solutions require you to think of geometry as living with a bounding box, Dendro makes working with volumes no different than handling any other geometry in Grasshopper. Dendro works with many native Grasshopper components, avoiding the 'blocking' found in other plugins, and allowing you to move in and out of volume operations very quickly.

INSTALLATION

1 DOWNLOAD AND UNBLOCK

Download the plug-in from www.ecrlabs.com/dendro
Right click the ZIP file and select "Properties" from the drop-down menu. Make sure ***"Unblock"*** is not check ed under the General tab.



2 COPY PLUGIN FILES

Copy all files from the ZIP to your Rhino/Grasshopper plug-in directory. This should be:

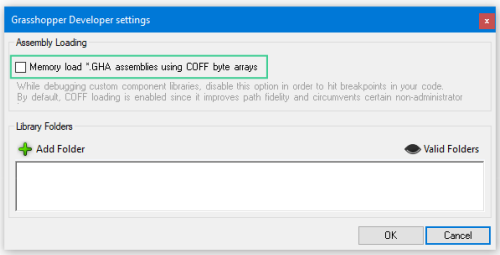
C:\Users\[UserName]\AppData\Roaming\Grasshopper\Libraries

3 DISABLE COFF BYTE ARRAYS

Open Rhino. Type the command:

GrasshopperDeveloperSettings

Make sure ***"Memory load *.GHA assemblies using COFF byte arrays"*** is unchecked.



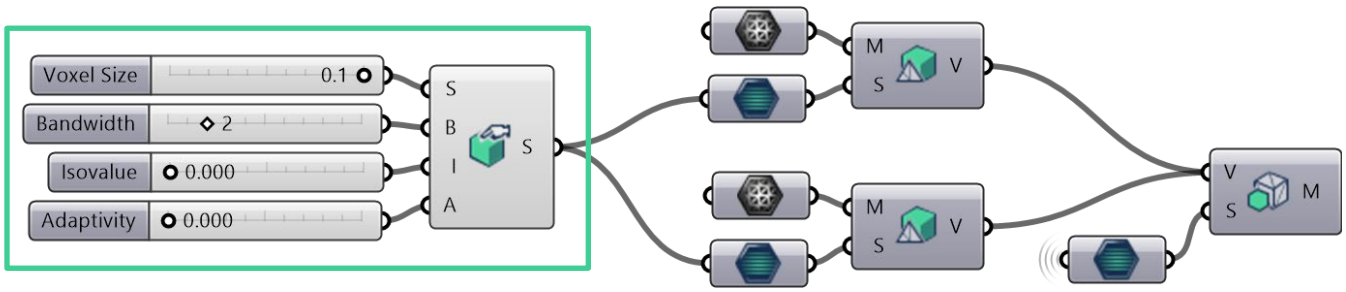
Failure to unblock the zip file or disable COFF byte arrays will result in the plug-in not working.

BASIC USAGE

The typical workflow for the Dendro plug-in is to generate a volume, run your operations on it and then output your volume as a mesh. Volume objects can be created from Meshes, Curves or Points.

GLOBAL VARIABLES

All the geometry converters within Dendro use a global settings parameter. There are fringe use-cases where you may want to use different settings for each converter, but in general, ***you should create one settings component and feed it into all of your components.***



SETTINGS COMPONENT INPUTS

Let's start with the settings component to create your first volume. The settings component has four inputs:

Voxel Size (S)

Used by:
- Mesh to Volume
- Curve to Volume
- Point Cloud to Volume

Description:
Voxel size is the x, y, z dimensions of the individual voxels filling the volume. Think of this as the resolution of the volume.

Suggestions:
Keep this value as large as possible while working and decrease it as a final step.

Bandwidth (B)

Used by:
- Mesh to Volume
- Curve to Volume
- Point Cloud to Volume

Description:
Bandwidth extends the available voxel field around your volume. Voxels within this band are set active, everything else is inactive.

Suggestions:
This controls the active voxel count so keep this value as small as possible in order to minimize computation time.

Isovalue (I)

Used by:
- Volume to Mesh

Description:
Isovalue is the accuracy of the resulting mesh to the original value. it can be abstractly thought of as a positive or negative offset.

Suggestions:
Typically you want to keep this at zero to maintain accuracy to the actual volume.

If you encounter "Invalid Mesh" issues, then setting the Isovalue to 0.002 is often a workaround.

Adaptivity (A)

Used by:
- Volume to Mesh

Description:
Adaptivity sets the uniformity of mesh faces. Values can range from 0-1, with a value 0 being more equalized and dense.

