

---

# Vapor Documentation

*Release 3.2.0.59c3322*

**John Clyne, Scott Pearce, Samuel Li, Stanislaw Jaroszynski**

**Aug 20, 2020**



---

## Contents:

---

<b>1</b>	<b>Downloads</b>	<b>3</b>
1.1	Weekly Build . . . . .	3
1.2	Current Stable Release: Vapor 3.2.0 . . . . .	3
1.3	Installation Instructions . . . . .	4
1.4	Sample Data . . . . .	4
1.5	Previous Releases . . . . .	4
<b>2</b>	<b>Quick Start Guide</b>	<b>7</b>
2.1	Start Vapor3 . . . . .	7
2.2	Import or Load Data . . . . .	7
2.3	Creating a Renderer . . . . .	8
<b>3</b>	<b>Getting Data into Vapor3</b>	<b>11</b>
3.1	Importing data . . . . .	11
3.2	Vapor Data Collection (VDC) . . . . .	11
3.3	WRF-ARW . . . . .	15
3.4	NetCDF and the CF Conventions . . . . .	15
3.5	Raw Binary Data . . . . .	16
<b>4</b>	<b>Using Vapor 3</b>	<b>19</b>
4.1	The Renderers . . . . .	19
4.2	Basic Renderer Controls . . . . .	28
4.3	Navigation Settings . . . . .	31
4.4	Global Settings . . . . .	40
4.5	Ancillary Tools . . . . .	40
<b>5</b>	<b>Contributing to Vapor</b>	<b>45</b>
5.1	Where to start? . . . . .	45
5.2	Bug Reports and Feature Requests . . . . .	46
5.3	Code Contributions . . . . .	46
5.4	Contributing to Vapor's Documentation . . . . .	51
5.5	Contributing to Vapor's Gallery . . . . .	52
<b>6</b>	<b>License and Citation</b>	<b>53</b>
6.1	License . . . . .	53
6.2	Citation . . . . .	54

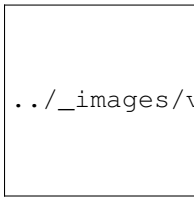


VAPOR is the Visualization and Analysis Platform for Ocean, Atmosphere, and Solar Researchers. VAPOR provides an interactive 3D visualization environment that can also produce animations and still frame images. VAPOR runs on most UNIX and Windows systems equipped with modern 3D graphics cards.

The VAPOR Data Collection (VDC) data model allows users progressively access the fidelity of their data, allowing for the visualization of terascale data sets on commodity hardware. VAPOR can also directly import data formats including WRF, MOM, POP, ROMS, and some GRIB and NetCDF files.

Users can perform ad-hoc analysis with VAPOR's interactive Python interpreter; which allows for the creation, modification, and visualization of new variables based on input model data.

VAPOR is a product of the National Center for Atmospheric Research's Computational and Information Systems Lab. Support for VAPOR is provided by the U.S. National Science Foundation (grants # 03-25934 and 09-06379, ACI-14-40412), and by the Korea Institute of Science and Technology Information



../../\_images/vaporBanner.png



---

**Note:** If you are looking for our legacy version of Vapor 2.6, follow this link to the bottom of the page.

---

## 1.1 Weekly Build

Vapor builds installers on a weekly basis. These have new features that have not been tested, and may be unstable.

[Download here](#)

## 1.2 Current Stable Release: Vapor 3.2.0

February 3, 2020

[Download here](#)

Release notes for VAPOR-3.2.0

New Features:

- Flow Renderer
- Model Renderer
- New Transfer Function Editor
- Off screen rendering
- Performance optimization to Vapor's DataMgr class
- Added support for stretched grids to vdccreate
- Added ability to color Volume Renderings with a secondary variable
- Increased Volume Rendering sampling rate maximum setting

- Updated 3rd party libraries

## 1.3 Installation Instructions

We encourage users of Vapor to install with the methods described here. If you're a developer and would like to contribute, see the *Building From Source* <*buildFromSource*> section below.

### Linux

Run the downloaded .sh script in a terminal window. It will prompt you as to where the binaries will be installed. For example:

```
% sh VAPOR3-3.0.0.beta-RH7.sh
```

### OSX

Double click on the downloaded .dmg file. Once the Finder window pops up, drag the Vapor icon into the Applications folder.

### Windows

Run the downloaded .exe file. A wizard will step you through the installer settings necessary for setup.

## 1.4 Sample Data

Dataset	Model	Grid Resolution	File Size
<a href="#">DUKU</a>	WRF	181 x 166 x 35	734 MB
<a href="#">Kauffman</a>	ROMS	226 x 642 x 43	495 MB

---

**Note:** Users can download a 500 meter resolution image of NASA's [BigBlueMarble](#) for use in Vapor's Image Renderer.

---

## 1.5 Previous Releases

### 1.5.1 Vapor 3.1.0

July 5, 2019

[Download here](#)



## Release notes for VAPOR-3.1.0

### New Features:

- 3D Variable Support
- Direct Volume Renderer
- Isosurfaces
- Slice Renderer
- Wireframe Renderer
- Python variable engine
- Geotiff creation from Vapor renderings
- Support for MPAS-A and MOM6 models

## 1.5.2 Vapor 2

If you are interested in using Vapor 2, it can be [downloaded after filling out a short survey](#).

Vapor 2 is deprecated, and we strongly encourage users to download the currently supported releases of Vapor 3.

[Legacy documentation for Vapor 2 can be found here](#). Please note that this website is no longer supported, and some links may be broken. Use at your own discretion.



# CHAPTER 2

---

## Quick Start Guide

---

Install the version of Vapor3 for your computer's operating system from our downloads page. Updating your system's graphics driver if you're not sure that it's current.

If you don't have data that's supported by Vapor3 yet, you can download a sample to get started with.

### 2.1 Start Vapor3

After following the installation instructions for Vapor3, run the application by doing the following:

**OSX** Vapor3 will be located within your Applications folder. Double click on Vapor3's icon.

**Windows** Unless you chose a different directory during installation, Vapor3 will exist in C:\Program Files\VAPOR\vapor.exe. Double click on the executable.

**Linux** From a BASH shell, navigate to the directory you installed Vapor3 into. Then issue the following command:


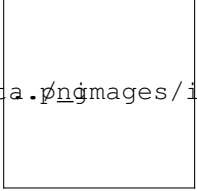
```
user@localhost:/vaporInstallDir$ bin/vapor
```

### 2.2 Import or Load Data

There are two ways to get data into Vapor3. Once you've loaded data, we can create your first *Renderer*.

To load data, do one of the following from the **File** menu:

1. **Load a .vdc file after converting your data into a VDC** File->Open VDC
2. **Import your data, if it's one of the natively supported data types (WRF, NetCDF-CF, MPAS)**  
File->Import->[dataType]

Loading a .vdc file		Importing data
	or	

## 2.3 Creating a Renderer

Now that we've loaded some data to visualize, we can create a new *Renderer*.

Vapor3 displays all of your renderers in a table in the upper left corner of the application. Next to this table are controls that let you create New renderers, Delete renderers, or Duplicate existing renderers.

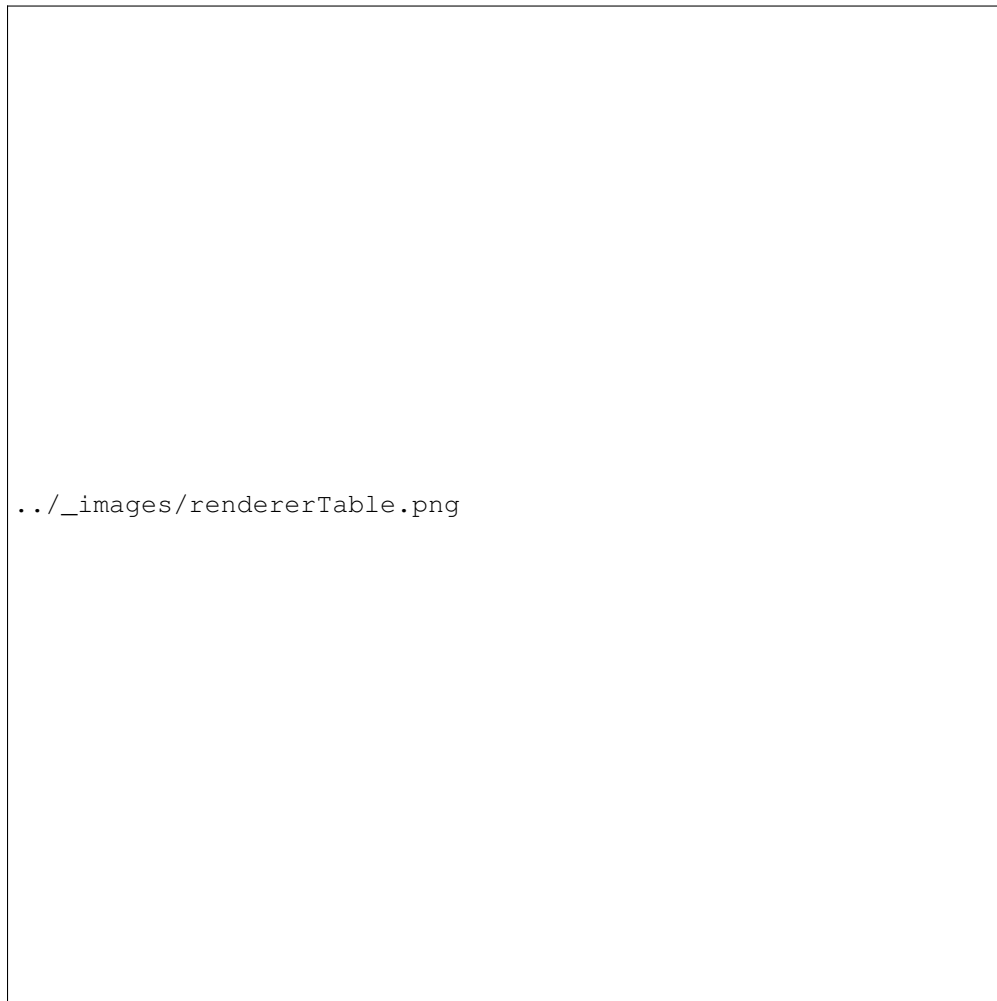


Fig. 1: Vapor3's Renderer Table.

