Volumes in Katana

Rendering volumes in Katana is similar to rendering other geometry: you'll need a volume primitive and a volume shader. The RenderMan Volume primitive is created in Katana using the PrmanVolume node. For a simple homogeneous volume such as fog you'll just need to assign the PxrVolume shader to the PrmanVolume primitive. More complex shading can be built on top of a PxrPrimvar pattern wired to connect volume data into the volume shader. RenderMan's volume shader PxrVolume along the PxrPrimvar and other patterns, provide a complete volume workflow for either OpenVDB volume data or geometric volume primitives. For more details on RenderMan's volume primitives see the RenderMan Volume Documentation.

Currently PrmanVolume supports two types of volumes:

- **Shaped Geometric Volumes** ('box') - this is most commonly a homogeneous volume used for effects such as fog. The PrmanVolume shader can be attached to any geometric primitive however using the PrmanVolume box primitive allows for effects such as lights in the volume, etc.
- **OpenVDB Implicit Field Plugin** ('vdb') - externally generated volume data which is imported into Katana and RenderMan using the OpenVDB file format.

PrmanVolume Node

The PrmanVolume node contains parameters for configuring the volume primitive as well as transform attributes for positioning and scaling the volume. The exposed parameters in the UI will depend on the type of volume selected:

Box Parameters

Voxel Resolution

Number of voxels in each dimension of the volume.

PrimVars

Dynamic list of primitive variables on the volume. For each prim var you must specify a name and type. If the variable is uniform then check the 'uniform' box and enter a constant value. RfK will then automatically generate the uniform data for the volume and send it to RenderMan. If your data is non-uniform you must generate the data and wire it into the primvar via OpScript or other means.

VDB Parameters

Filename

Full path to the OpenVDB data file.

Density Grid

A VDB grid is a sparse tree representation of voxel data. For instance, a float grid stores one single-precision floating point value per voxel. After the OpenVDB file is loaded, a list of grid names become available in the pulldown menu. Pick the grid name you want to sample. All grids are sent to the renderer as primvars, however only those wired into the PxrVolume shader will be used.

Density Multiplier

Scales the values in the density grid of the volume. This should be more efficient than scaling the density in the PxrVolume shader.

Density Rolloff

Values greater than 0 produce a logarithmic falloff in density values. Smaller rolloff values produce greater falloff.

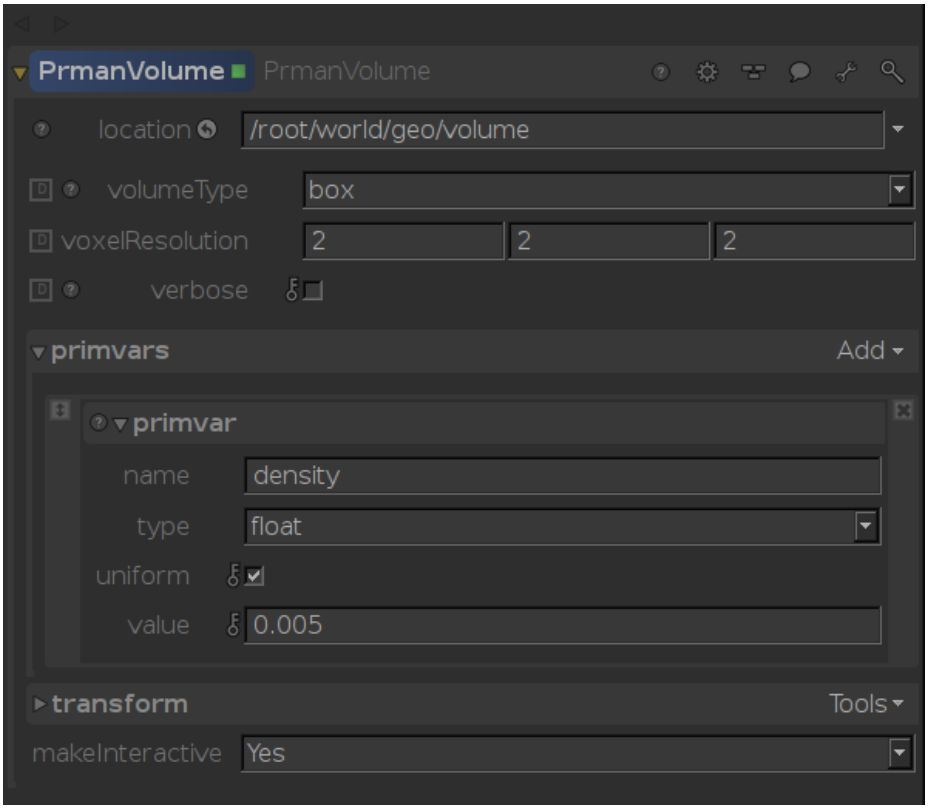
Filter Width

If set to 0, this disables mipmapping. If greater than 0, values less than 1 bias towards finer levels of the mipmap and values larger than 1 bias towards coarser levels.