Suggestions:
Higher adaptivities will allow more variation in polygon size, resulting in fewer polygons and quicker calculations.

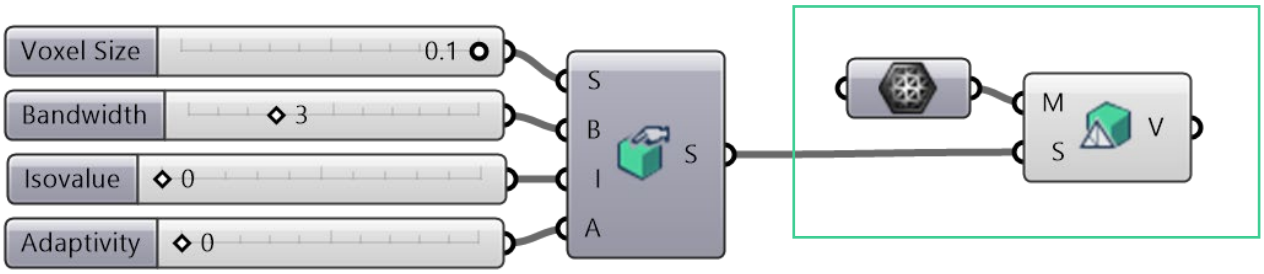
CONVERSION

With an understanding of the basics of volume settings, you can use the Dendro conversion components to switch to and from volume data. For converting geometry into volumes, you can use the ***Mesh to Volume***, ***Curves to Volume*** and ***Point Cloud to Volume*** components. Use the ***Volume to Mesh*** for getting your volume back into a closed mesh.

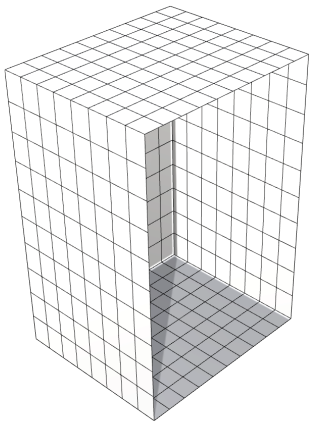
MESH TO VOLUME

The ***Mesh to Volume*** component takes any closed mesh as input and converts it into a volume. For the ***Mesh to Volume*** component to work it must have a Settings input and a "closed" mesh.

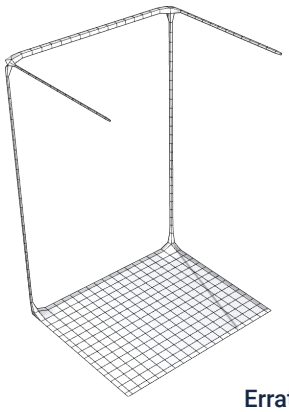
You can generate valid volumes from "open" meshes if your voxel size is larger than any gaps in the mesh. This feature can be useful for mesh repair or for creating a watertight mesh for 3D printing.



Mesh to Volume only works on closed meshes. If a mesh is open you will get erratic results. In the example to the right, the input mesh box (left) is missing an entire face and as a result the output (right) is incorrect.