Click on **New**. This will raise a window that will let you choose from the currently available renderers. Pick the **Slice** Renderer by double-clicking on the **Slice** button.



../\_images/newRenderer.png

Fig. 2: Vapor3's New Renderer Dialog

Notice that your new Slice Renderer has been added to the Renderer Table. By default, all renderers are disabled after being created. To enable your Slice renderer, click the `Enabled` checkbox in the Renderer Table that's in the same row as your new Slice.

Now that you have your first Renderer, you can do the following:

1. Change the displayed variable in the *Variables Tab*
2. Change the color mapping of your variable in the *Appearance Tab*
3. Modify the orientation and region that your renderer is drawn to in the *Geometry Tab*
4. Add annotations and color bars in the *Annotations Tab*

At this point, we've created our first renderer. To customize it, we need to get familiar with the four tabs listed above. The *Variables Tab* and *Appearance Tab* are the most important to get started with. We also encourage you to watch active demonstrations in our Video archive.

If you have any questions, bug reports, or feature requests, see our Help section. Thank you.

---

### Getting Data into Vapor3

---

Vapor can directly import the datasets listed below. Some of them can be converted into the VDC format if you are experiencing a performance bottleneck. We strongly recommend that users start by importing their data and only convert VDC if necessary.

- NetCDF files that follow the CF Convention (NetCDF-CF)
- WRF-ARW wrf
- MPAS mpas

Raw binary data must be converted to VDC before loading into Vapor.

### 3.1 Importing data

Users can import their data through Vapor's `File` menu.

### 3.2 Vapor Data Collection (VDC)

In most cases, directly importing data is sufficient for an interactive user experience. However, if rendering times keep the application from being interactive, users may want to consider using the VDC data format.

The VDC data format allows users to render their data at different levels of compression. Viewing compressed data reduces the time a rendering takes to complete, improving interactivity.

With VDC, users can configure their renderers quickly at low fidelity, and then turn off compression for their final renderings. Being able to interact with your data becomes important when rendering takes many seconds, minutes, or even hours to complete.

#### 3.2.1 Reference Documentation



Fig. 1: Importing WRF data into Vapor.



**cf2vdc**

**cfvdccreate**

**ncdf2wasp**

**raw2vdc**

**raw2wasp**

**tiff2geotiff**

**vdc2raw**

**vdccreate**

**vdcdump**

**wasp2ncdf**

**wasp2raw**

**waspcreate**

**wrf2vdc**

**wrfvdccreate**

### 3.2.2 Using the Command Line Tools

Creating a VDC requires command line tools that come bundled with Vapor. These tools can be found and issued in the installation directory. For example, Windows users may find it in C:Program FilesVAPORand Linux users may find it in /home/john\_doe/vapor/bin/wrf2vdc.

Alternatively, the tools can be added to the user's system path by clicking on the *Tools* menu, and selecting *Install Command Line Tools*.After doing this, users will be able to issue the command line utilities from any directory in their terminal or command prompt.



Fig. 2: Installing the command line tools for VDC creation, through Vapor's GUI.

The two step VDC creation process is as follows:

Step 1) Create a .vdc metadata file that describes the structure of your data

Step 2) Transform the data values into VDC format

This process is supported with WRF-ARW, NetCDF-CF, and *raw binary data* <binary>.

Once the conversion is complete, users can load VDC files into Vapor. Read on for instructions for your data type.



Fig. 3: Loading a VDC into Vapor.

### 3.3 WRF-ARW

Vapor supports WRF-ARW model output, so it can be directly imported.

The two tools for converting WRF-ARW into VDC are `wrfvdccreate` and `wrf2vdc`. If either of these commands are issued by themselves, advanced options will be listed to the terminal. These advanced options are usually not necessary.

In the directory where Vapor 3 is installed, there is a command line utility called `wrfvdccreate`. Issue this command in a terminal (Unix) or command prompt (Windows), followed by your WRF-ARW files, and finally the name of the `.vdc` file to be written.

```
wrfvdccreate wrfout_d02_2005-08-29_02 katrina.vdc
```



Fig. 4: Creating a `.vdc` metadata file with `wrfvdccreate`.

Once we have a `.vdc` file, the metadata has been recorded and we can transform the data into the VDC format. From Vapor 3's installation directory, issue the command `wrf2vdc`, followed by your WRF-ARW files, and finally followed by the `.vdc` file that was made in Step 1.

```
wrf2vdce wrfout_d02_2005-08-29_02 katrina.vdc
```



Fig. 5: Applying the VDC transform to WRF-ARW data with `wrf2vdc`

### 3.4 NetCDF and the CF Conventions

If your NetCDF data follows the CF conventions, then the process of converting it to VDC is nearly identical to the WRF conversion process. The commands that will be used are named `cfvdccreate` for `.vdc` metadata creation, and `cf2vdc` for applying the transform.

The [CF Conformance Checker](#) may be used to test if your data adheres to the standard. Passing this checker does not guarantee that Vapor will read the data, but a failer is a strong indicator that Vapor will not be able to read the data.

In order for NetCDF data to be compliant with the CF conventions, the following conditions must be met for the file's vector, scalar, and coordinate variables.

What is a "Coordinate Variable"? From the CF1.X Definition:

*We use this term precisely as it is defined in section 2.3.1 of the NUG. It is a one- dimensional variable with the same name as its dimension [e.g., time(time)], and it is defined as a numeric data type with values that are ordered monotonically. Missing values are not allowed in coordinate variables.*

Each coordinate variable must have an `axis` attribute as follows:

- X coordinate variables must contain an `axis` attribute that is equal to 0 or X.
- Y coordinate variables must contain an `axis` attribute that is equal to 1 or Y.
- Z coordinate variables must contain an `axis` attribute that is equal to 2 or Z.
- Time coordinate variables must contain an `axis` attribute that is equal to 3 or T.

The Time coordinate variable **must** have a `units` attribute which must be identifiable by the Udunits library. Suitable `units` attributes include:

- seconds
- s
- days since 0001-01-01 00:00:00
- seconds since 2011-01-01 00:00:00

Coordinate variables for the X, Y and Z axes may have an attribute that defines the units they are measured in. These units will assist vapor in creating renderings that are accurate between multiple datasets. Some suitable values for the `units` attribute are:

- degree\_east
- meters
- m
- km

*Need to elaborate. We currently support `ocean_s_coordinate_g1` and `ocean_s_coordinate_g2` when the vertical coordinate is dimensionless. What about other cases?*

Variables with missing data values must have the attribute `_FillValue` or `missing_value` specified. See section 2.5.1 of the CF 1.6 specification for more information.

Once your NetCDF files have attributes that make them CF compliant, you may produce VDC by doing the following.

In the directory where Vapor 3 is installed, there is a command line utility called `cfvdccreate`. Issue this command in a terminal (Unix) or command prompt (Windows), followed by your NetCDF-CF files, and finally the name of the `.vdc` file to be written.

Typing the command `cfvdccreate` alone will display the optional arguments that may be used, but these are usually not necessary.

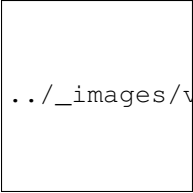
## 3.5 Raw Binary Data

Converting raw binary data to VDC is a complex process for converting data in Vapor 3. With WRF-ARW and NetCDF-CF data, Vapor can read the files and extract metadata that describes the grid that the data exists within. With raw binary data, we need to define that metadata ourselves in step 1.

To make a VDC from scratch, users need to carefully read all options in the `vdccreate` utility, and define their `.vdc` metadata file accordingly.

In this [sample dataset of a sphere](#), we have a 64x64x64 rectilinear grid with one timestep, and one 3D variable. For this example, we can create a `.vdc` metadata file with the following flags from `vdccreate`. Note that we are not using the raw data file yet, just defining the grid, time dimension, and variables.

```
vdccreate -dimension 64x64x64 -numts 1 -vars3d exampleVar sphere64.  
vdc
```

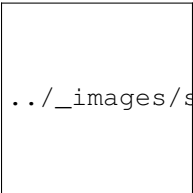


../\_images/vdcreate.png

Fig. 6: Command line arguments for `vdcreate`, seen by issuing the command without any arguments

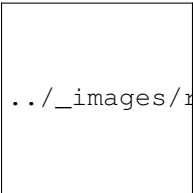
Now that a `.vdc` metadata file has been created, the VDC transform can take place. Each variable must be converted individually with `raw2vdc`, and this must be done one timestep at a time.

```
raw2vdc -ts 0 -varname exampleVar sphere64.vdc sphere64.raw
```



../\_images/sphere.png

Fig. 7: A volume rendering of our sphere, converted from raw binary data.