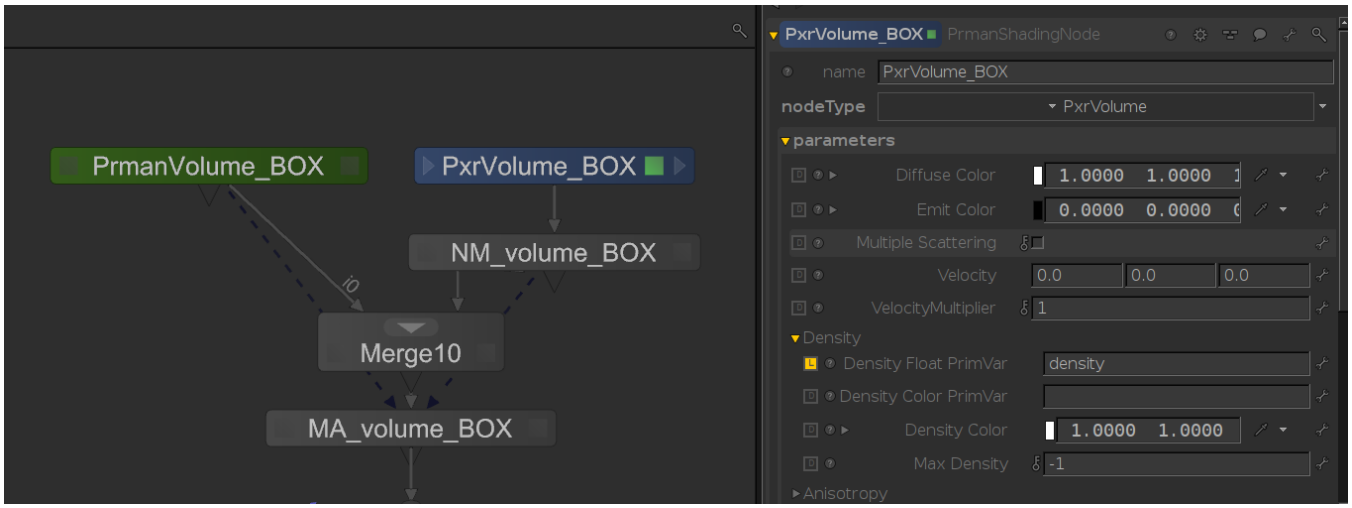
> ⚠ Blobbies are currently not supported by the PrmanVolume node.