Open mesh example

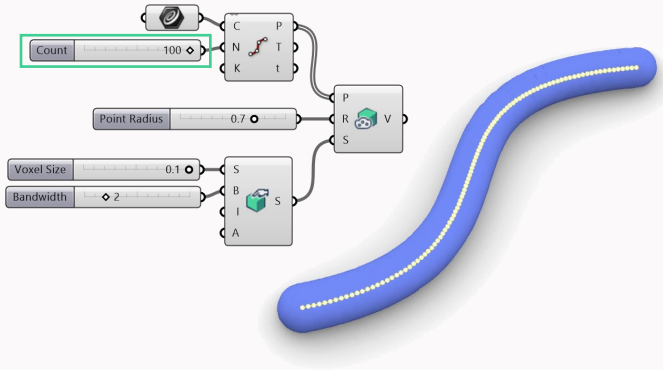


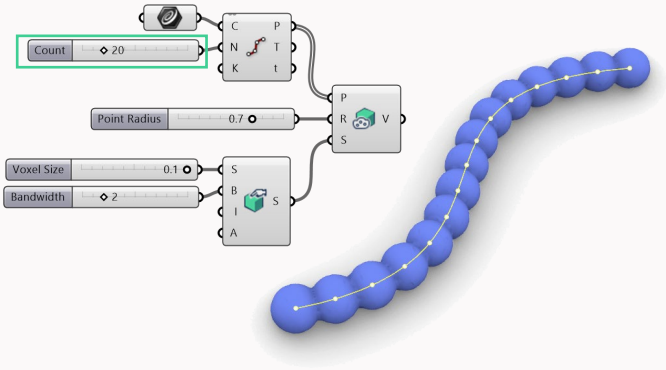
Erratic output from Mesh to Volume

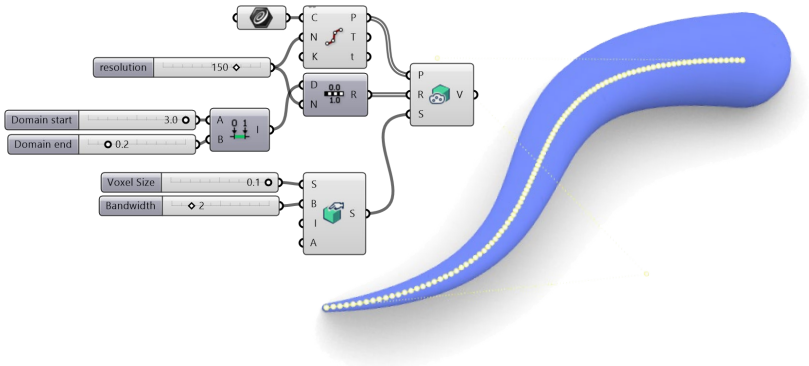
POINT CLOUD TO VOLUME

The ***Point Cloud to Volume*** component generates volumes from point data. Similar to ***Curve to Volume,*** this component takes in a radius and creates a volume sphere at each point. You can supply one radius value for everything or provide a list of radius values for each point.

This component can be used to replicate the features of the ***Curve to Volume*** but with more control. Below you can see what happens as segment increases are applied to a curve with the resulting points used to create a volume.



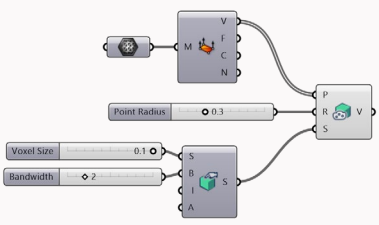


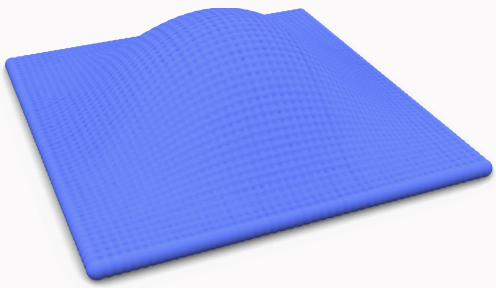


Additionally, the Point Cloud to Volume component can take in a list of radius values, one for each point provided. This allows you to make complex variable volumes from an input curve.
In this example points along a curve are given a radius from 0.2 up to 3.0.

Points can also be used to create a thickened volume from a mesh or surface by supplying points.

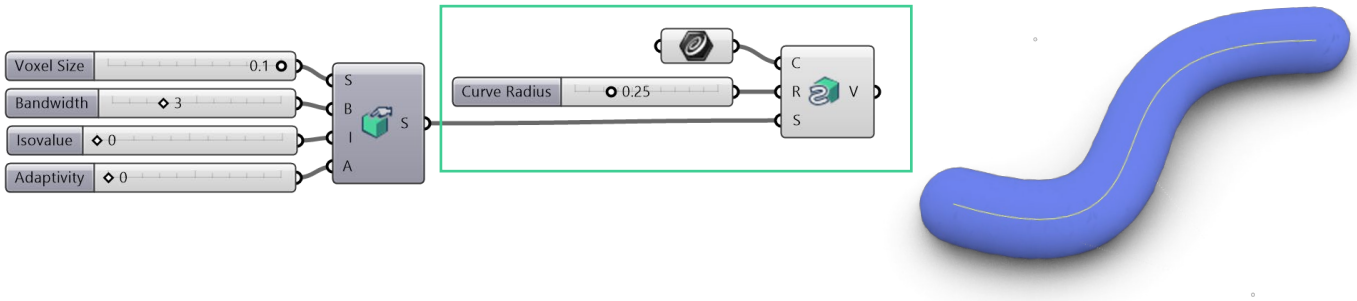
Here all the vertices of a mesh surface are extracted and assigned a radius to create the resulting volume.