../\_images/raw2vdc.png

Fig. 8: Command line arguments for `raw2vdc` wavelet transform, seen by issuing the command without any arguments.



## CHAPTER 4

---

### Using Vapor 3

---

Vapor 3 is comprised of a set of tools called Renderers. Each Renderer visualizes your data in different ways based on your specifications.

We recommend that all users start by working through the quickStartGuide, or watching the video tutorials in the guides for our renderers.

If any feature in Vapor is not sufficiently self describing in the application, this is where to find elaboration. Please contact our team if you think you may have found a bug, usability issue, or you'd like to request an enhancement.

## 4.1 The Renderers

Vapor's main utility comes from its suite of Renderers, each of which display simulation data with color and opacity according to user defined parameters.

Renderers can be captured as still images or animations for analysis, publication, and presentation.

Below, you can find descriptions of each Renderer and tutorials on how to use them.

### 4.1.1 Contours

## Description

Displays a series of user defined contours along a two dimensional plane within the user's domain.

## Basic Controls

This renderer contains all of Vapor's standard renderer controls: the *Variables*, *Appearance*, *Geometry*, and *Annotation* tabs.

Like the Image and TwoDData renderers, Contours may be offset by a height variable.

## Specialized Controls

The Contour Renderer has specialized controls in its Appearance tab, under the heading "Contour Properties".

*Spacing* controls the incremental increase in data value between contours

*Count* controls how many contours are currently being drawn

*Contour Minimum* sets the value of the lowest valued contour in the series

## 4.1.2 Two Dimensional Data

### Description

The Two Dimensional Data Renderer displays the user's 2D data variables along the plane described by the source data file.

### Basic Controls

Like the Contour and Image renderers, 2D variables may be offset by a height variable.

This renderer contains all of Vapor's standard renderer controls: the *Variables*, *Appearance*, *Geometry*, and *Annotation* tabs.

## 4.1.3 Image Renderer

### Description

The Image Renderer displays a georeferenced image that is automatically reprojected and fit to the user's data, as long as the data contains georeference metadata.

The Image Renderer may be offset by a height variable to show bathymetry or mountainous terrain.



## Basic Controls

This renderer contains a subset of Vapor's standard renderer controls: the *Variables*, *Appearance*, and *Geometry* tabs. The Image Renderer's Appearance tab does not contain a transfer function since it is not rendering any variable with color or opacity.

## Specialized Controls

Georeferenced images may be selected for rendering in the Appearance tab. Vapor comes bundled with two georeferenced image products: *NaturalEarth*, and *BigBlueMarble*.

If an image contains transparency, this transparency can be toggled with the *Ignore transparency* checkbox. Vapor's BigBlueMarble contains transparency over its ocean regions so that sub-surface ocean models may be viewed alongside land masses.

### 4.1.4 Volume Renderer

#### Description

The Volume Renderer displays the user's 3D data variables within a volume described by the source data file, according to color and opacity settings defined by the user.

#### Basic Controls

This renderer contains all of Vapor's standard renderer controls: the *Variables*, *Appearance*, *Geometry*, and *Annotation* tabs.

The Volume Renderer additionally has specialized controls in its Variables and Appearance tabs.

#### Specialized Controls

The Volume Renderer's Variables tab allows users to color the volume by a secondary variable. This means that the opacity of the volume can be controlled by a primary variable, and the color can be attributed to a secondary variable.

Under the Appearance tab, the Volume Renderer also allows users to modify its Raytracing and Lighting parameters.

#### Raytracing Parameters

*Rendering Algorithm* - Defaults to either Curvilinear or Regular, depending on the topology of the current variable's grid. If using an older graphics card, the Volume Renderer may default to Regular due to the increased complexity of the Curvilinear raytracer.

*Sampling Rate Multiplier* - The volume renderer takes samples along the projected camera rays to determine the volume density at a point and the multiplier increases the number of samples which can give a more accurate visualization in certain cases.

*Volume Density* - A global opacity value applied to all voxels within the Volume Rendering.

## Lighting Parameters

*Apply Lighting* - Controls whether Vapor's default light sources are emitted onto the volume during rendering. The default lights are directional.

*Ambient* - Controls the amount of ambient light that is absorbed by the volume. Ambient lights do not have a single directed source, and controls the overall brightness of the scene.

*Specular* - Controls the angle of reflection for incident light on the volume. In general, if a volume is very specular, it will appear to have more detail and depth. It will appear flatter with smaller Specular parameterization.

*Shininess* - Controls the overall reflection of light incident upon the volume.

## 4.1.5 Isosurfaces

### Description

The Isosurface Renderer displays the user's 3D data variables within a volume described by the source data file, according to color and opacity settings defined by the user.

### Basic Controls

This renderer contains all of Vapor's standard renderer controls: the *Variables*, *Appearance*, *Geometry*, and *Annotation* tabs.

### Specialized Controls

The Isosurface Renderer's Variables tab allows users to color the volume by a secondary variable. This means that the opacity of the volume can be controlled by a primary variable, and the color can be attributed to a secondary variable.

Under the Appearance tab, the Isosurface Renderer also allows users to modify its Raytracing and Lighting parameters.

### Raytracing Parameters

*Rendering Algorithm* - Defaults to either Curvilinear or Regular, depending on the topology of the current variable's grid. If using an older graphics card, the Isosurface Renderer may default to Rectilinear due to the increased complexity of the Curvilinear raytracer.

*Sampling Rate Multiplier* - The volume renderer takes samples along the projected camera rays to determine the volume density at a point and the multiplier increases the number of samples which can give a more accurate visualization in certain cases.

### Lighting Parameters

*Apply Lighting* - Controls whether Vapor's default light sources are emitted onto the volume during rendering. The default lights are directional.

*Ambient* - Controls the amount of ambient light that is absorbed by the volume. Ambient lights do not have a single directed source, and controls the overall brightness of the scene.

*Specular* - Controls the angle of reflection for incident light on the volume. In general, if a volume is very specular, it will appear to have more detail and depth. It will appear flatter with smaller Specular parameterization.

*Shininess* - Controls the overall reflection of light incident upon the volume.

~

## 4.1.6 Flow

### Description

The Flow Renderer creates either *Streamlines* or *Pathlines* within a user's data domain. *Streamlines* are time-invariant trajectories that follow the user's defined field variables. *Pathlines* are time-variant, following trajectories of the user's field variables as time progresses.

### Basic Controls

The Flow Renderer uses Vapor's standard controls located in the [Variables](#), [Appearance](#), [Geometry](#), and [Annotation](#) tabs. In addition, the Flow renderer contains a Seeding tab, described below.

### Specialized Controls

The Flow Renderer Seeding Tab contains four sets of parameters as follows:

### Integration Settings

**Flow type:** Specifies whether the Flow Renderer is advecting *Streamlines* or *Pathlines*.

*Streamlines* draw trajectories along a vector field at a single timestep. These trajectories are time-invariant, and are best used to analyze vector fields at a single time step.

*Pathlines* draw trajectories along a vector field as it changes in time. If you were to throw a feather into a tornado, the path of the feather would be analogous to a Pathline.

### Streamline Settings

**Flow direction:** Determines whether the advection of the *Streamlines* will go forward along the vector field, backwards against it, or bidirectionally.

**Integration steps:** Specifies the number of integration steps to advect the *Streamline* along.

## Pathline Settings

**Pathline length:** Specifies the length of each individual pathline to be shown in the unit of timesteps. If set to 1, the trajectory of a particle travels from the previous timestep (t-1) to the current timestep (t) will be displayed. If set to 2, the trajectory of a particle travels from the past two timesteps (t-2 and t-1) to the current timestep (t) will be displayed.

**Injection Interval:** This controls the frequency at which pathlines are injected into the scene. A value of 0 will only inject pathlines at the initial time step of the data set. A value of 1 will inject at every timestep, 2 will inject at every 2nd timestep, etc.

Both *Streamline* and *Pathline* integration contain the following parameters:

**Vector field multiplier:** Allows the user to multiply their vector field values by a scalar. This is useful in the situation where the data domain and the vector field are in different units. For example, if the data domain is in units of km, and the vector field is in units of m/s, the user may multiply their vector field by 1000 so that all units are congruent with each other.

**Axis Periodicity:** If the X, Y, or Z axes are periodic, flow lines that exit the domain along these axes will be re-inserted on the opposite side of the domain. Flow lines will continue advecting from that point.

## Seed Distribution Settings

**Seed distribution type:** This parameter determines the way that the Flow Renderer places seeds within the scene. The options are:

*Gridded:* Users can specify a grid of seeds that are distributed on the X, Y, and Z axes. The Z axis seed option is removed when rendering 2D flow.

*Random:* Users select a number of seeds that will be randomly placed within the Flow Renderer's Rake region.

