
CinemaScience Documentation

Release 0.0.1

Terry Turton

Jun 18, 2021

Contents:

1	What is Cinema?	3
1.1	Cinema Citation	4
2	Getting Started	5
2.1	Tutorial	5
3	CinemaScience Specifications	7
3.1	Spec D specification	7
3.2	Cinema Image Set Specification	9
3.3	Deprecated Specifications	9
4	CinemaScience Viewers	11
4.1	Cinema:View	11
4.2	Cinema:Explorer	12
4.3	Cinema:Scope	12
4.4	Other Viewers	13
5	Cinemasci Python Toolkit	15
5.1	Requirements	15
5.2	Installation	15
5.3	Cinemasci submodules	16
6	CinemaScience Algorithms, Libraries, and Tools	17
6.1	cinema_lib Library	17
6.2	cinema_components Library	22
6.3	cinema_movie Tool	26
7	Tutorial: Cinemasci	29
8	Tutorial: Cinema Workflows	33
8.1	Custom Script	33
8.2	Post-Processing via ParaView 5.9 Export Inspector	34
8.3	Post-Processing via ParaView 5.7 Export Inspector	37
8.4	Post-Processing via ParaView 5.6 Cinema Export Scene	39
8.5	In Situ via ParaView Catalyst	40
8.6	Post-Processing via VisIt Cinema Export Wizard	41
8.7	In Situ via VisIt LimSim	43

8.8	In Situ via ALPINE Ascent	44
9	Tutorial: Cinema Viewers	45
9.1	Viewing Cinema Databases	45
9.2	Cinema:View	45
9.3	Cinema:Explorer	48
9.4	Cinema:Scope	52
10	Tutorial: Other Useful information	57
10.1	Converting Spec A to Spec D databases	57
10.2	A Note on Browser Security	57
11	Cinema Publications	59
12	Indices and tables	61

CinemaScience is an ecosystem for large scale data analysis, exploration, and visualization.

What is Cinema?

Cinema is an innovative way of capturing, storing, and exploring extreme scale scientific data – either simulation data or experimental data. It is a highly interactive approach to data analysis and visualization that promotes flexible investigation of large scientific datasets. The Cinema ecosystem consists of database specifications, writers, viewers, and algorithms. Cinema databases consist of data abstracts that are accessed via Cinema viewers in a browser-based approach. While earlier versions of Cinema focussed on images, the current version of Cinema expands the concept of data abstracts to include images, variables, parameters, metadata, mesh files, and csv files.

Cinema enables a multitude of analysis approaches. At the simplest level, a Cinema viewer can be used to visualize a Cinema database, exploring the data through pre-rendered images as if one were using the original 3D representation – but rendering the visualization much faster than possible on a full 3D simulation. Beyond exploring the data, sophisticated analysis algorithms can be applied to the saved raw data variables. The results of that analysis could be a new visualization or it could be a single measurement abstracted from the data at each time step. Computer vision techniques can be used on the raw data to detect, measure, and track features in the data, revealing the scientifically meaningful temporal or spatial evolution of those features. Image processing techniques can be applied to clean up noisy experimental images. Sampling or change detection techniques can be used to identify interesting time steps or spatial regions of a simulation. Cinema can also be used to curate parameters from experimental or simulation runs. Used in conjunction with analysis approaches, the scientist can explore and identify run parameters of particular interest, potentially saving time spent on experimental runs or driving the next round of simulation runs.

Cinema writers are available through common visualization applications. A Cinema database export wizard leveraging the *Cinemasci Python Toolkit* is accessible interactively in *ParaView*. *VisIt* has Cinema export capabilities. Or a Cinema database can be produced through *ParaView*’s Catalyst or *VisIt*’s LibSim in situ libraries. The *Ascent* infrastructure can also be used to export Cinema databases. A Cinema python library, *cinema_lib*, a command line tool is also available. Lastly, application-specific writers to create Cinema databases are also straightforward to implement in Python or other scripting languages.

The bounds of data exploration with Cinema are limited only by one’s imagination. Details of the Cinema database specifications are in *CinemaScience Specifications* and basic viewers are described in *CinemaScience Viewers*.

Cinema is developed by the *Data Science at Scale Team* at *Los Alamos National Laboratory*.

All of our software is open source, and is available at the *CinemaScience GitHub* page. We invite the community to contribute code, documentation, examples, bug fixes, etc. Simply issue a pull request.

1.1 Cinema Citation

If using Cinema in your analysis, please cite the original publication introducing Cinema:

- James Ahrens, Sébastien Jourdain, Patrick O’Leary, John Patchett, David H. Rogers, and Mark Petersen. An image-based approach to extreme scale in situ visualization and analysis. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC ‘14). IEEE Press, Piscataway, NJ, USA, 424-434, 2014. DOI [10.1109/SC.2014.40](https://doi.org/10.1109/SC.2014.40)

CHAPTER 2

Getting Started

Cinema is an application-neutral approach to large data analysis, visualization and exploration. The foundation of Cinema is a database specification that provides a way for common data to be written and read by any application. We provide reference implementations of exporters and viewers, but these are by no means intended to be the only applications within the Cinema community.

A good way to get started is to read the [Supercomputing Paper](#) that first described Cinema’s approach. From there, you can move on to reading the specifications, creating databases with our tools, or including Cinema-compliant components in your own workflows.

Video examples, sample data, and viewer downloads are available on the [CinemaScience website](#).

2.1 Tutorial

The most recent interactive tutorial can be downloaded from the [CinemaScience GitHub](#) page.

- [2020 Supercomputing Tutorial](#)

CinemaScience Specifications

A Cinema database is a collection of data extracts that can be accessed for visualization or interactive data exploration via a viewer application.

- **Spec D** : With Cinema release v1.3 and later, Cinema converged on a specification, *Spec D specification*, that incorporates the flexibility and functionality of previous releases while expanding the Cinema data specifications. Official [Release Notes](#) are available on the [CinemaScience GitHub](#) page. This specification includes multiple files per database entry and arbitrary data. It has a suite of flexible viewers and view components for a wide range of analysis approaches.
- **CIS** : A second specification is the new Composable Image Set specification, *Cinema Image Set Specification*, that supports advanced and flexible post hoc colormapping.

3.1 Spec D specification

The early Cinema specifications focussed on an image-based approach. The philosophy of Spec D embraces a wide range of data abstracts to leverage and promote data analysis in addition to visualization. This is a brief overview of the Spec D specifications and we encourage the reader to consult the full document: [Dietrich \(Spec D\)](#).

Spec D is based on a Comma Separated Values (CSV) database file. A Cinema database is a directory with a name of the form *database_name.cdb* that contains a *data.csv* file:

The *database_name.cdb* directory may optionally contain other data files references by *data.csv* or may contain optional directories. An *image/* directory is a common subdirectory to contain rendered images.

```
$ ls -l sphere.cdb
... image           # a directory for images
... data.csv
```

The first row of the *data.csv* file is a required row of unique non-empty strings that are the column headers for each column. Each column is a data abstract – a variable, parameter, (optional) image file, or (optional) data file such as a vti file, or a csv file. All data values must be floats, integers, strings, or empty (missing) values and must have the same values for all rows of that column (except when empty).

The last header values may either be `FILE` or start with `FILE` (e.g., `FILE,FILE1,FILE2`). No other non-`FILE` columns may be placed after the first `FILE` column (`FILE` columns are always last).

Each row after the header is a data row. At least one data row must exist. The first non-empty value in each column establishes the data type. The database can be extended simply by appending the `data.csv` file.

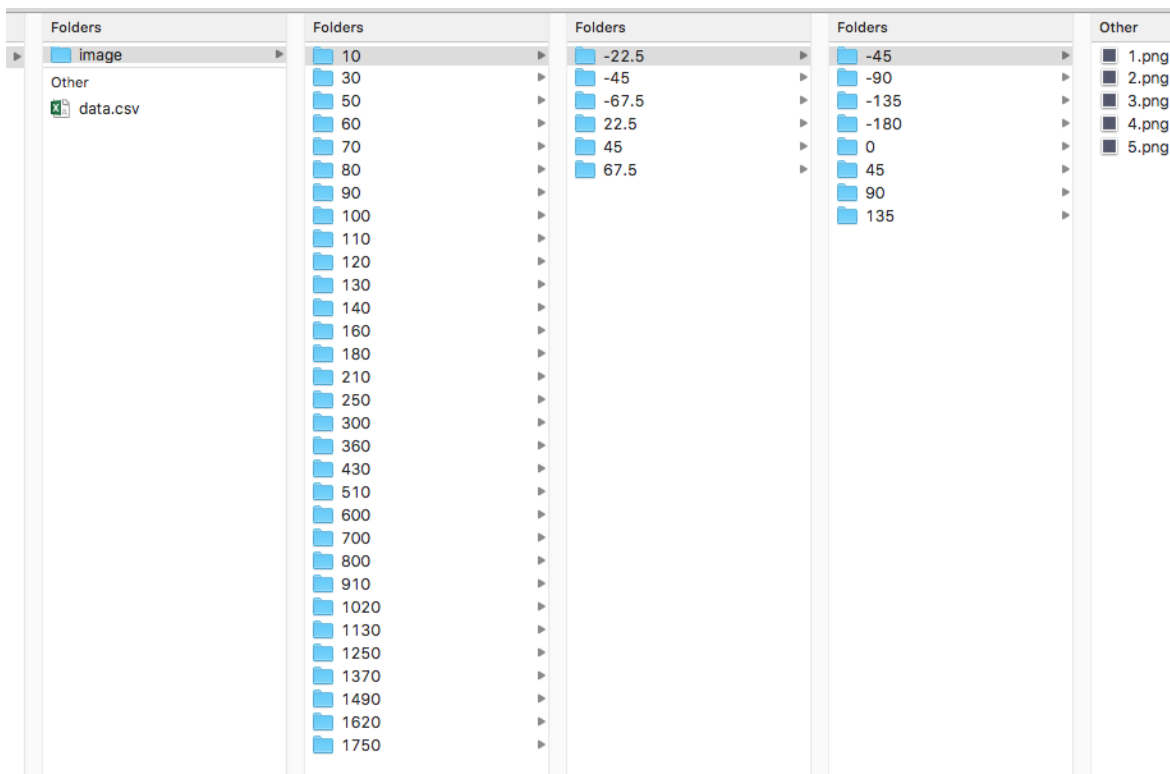
A data value representing a file path must be a string containing either a) a POSIX file path, or b) a URL. Files can be of any type, as indicated by MIME name extension, e.g., `.png`, `.vti`, `.txt`, etc.

3.1.1 Example: Nyx Cosmology Simulation

This example shows the header row and first few rows of the `data.csv` file for a Nyx Cosmology simulation. The images show dark matter density for 31 time steps, 6 phi values, 8 theta values and 5 density isosurfaces. The columns include variables that define the image: time, phi, theta, and iso, along with data derived from the simulation or from the images themselves: entropy, sample number, Hu moments and a Canny edge calculation. The last column (`FILE`) points to the image described by that row.

```
phi,theta,iso,time,sample,entropy,canny,hu0,hu1,hu2,hu3,hu4,hu5,hu6,FILE
-180,-67.5,1,10,2486,0.008196755,420,0.485110902,1.28E-07,1.04E-07,3.82E-09,-7.62E-17,
↪1.37E-12,-9.95E-19,image/10/-67.5/-180/1.png
-135,-67.5,1,10,7054,0.008344357,464,0.485084112,1.48E-07,1.14E-07,7.26E-09,-2.09E-16,
↪2.80E-12,-7.60E-18,image/10/-67.5/-135/1.png
-90,-67.5,1,10,238,0.008164505,385,0.485064888,1.08E-07,7.63E-08,9.75E-09,-2.66E-16,3.
↪20E-12,-1.43E-18,image/10/-67.5/-90/1.png
-45,-67.5,1,10,4928,0.007957556,449,0.485064602,1.05E-07,7.44E-08,9.67E-09,-2.59E-16,3.
↪13E-12,-2.23E-18,image/10/-67.5/-45/1.png
```

The file structure for the images is thus arranged by time step then phi then theta and finally has the images for each of the five density isosurfaces:



The Cinema:View and Cinema:Explorer viewers are the basic viewers used with Cinema databases, *Cinema:Viewers*.

Fig. 3.1: An example of the file system organization for a large Cinema image database (from left to right): top cdb directory, time steps, phi, theta and isosurfaces.

The
view-
ers
are

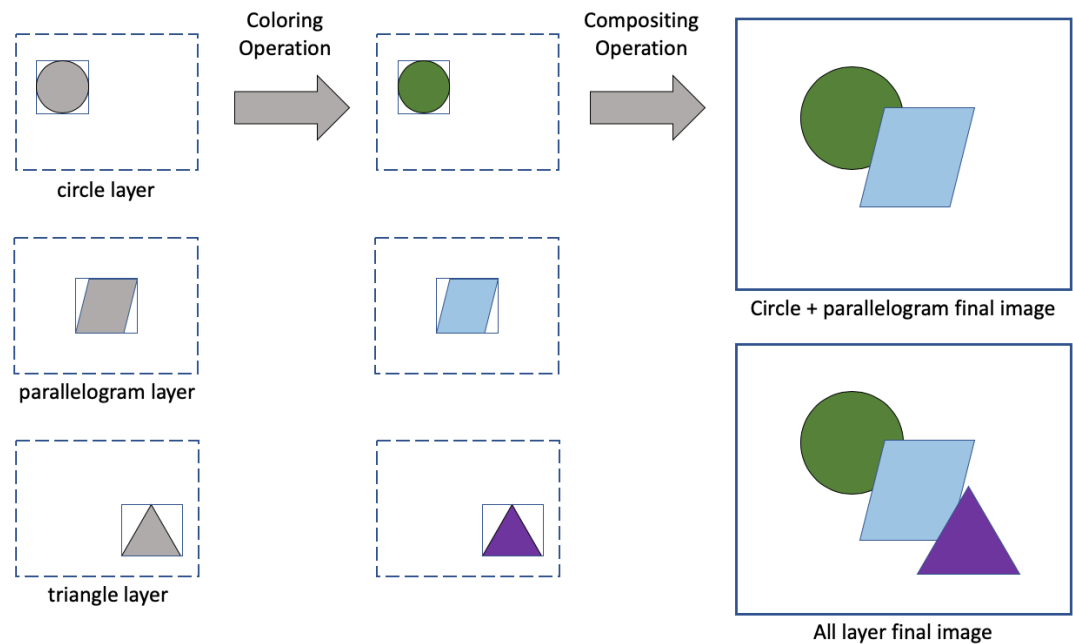
based on Spec D and provide the user with different approaches to viewing and exploring the data. The user is free to develop other viewers and analysis approaches specific to their data. The [CinemaScience Github](#) contains several repositories to support users in developing their own Cinema viewers. An example using the Cinema:Explorer viewer is shown in *Cinema:Explorer*.

3.2 Cinema Image Set Specification

The Cinema Image Set (CIS) specification extends the *Spec D specification* to enable explorable image functionality. A CIS database is a logical collection of related images or *elements*. A single CIS image is generated by coloring and compositing a set of elements within the CIS database file. The results of the coloring and compositing steps are dependent upon the information contained in the CIS file, choices by the user, and the capabilities of the consumer application (e.g., a Cinema viewer).