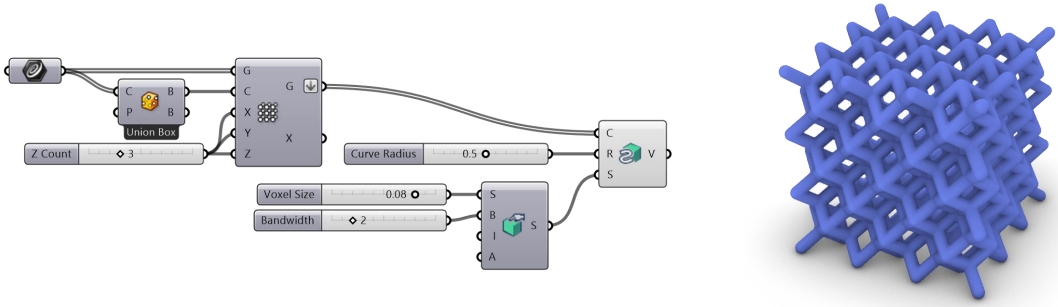


CURVE TO VOLUME

The ***Curve to Volume*** component takes any curve as input and generates a simple volume from it. The curve is wrapped with a spherical profile of a given radius. The radius input defines the size of these spheres based off of your global Rhino units.



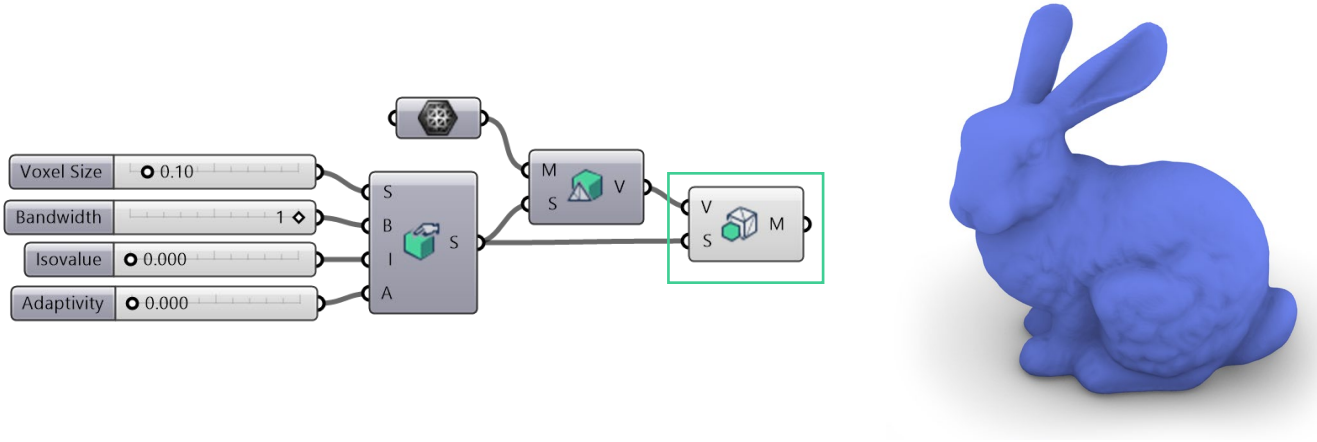
Curves can also be used to effectively create lattice structures from existing line data when used in conjunction with other plug-ins such as Crystallon or Interlattice, potentially with better reliability and performance characteristics than their built in mesh generators.



VOLUME TO MESH

Volume to mesh is the single output component to get volumetric data back into a format that is native to Grasshopper. The Volume to Mesh component will take single or multiple Volume (V) inputs and convert them into mesh output.

The settings driving the mesh generation are the Isovalue and Adaptivity that come out of the Settings component *(see the table on page 2 for more information)*.