*Random w/ bias:* Like a Random distribution, except that users also specify a bias variable that influences the random distribution. If the bias is positive, the seed distribution will prefer regions where there are greater values of the selected variable. If the bias is negative, the distribution will prefer regions where the variable is lesser in value.

*List of seeds:* A user may specify a list of seeds to be read from the renderer. The file must contain lines of comma separated values that represent the locations of the seeds on the X, Y, and Z axes. Users may optionally add a time value after the Z coordinate. Empty lines are ignored, and lines beginning with the # character are comments.

The following example would place a seed at spatial location of (.5, .8, .25) with timestamp .5.:

```
# X, Y, Z, T
.5, .8, .25, 0.5
```

## Rake Region

If the Flow Renderer is using a Gridded, Random, or Random w/ Bias seed distribution, users may constrain the region of seed injection with the Flow Rake. By default, the Rake is as large as the entire domain. If there is a specific region of interest, users should constrain the Rake to only distribute seeds within that region.

The Rake can be adjusted through the left-hand control panel by either moving the sliders corresponding to the desired axis, or typing explicit values. Alternatively, the current Rake can be rendered and manipulated within the scene. This is done by clicking on the Navigation drop-down menu at the top left of the application, and selecting "Region". Users must be in the Seeding tab with the Region mode activated to show the Flow Rake within the scene.

## Write Flowlines to File

Users may write the geometry of the currently rendered flow lines by selecting a text file, and clicking Write to file. The data format of the file is a CSV containing values as follows:

#	ID,	X-position,	Y-position,	Z-position,	Time,	Value+
---	-----	-------------	-------------	-------------	-------	--------

+Value is the value of the currently selected color-mapped variable.

### 4.1.7 Slices

#### Description

The Slice Renderer displays an axis-aligned slice or cutting plane through a 3D variable. Slices are sampled along the plane's axes according to a sampling rate defined by the user.

#### Basic Controls

This renderer contains all of Vapor's standard renderer controls: the *Variables*, *Appearance*, *Geometry*, and *Annotation* tabs.

#### Specialized Controls

The Slice Renderer contains a *Quality* adjustment parameter in the Appearance tab. This renderer operates by sampling data values along a plane. The *Quality* parameter will increase the sampling rate used in generating the Slice.

### 4.1.8 Barbs

#### Description

The Barb Renderer displays an array of arrows with the users domain, with custom dimensions that are defined by the user in the X, Y, and Z axes. The arrows represent a vector whose direction is determined by up to three user-defined variables. Barbs can have a constant color applied to them, or they may be colored according to an additional user-defined variable.

#### Basic Controls

This renderer contains all of Vapor's standard renderer controls: the *Variables*, *Appearance*, *Geometry*, and *Annotation* tabs.

The Barb Renderer can operate on either two-dimensional, or three-dimensional field variables. This is defined in the Variables tab.

The Barb Renderer may be offset by a height variable, if the barbs are referencing two-dimensional variables.

## Specialized Controls

The Barb's Appearance tab contains controls as follows:

*X Dimension* - Controls how many barbs to uniformly distribute along the X axis of the Barb Renderer's region.

*Y Dimension* - Controls how many barbs to uniformly distribute along the Y axis of the Barb Renderer's region.

*Z Dimension* - Controls how many barbs to uniformly distribute along the Z axis of the Barb Renderer's region. This field is only enabled when rendering barbs with 3D variables.

*Length Scale* - A unitless value that controls how long the barbs are. This value is derived by the values of the field variables at the data set's initial timestep, or the timestep at which *Recalculate Scales* is pressed.

*Thickness Scale* - A unitless value that controls how thick the barbs are. This value is derived by the size of the data set's domain.

*Recalculate Scales* - Recalculates the unitless values for *Length Scale* and *Thickness Scale* at the current timestep.

## 4.1.9 Wire Frames

### Description

The Wire Frame Renderer displays a wireframe of the mesh for the selected variable. Wireframes may be rendered for either two-dimensional or three dimensional variables.

### Basic Controls

This renderer contains all of Vapor's standard renderer controls: the *Variables*, Appearance, *Geometry*, and *Annotation* tabs.

## 4.1.10 3D Models

The Model Renderer can display 3D geometry files within the Vapor scene. This is commonly used for rendering models of wind turbines or cityscapes in Large Eddy Simulations.

The model renderer is used to import and display 3D models alongside your data. It supports most common 3D model files (a full list can be found at [github.com/assimp/assimp](https://github.com/assimp/assimp)). It can also display more complex 3D scenes that can be animated with your data timesteps by using .vms files.

### Importing a 3D Model

To import a 3D model file, create a model renderer and use the "Select" button next to the 3D Model/Scene dialog. Enable the renderer to view it. If you cannot see anything, make sure your 3D model is an appropriate size for your dataset. You can move/rotate/scale the model under the "Geometry" tab.

### Importing a 3D Scene

Importing a 3D scene follows the same process as importing a 3D model. See below for documentation on creating a 3D Scene.

## Creating a 3D Scene

3D scenes are stored in .vms files containing an XML description of a 3D scene.

### .vms XML layout

A .vms file must have a top-level node called `scene`. Everything else is a child of this node.

### Instances

Every model displayed in the scene is called an `instance`. The associate tag is called `instance_<name>`. Any time you add an `instance_<name>` tag with a new `<name>`, you create a new instance. When creating a new instance, you need to specify the file from which to load the 3D model data for that instance. This is done with the attribute `file` within the instance tag. You can optionally transform the instance by adding child Transformation nodes. While instances can be created outside of a timestep, they will only be displayed if they are referenced inside a timestep.

### Time

The `time_<timestep #>` tag represents a timestep. The `<timestep #>` is an integer representing the timestep as it appears in the vapor timestep selector. Time nodes can contain instances which will be displayed during the referenced timestep. When rendering, the most recent valid timestep is rendered. For example, if you want to have the same scene displayed for every timestep, create a single `time_0` tag and create your instances inside of it. If you want to stop rendering after a certain timestep, `<N>`, create an empty tag `time_N`.

### Transformations

There are 4 transformation tags: `translate`, `rotate`, `scale`, and `origin`. Each tag results in the same transformation as Vapor's renderer/dataset transform settings. Each transformation tag has three possible attributes: `x`, `y`, and `z`. The rotate `x`, `y`, and `z` values rotate round the corresponding axis by the value given in degrees. If an origin is specified when creating a new instance, that origin will be used for all subsequent transformations of that instance unless otherwise specified.

### .vms File Example

```
<scene>
  <instance_blade file="turbine-blade.stl">
    <origin z="90" />
  </instance_blade>
  <instance_tower file="turbine-tower.stl" />
    <time_0>
      <instance_blade1>
        <translate x="420" y="300" z="0" />
      </instance_blade1>
      <instance_tower>
        <translate x="420" y="300" z="0" />
      </instance_tower>
    </time_0>
    <time_1>
      <instance_blade>
        <translate x="420" y="300" z="0" />
      </instance_blade>
    </time_1>
  </instance_tower>
</scene>
```

(continues on next page)

(continued from previous page)

```
        <rotate x="-72.6" />
    </instance_blade>
    <instance_tower>
        <translate x="420" y="300" z="0" />
    </instance_tower>
</time_1>
</scene>
```

## 4.2 Basic Renderer Controls

Each of Vapor's renderers has a common set of controls that will create imagery of your variables according to color, opacity, and region of interest.

While each renderer is unique, controlling their parameters is mostly the same. All renderers are controlled by four tabs:

- *Variables*
- *Appearance*
- *Geometry*
- *Annotation*

See the [Renderers](#) section for more information on how each of these tabs work for a given renderer. Again, they all operate in the same way for the most part.

### 4.2.1 Variables Tab

The Variables Tab allows the user to define what variables are used as input to a renderer. The options presented to the user in this tab depend on the renderer currently being used.

Users that have converted their data into VDC will have a fidelity controller, which allows them to view compressed data to speed up their rendering time. Making a visualization interactive lets you change parameters faster, so you can crank up the fidelity of your data for a final visualization after exploring first.

### 4.2.2 Appearance Tab

The appearance tab controls the color, opacity, and any renderer-specific parameters of your renderer. Color and opacity are controlled by the Transfer Function. Renderer-specific parameters will be grouped together within the Appearance Tab. See the [Renderers](#) section for more info on renderer-specific parameters.

The Transfer Function consists of a [Probability Density Function \(PDF\)](#) of your currently selected variable. Underneath the PDF is a color bar that shows the colors that get applied to the values located directly above it.

In the figure above, we can see that our transfer function is operating on the variable P. The range of values within the transfer function are -1314.76 to 1268.32. All values of P less than 1314.76 are colored deep blue. The coloration transitions into red at the high end of the PDF, until becoming saturated at values of 1268.32 and higher.

Below the histogram is a button to update the histogram, which is calculated only when the user requests it to save on compute time. Options to change the color interpolation type are also available.





Fig. 1: Variables tab for the Slice renderer

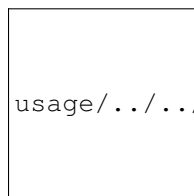


Fig. 2: Vapor 3's Transfer Function editor



Fig. 3: Additional options for the Transfer Function

## Controlling Color

Vapor's default color map is called CoolWarm. This is arbitrary, and may not suit your needs. Vapor bundles several other color maps that can be found by pressing the "Load TF" button at the top of the Appearance tab.