Fig. 3.2 illustrates how multiple layers of a CIS file can be composited into different final images.

A full description for the CIS Specification can be found in the [CinemaScience Github](#). Please see the [CIS specification](#) for up-to-date information and examples.



3.3 Depreciated Specifications

These specifications are no longer supported.

Fig. 3.2: Diagram of layers and possible composited images. Layers can be combined together in many ways, depending upon the information that is included, as well as the capabilities of the consumer of the data.

• Astaire (Spec A) : This was a

basic embodiment of the original Cinema image-based approach. It includes both static and spherical cameras, and includes the capability to turn elements on and off as detailed in the SC paper. The resulting Spec A database is a colormapped visualization that can be viewed interactively with a standalone application viewer.

- Chaplin (Spec C) : This specification included mobile cameras, floating point images, and data abstracts, and the ability to change and apply colormaps.

CHAPTER 4

CinemaScience Viewers

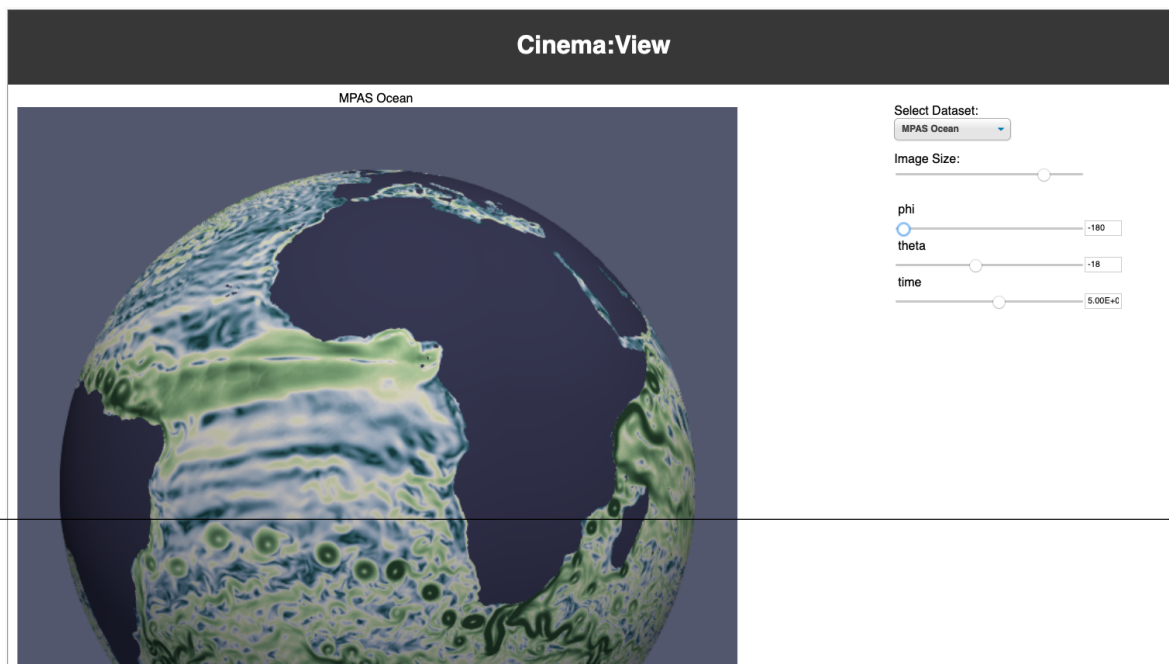
CinemaScience provides reference viewers and flexible components for users to build application specific viewers. The basic viewers are introduced here. A full viewer tutorial is available at [Tutorial: Cinema Viewers](#).

4.1 Cinema:View

Note: Cinema:View is the new version of Cinema:Compare – a now deprecated viewer.

Note: To use browser based viewers, you need to allow local file access. See [A Note on Browser Security](#) for more information.

Cinema:View is an interactive visualization approach to exploring Cinema databases. It can be used with single databases to rotate around a visualization as with the MPAS-Ocean simulation shown in [Fig. 4.1](#).



Cinema:View can also be used to compare multiple databases as with

the
Warp
plasma
ac-
cel-
er-
a-
tor
vi-
su-
al-
iza-
tion,
[Fig. 4.2](#)
that
com-
pares
dif-
fer-

ent approaches to finding isocontours in the simulation.

The list of databases available in the viewer is controlled via a `databases.json` file in the `cinema_view/data/` directory. The sets of single or multiple databases are defined in the `databases.json` file and appear as a list in the dropdown menu above the control sliders. Titles can also be specified for the each Cinema database in the comparison view. Cinema:View can be found at [cinema_view](#) and its tutorial is at [Cinema:View](#).

Fig. 4.2:

A

4.2 Cinema:Explorer

com-
par-
i-

Cinema:Explorer is a browser based viewer for Spec D databases. It includes a parallel coordinates plot and a scatterplot view. The columns are the data artifacts or derived quantities that are defined in the `data.csv` file for the two Spec D database. [Fig. 4.3](#) shows the baryon density from a Nyx cosmological simulation. Note how one can choose a subset of the images to view by selecting a region along one of the variable axes. The parallel coordinates view supports linking and brushing allowing the user to query a subset of the data and view the resultant set of images.

Cinema:Explorer uses a similar `databases.json` approach to define a list of available databases which can be explored. `databases.json` is located within the `cinema_explorer/cinema/explorer/1.9` directory. Cinema:Explorer can be found at [cinema_explorer](#) and its tutorial is at [Cinema:Explorer](#).

ac-
cel-
er-
a-
tor
sim-
u-
la-

4.3 Cinema:Scope

Cinema:Scope is a prototype cross-platform viewer application that allows the user to interactively explore a Cinema database of images, through both sliders and mouse controls. Cinema:Scope is compatible with the CSV specification of Cinema databases. In [Fig. 4.4](#), a Sedov blast wave is viewed with Cinema:Scope. The oneparameters linked to the mouse controls can be changed by the user. For databases with multiple artifacts, the on image artifact can be changed by the user to change the view to a different set of images.

the
left,
a
data-
driven
topo-
log-

12
cal
anal-
y-
sis

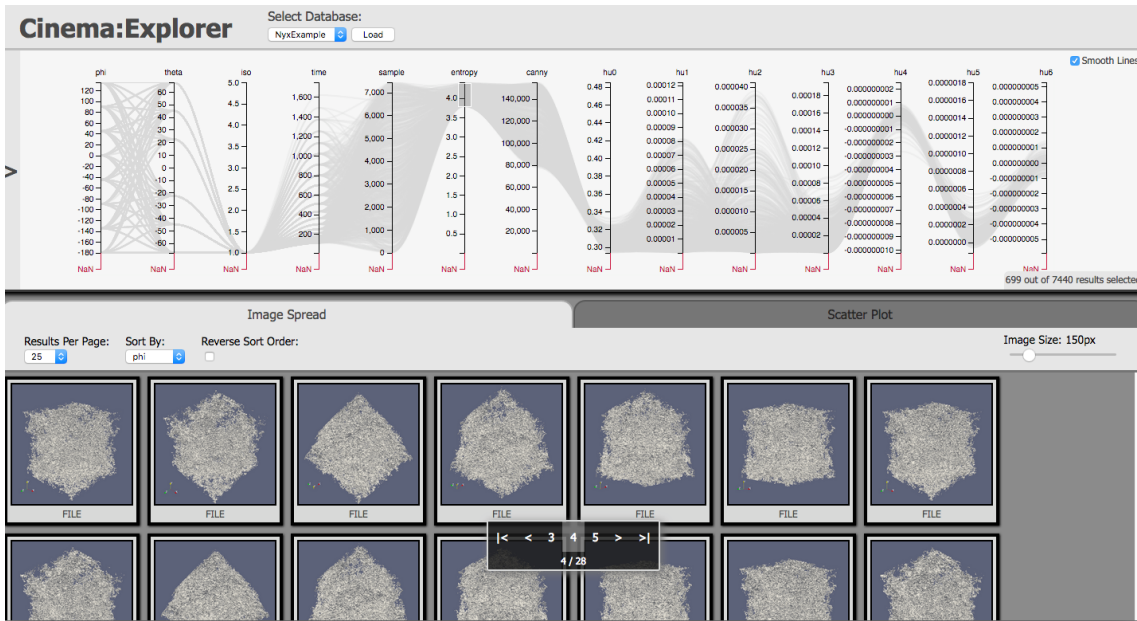


Fig. 4.3: A large Nyx cosmological simulation is viewed within Cinema:Explorer. The database has had a series of computer vision algorithms applied to extend the database (see [cinema_lib Library](#)).

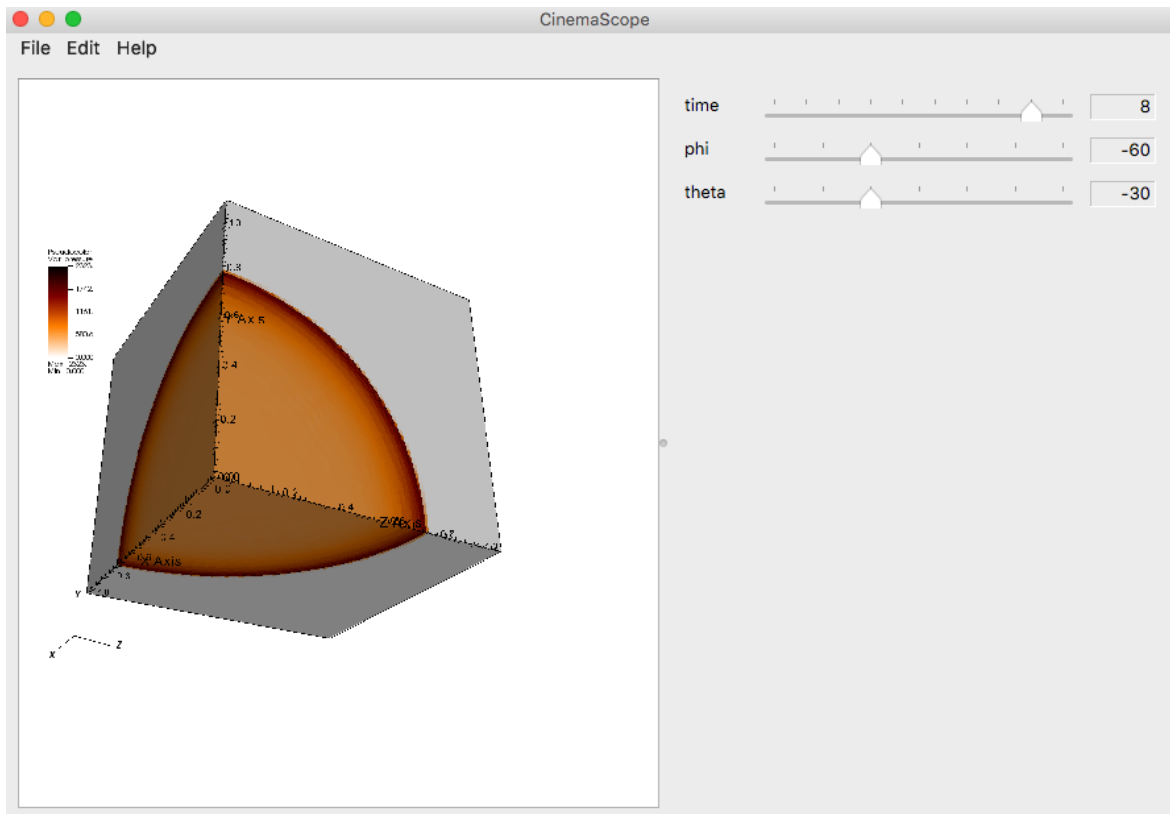


Fig. 4.4: A Sedov blast wave is viewed within Cinema:Scope. The database can be explored either through slider control or through intuitive mouse controls.

CinemaScope can be found at [cinema_scope](#) and its tutorial is at [Cinema:Scope](#).

4.4 Other Viewers

The CinemaScience GitHub page

viewers and modules that provide useful examples for developers. In particular, [cinema_components](#) provides a javascript library of components that can be used to extend CinemaExplorer or individually assembled to create an application-specific viewer (see [cinema_components Library](#)).

The Cinema viewer projects include:

- [cinema_components](#) A javascript library containing prebuilt components for viewing and querying Cinema CSV-based databases.
- [cinema_newsfeed](#) A pipeline approach to present analysis results to the user.
- [cinema_quest](#) An interactive visual tool for querying Cinema Database ensembles.
- [cinema_bandit](#) A multi-view application for analysis and visualization of experimental data.
- [cinema_debye_scherrer](#) An interactive web-based tool to visualize multiple datasets.
- [cinema_simpleviewers](#) A set of simple viewers to be used as examples to create custom Cinema viewers.
- [cinema_unityviewer](#) An experimental viewer based on the Unity game engine.
- [cinema_jnc](#) A prototype Jupyter notebook-based viewer.

Cinemasci Python Toolkit

cinemasci is a set of python tools for reading, writing and viewing Cinema databases

5.1 Requirements

Minimum Requirements are:

- Python 3.7 or above

This project uses coding standards spelled out in [PEP8](#).

5.2 Installation

The cinemasci module is available on the [CinemaScience GitHub](#): at [cinemasci](#). It can be installed in multiple ways.

Clone the repository:

```
$ git clone --recurse-submodules https://github.com/cinemascience/cinemasci.git
or
$ git clone https://github.com/cinemascience/cinemasci.git
$ cd cinemasci
$ git submodule init
$ git submodule update
```

Alternately, the latest release of this module is available on [pypi.org](#), and can be installed with `pip`:

```
$ pip install cinemasci
```

Or it can be installed locally from source using the `setup.py` file.

5.3 Cinemasci submodules

The `cinemasci` submodules are used for common Cinema tasks:

- `cdb`: Tools for reading, writing and manipulating a cinema database.
- `cis`: Tools for reading, writing and manipulating composable image sets.
- `pynb`: A Jupyter notebook viewer for simple databases.
- `server`: A simple server to help view databases, using the `viewers` submodule.
- `viewers`: Viewers (submodule)

Please see the `cinemasci` tutorial, *[Tutorial: Cinemasci](#)*, for examples of using Cinemasci functionality.

CinemaScience Algorithms, Libraries, and Tools

CinemaScience includes algorithms, code libraries, and tools to facilitate analysis in situ and postprocessing, available on the [CinemaScience GitHub](#) page. The Cinema algorithm, library, and tool projects include:

- [cinema_change_detection](#) A R command line tool that takes as input a Cinema database and produces a new Cinema database with change detection results using Myers et al. This tool also produces a change detection png image and json file to be used with Cinema:Newsfeed.
- [cinema_asqlpy](#) An apsw/SQLite virtual table interface to Cinema Spec A databases.
- [cvlib](#) A Cinema Viewer library which provides a JavaScript API to Spec A Cinema databases for visualization in the browser.
- [cvlibd](#) A Javascript framework to facilitate development of viewer applications for Spec D Cinema databases in the browser.
- [cinema_lib](#) Python 3.6 library and command line tool for Cinema specifications A and D.
- [cinema_components](#) A javascript library containing prebuilt components for viewing and querying Cinema SpecD databases.

6.1 cinema_lib Library

Cinema_lib is a set of tools and a library for interacting with Spec A and Spec D Cinema databases (CDBs) through Python and the command line tool, *cinema*.