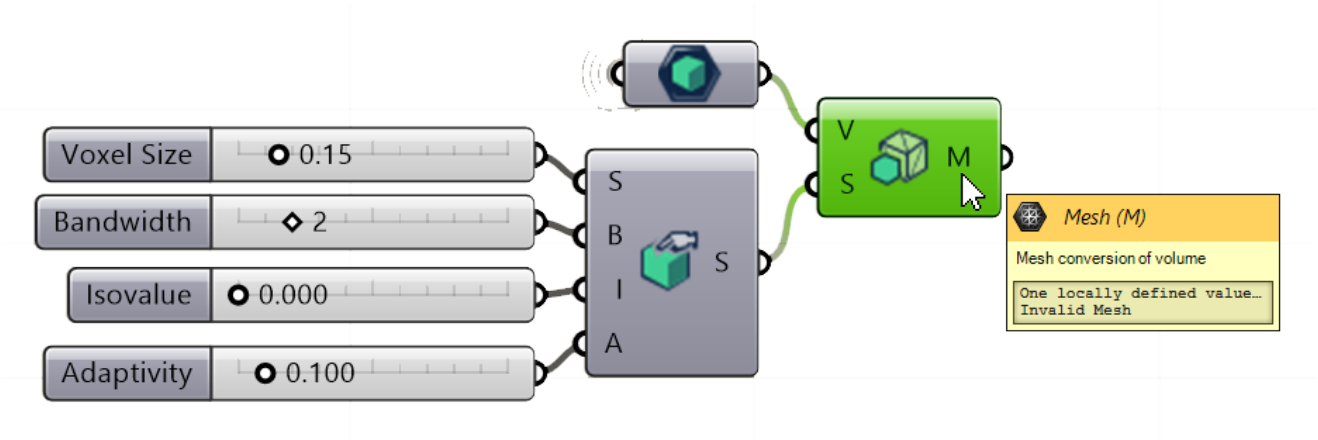


In some instances, converting a volume into a mesh may result in an "Invalid Mesh." Invalid Meshes often appear to output correctly and render from the Grasshopper canvas without issue but fail on subsequent operations and will not bake. As a result, they can be frustrating to spot and diagnose.

The easiest fix is to make a change to your Isovalue and take it from 0 to a small number such as 0.001 or 0.002 which will usually result in a valid mesh output. If there is still a failure to try tweaking your Adaptivity settings.

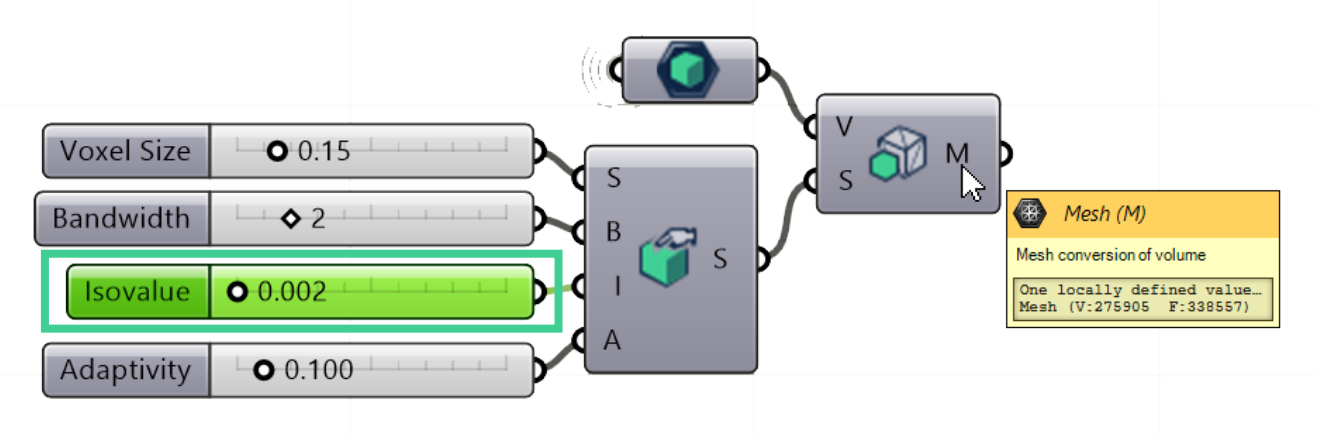
INVALID MESH

In this example the Volume to Mesh component is outputting an invalid mesh. This is not obvious until the mouse is hovered over the "M" of the output triggering the yellow breakout info box.



VALID MESH

The invalid state was resolved by increasing the Isovalue from 0 to 0.002. This is the most common fix for any invalid mesh output.



FILTERS

Filters are a group of Dendro components that modify volume objects. Filters offer some amazing flexibility in modifying volumetric input data. Using other volumes as masks, you can even isolate specfic areas to apply filtering to.

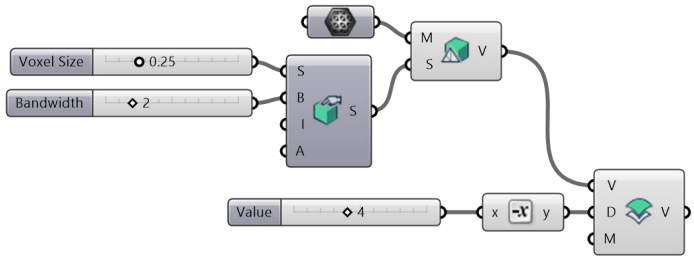
SMOOTH VOLUME

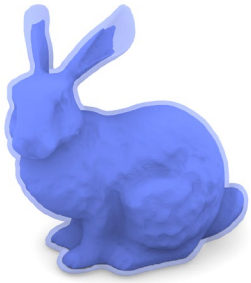
The ***Smooth*** component runs a smoothing operation on an input volume. The Smooth component has the following inputs:

Volume (V)	Type (T)	Width (W)	Iterations (I)
Usage: This is the input volume that the smooth operation will run on. Input can be single or multiple volumes.	**Usage:** There are four smoothing types available: Gaussian (0), Laplacian (1), Mean (2) and Median (3). Values supplied and integers.	**Usage:** Width can be thought of as the scale of the smoothing effect on the input volume. Input must be a positive integer. ***Width input has no effect on Laplacian Type smoothing.***	**Usage:** Iteration is the number of times the smoothing operation runs on the input body.

OFFSET VOLUME

The ***Offset*** component offsets the exterior boundary of a volume. The Distance (D) input specifies the offset amount in your document world units. Offset units may also be positive or negative which is very helpful in shelling an object as well as making molds for casting.





BLEND VOLUME

The ***Blend*** component takes any two volume inputs and morphs between them. The position to evaluate the blend at is controlled by the parameter (t) input (interval of 0-1). The End Time (E) number establishes the upper boundary of the time step and is tied to voxel size.

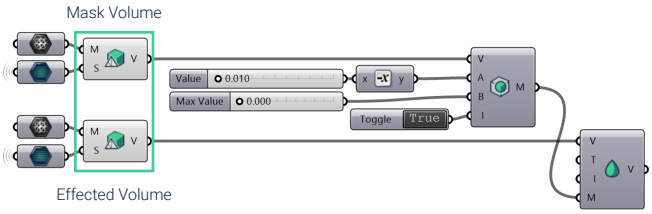
To find the correct End time, attach your volume inputs and then set your parameter (t) to 1. Then increase your End Time until the output looks exactly like the (B) volume input. Now you can adjust the parameter (t) to blend between the two input volumes.

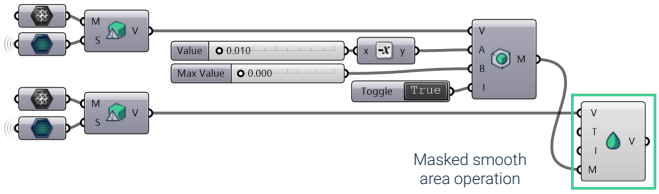
MASK FILTER

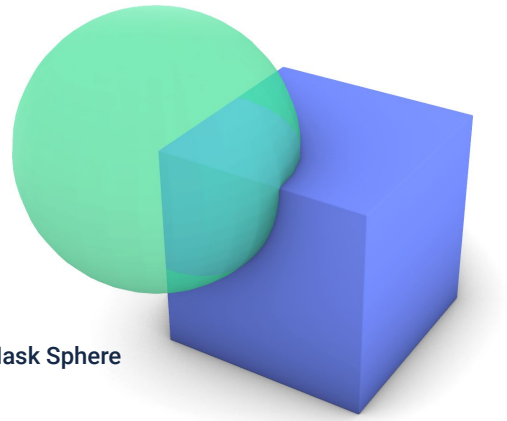
Dendro's filter components (Smooth, Offset and Morph) all have a "Mask (M)" input, which allows you to perform the filter operation in specific areas. Mask data is created by the Create Mask component. The mask is defined by a volume which represents the bounds of the mask.

Masks function similarly to the mask feature found in a 2D image editor, allowing the user to define or limit an area that will be affected by a filter.

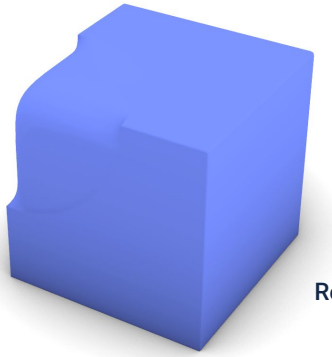
Masks may seem confusing at first but getting comfortable with them allows for critical control of filter effects in the Dendro volume workflow.