The colors in the color map be moved by creating a color-control-point, and dragging it. To create a new color-control-point, right click on the Colorbar, and then click "New Color Control Point." The color at this control point may now be dragged to suit your needs.

Fig. 4: Adding and moving color control points in the Colorbar

These control points may also be given direct color values by either double clicking them, or right-clicking and selecting "Edit color control point". After a color has been changed, Vapor will interpolate between control points to give a smooth color transition.

## Controlling Opacity

Opacity is controlled by the green line on top of the PDF. The higher this green line is on the PDF's Y axis, the more opaque the colors will be at that point. For example, the green bar is set to  $Y=0$  over the blue values in the image below. All of these values will be masked out. The green bar then ramps up, and the values become more opaque, until we reach full opacity in the red region.

### 4.2.3 Geometry Tab

The Geometry tab controls where your renderer is drawing, within the space of your simulation. By excluding regions of data from being drawn, occluded features may be seen more clearly. Compute time will also be reduced, as well as the memory needed for a given renderer.

If you have a region of interest in another renderer, that region can be copied in the Geometry tab.

Users can apply transforms to scale, translate, and rotate their renderers on X, Y, or Z. The origin used for these transforms may also be adjusted.

Users may also control the geometry of their renderer by using the *Region Mouse Mode*, located at the top left corner of the application. This will enable a red box with handlebars that can be right-clicked to grow or shrink the region being rendered on any axis.

### 4.2.4 Annotation Tab

Quantifying the colors to your viewers can be done by adding a colorbar in the Annotation tab.

## 4.3 Navigation Settings

At the top level of Vapor's control menu, there is a top-level tab called Navigation, which contains settings that help users identify and visualize where they are in the scene. The Navigation tab is composed of an Annotation tab, and a Viewpoint tab.

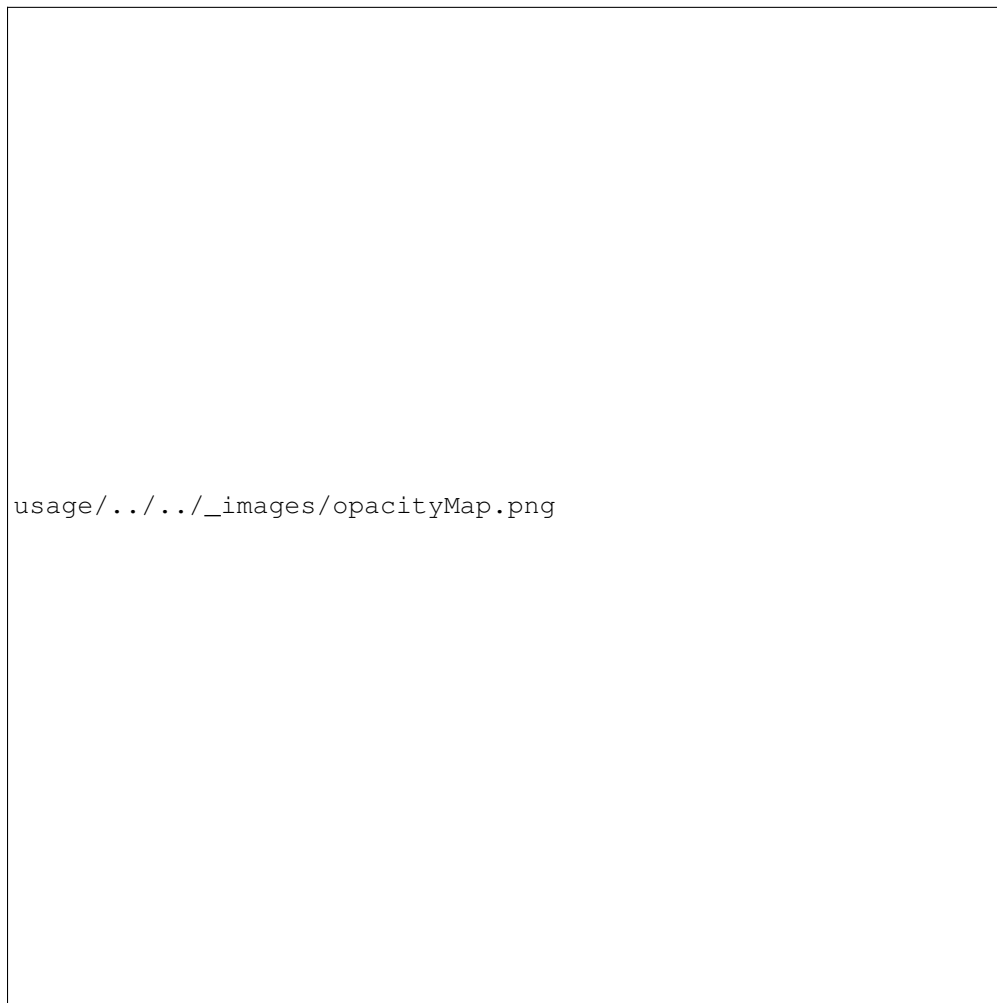


Fig. 5: Blue values are hidden completely. White values ramp up from transparent to opaque, and red values are fully opaque.



usage/../../../../\_images/geometryWidget.png

Fig. 6: Coordinate selector in the Geometry Tab

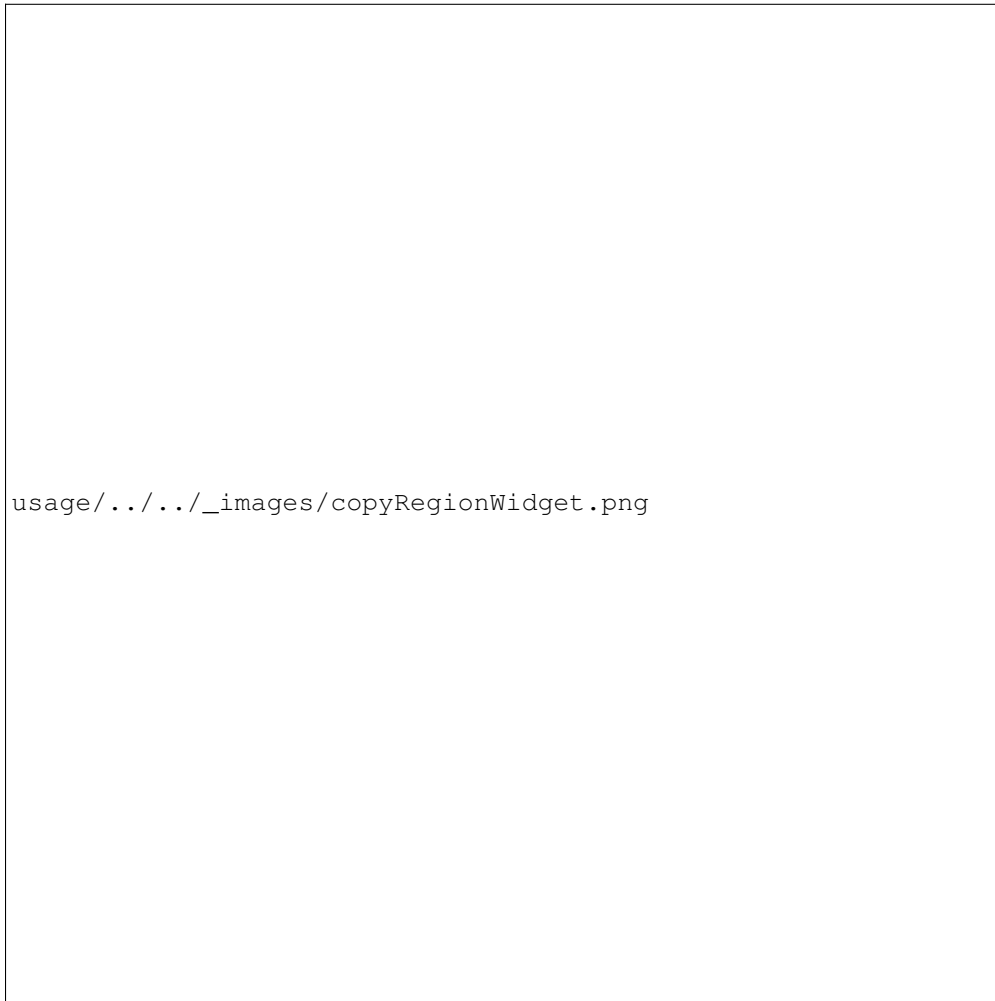


Fig. 7: Copy geometry from one renderer to another



Fig. 8: Transformation options within the Geometry widget



Fig. 9: Select the Region Mouse Mode for interactive geometry adjustment





Fig. 10: Interactive geometry controls alongside a Barb renderer, after activating the Region Mouse Mode



Fig. 11: Colorbar size and position controls, located in the Annotation tab



Fig. 12: An exmaple colorbar

### 4.3.1 Annotations

In the Annotations tab, users can add Axis Annotations, Time Annotations, and 3D arrows that indicate which direction the X, Y, and Z axes are oriented in. Users can also control whether they want to render bounding boxes that indicate the extents of their domain.

### 4.3.2 Animation

Currently in development.