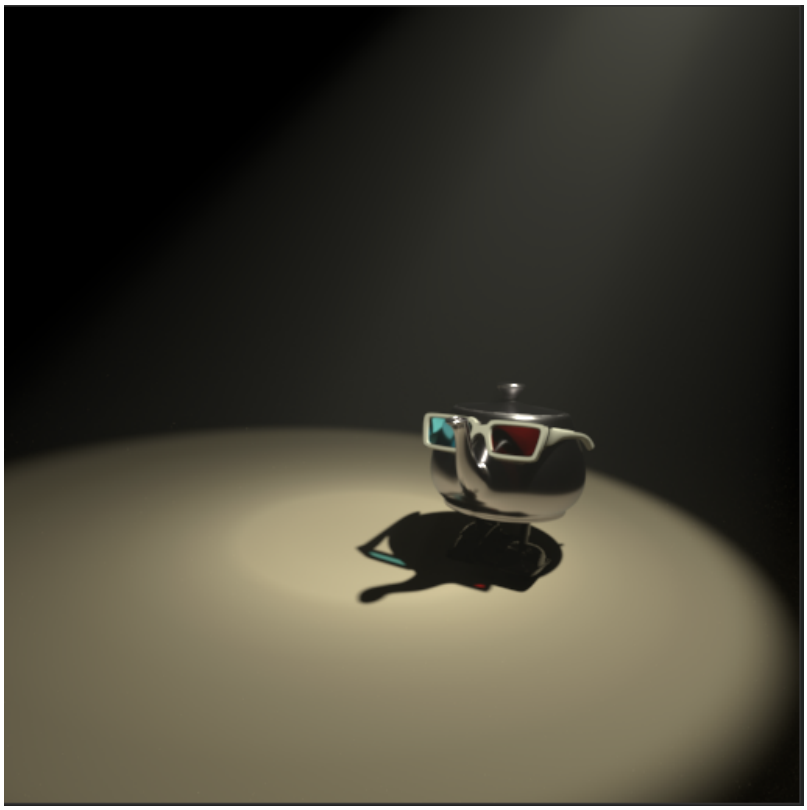
Box Volumes

The PxrVolume shader can be attached to any closed piece of geometry for simple effects within a shaped region, however for effects such as fog that enclose the camera or envelop light sources RenderMan requires that the PxrVolume shader be attached to an RiVolume object. This is accomplished in Katana by selecting the ‘box' volume type in the PrmanVolume node:



The default settings of the node will create a uniform fog effect with a primvar called “density”. Below is the Katana recipe for creating and shading a fog volume:



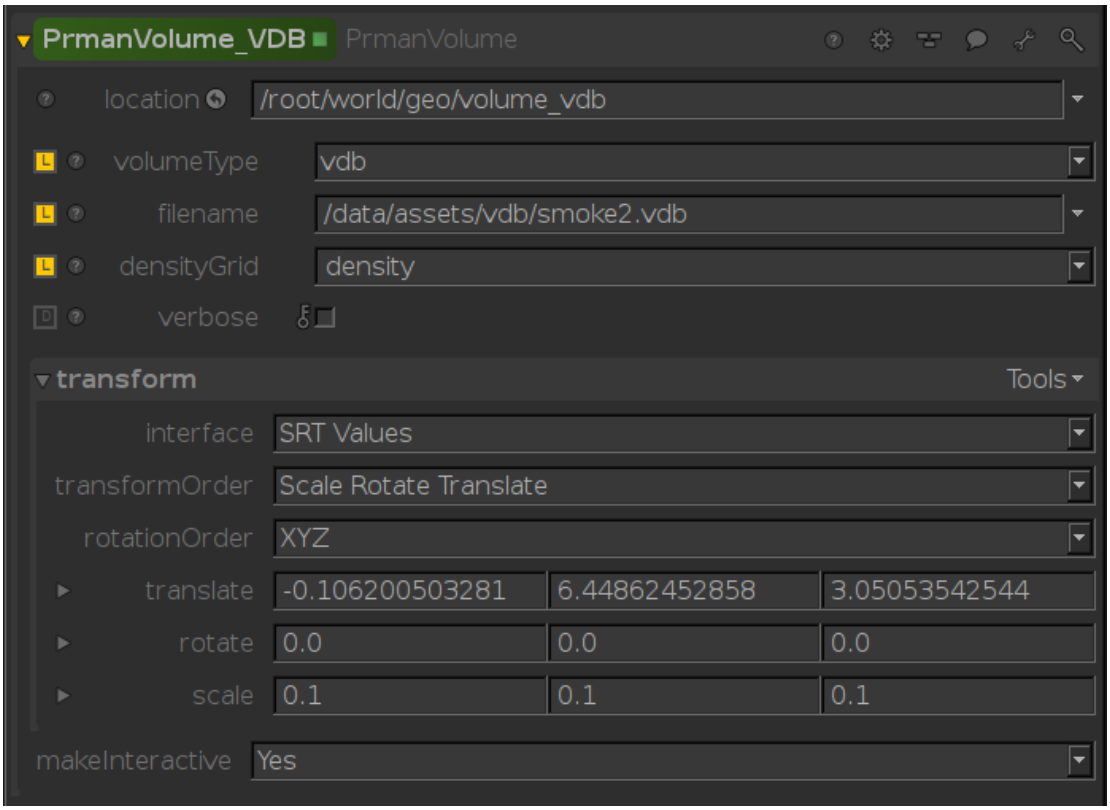


More complex effects can be set up on a Box volume by adding the primvars via the UI and setting up a corresponding PxrPrimvar pattern to be wired into the PxrVolume shader.