6.1.1 Requirements

Minimum Requirements are:

- Python 3.6

Optional requirements are:

- numpy >=1.13 - image capabilities - OpenCV capabilities

- `scikit-image` $\geq 0.13.1$ (newer versions may cause regression tests to fail due to changing numerics and implementations of algorithms) - image capabilities
- `opencv-python` ≥ 3.4 (newer versions may cause regression tests to fail due to changing numerics) - OpenCV capabilities

6.1.2 Installation

To run the command line tool directly from the repository, first clone from the [CinemaScience GitHub](#):

```
$ git clone https://github.com/cinemascience/cinema_lib.git
$ cd cinema_lib
$ ./cinema
```

Or install with `pip`:

```
$ git clone https://github.com/cinemascience/cinema_lib.git
$ cd cinema_lib
$ pip install .
$ cinema
```

6.1.3 Commands

cinema_lib provides database manipulation commands, image manipulation commands and computer vision techniques. Many of these actions result in the generation of a modified or a new CDB.

Help Command

- List all of the currently implemented commands. The abbreviated help header is shown:

```
$ cinema --help

usage: cinema [-h] [--version] [-v] [-q] [-a DB] [-d DB] [-l STR] [-t] [-i]
              [--a2d] [--d2s] [--s2d DB] [--image-grey N] [--image-mean N]
              [--image-stddev N] [--image-unique N] [--image-entropy N]
              [--image-joint N] [--image-canny N] [--image-firstq N]
              [--image-secondq N] [--image-thirdq N] [--image-90th N]
              [--image-95th N] [--image-99th N] [--cv-grey N]
              [--cv-box-blur N] [--cv-gaussian-blur N] [--cv-median-blur N]
              [--cv-bilateral-filter N] [--cv-canny N]
              [--cv-contour-threshold N] [--cv-fast-draw N] [--cv-sift-draw N]
              [--cv-surf-draw N]
```

Examples: Database Manipulation

- Validate a Spec A/D database:

```
Spec A:
$ cinema -t -a path_to_database/database_name.cdb

Spec D:
$ cinema -t -d path_to_database/database_name.cdb
```

- Return the header (column labels) for a Spec D database:

```
$ cinema -i -d path_to_database/database_name.cdb
```

- Validate a Spec D database quickly (without validating the row data), reporting the header verbosely:

```
$ cinema -itvq -d path_to_database/database_name.cdb
```

- Validate a Spec A and then convert it to a Spec D database (this will result in the generation of the required `data.csv` file):

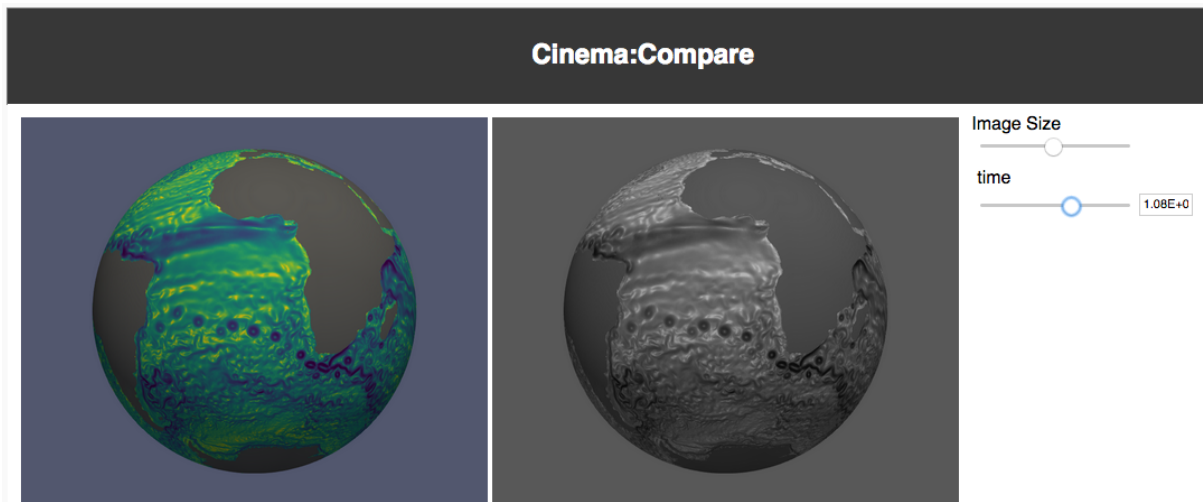
```
$ cinema -t --a2d -a path_to_database/database_name.cdb
```

Image Color Conversion and Statistics

- Convert RGB images in a Spec D database to greyscale images:

```
$ cinema -d path_to_database/database_name.cdb --image-grey 2
```

Using *Cinema:View*, one can see that the original Viridis colormap has been converted to greyscale:



Let's look at a sample workflow using an MPAS-Ocean static (no rotation) CDB with 173 time steps to perform some image-based analysis. The workflow starts with validating the database and checking the location of the image `FILE` column:

```
$ cinema -itv -d mpas_static.cdb
INFO: Checking database "mpas_static.cdb" as Spec D.
INFO: Opening CSV file "data.csv".
INFO: Header is ('time', 'FILE').
INFO: Number of columns are 2.
INFO: First data row is ('0.00E+00', 'image/0.000000e+00.png').
INFO: Data types are ('FLOAT', 'STRING').
INFO: FILE column indices are [1].
INFO: Number of data rows are 173.
INFO: 173 files validated to be present.
INFO: Check succeeded.
0: time
1: FILE
```

Note that the FILE column is N=1. Let's convert the original Viridis colormap to greyscale:

```
$ cinema -d mpas_static.cdb --image-grey 1
```

and do a quick validation to verify the new image FILE column:

```
$ cinema -itvq -d mpas_static.cdb
INFO: Checking database "mpas_static.cdb" as Spec D.
INFO: Opening CSV file "data.csv".
INFO: Header is ('time', 'FILE', 'FILEimage greyscale').
INFO: Number of columns are 3.
INFO: First data row is ('0.00E+00', 'image/0.000000e+00.png', 'image/0.000000e+00_
→image_grey.png').
INFO: Data types are ('FLOAT', 'STRING', 'STRING').
INFO: FILE column indices are [1, 2].
INFO: Doing a quick check. Not checking row data.
INFO: Check succeeded.
0: time
1: FILE
2: FILEimage greyscale
```

Choose the new FILEimage greyscale image column and calculate the mean image data:

```
$ cinema -d mpas_static.cdb --image-mean 2
```

Note that each image action may result in one or more columns of output data being added to the data.csv file so validation is a useful step to keep in mind.

The full list of image operations is:

```
--image-grey N      COMMAND: convert and write image data to greyscale PNG
                    in column number N, using scikit-image color.rgb2grey.
                    New image files are named
                    "<old_base_filename>_image_grey.png"
--image-mean N      COMMAND: add image mean data calculated from images in
                    column number N
--image-stddev N     COMMAND: add image standard deviation data calculated
                    from images in column number N
--image-unique N     COMMAND: add unique pixel count data calculated from
                    images in column number N
--image-entropy N    COMMAND: add image Shannon entropy data calculated
                    from images in column number N, using a histogram with
                    131072 bins
--image-joint N      COMMAND: add the joint entropy (multi-dimensional
                    Shannon entropy) data calculated from images in column
                    number N, using 1024 discretization levels per
                    dimension
--image-canny N      COMMAND: add Canny edge pixel count data calculated
                    from images in column number N
--image-firstq N     COMMAND: add the first quartile data calculated from
                    images in column number N
--image-secondq N    COMMAND: add the second quartile data calculated from
                    images in column number N
--image-thirdq N     COMMAND: add the third quartile data calculated from
                    images in column number N
--image-90th N       COMMAND: add the 90th percentile data calculated from
                    images in column number N
--image-95th N       COMMAND: add the 95th percentile data calculated from
```

(continues on next page)

(continued from previous page)

```

--image-99th N      images in column number N
                    COMMAND: add the 99th percentile data calculated from
                    images in column number N

```

Computer Vision Commands

In addition to standard image operations and statistics, *cinema_lib* includes a set of Computer Vision commands. As an example, one can find features identified by a FAST algorithm. Let's start with the original MPAS images:

```

$ cinema -itv -d mpas_static.cdb
INFO: Checking database "mpas_static.cdb" as Spec D.
INFO: Opening CSV file "data.csv".
INFO: Header is ('time', 'FILE').
INFO: Number of columns are 2.
INFO: First data row is ('0.00E+00', 'image/0.000000e+00.png').
INFO: Data types are ('FLOAT', 'STRING').
INFO: FILE column indices are [1].
INFO: Number of data rows are 173.
INFO: 173 files validated to be present.
INFO: Check succeeded.
0: time
1: FILE

```

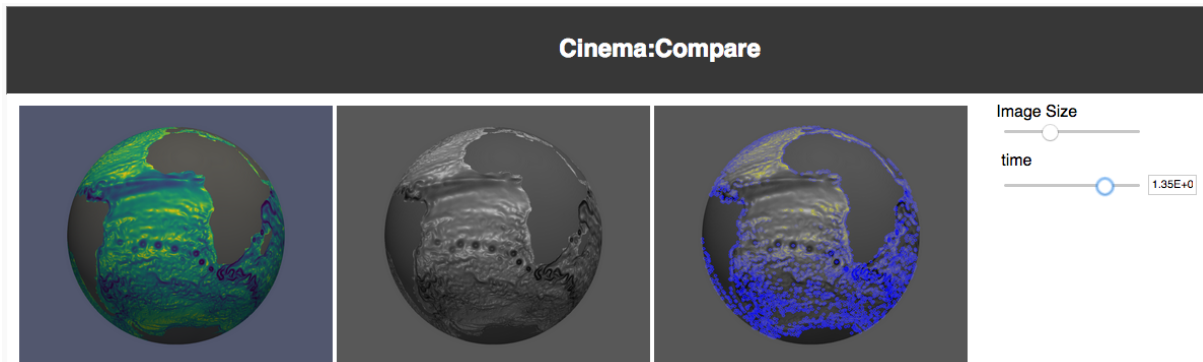
First use a *cinema_lib* function to convert the Viridis colormap to greyscale and then run the *cinema* cv-fast-draw option:

```

$ cinema -d mpas_static.cdb --cv-grey 1
$ cinema -d mpas_static.cdb --cv-fast-draw 2 --label FAST

```

Once again, using *Cinema:View*, one can see the original Viridis colormap, its greyscale counterpart, and the FAST features added from the Computer Vision suite of functionality. The FAST algorithm finds the eddies, land masses, and ocean currents:



The full list of computer vision operations is:

```

--cv-grey N      COMMAND: convert and write image data to greyscale PNG
                  in column number N, using OpenCV cvtColor. new files
                  are named "<old_base_filename>_cv_grey.png"
--cv-box-blur N  COMMAND: apply box blur to image data in column number
                  N. new files are named
                  "<old_base_filename>_cv_box_blur.png"
--cv-gaussian-blur N COMMAND: apply Gaussian blur to image data in column

```

(continues on next page)

(continued from previous page)

```

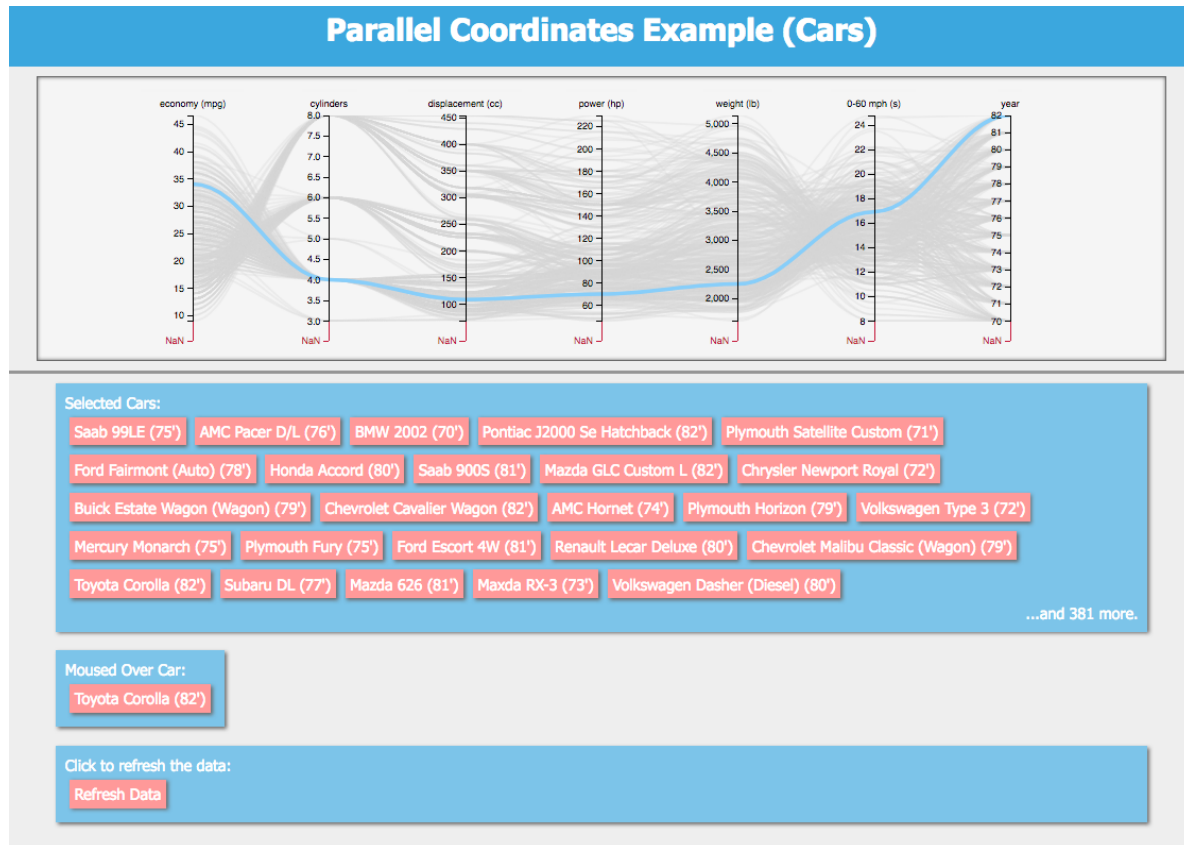
number N. new files are named
"<old_base_filename>_cv_gaussian_blur.png"
--cv-median-blur N  COMMAND: apply median blur to image data in column
number N. new files are named
"<old_base_filename>_cv_median_blur.png"
--cv-bilateral-filter N  COMMAND: apply bilateral filter to image data in
column number N. new files are named
"<old_base_filename>_cv_bilateral_filter.png"
--cv-canny N  COMMAND: apply Canny edge detector to image data in
column number N. new files are named
"<old_base_filename>_cv_canny.png"
--cv-contour-threshold N  COMMAND: draw contours around image thresholds on
image data in column number N. new files are named
"<old_base_filename>_cv_contour_threshold.png"
--cv-fast-draw N  COMMAND: draw FAST features on image data in column
number N. new files are named
"<old_base_filename>_cv_fast_draw.png"
--cv-sift-draw N  COMMAND: draw SIFT features on image data in column
number N. new files are named
"<old_base_filename>_cv_sift_draw.png"
--cv-surf-draw N  COMMAND: draw SURF features on image data in column
number N. new files are named
"<old_base_filename>_cv_surf_draw.png"