### 4.3.3 Viewpoint

The Viewpoint tab contains tools that let the user apply global transforms to datasets that they have loaded. This is similar to how individual renderers can be transformed, but in this case the transform applies to all renderers in a dataset.

Projection strings can also be modified if a dataset is georeferenced.

Finally, camera position and direction values are displayed here and may be changed numerically for convenience.



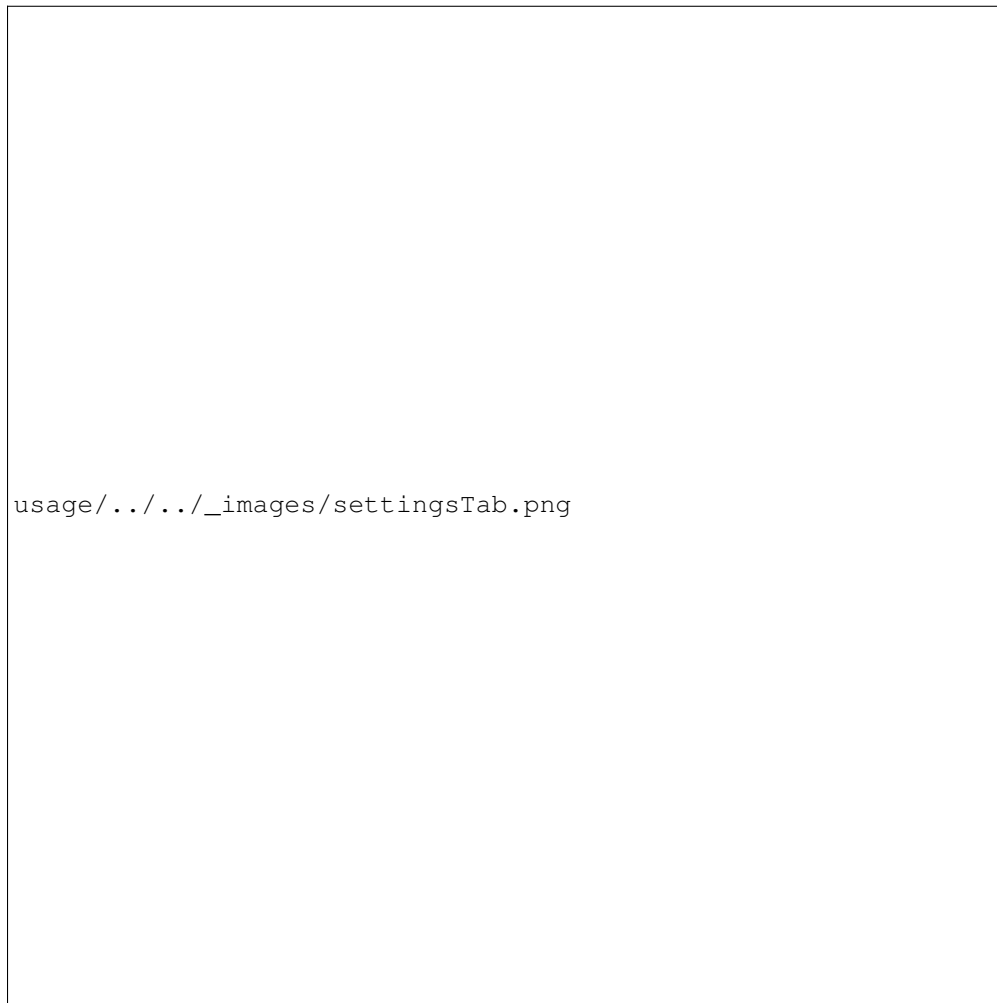
Fig. 13: The Viewpoint Tab, within the top-level Navigation tab

## Off Screen Rendering

Sometimes users will want to produce snapshots or animations at a higher resolution than their current display. A rendering can be captured at 4k, or any user-defined resolution by modifying the **Framebuffer Settings** section of the Viewpoint tab. Simply check the “User Custom Output Size” checkbox, and enter the width and height of the image sequence you would like to capture.

## 4.4 Global Settings

The last top-level tab next to the Renderers and Navigation tabs is called Settings. This is where Vapor’s session file save frequency is set, as well as programatic settings like window sizes and cache sizes.



usage/../../../../\_images/settingsTab.png

Fig. 14: The top-level Settings Tab

## 4.5 Ancillary Tools

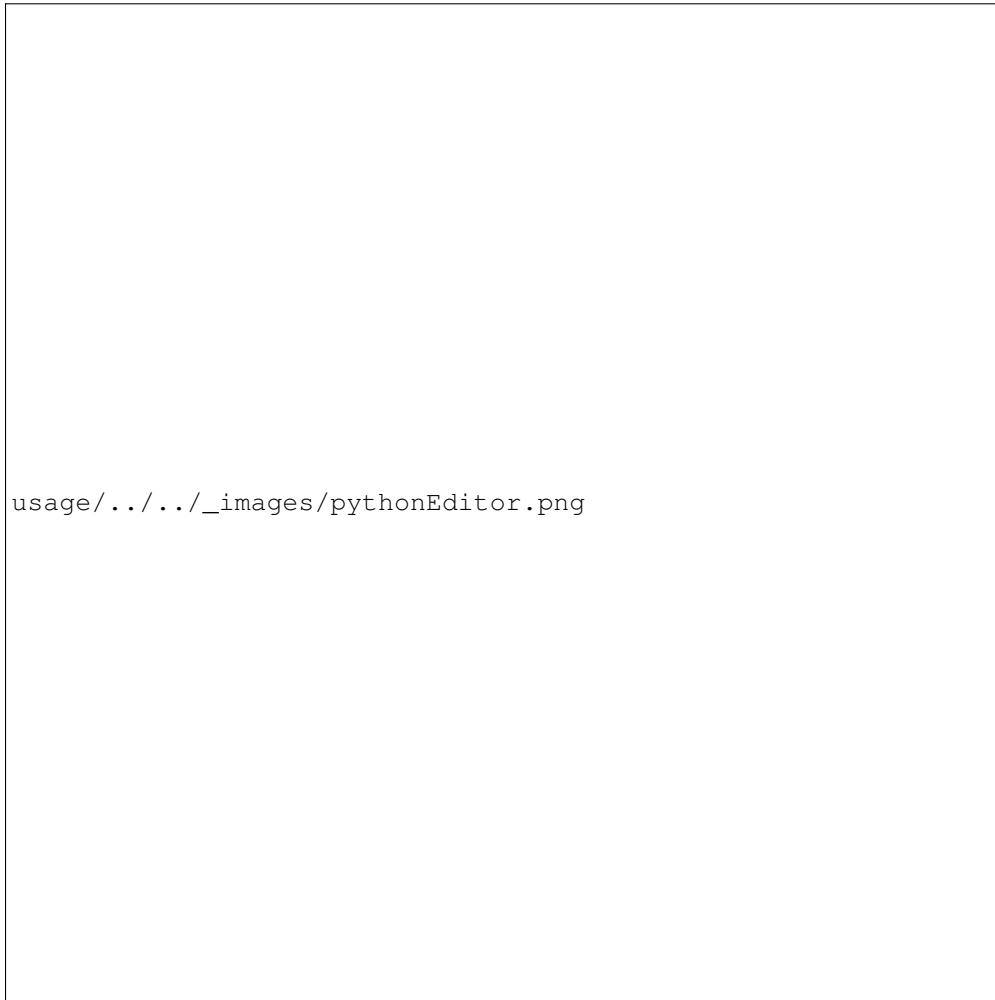
Vapor comes with a Tools menu that provides utilities that can help with visualization and analysis.

### 4.5.1 Python Engine

The Python Engine is a tool that allows users to derive new variables based on the data that exist in their files. Users need to select input variables that will be read in their script, and they will need to define an output variable. If the script successfully run by the Python Engine, the output variable will be usable in the same way as the native variables are in the dataset.

The modules *numpy* and *vapor\_utils* are available for importation in the Python Engine.

Note: Input variables must exist on the same grid to produce a valid output.



### 4.5.2 2D Plots

Users can generate two-dimensional line plots of their variables using the Plot Utility. Line plots can be done either through two points in space at a single timestep, or through a single point across a timespan.