Mask Sphere



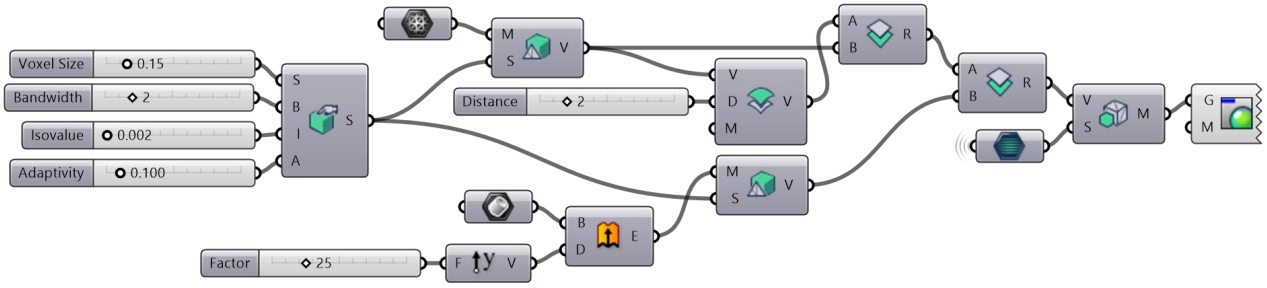
Resulting volume

Volume (V)	Min Value (A)	Max Value (B)	Mask Invert (I)
Usage: This is the input volume that will be your mask area. **Example:** In the above example the green sphere is the mask volume. This volume solely exists to define a filter area and is not seen in the resulting filter operation. A mask must overlap a target volume in order to be of use.	**Usage:** Min Value should be a negative number. This value can not be zero. **Example:** The Min Value defines how the mask effect will transition inside of the masked volume boundary. Start with a low negative number such as -0.001 and work your way up to see effect.	**Usage:** MaxValue should be a positive number. This number can be zero. **Example:** The max value defines how the masked effect will transition outside of the volume boundary. It is easiest to maintain this at 0 unless you are looking for a specific effect.	**Usage:** True or False boolean input that defines which side of the mask volume a filter effect will influence. **Example:** A False boolean will mean a filter will only apply to the target volume on the inside of the mask. A True input will apply the filter to any part of the volume outside of the mask.

BOOLEAN

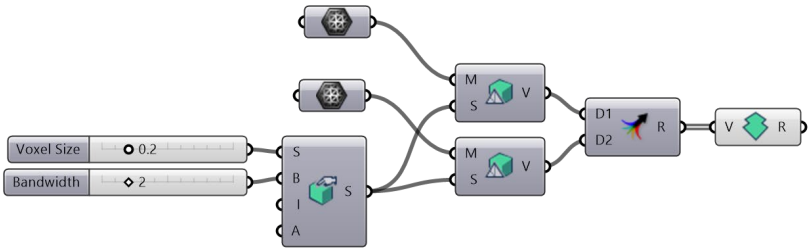
Boolean operations are one of the more compelling features within Dendro. Where mesh (and even NURBS) booleans can be prone to failure, volume booleans are incredibly stable and can be computationally lighter. The standard boolean operations available are Union, Difference, and Intersection.

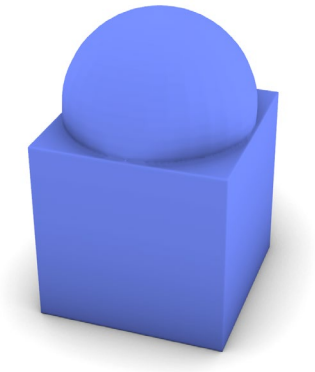
When used with Filters the booleans can be extremely useful at shelling objects and/or making molds.



UNION

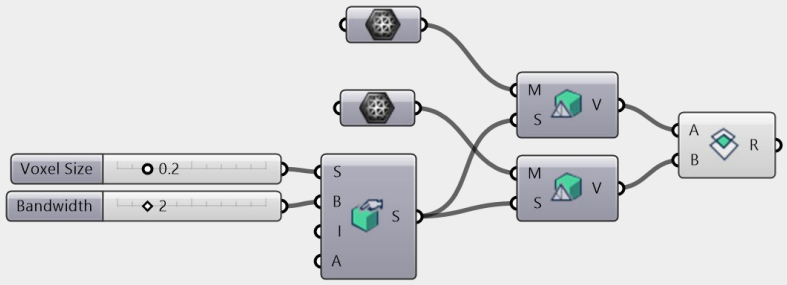
Union combines any overlapping input volumes into a single body. The component must have multiple input volumes on the same branch of a tree in order to run correctly.