```

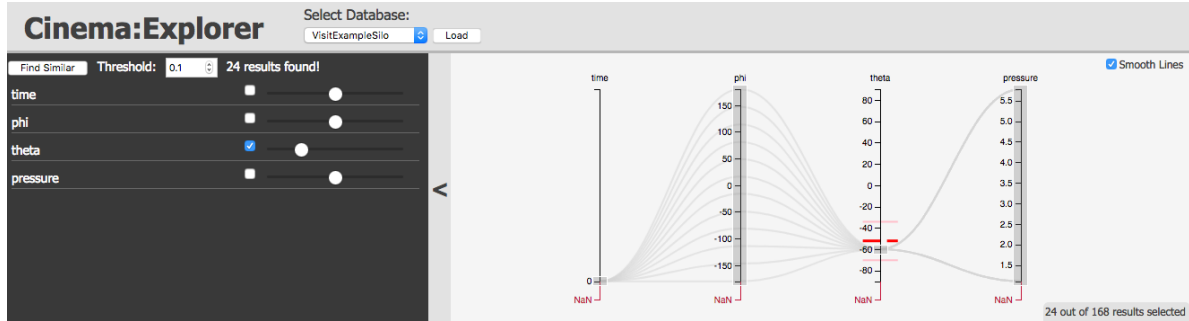
6.2 cinema_components Library

CinemaScience includes a library of viewer components that can be added by the user to create analysis and data specific viewers. These components include:

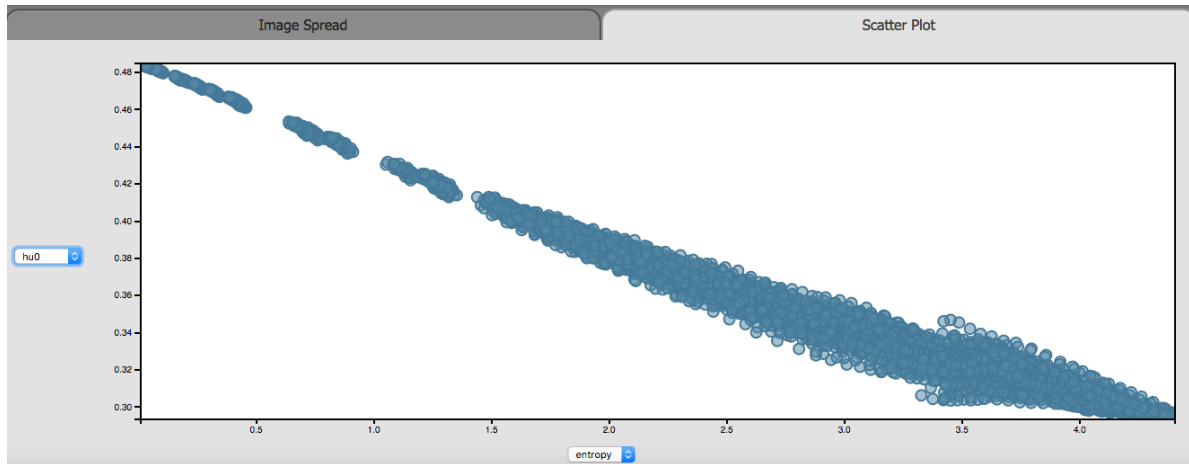
- `PcoordSVG` – A component for viewing and browsing a database on a Parallel Coordinates Chart (rendered with SVG).
- `PcoordCanvas` – A component for viewing and browsing a database on a Parallel Coordinates Chart (rendered with Canvas).



- **Glyph** – A component for viewing data on a Glyph Chart.



- ScatterPlotSVG – A component for viewing data on a Scatter plot (rendered with SVG).
- ScatterPlotCanvas – A component for viewing data on a Scatter plot (rendered with Canvas).



6.2.1 Example Use Case

Cinema viewers are JavaScript/HTML/CSS based and use D3 to link data and user actions. Below is a simple example of a browser page that uses a `pcoordSVG` component to control the display of an `ImageSpread` component for a database name `mydata.cdb` located in the same directory:

```
<html>
  <head>
    <!--Import D3-->
    <script src="lib/d3.min.js"></script>
    <!--Import Cinema Components Library-->
    <script src="CinemaComponents.js"></script>
    <!--Include Component's CSS-->
    <link rel='stylesheet' href='css/CinemaComponents.min.css'>
  </head>
  <body>
    <!--The component will be placed inside container-->
    <div id="pcoord_container" style="width:500px;height:400px;"></div>
    <div id="spread_container" style="width:100%;height:400px;"></div>
    <script>
      var chart, spread;
      //First create a database
      var database = new CINEMA_COMPONENTS.Database('mydata.cdb',
    ↪function() {
      //This callback function is called when the database has_
    ↪finished loading
    (continues on next page)
```

(continued from previous page)

```

        //Use it to create your components
        chart = new CINEMA_COMPONENTS.PcoordSVG(document.
↪getElementById('pcoord_container'), database);
        spread = new CINEMA_COMPONENTS.ImageSpread(document.
↪getElementById('spread_container'), database);

        //Using dispatch events, components can communicate with each_
↪other

        chart.dispatch.on('selectionchange', function(selection) {
            spread.setSelection(selection);
        });
    });
</script>
</body>
</html>

```

Full details on the use of `cinema_components` is on the [cinema_components GitHub](#) page.

6.3 cinema_movie Tool

cinema_movie is a tool to create a movie from a Cinema database (CDB). This is a new functionality currently under development. Updates on this documentation page may not completely reflect current capabilities. For the latest information on `cinema_movie`, please check out the README the [cinema_movie GitHub](#) page.

6.3.1 Requirements

Minimum Requirements are:

- Python 3.7
- pandas, numpy, opencv-python

6.3.2 Files

```

cinema_movie - main program
cmovie - movie production module

```

The `cinema_movie` script takes in a Cinema database (CDB) and creates a movie based on the set of frames described in the `frames.csv` control file. The `cmovie` module contains the functions needed to create the movie.

6.3.3 Command Line Control Parameters

A series of command line arguments can be used to modify the functionality of `cinema_movie`:

```

cdb:      Set input path and Cinema database name (default: ./data/example_data.cdb)
frames:   Set input csv file name to choose views in the movie; assumes path is cdb_
↪(default: ./data/example_data.cdb/frames.csv)
FILE:     Set the image column used from the CDB (default: FILE)
fps:      Set the frame rate for the movie (default: 5 fps)
opath:    Set/creates path to output movie (default: ./)
movie:    Set output movie name (default: cinema.mp4)

```

There are error checks on the path and database name and to verify the database columns that will be used in the movie. If there are no images found that satisfy the requested movie parameters, a warning message will print.

6.3.4 Usage

Make a movie by running the script with any modified arguments. Examples of usage

```
$ ./cinema_movie
$ ./cinema_movie --cdb ./tmp/my_cdb.cdb --fps 10 --movie mymovie.mp4
```


CHAPTER 7

Tutorial: Cinemasci

This tutorial uses the [Supercomputng Cinema tutorial](#).

The jupyter notebook workflow needs the following installed:

- Python 3.7 or above
- pandas, numpy
- os, shutil
- cinemasci v1.4
- openCV 4.4 (opencv-python)
- skimage (scikit-image)
- notebook, jupyterLab

Start by cloning the full tutorial repository and navigating to the pngviewer subdirectory:

```
$ git clone https://github.com/cinemasience/cinema_tutorial_2020-SC
$ cd cinema_tutorial_2020-SC/cinema_jnc/pngviewer
$ jupyter notebook
```

In the jupyter notebook, open `cinema_tutorial.ipynb`. The first cell, [Fig. 7.1](#) uses cinemasci to load a volume Cinema database from the [Nyx](#) cosmology simulation show the formation of dark matter halos in the universe over time. The sliders can be used to select the phi and theta view angles and to explore the database over time.



```
In [1]: import cinemasci
import cinemasci.pynb
import os

# # create a viewer object
viewer = cinemasci.pynb.CinemaViewer()
viewer.setUIValues({'image size': 200})
viewer.hideParameterControl('producer')

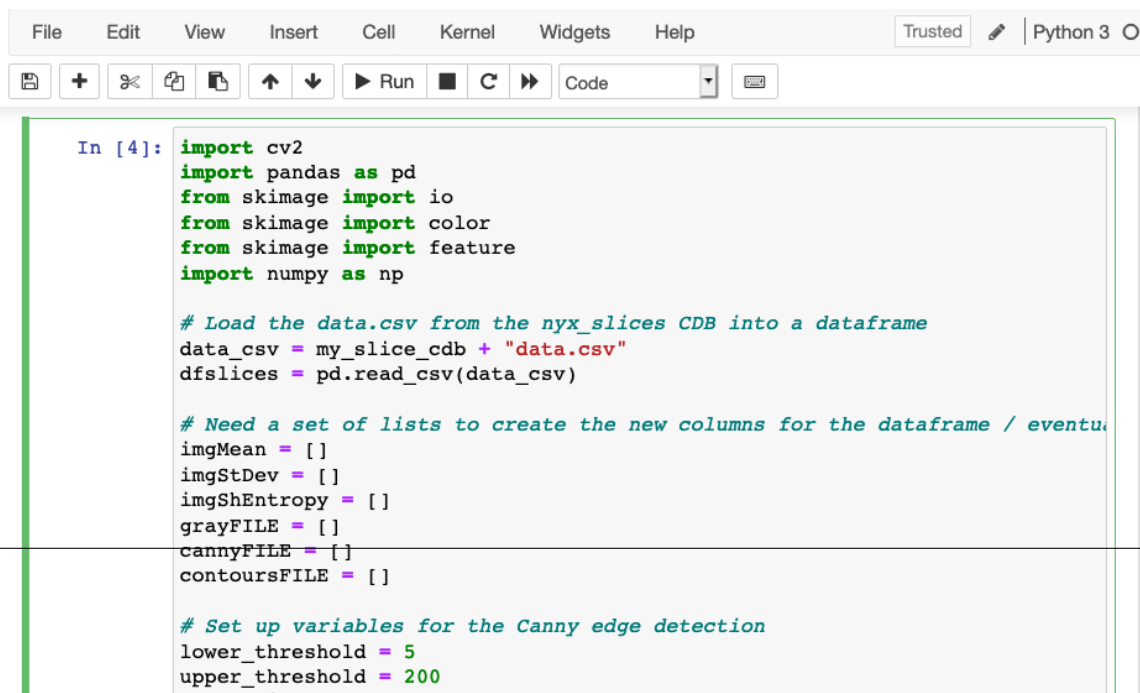
my_cdb = "data/nyx_volume.cdb"
viewer.load(my_cdb)
```

The second cell loads a Cinema database

with
two
slices
of
the
Nyx
sim-
u-
la-
tion
(de-
noted
7
and
12)
over
18
time
steps,
[Fig.
7.2.](#)
The
next
cell,

[Fig. 7.3](#), runs an image-based analysis using the Python libraries, OpenCV and skimage. The Cinema database CSV file is loaded into a pandas dataframe for easy manipulation. The image statistics and output image names are saved into new lists.

Fig. 7.2: The second cell in the jupyter notebook workflow creates a Cinema viewer object and loads a Nyx database with two slices of the Nyx simulation.



```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
[Icons] Run [Icons] Code [Dropdown] [Icon]

In [4]: import cv2
import pandas as pd
from skimage import io
from skimage import color
from skimage import feature
import numpy as np

# Load the data.csv from the nyx_slices CDB into a dataframe
data_csv = my_slice_cdb + "data.csv"
dfslices = pd.read_csv(data_csv)

# Need a set of lists to create the new columns for the dataframe / eventually
imgMean = []
imgStDev = []
imgShEntropy = []
grayFILE = []
cannyFILE = []
contoursFILE = []

# Set up variables for the Canny edge detection
lower_threshold = 5
upper_threshold = 200

```

Finally, Fig. 7.4, the image statistics and output image names are added

to
the
orig-
i-
nal
dataframe
and
re-
ordered
per
*Cin-
e-
ma-
Science
Spec-
i-
fi-
ca-
tions*
to
cre-
ate
the
new
Cin-
ema
database.
The
work-
flow
fin-
ishes
by

launching a HTML file in the browser to view the databases in the workflow.

The [CinemaScience GitHub](#) page and the [CinemaScience website](#) are useful sources for more information and ideas.

Fig.
7.4:
The
new
dataframe
is
writ-
ten
to
a
CSV
file
to
cre-
ate
the
up-
dated
Cin-
ema
database.

Note:
To

Tutorial: Cinema Workflows

This tutorial will help the user explore the CinemaScience ecosystem and give examples for possible workflows, including how to generate or export Cinema databases (CDBs).

The [CinemaScience GitHub](#) page and the [CinemaScience website](#) are useful sources for more information and ideas.

8.1 Custom Script

A basic approach is a custom script that generates a database according to the Cinema Specifications, *CinemaScience Specifications*. This approach may be appropriate for both simulation and experimental data, run statistics, or other already extant datasets. Taking in one CDB, performing analysis and outputting an updated CDB is a common workflow.