### 4.5.3 Statistics

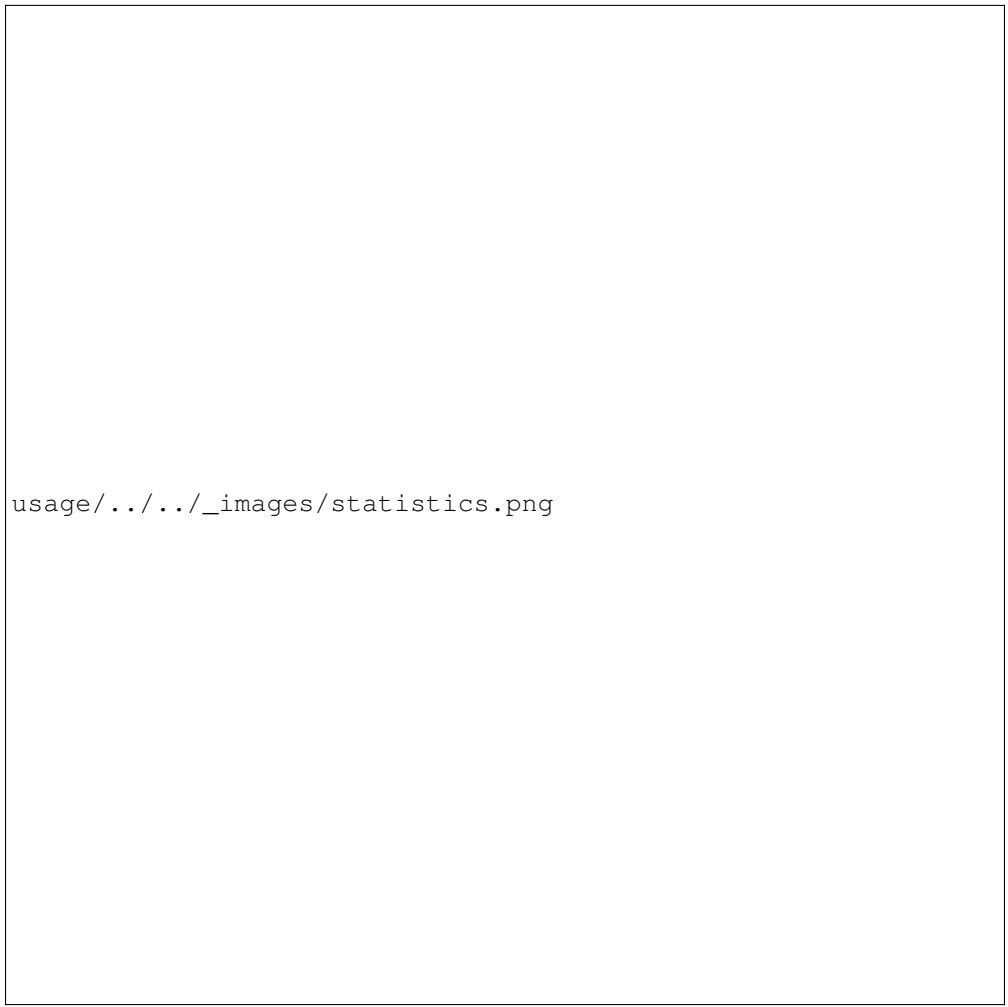
Statistical values can help users select meaningful values for renderer color extents, isosurface values, and contour values. Vapor currently supports calculating the minimum, maximum, mean, median, and mode for variables. The spatial and temporal extents of the variables being queried are adjustable by the user.



Fig. 15: The user interface for hte Plot Utility



Fig. 16: An example of a line plot of Pressure through the spatial domain, at timestep 0



usage/../../../../\_images/statistics.png



---

## Contributing to Vapor

---

### Table of contents:

- *Where to start?*
- *Bug Reports and Feature Requests*
- *Code Contributions*
- *Contributing to Vapor's Documentation*
- *Contributing to Vapor's Gallery*

---

**Note:** Parts of this document are derived from the [xarray Contributing Guide](#).

---

## 5.1 Where to start?

All contributions are welcome, and can include:

- Bug reports
- Feature requests
- Code contributions
- Documentation updates
- Your own visualizations to share in Vapor's gallery

Vapor's current to-do list can be found on our [GitHub "issues" tab](#).

If you've found an interesting issue that you would like to help fix, write a comment stating that you would like to be assigned to it. Assigning issues to one or more individuals helps coordination among developers.

After assignment, you can refer to this document to set up your development environment, and then make your contribution. The purpose of this guide is to help developers understand what part of Vapor’s architecture is relevant to their assigned issue, and how to get their fix incorporated into our *master* branch.

Feel free to ask questions on the [Vapor Discourse Forum](#).

## 5.2 Bug Reports and Feature Requests

To submit a new bug report or feature request, you can click the “New issue” button in the upper right of our [GitHub issue tracker](#), or by [clicking on this link](#).

Bug reports are an important part of making Vapor more stable. A bug report should have a brief description of the problem, as well as a list of steps to reproduce the problem. See [this stackoverflow article](#) for tips on writing a good bug report.

If you’re building Vapor from its source code and find a bug, always try to reproduce your bug on the *master* branch.

It is also worth searching for existing bug reports and pull requests to see if the issue has already been reported and/or fixed. It’s possible that a bug may be fixed, but it has not been merged into *master* or incorporated into a current installer.

## 5.3 Code Contributions

If you’ve found an issue you’d like to fix, you’ll need to compile Vapor, fix the issue at hand, and submit your changes for review and approval. Prerequisite software includes:

- Git for Vapor’s version control
- CMake (version 3.2 or higher) for Vapor’s build system
- One of the following compilers, dependent on which operating system you’re using:
  - OSX - LLVM 10.0.1 (clang-1001.0.46.4)
  - Ubuntu/CentOS - g++ 7.5.0 or higher
  - Windows - Microsoft Visual Studio 2015, version 14

### 5.3.1 Version control, Git, and GitHub

To a new developer, working with Git is one of the more daunting aspects of contributing to *Vapor*. It can very quickly become overwhelming, but sticking to the guidelines below will help keep the process straightforward and mostly trouble free. As always, if you are having difficulties please feel free to [ask for help](#).

Vapor’s code is hosted on [GitHub](#). To contribute you will need to sign up for a [free GitHub account](#). We use [Git](#) for version control to allow many people to work together on the project. Git can also be acquired by package managers like apt-get (Ubuntu), yum (RedHat), and choco (Windows).

The [GitHub help pages](#) are a great place to get started with Git.

[GitHub also has instructions](#) for installing git, setting up your SSH key, and configuring Git. All these steps need to be completed before writing code for Vapor.

### 5.3.2 Forking Vapor's code

After installing Git and registering with GitHub, it's time to "Fork" Vapor's code base by clicking the Fork button on the upper right corner of [Vapor's GitHub repository](#). This creates your own repository on GitHub that contains a copy of Vapor's current master branch.



Fig. 1: Click the "Fork" button in the top-left corner of we website.



Fig. 2: The newly created fork, based off Vapor's master branch. Note the new repository name (sgprowse/VAPOR). This is the repository you will clone from.

Clone your forked repository to a suitable location on your local work machine. This new remote repository is what will be merged with Vapor's master branch once your changes have been made.

After completing your work, your changes can be submitted for review through a Pull Request from your Fork, into Vapor's master repository. This is done under the Pull Requests tab in Vapor's github repository. From this tab, create a new pull request that brings the changes from your forked repo into Vapor's master repo. More details on this step are included in the Submitting Your Changes section of this document.

For more information on the Forking Workflow, please see [Atlassian has a tutorial](#) on basics and best practices.

### 5.3.3 Third-Party Libraries

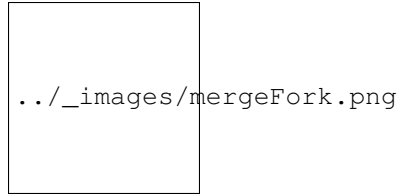


Fig. 3: Submitting a pull request to Vapor’s master branch.

## Download pre-built Third-Party Libraries

This is the recommended approach for acquiring the third-party libraries that Vapor depends on. If you require additional libraries, or custom settings to the libraries currently used by vapor, see the Building Third-Party Libraries section.

After forking your new Vapor repository and cloning from it, you can download pre-built third-party libraries from the links below. Be sure to select the correct libraries for the operating system you’re building on.

---

**Note:** These libraries must be placed in the directory specified above their respective links.

---

### *Windows*

Unzip the following file linked below into the root of your C:\ directory.

[Windows third party libraries](#)

### *Linux and OSX*

If building on Linux or OSX, the third party libraries must be placed in /usr/local/VAPOR-Deps/.

[OSX third-party libraries](#)

[Ubuntu third-party libraries](#)

[CentOS third-party libraries](#)

## Building Third-Party Libraries

This is an alternative to downloading our pre-built libraries that allows you to configure and store them wherever you want. This is a more complex exercise. If you choose to do this, you must also configure Vapor’s CMake configuration to point to your custom directory.

If you wish to go down this route, you may follow the build instructions for [Windows](#) and [UNIX](#).

---

**Note:** In order for Vapor to generate installers, all third-party libraries must be built in the same directory.

---

---

**Note:** The file <vapor-source>/site\_files/site.NCAR may be used in lieu of CMake to specify the location of your party libraries. Edit this file to specify your choice. CMake variables passed over the command line may be overwritten by this file’s presets.

---

<i>Vapor 3 was built with the following third party library configuration.</i>	
Library	Version
assimp	4.1.0
freetype	2.10.1
glew	2.1.0
hdf5	1.10.5
jpeg	9c
libgeotiff	1.5.1
udunits	2.2.26
netCDF	4.7.0
tiff	4.0.10
proj	6.1.1
python	3.6.9
Qt	5.13.2

---

**Note:** Different versions of Qt have presented bugs. It's recommended that developers use the same library versions as those listed above, especially for Qt.

---

The source code for these libraries can be downloaded [here](#).

### 5.3.4 Install System Libraries

Building Vapor from source requires system libraries that are not natively available on all UNIX platforms. The following commands can be used to acquire these libraries.