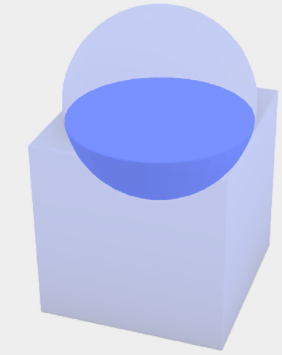




INTERSECT

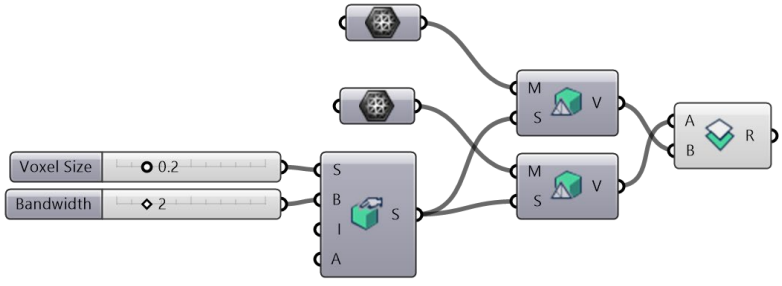
Intersection outputs any overlapping volumetric area from both the A and B inputs. Inputs can be single volumes or lists.

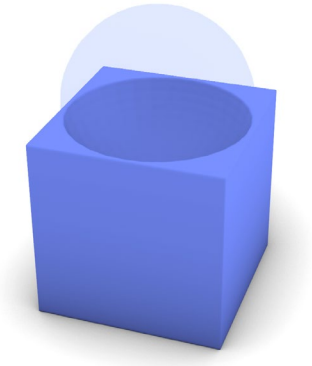




DIFFERENCE

This is the basic subtraction boolean in Dendro. The B input is subtracted from the A input. Inputs can be single volumes or lists.





IO

Dendro has basic read and write capabilities for OpenVDB binary files. This can be utilized to interact with OpenVDB files generated in programs outside of Rhino/Grasshopper such as Houdini or Maya.

WRITE VDB FILE

Saves a .VDB file to the specified File Path (F). File path must be fully specified valid location, file name and end in ".vdb". Boolean set to "True" triggers file output.

READ VDB FILE

Loads a .VDB file from specified location. File must have .vdb extension.

COMPONENT COMPATIBILITY

Dendro works with most Grasshopper components giving it the ability to fully integrate into the standard Grasshopper workflow and logic. This helps to make the learning curve for Dendro voxel usage extremely low.

That being said, voxel systems are not natively supported by either Rhino or Grasshopper so some components can have unexpected and unpredictable results. Below is a compatibility chart for native Grasshopper components that accept Geometry (G) or Content (C) inputs with Dendro Volume (V) outputs.

Functioning Native Components

Sets
All List items working
All Set items working
All Tree items working

Surface
• Primitive / BBox

Transform
• Affine / Scale
• Array / Box Array
• Array / Curve Array
• Array / Linear Array
• Array / Polar Array
• Array / Rectangular Array
• Euclidean / Mirror
• Euclidean / Move
• Euclidean / Move to Plane
• Euclidean / Orient
• Euclidean / Rotate
• Euclidean / Rotate 3D
• Euclidean / Rotate Axis
• Euclidean / Rotation Dir

• Util / Compound
• Util / Split
• Util / Inverse Trans
• Util / Transform
• Util / Group
• Util / UnGroup

Non-Functioning Native Components

Transform
• Affine / Camera Obscura
• Affine / Scale NU
• Affine / Shear
• Affine / Shear Angle
• Affine / Box Mapping
• Affine / Orient Direction
• Affine / Project
• Affine / Project
• Affine / Rect Mapping
• Affine / Tri Mapping
• Array / Kaleidoscope
• Euclidean / Move Away From (partially working)
• Morph / (none of the components in this group are compatible)

Vector
• Point / Project point

SPECIAL THANKS

The MachineHistories team for all their help in bringing this altogether and for their insights into making this a much more thoughtful plug-in.

McNeel for their amazing documentation which helped in integrating these tools more seamlessly into the Grasshopper workflow.

ADDITIONAL INFO

OPENVDB

Dendro is based built on top of the OpenVDB library which is developed and maintained by DreamWorks Animation for use in special effects applications for feature film productions. For more information please visit:
www.openvdb.org

GITHUB REPOSITORY

Dendro is an open source project. The repository is up and available at GitHub:
www.github.com/ecrlabs/dendro

FOOD4RHINO

Questions on using the plug-in? We frequently check our Food4Rhino forum page and will get back to you asap:
https://www.food4rhino.com/app/dendro

CONTACT

Want to talk to us directly? No problem:
dev@ecrlabs.com

The Dendro plug-in is a beta product being actively updated so please check to ensure you have the most up-to-date version at: **www.ecrlabs.com/dendro**

Thanks for the support!
-The ECR Labs Team