Any programming language can be used but here we demonstrate a pandas dataframe approach. In this pseudo-script, an input CSV file is read in, manipulated, has a final image FILE column added and then is written out to a CDB. Along the way, the necessary directories are created and the images are moved over from an input directory to the data/image directory.

```
#!/usr/bin/env python
# Psuedo code to generate a custom Cinema Database

import sys, os
import pandas as pd

#####
# writes the data frame to the CDB data.csv file
#####
def write_data(fname, df):
    sys.stderr.write ("Writing Cinema database data.csv...\n")

    with open(fname, "w") as output:
        df.to_csv(fname, mode='w', sep=',', index=False)
#####
```

(continues on next page)

(continued from previous page)

```

# get database parameters from sys.argv
runParams = [180101, 00] # Default run params
if len(sys.argv) == 3:
    runParams[0] = int( float(sys.argv[1]) )
    runParams[1] = int( float(sys.argv[2]) )
else:
    sys.stderr.write ( ("Warning, no run information, using defaults: {};\n").
↳format(runParams[0],runParams[1]) )

# Setup CDB directory structure and default image for file
sys.stderr.write ( 'Run Params: {}_{}\n'.format(runParams[0],runParams[1]))
sys.stderr.write ( 'Creating Cinema Database directory structure ...\n' )
inputDir = 'input/'
dataDir = 'data/run_{}_{}.cdb'.format(runParams[0],runParams[1] )
imageDir = dataDir+'/image'
defaultFILEimg = 'cinemaNoFILE.png'

# check for dataDir; copy default File img to image directory
os.makedirs(dataDir, exist_ok=True)
os.system('/bin/cp ' + defaultFILEimg + ' ' + imageDir+'/.')

# Read in the already-existing dataset
sys.stderr.write ("Reading in datafile file...\n")
inputFile = inputDir + 'run_{}_{}.csv'.format(runParams[0],runParams[1] )
dfRun = pd.read_csv(inputFile, sep=',')

# manipulate the dfRun dataframe:
sys.stderr.write ("Analysis and Image File Code Block...\n")
dfRun['newVar'] = dfRun['var1']/dfRun['var2'] # perform some analysis
if 'uselessProperty' in dfRun.columns:      # delete a column
    dfRun = dfRun.drop(['uselessProperty'], axis =1)
# ...etc...
dfRun['FILE'] = defaultFILEimg                # add the final FILE column for_
↳images
for index, row in dfRun.iterrows():          # select an image to show for that_
↳row
    thisValue = row['newVar']
    thisImage = inputDir + 'run_{}_{}_{}.png'.format(thisValue, runParams[0],
↳runParams[1])
    if os.path.isfile(thisImage):             # check if it exists and if so
        row['FILE'] = imageDir + thisImage    # set the FILE variable to that image
        os.system('/bin/mv ' + inputDir+thisImage + ' ' + imageDir+'/.') # and move_
↳it to the image directory

# Write out the dataframe to a Cinema database data.csv
datafilename = dataDir+'/data.csv'
write_data( datafilename, dfRun)

```

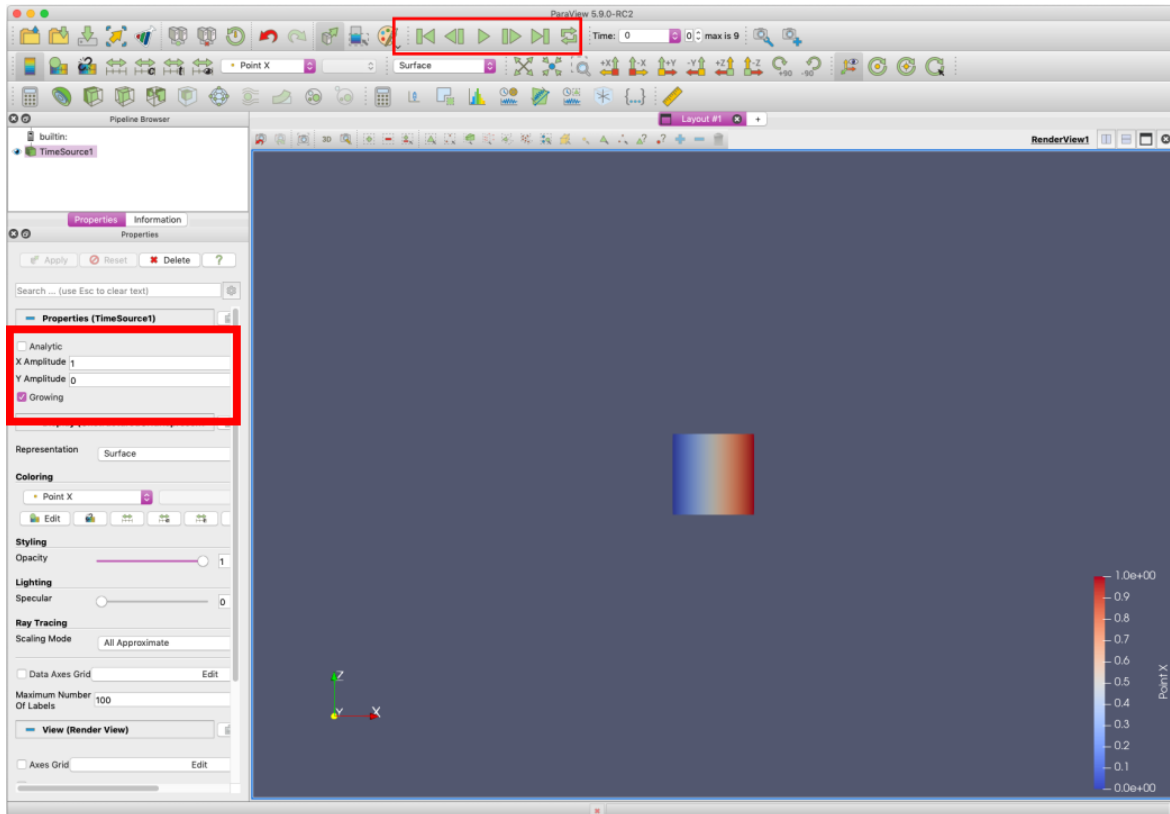
Alternately, one can use a text editor or spreadsheet program to generate the necessary CSV files.

8.2 Post-Processing via ParaView 5.9 Export Inspector

ParaView provides extract functionality for Cinema databases. This can be used in a post-processing workflow to create Cinema databases from any dataset that can be loaded into ParaView. In ParaView 5.9 (currently in a release

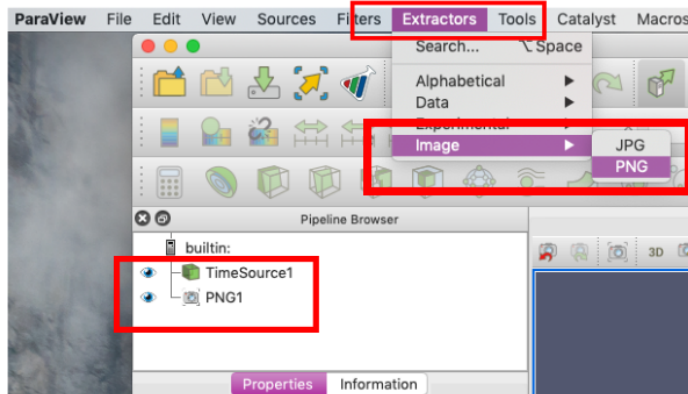
candidate version), the Cinema export has been moved into an updated Extractor scheme.

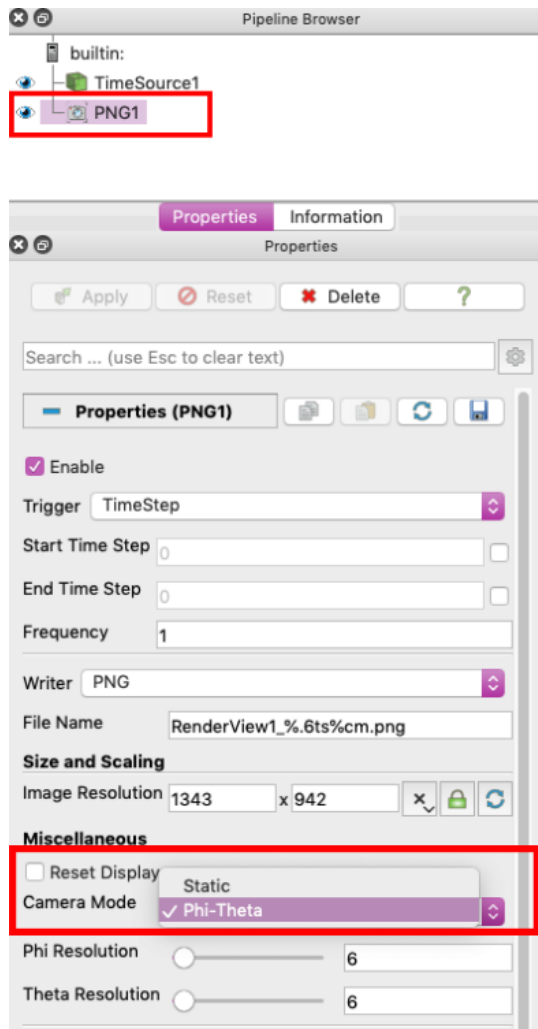
This can be demonstrated with one of the sources available within ParaView. Open ParaView 5.9 (currently RC2). From the ParaView menu, select Sources -> Alphabetical -> Time Source. We want to see time-varying behavior so under the Properties tab, change the X Amplitude to 1 and click on the Growing checkbox (see red box on left). Under Coloring, choose Point X and a Surface Representation. Make sure the visualization object is sufficiently small so that when you run the time animation (see red box on the top), you can see the time source expand and contract. The image below illustrates this setup.



The Cinema database export functionality is accessed via the Extractor menu. Click on Extractors -> Image -> PNG (or

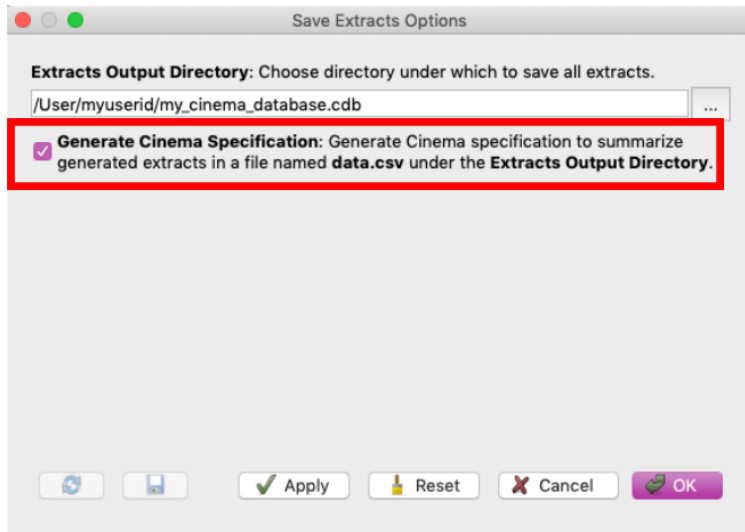
JPG). This will add a PNG (or JPG) element to the Pipeline Browser. Within the Properties, select the Camera Mode, either Static or Phi-Theta. Phi-Theta will produce images with the camera rotated at regularly spaced phi, theta intervals.





To export the Cinema database:

- Create an appropriate output directory, e.g.: `/<path_to_dir>/my_cinema_database.cdb`
- From the File menu, select `Save Extracts`
- In the `Save Extracts Options` dialog box (shown below), enter the path to the extracts output directory created above or use the browse menu (three dots) to navigate to the directory.
- Click on the `Generate Cinema Specification` box to generate the `data.csv` file

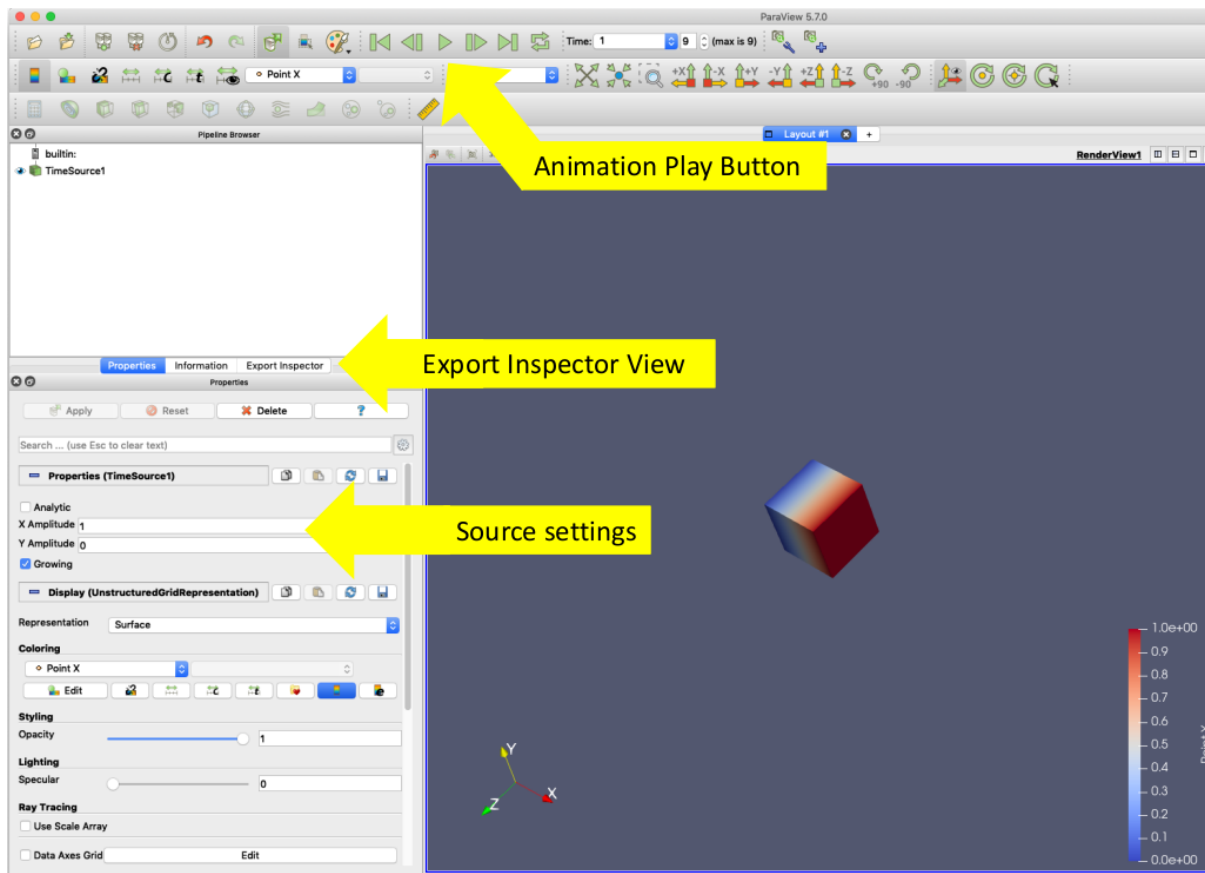


After generating the Cinema database, you can view it in one of the Cinema viewers *CinemaScience Viewers*

Note: for users not familiar with ParaView, we refer you to the [ParaView Tutorial](#) to get started.

8.3 Post-Processing via ParaView 5.7 Export Inspector

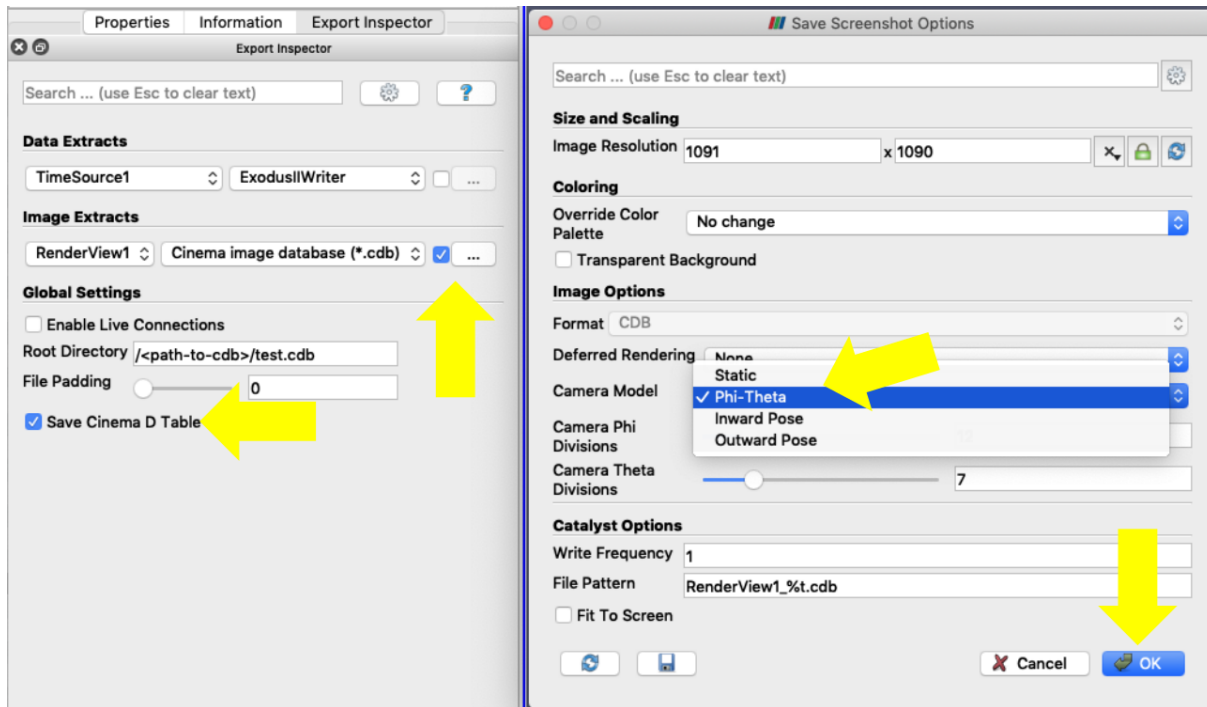
ParaView provides Cinema database export functionality which can be used in a post-processing workflow to create Cinema databases from any dataset that can be loaded into ParaView. This can be demonstrated with one of the sources available within ParaView. Open ParaView 5.7.0. From the ParaView menu, select *Sources -> Time Source*. We want to see time-varying behavior so under the Properties tab, change the X Amplitude to 1 and click on the Growing checkbox. Under Coloring, choose Point X and a Surface Representation. Make sure the visualization object is sufficiently small so that when you run the time animation (play button on the animation menu), you can see the time source expand and contract. The image below illustrates this setup.