**Ubuntu:** `sudo apt-get install git freeglut3-dev libexpat1-dev libglib2.0-0 libdbus1-3`

**CentOS:** `sudo yum install dbus make freeglut-devel expat-devel libquadmath-devel libXrender-devel libSM-devel fontconfig-devel`

---

**Note:** Vapor uses docker images as a basis for its continuous integration test suite, which are built from Dockerfiles located in `<source-directory>/share/docker`. The docker files in this directory can be used as a reference to how our different test systems build Vapor.

---

### 5.3.5 Build Vapor with CMake

On all operating systems, create a directory where the build will take place. One option is to put the build directory inside of the Vapor source code directory.

```
> cd VAPOR > mkdir build > cd build
```

**Windows** Enter your build directory as the “Where to build the binaries” field in the CMake GUI. Click *Configure*, *Generate*, and then *Open Project* in that order. Visual Studio will open, and you can build the target *PACKAGE* to compile the source code.

**UNIX:** Navigate to your build directory and run the command `cmake <source_directory>`, where `<source_directory>` is the root directory of Vapor's source code. If the configuration was successful, you can then run `make` to compile Vapor.

```
cmake .. && make
```

If compilation is successful, you can find Vapor’s executable in the *bin* directory within your *build* directory.

Some users may want their build to target a different library than what is distributed with Vapor’s 3rd party library bundle. Different libraries can be targetted in two ways, 1) through the *ccmake* tool, and 2) by editing the file located in `<source-directory>/site_files/site.NCAR`.

Cmake provides an interface to set build variables called *ccmake*. From your build directory, you can issue the *ccmake* command, followed by the path to Vapor’s source code. If your build directory is in `<source_directory>/build`, issuing *ccmake* from this directory would look like this:

```
ccmake ..
```



Fig. 4: *ccmake*’s interface for changing build variables after issuing “*ccmake ..*” on Vapor’s source directory. This is assuming your build directory is in `<vapor_source>/build`.

The above interface allows you to set targets for some (but not all) of Vapor’s libraries. [More information on \*ccmake\* can be found here](#). Your changes will be saved in your build directory in a file named `CMakeLists.txt`. If this file gets deleted, your changes will be lost. To set your libraries in a more permanent fashion, you can edit the `site.NCAR` file, described below.

The `site.NCAR` file is what is used by the Vapor team to define what third party libraries a build should link to. This file is located at `<vapor_source>/site_files/site.NCAR`, and contains conditionals for building on Darwin (OSX), Windows, and Linux (Ubuntu/CentOS). There are also conditionals for building on NCAR’s visualization cluster, Casper.

The `THIRD_PARTY_DIR` variable in this file may be overloaded to re-target the location of Vapor’s libraries. Special values exist for the Qt and Python libraries because they are built outside of the `THIRD_PARTY_DIR`, and must be manually overridden. These variables are:

QTDIR	Directory where the Qt library has been built/installed
Qt5Core_DIR	Directory where the Qt5Core shared library was built/installed
QT_QMAKE_EXECUTABLE	Location of Qt’s QMake executable
PYTHONDIR	Directory containing Python’s libraries and headers
PYTHONPATH	Directory containing Python’s site packages
PYTHONVERSION	Version of Python
NUMPY_INCLUDE_DIR	Directory containing numpy’s header files

### 5.3.6 Adding to the Code Base

After successfully compiling Vapor, you can make changes to the code base. Make sure to follow Vapor’s [Code Conventions](#). If building on a UNIX system, eliminate all compiler warnings.

What pieces of code you add or modify will depend on the issue you’re trying to fix. Most often, contributors will be doing one of two things:

#### Creating a Data Reader

Coming soon: How to create a data reader

## Creating a Renderer

Coming soon: How to create a renderer

### 5.3.7 Build and Test an Installer

Before submitting your changes for review, it's worth the time to build an installer to see if libraries are properly linked, and optimized code works correctly.

To build an installer, run `ccmake <vapor-source-dir>` so that the field `CMAKE_BUILD_TYPE Debug` is changed to `CMAKE_BUILD_TYPE Release`. Also change the field `DIST_INSTALLER OFF` to be `DIST_INSTALLER ON`. Alternatively to `ccmake`, you can hand-edit the file `CMakeLists.txt`, which is located in the root of Vapor's source directory.

On Windows, make sure that the Visual Studio setting for the build is in *Release* mode, not *Debug*, and build the target `PACKAGE`.

On OSX, run `cmake <vapor-source-dir> && make && make installer` from your build directory.

On Linux, run `cmake <vapor-source-dir> && make linuxpreinstall && make installer` from your build directory.

### 5.3.8 Submitting Your Changes

After your implementation is complete, push your commits to your forked repository on GitHub. Then submit your changes you've made on your forked branch to Vapor's master branch as a GitHub Pull Request.

Vapor will automatically run a small set of tests to see if its internals are in tact, and check for warnings. If these tests pass, Vapor's team will review the Pull Request to make sure that Vapor's [Code Conventions](#) were honored, and that the logic and structure of the code is sound.

After review, further changes may be requested. If everything looks good, the Pull Request will be merged into Vapor's master repository.

### 5.3.9 Vapor Coding Conventions

## 5.4 Contributing to Vapor's Documentation

Vapor uses the [Sphinx](#) documentation generator.

To contribute to Vapor's documentation, first follow these [instructions for installing Sphinx](#).

After installing sphinx, you will have a program called `sphinx-build` available. You can find its location by typing the command `which sphinx-build`. This location needs to be applied to the `SPHINXBUILD` variable, at line 6 in `<vapor-install-dir>/docs/sphinx/Makefile`.

Once the makefile has been modified, navigate to `<vapor-install-dir>/docs/sphinx`. Within this directory you will find `.rst` files that Sphinx uses to generate the HTML used for Vapor's documentation. You can modify these `.rst` files, or add new ones.

Once the `.rst` file changes are complete, type `make ..` from the `<vapor-build-dir>/docs/sphinx` directory. This will generate the HTML, which will be saved in `<vapor-install-dir>/docs`. The HTML and `.rst` files can then be committed in Git, and added to a pull request for review.

## 5.5 Contributing to Vapor's Gallery

Contributing to [Vapor's visualiation gallery](#) helps us understand how the application is being used, and can drive future requirements. If you'd like to share your visualizations with us, you can submit them to our gallery by filling out the following form:

[Submit a visualization to Vapor's gallery](#)



## 6.1 License

PLEASE READ THIS SOFTWARE LICENSE AGREEMENT (“LICENSE”) CAREFULLY BEFORE DOWNLOADING THE SOFTWARE. BY DOWNLOADING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, YOU ARE NOT AUTHORIZED TO DOWNLOAD THIS SOFTWARE.

Vapor 3 is released under the permissive Apache v2.0 license.

### 6.1.1 Third Party Software

Vapor 3 makes use of the third party software listed below. In addition to the license above, users agree to the licenses outlined by the following libraries, frameworks, and source files:

Library/Framework	License
GLEW	Custom
IrrXML	Custom
OpenImageDenoise	Apache v2.0
assimp	Custom
embree	Apache v2.0
Freetype	Custom
geotiff	Custom
glfw3	Custom
h5bzip2	Custom
hdf5	Custom
jpeg	Custom
Microsoft Redistributable dll's	Custom
netCDF	Custom
NSIS	CPL v1

Continued on next page

Table 1 – continued from previous page

ospray	Apache v2.0
png	Custom
proj	Custom
python	python
Qt	LGPL v3
sqlite3	Public Domain
SZIP (szlib)	HDF License
TBB (threading)	Apache v2.0
tiff	Custom
udunits	Custom
Source code	
patchelf	Custom
nanoflann.hpp	Custom
SWT (wavelets)	GPL v2.1
Trackball.cpp	LGPL v2
geodesic.h	MIT
geo_ctrans.h	MIT
GetGitRevisionDescription.cmake	Boost V1

## 6.2 Citation

VAPOR is developed as an Open Source application by the National Center for Atmospheric Research, under the sponsorship of the National Science Foundation. Continued support for VAPOR is dependent on demonstrable evidence of the software’s value to the scientific community. You are free to use VAPOR as permitted under the terms and conditions of the license. We kindly request, however, that you cite VAPOR in your publications and presentations. We suggest the following citations as appropriate:

For journal articles, proceedings, etc., we request:

Li, Shaomeng; Jaroszynski, Stanislaw; Pearse, Scott; Orf, Leigh; Clyne, John. 2019. “VAPOR: A Visualization Package Tailored to Analyze Simulation Data in Earth System Science.” *Atmosphere* 10, no. 9: 488.

```
@Article{atmos10090488,
AUTHOR = {Li, Shaomeng and Jaroszynski, Stanislaw and Pearse, Scott and Orf, Leigh
↪and Clyne, John},
TITLE = {VAPOR: A Visualization Package Tailored to Analyze Simulation Data in Earth
↪System Science},
JOURNAL = {Atmosphere},
VOLUME = {10},
YEAR = {2019},
NUMBER = {9},
ARTICLE-NUMBER = {488},
URL = {https://www.mdpi.com/2073-4433/10/9/488},
ISSN = {2073-4433},
DOI = {10.3390/atmos10090488}
}
```

For presentations, posters, etc., we suggest:

Imagery produced by VAPOR (www.vapor.ucar.edu), a product of the Computational
↪Information Systems Laboratory at the National Center for Atmospheric Research

or simply the URL when space does not permit otherwise:

`www.vapor.ucar.edu`



## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`