OpenVDB Volumes

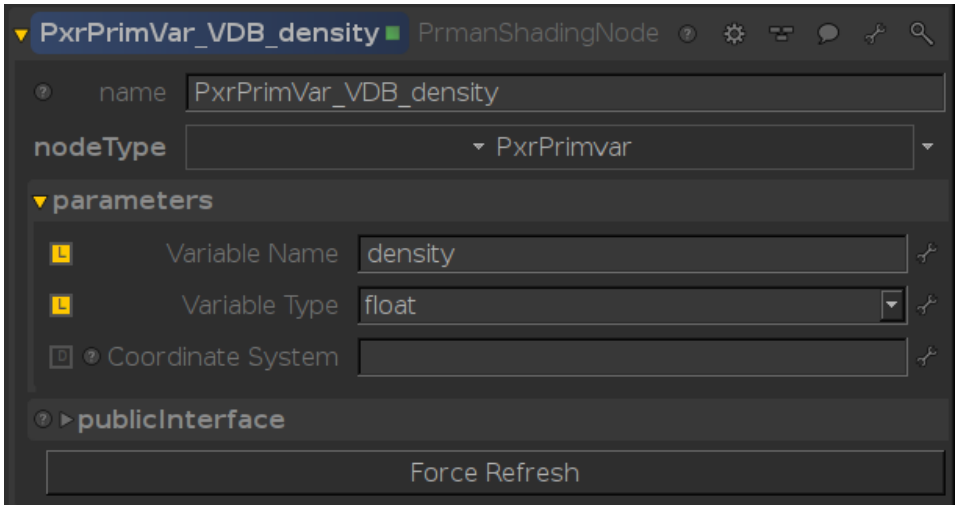
PrmanVolume also supports generated volume data via OpenVDB files. OpenVDB is an open source hierarchical data structure for volumes. It has become the standard for interchange of volumetric data between applications. For more information about OpenVDB, see the OpenVDB FAQ .

With the PrmanVolume type parameter set to ‘vdb’ you’ll have parameters available to enter the OpenVDB filename then select the base density grid:

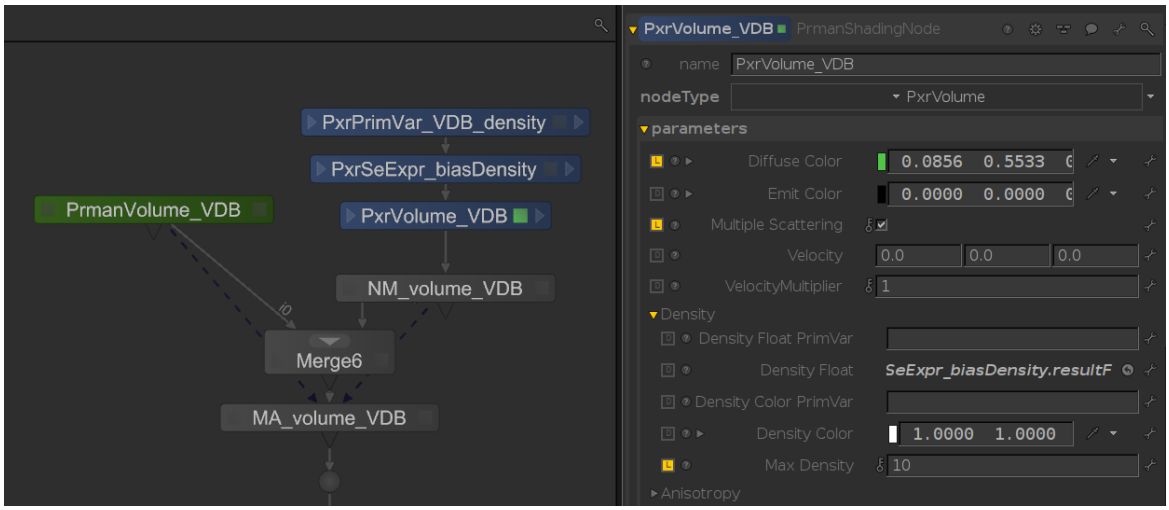


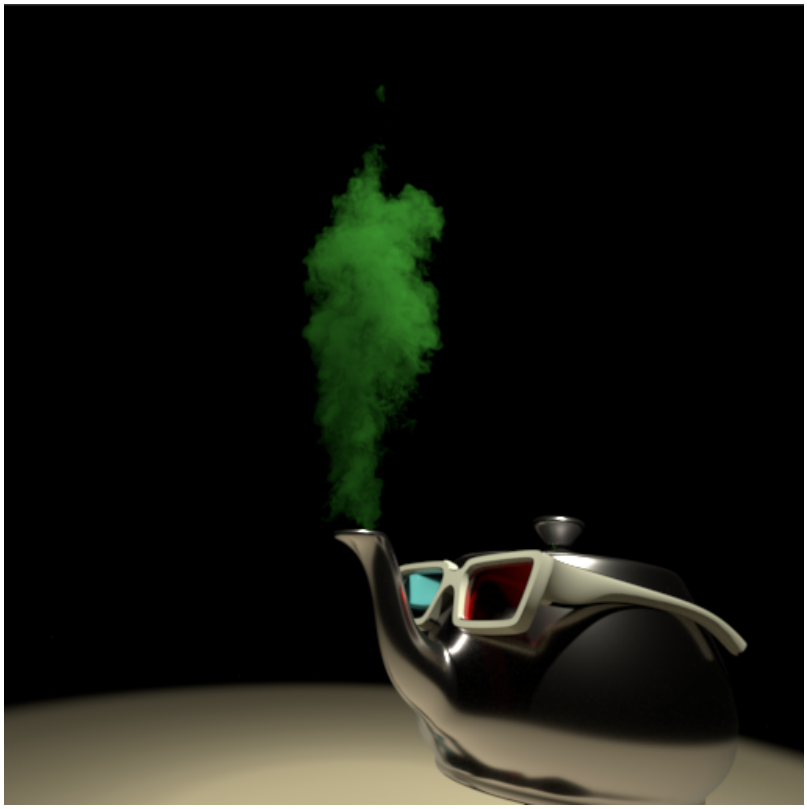
OpenVDB Shading Network

RfK will internally create primvars for all grids found in the VDB file (float or vector3). There is no penalty if they are not used. PxrVolume can look up the density and velocity primvars by name. To access other primvars or to modify the existing density and velocity, primvars can also be accessed by the shading network via the PxrPrimvar pattern node:

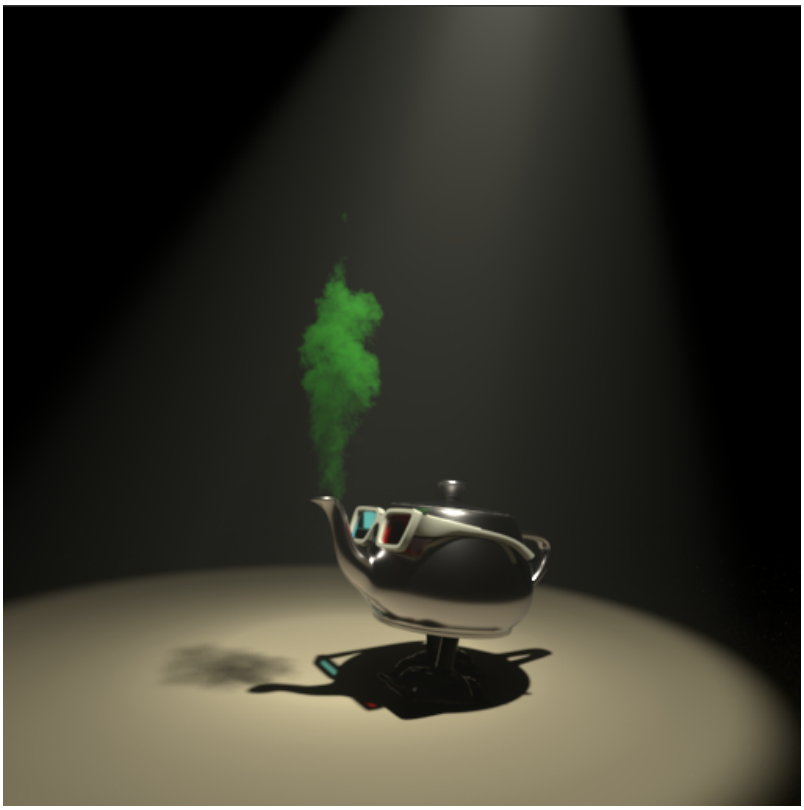


Connect the result of PxrPrimvar directly to the PxrVolume shading node, or wire it through other shading nodes, as in the image below:



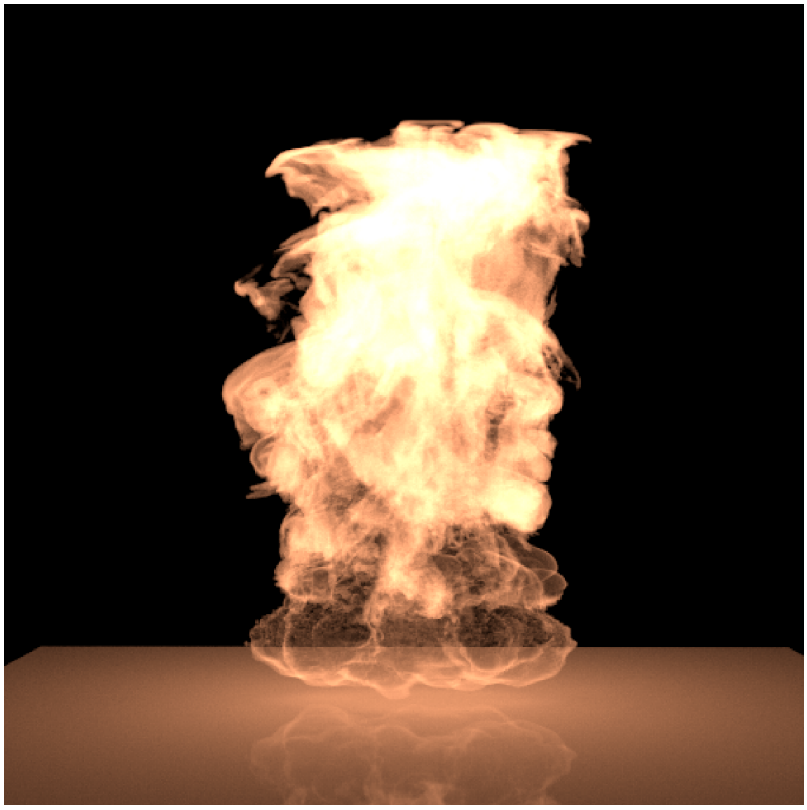


With the two networks wired together we can combine the overlapping volumes in a single render:

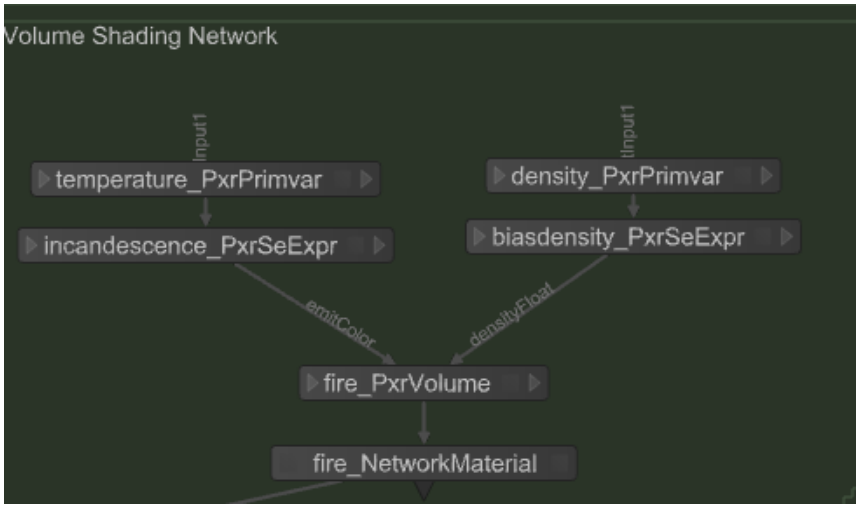


The example scene using the PrmanVolume primitive and PxrVolume shader can be found here.

OpenVDB Fire



An example of using multiple vdb grids as primvar input to the PxrVolume shader is shown below:



The expressions that you use in the PxrSeExpr node are going to be look dependent. When creating the file for the documentation, below is what was used for the incandescence and the density bias.

Incandescence:

```
$temperature = floatInput1;
$incandescence = (1.0 - smoothstep(bias($temperature, 0.8), 0.143, 0.857)) * [20, 7.776, 3.702];
$incandescence
```

Density bias:

```
$density = bias($density,0.815);
$density1 = 0.9 * smoothstep($density, 0.136, 0.15);
$density2= 0.9 * (1 - smoothstep($density, 0.15, 0.857));

if ($density < 0.15) {
    $density = $density1;
} else {
     $density = $density2;
}
$density
```