The Cinema export functionality can be found on the `Export Inspector` view which is available as one of the default tabs when ParaView is opened (see above image) or the Export Inspector can be opened from the ParaView View menu. Click on the Export Inspector tab where you will see a set of default options. To export a Cinema Database:

- Under `Image Extracts`, make sure the correct view is selected. The default `RenderView1` will usually be the right one.
- In the dropdown menu on the right, select `Cinema image database (*.cdb)` and click the checkbox to the right to establish that option.
- Open the ellipses menu to the right of the checkbox to get the `Save Screenshot` option menu. When `Cinema image database` is chosen, it shows the Cinema options. To get multiple camera angles, choose `Phi-Theta` from the `Camera Model` dropdown menu.
- Under `Root Directory`, add the complete path and output database name: `/<path-to-cdb>/test.cdb`.
- Under `File`, select `Export Now` to start the Cinema database export.

After generating the Cinema database, you can view it in one of the Cinema viewers *CinemaScience Viewers*

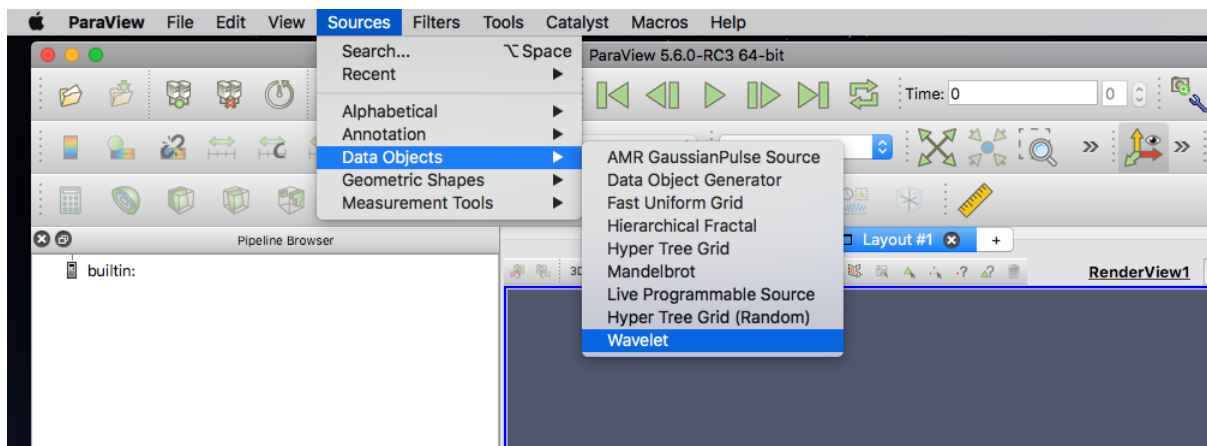


Note: for users not familiar with ParaView, we refer you to the [ParaView Tutorial](#) to get started.

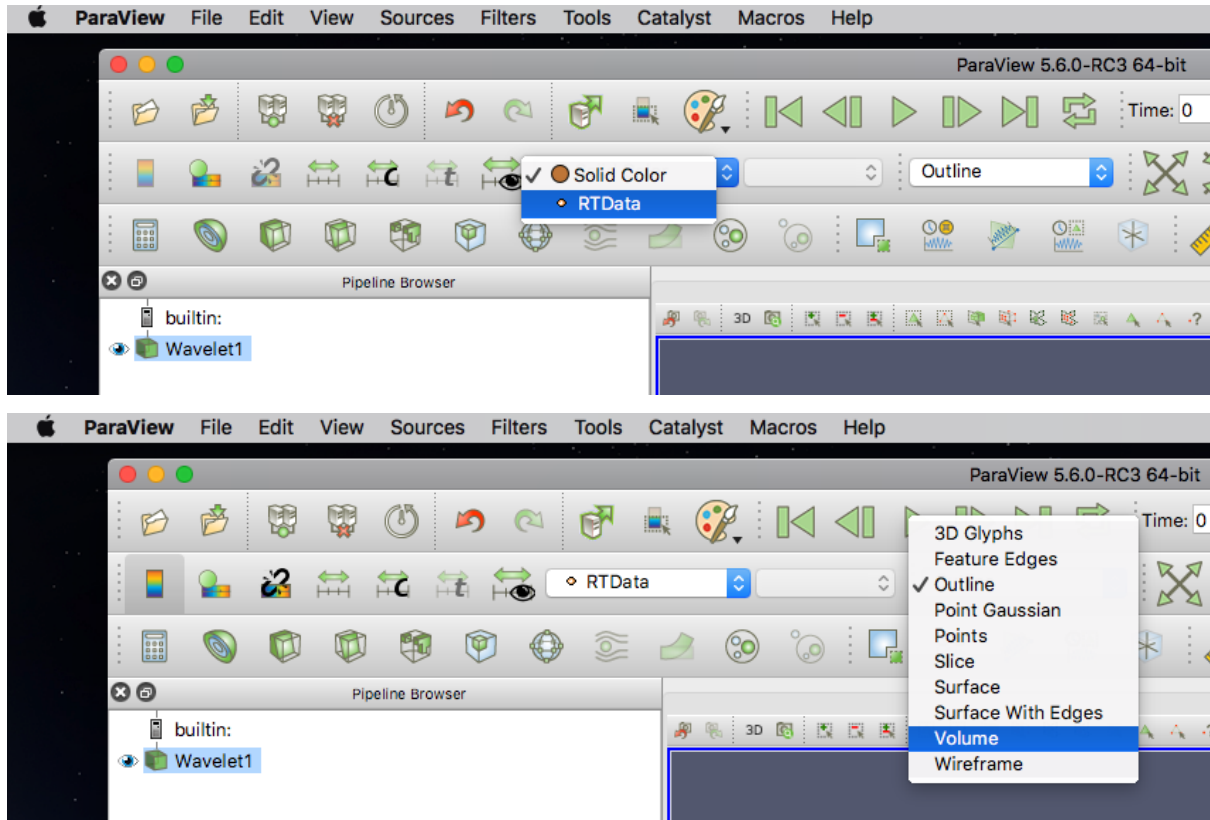
8.4 Post-Processing via ParaView 5.6 Cinema Export Scene

The current ParaView release v5.6 (in Release Candidate status as of this writing) has a Cinema Export Wizard that outputs Cinema Spec A databases. These can be converted to Cinema Spec D databases through the `cinema_lib` command line interface. The Cinema Spec D export wizard will be included in an upcoming ParaView release. This tutorial will be updated when that change takes place. The basic functionality will be similar to the following.

A Cinema database can be exported directly from ParaView. This can be demonstrated with a wavelet source. Open ParaView, select Sources -> Data Objects -> Wavelet. Click on Apply to load a basic wavelet.



Select `RTData` as the variable of interest and choose a `Volume` representation (answer Yes when it asks if you want to change the representation type):



File, open Export Scene. This will bring up the export dialog to input the CDB name and location. The Cinema Export Wizard will pop up. On the left is the default export dialog. A typical set of answers to generate a CDB placing cameras around the globe is given on the right. The Cinema export will automatically cycle through all time steps present in the data.

This Spec A data can be converted to a Spec D CDB using the `cinema_lib` tools. Please see [Converting Spec A to Spec D databases](#) for details. Once converted, the Spec D database is available for viewing in one of the Cinema Viewers as explained in the Viewer tutorial, [Tutorial: Cinema Viewers](#).

8.5 In Situ via ParaView Catalyst

ParaView's in situ Catalyst library can be used to output Cinema Spec D databases. Within ParaView, load the data and create the visualization you wish to generate in situ.

If the `Export Inspector` view is not already opened, it can be opened from either the `View` menu by selecting `Export Inspector` or from the `Catalyst` menu by selecting `Define Exports`. Set up the Cinema database export as described in the post-processing Cinema ParaView workflow, [Post-Processing via ParaView 5.7 Export Inspector](#)

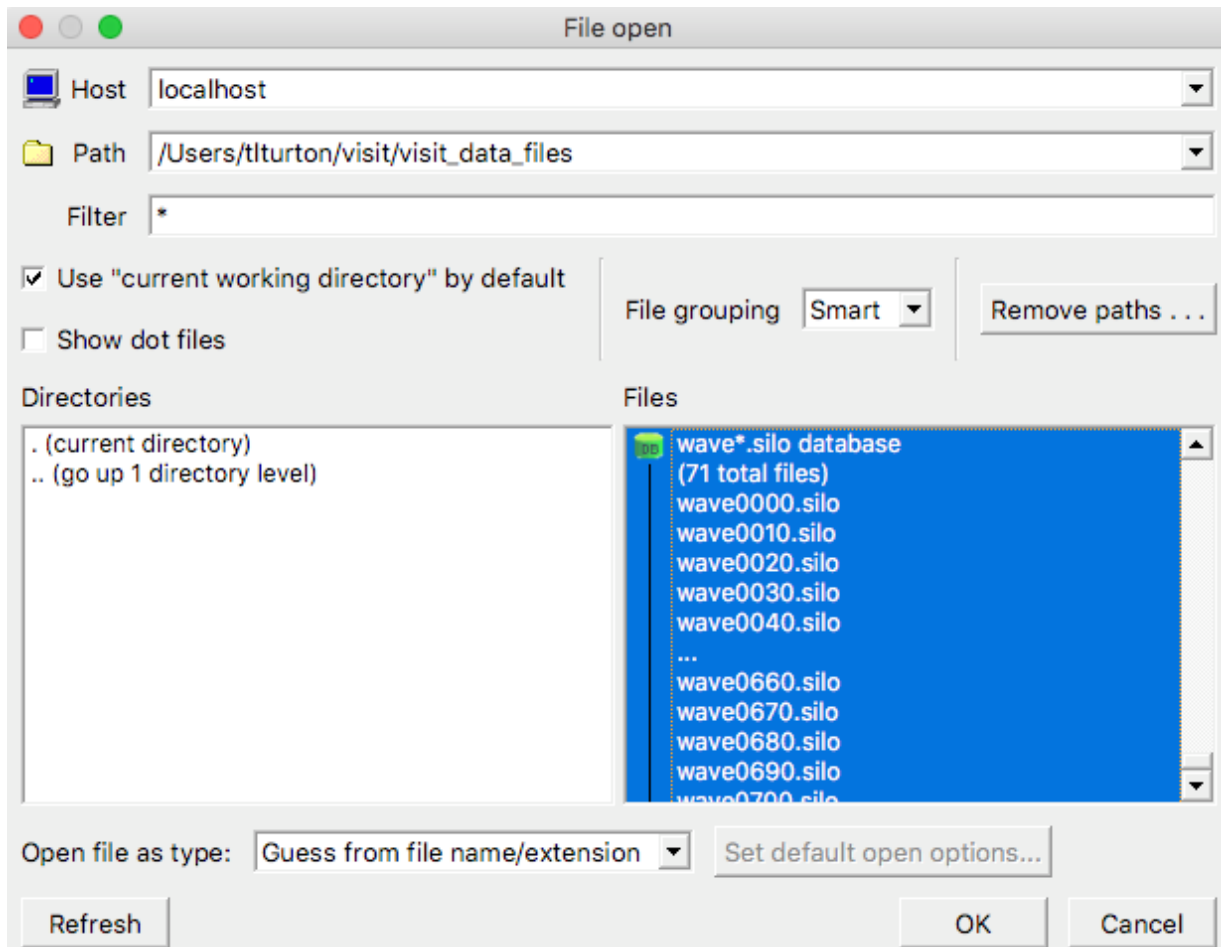
- Under `Image Extracts`, make sure the correct view is selected.
- Select `Cinema image database (*.cdb)` and click the checkbox to the right to establish that option.
- Open the ellipses menu to the right of the checkbox to select the Cinema database options via the `Save Screenshot` menu.
- Under `Root Directory`, add the complete path and output database name: `/<path-to-cdb>/test.cdb`.

- From the ParaView Catalyst menu, select `Export Catalyst Script` and enter a location and name for the exported python script. The exported script can be edited to fine-tune as needed. This script can then be integrated into an in situ pipeline.

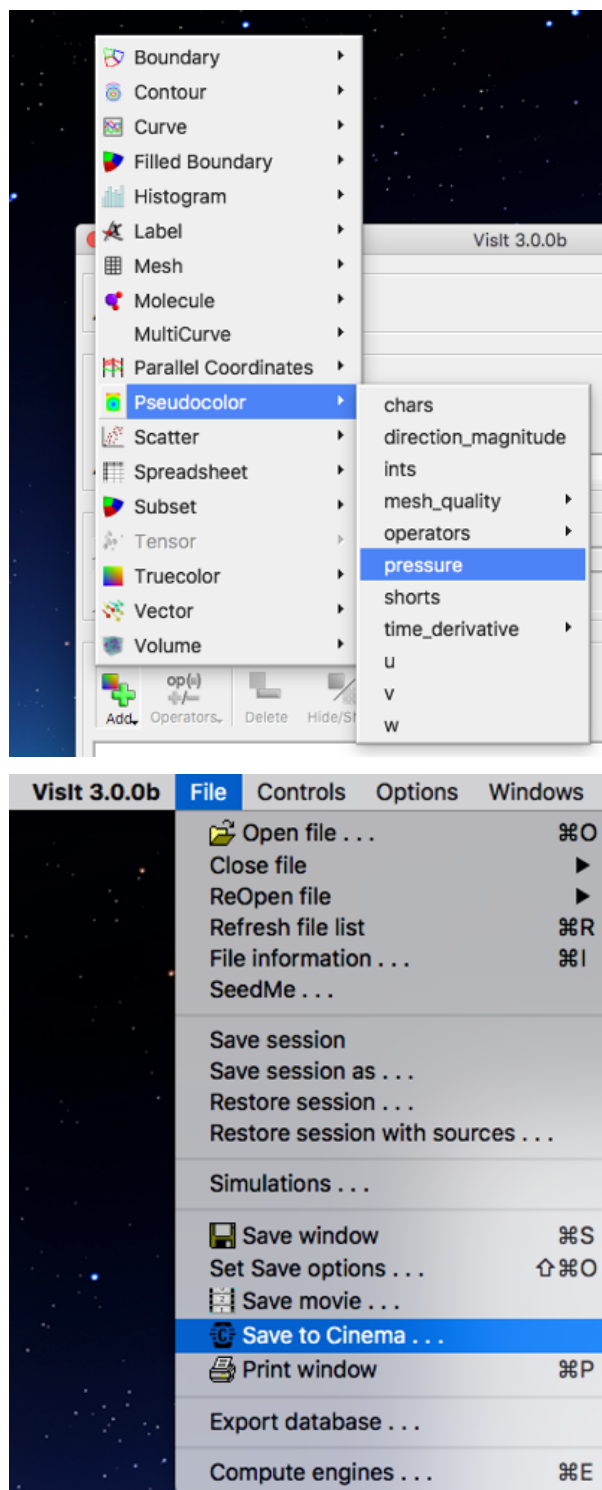
The [ParaView Python Documentation](#) is an excellent source of information on how to create a Catalyst pipeline.

8.6 Post-Processing via VisIt Cinema Export Wizard

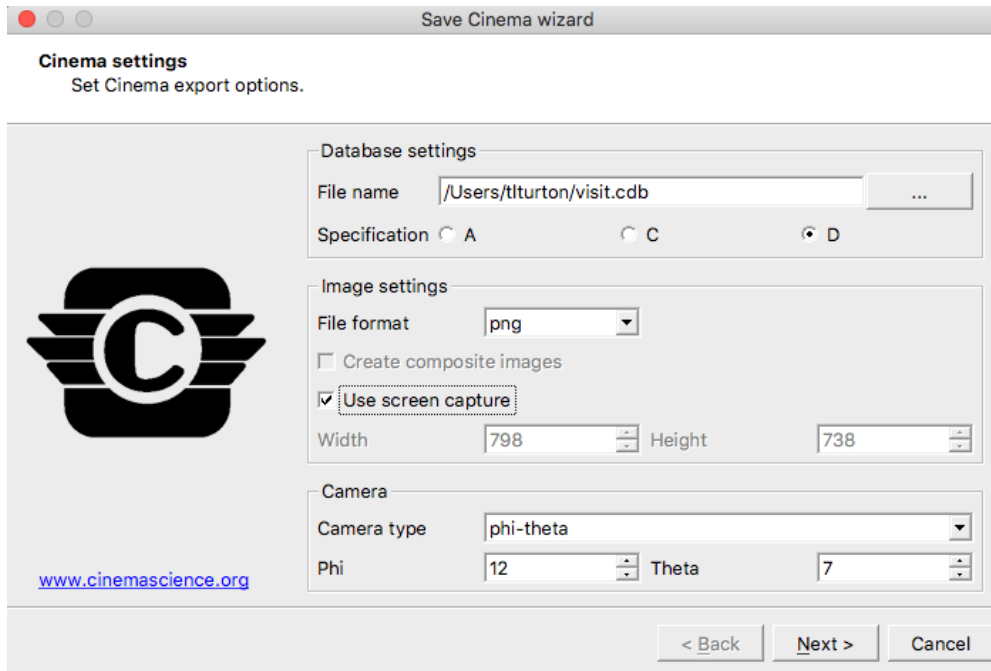
A Cinema Export Wizard is available in VisIt v3.0.2 to directly export a Cinema database from VisIt. This can be demonstrated with sample data that can be downloaded from the [VisIt tutorial](#) page. Start by clicking on `Open` to bring up the VisIt `File open` dialog box. Navigate to the correct directory, select the data you wish to visualize and click `OK`:



Click `Add`, select `Pseudocolor` and choose an appropriate variable such as `pressure`. On the control bar, click `File -> Save To Cinema` to bring up the VisIt Cinema database dialog:



In the Cinema wizard, change the database name and path as needed, select Spec D for the current Cinema specification and choose Use screen capture to select the entire VisIt viewing window. The Camera type can be either static (no rotation, only time evolution of the currently rendered view) or phi-theta for the default rotations. Modify the number of phi and theta camera locations as desired. Click Next. On the next window, choose the number of frames (time steps) to include and click Finish.



Save Cinema wizard

Cinema settings
Set Cinema export options.

Database settings

File name ...

Specification ☐ A ☐ C ☒ D

Image settings

File format

☐ Create composite images

☒ Use screen capture

Width Height


Camera

Camera type

Phi Theta

www.cinemascience.org

< Back Next > Cancel



Save Cinema wizard

Choose length
Choose start/end frame and stride.

Frame start

Frame end

Frame stride

www.cinemascience.org

< Back Finish Cancel

VisIt may request permission to access a terminal window which provides verbose output on CDB export progress.

Note: for users not familiar with VisiT, we refer you to the [VisIt Documentation](#) to get started.

8.7 In Situ via VisIt LimSim

Not currently available.

8.8 In Situ via ALPINE Ascent

Ascent is a many-core capable lightweight in-situ visualization and analysis infrastructure for multi-physics HPC simulations. Ascent is under development as part of the Exascale Computing Project's ALPINE project. For simulation codes instrumented with the [ALPINE Ascent](#) infrastructure, Cinema databases can be exported as one of the available `scenes`. Currently Ascent only outputs Spec A (the deprecated json-based specification). Spec A can be converted to Spec D databases (see [Converting Spec A to Spec D databases](#)). There are plans to update this to Spec D.

Tutorial: Cinema Viewers

This tutorial will help the user explore the CinemaScience ecosystem. It will discuss how to view Cinema databases (CDBs) with the standard Cinema Viewers.

The [CinemaScience GitHub](#) page and the [CinemaScience website](#) are useful sources for more information and ideas.

Note: To use browser based viewers, you need to allow local file access. See [A Note on Browser Security](#) for more information.

9.1 Viewing Cinema Databases

The basic Cinema viewers operate on the Cinema Spec D specification. In each case, CDBs are assumed to reside in a `data/` directory. Each CDB consists of a default `data.csv` file with columns of data abstracts following Spec D requirements (see [CinemaScience Specifications](#)), and any subdirectories needed for the data abstracts such as images, vti files, or additional csv files. The database viewers are described below.

9.2 Cinema:View

Cinema:View is designed to access the images within one or more CDBs, providing sliders spanning the values in the databases in order to select a specific image or set of comparison images. Cinema:View can be downloaded from the [cinema_view](#) GitHub page. This download will result in several directories and a `cinema_view.html`:

```
$ ls -l
  cinema
  data
  doc
  cinema_view.html
```

The `data/` directory includes a `databases.json` file and a test CDB, `sphere.cdb` with its required `data.csv`, and an `image/` subdirectory with the images arranged into subfolders by the `phi` variable.

```
cinema_view/data/sphere.cdb/data.csv
cinema_view/data/sphere.cdb/image/-18/0.png
```

In the default sphere example, there are 20 phi values and 1 theta value saved in the `data.csv`:

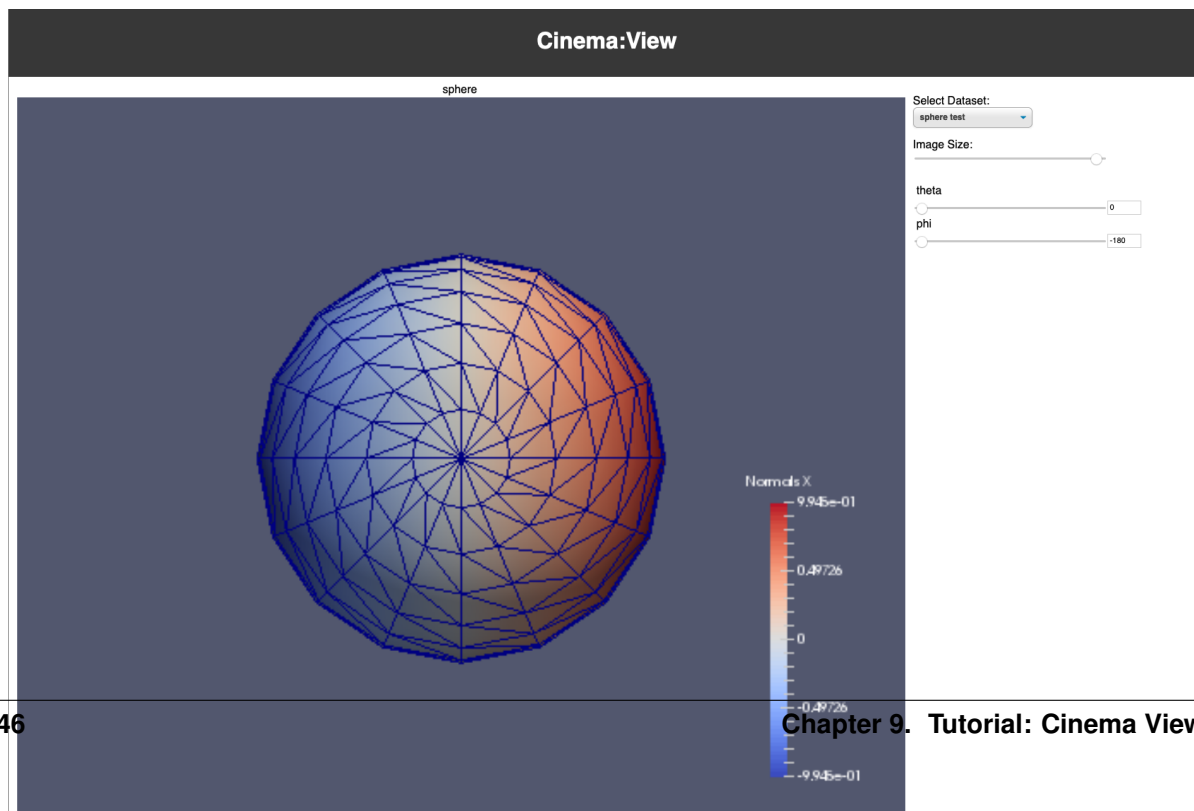
```
theta,phi,FILE
0,-180,image/-180/0.png
0,-162,image/-162/0.png
0,-144,image/-144/0.png
0,-126,image/-126/0.png
0,-108,image/-108/0.png
0,-90,image/-90/0.png
0,-72,image/-72/0.png
0,-54,image/-54/0.png
0,-36,image/-36/0.png
0,-18,image/-18/0.png
0,0,image/0/0.png
0,18,image/18/0.png
0,36,image/36/0.png
0,54,image/54/0.png
0,72,image/72/0.png
0,90,image/90/0.png
0,108,image/108/0.png
0,126,image/126/0.png
0,144,image/144/0.png
0,162,image/162/0.png
```

The `cinema_view.html` file defines the set of databases to display in the `dataSets` variable.

Opening `cinema_view.html` in Firefox:

```
$ open cinema_view.html -a Firefox
```

will bring up the Cinema:View viewer, shown in [Fig. 9.1](#). The sliders allow you to change the image size and the orientation.



This basic viewer allows the user to view a single database or a set of databases, e.g.,

Fig.
9.2,
cre-
ated
with
the
same
pa-
ram-
e-

ter

set.

The set of databases can be changed by editing the `data/databases.json` file (example below). The dropdown menu allows the user to select the database or set of databases to view. If viewing a set of databases, the sliders control all three databases in common.

9.
U
C
er
to
ex
pl
a
m
ti-
pl
C
er
da
th
vi
su
al
iz
tic
of
a
Se
de
bl
w

```
[
  {
    "database_
    ↪name-comment" : "The name of the dataset. It will appear in the dropdown.",
    "database_name": "sphere test",

    ↪"databases-comment" : "JSON array containing list of datasets to compare",
    "datasets":
      [
        {
          ↪"name-
          ↪comment": "Each dataset must have a name that will appear at the top of the image",
          "name": "sphere",

          ↪"location-comment
          ↪" : "Location of the cinema database relative to cinema root or absolute paths",
          "location": "data/sphere.cdb"
        }
      ],
    {
      "database_name": "sedov test",
      "datasets":
        [
          {
            "name": "sedov 1",
            "location": "data/sedov1.cdb"
          },
          {
            "name": "sedov 2",
            "location": "data/sedov2.cdb"
          },
          {
            "name": "sedov 3",
            "location": "data/sedov3.cdb"
          }
        ]
      }
    ]
  }
]
```

(continues on next page)

(continued from previous page)

```
}  
]
```

Note: Mistyping a database name or forgetting the `data/` directory part will result in a `TypeError`. Open the console window if nothing appears and check for the error. If so, check the `data/databases.json` file for errors.

```
TypeError: results is undefined
```

9.3 Cinema:Explorer

CinemaExplorer is a parallel coordinates approach to selecting and viewing data in a Cinema database. CinemaExplorer can be downloaded from the [cinema_explorer](#) GitHub page. This download will result in several directories and an `cinema_explorer.html`:

```
$ ls -l  
  cinema  
  data  
  doc  
  cinema_explorer.html
```

The default `data/` directory contains subdirectories with example CDBs to illustrate the range of functionality of CinemaExplorer. Again, each CDB consists of a `data.csv` file and any needed subdirectories for its data artifacts. An `image/` directory is common and the `file_types.cdb` includes a `wavelet/` subdirectory with `vti` files that can be viewed with CinemaExplorer.

```
bogus/  
  big_bogus_1.cdb/  
  big_bogus_2.cdb/  
  big_bogus_3.cdb/  
  big_bogus_4.cdb/  
  bogus_1.cdb/  
  bogus_2.cdb/  
  bogus_3.cdb/  
  bogus_4.cdb/  
file_types.cdb/  
  image/  
    wavelet/*.vti  
sphere_multi-image.cdb/  
sphere.cdb/
```

The set of databases for CinemaExplorer is defined in a `databases.json` file found in:

```
cinema_explorer/cinema/explorer/1.9/databases.json
```

`databases.json` follows JSON syntax rules. Each database entry minimally requires a name and a directory.

```
[  
  {  
    "name" : "sphere",  
    "directory" : "data/sphere.cdb"  
  },  
]
```

(continues on next page)

(continued from previous page)

```
{
  "name" : "Many File Types",
  "directory" : "data/file_types.cdb"
}
```

Opening the `cinema_explorer/cinema_explorer.html` file in Firefox

```
$ open cinema_explorer.html -a Firefox
```

will bring up Cinema:Explorer in a browser window. The default view has a parallel coordinates display of the `data.csv` columns. Each column corresponds to an axis. Fig. 9.3 shows a simple sphere in Cinema:Explorer.

By default, the first database listed in `databases.json` will initially load. All databases in `databases.json` will appear in a dropdown, Fig. 9.4 (left), menu under **Select Database:** in the CinemaExplorer browser window. After selecting a CDB, click on the **Load**, Fig. 9.4 (right), button to switch to that CDB.

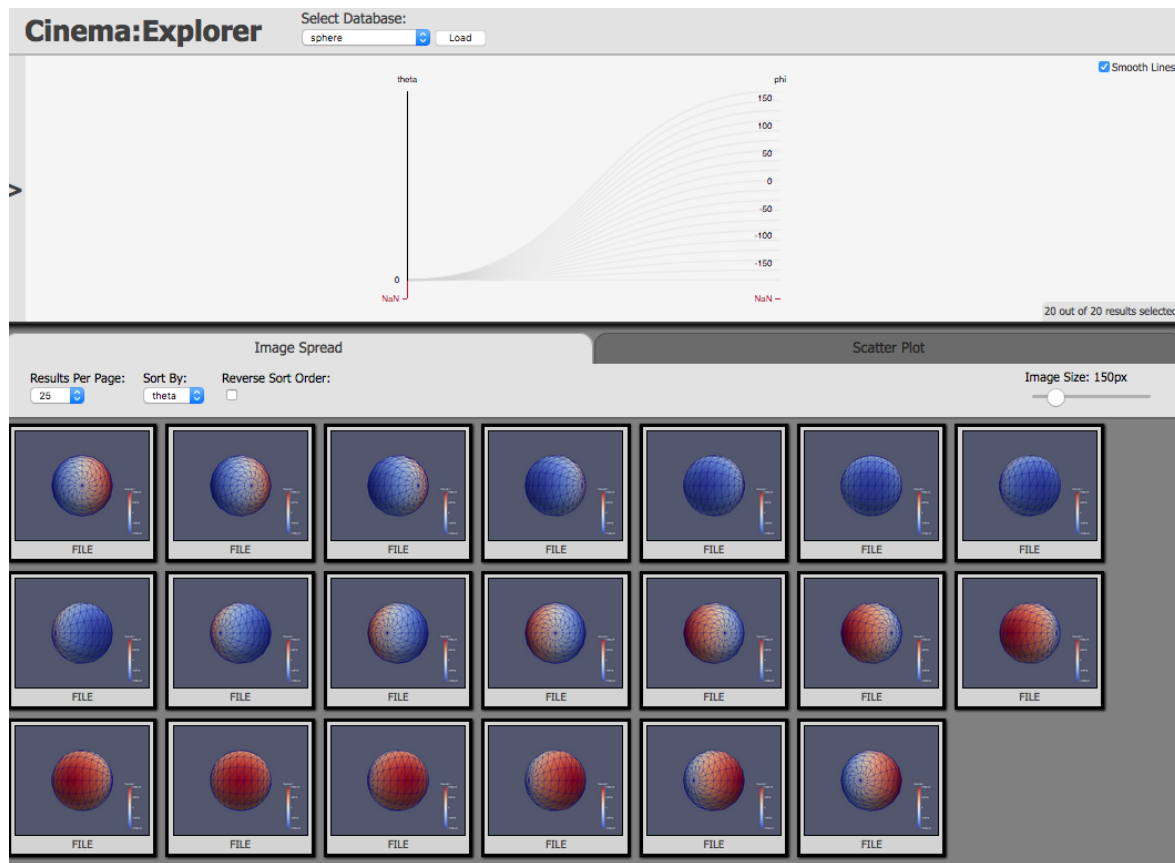
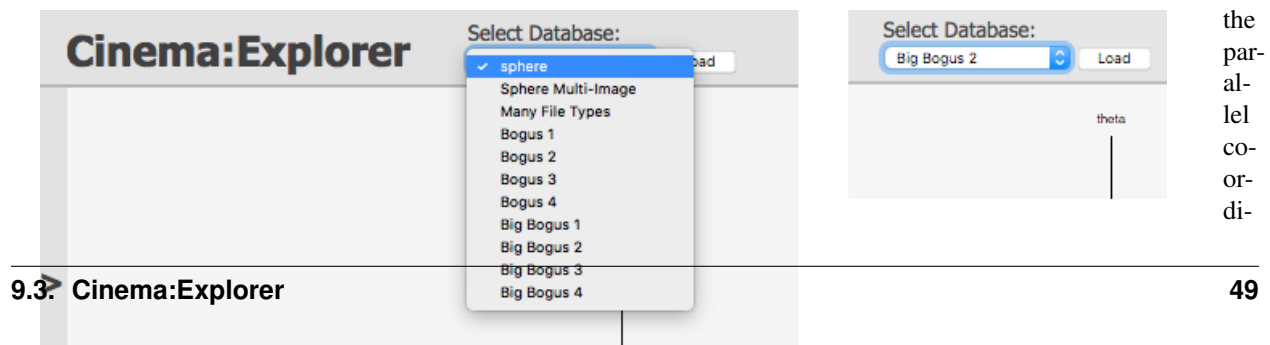


Fig. 9.3: A simple sphere database in a Cinema:Explorer window



Under
the
par-
al-
lel
co-
or-
di-

Fig. 9.4: A drondown menu displays databases listed in the `databases.json` control file. Don't forget to

notes,
the
de-
fault
tab
is
the
Image

Spread component. The image spread includes controls to change the image size, the results per page, and the sort variable and order. Let's switch to the Big Bogus 2 database, Fig. 9.5. It has several additional axes of (bogus) variables and more images than can fit in a single page. Note that CinemaExplorer switches between Canvas and SVG versions to accommodate the size of the database.

The second tab, on the right, is a ScatterPlot component. Each axis variable can be chosen from a dropdown menu of all axes so each variable can be plotted against any other variable.

The parallel component view provides a standard set of flexible actions to select and highlight data. Hovering over a specific data point or image in the CDB highlights its trace in the parallel coordinates plot and brings up a card with the detailed information from that database row, Fig. 9.7.

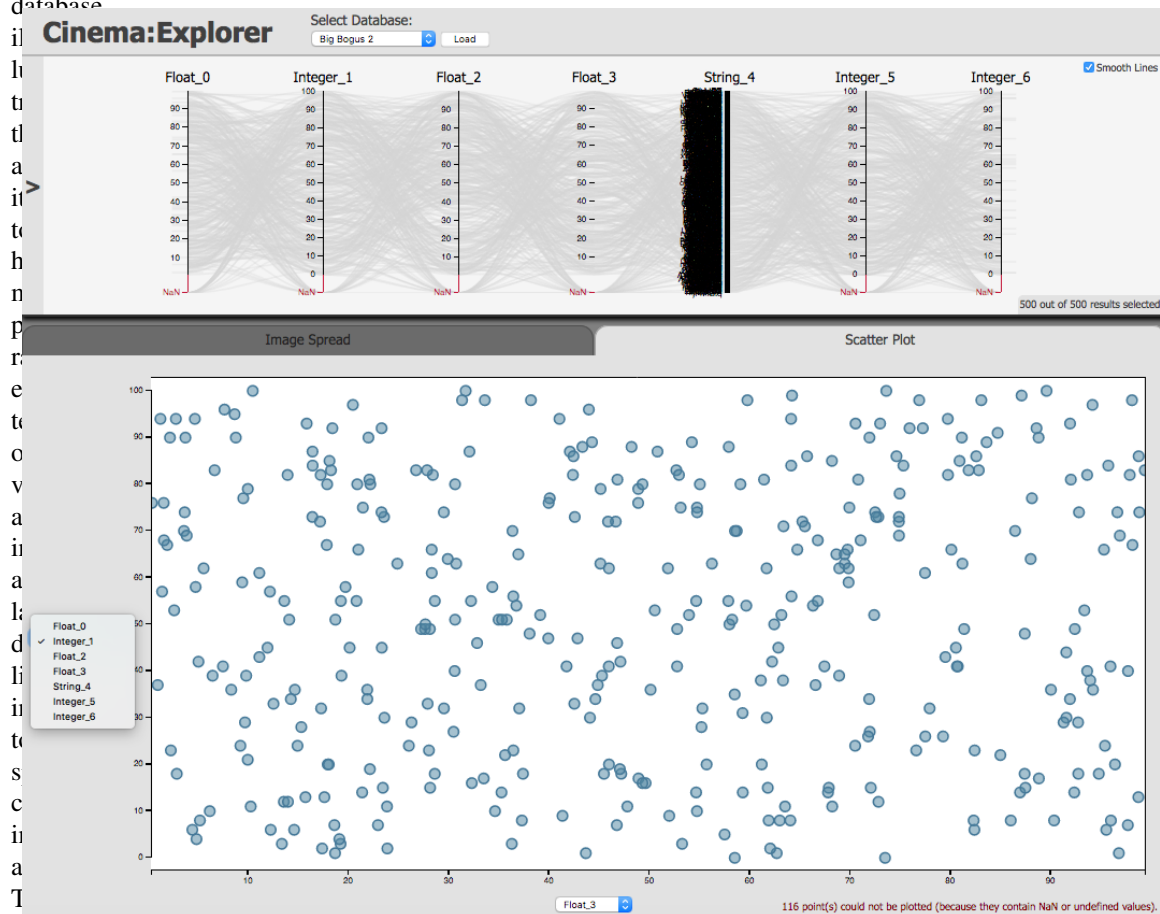
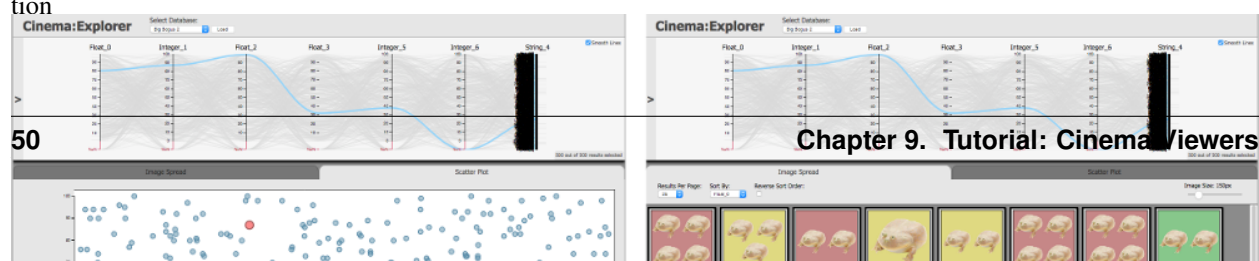


Fig. 9.6: The scatter plot tab is part of the default Cinema:Explorer capability. Dropdown menus for each axis follows the user to select a specific variable for that axis.



Another
use-
ful

feature is a modal view. Clicking on a single image will bring

up that image for closer inspection. Clicking anywhere in the main screen will dismiss the modal image. In Fig. 9.8, we select the image highlighted above.

A subset of the data can be selected via the parallel component axes. Left-mouse-click-hold-and-drag to select a range on an axis. A subset of six of the original 20 images is now visible in Fig. 9.9. That range can be shifted by hovering over the selected range then left-mouse-click and hold-and-drag. Or it can be modified by selecting one edge and dragging just that edge to increase or decrease the selection range. In Fig. 9.10, the range has been decreased to only select four of the database rows/images:

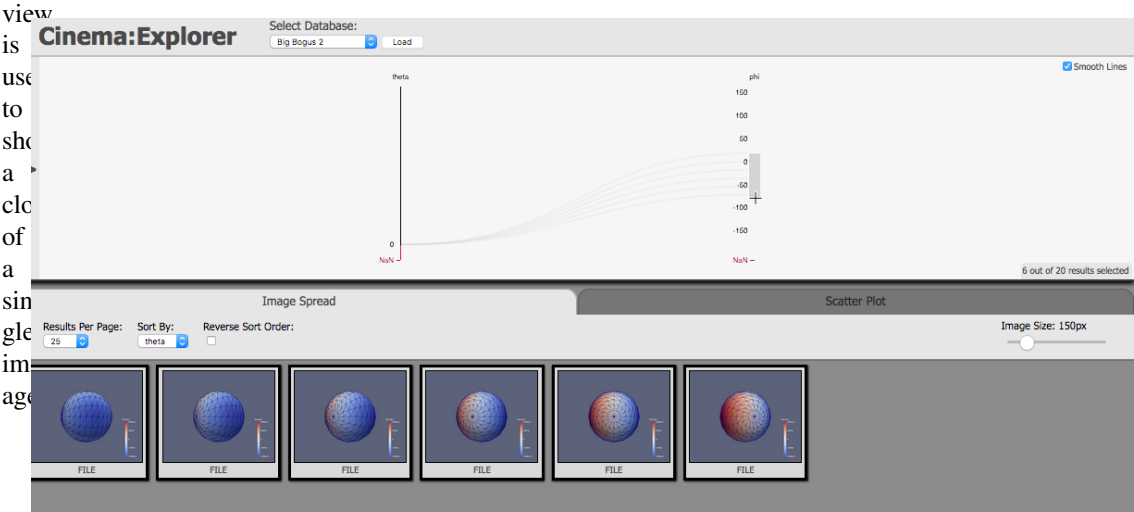
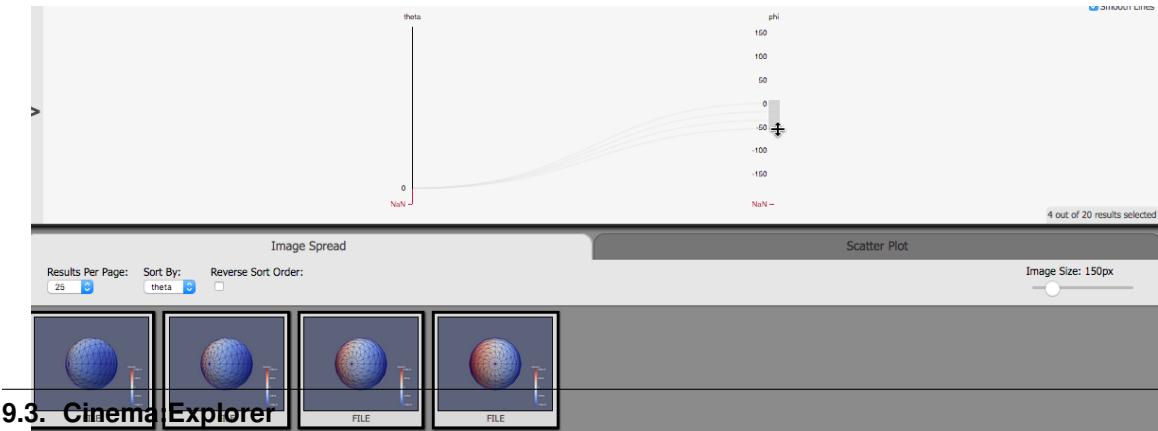


Fig. 9.9: Selecting a range on the phi axis shows only those images linked to the phi values in that range.



The selection can be cancelled by clicking on

Fig. 9.10: The range has been decreased to only include four images.

the
pre-
vi-
ously
se-
lected

axis (in an unselected area). Defining and selecting ranges on the axes is a particularly useful feature to, e.g., identify and explore outliers in the data.

9.3.1 Optional Control Fields for databases.json

There are three optional control fields that can be implemented within the `database.json` file to control the data viewed on the parallel coordinates axes.

- `filter` is a JSON regular expression which removes the specified axes (columns) whose header matches the regex. Note that adding `filter` will override the default filtering for any `FILE` columns so those will need to be explicitly removed. This example with the Bogus 1 data removes the `Float_6` axis and all axes that begin with `FILE`:

```
{
  "name" : "Bogus 1",
  "directory" : "data/bogus/bogus_1.cdb",
  "filter" : "(^Float_6)|(^FILE)"
}
```

- `query` queries the database and only displays those rows that match the criterion for the columns queried. This example with the Bogus 1 data displays only those rows where `Float_6` is within the range [20-70] and `Integer_2` is within the range [40, 60]:

```
{
  "name" : "Bogus 1",
  "directory" : "data/bogus/bogus_1.cdb",
  "query": {
    "Float_6" : [20, 70],
    "Integer_2": [40, 60]
  }
}
```

- `selection` applies an axis selection when the database is loaded but loads all database rows. This example with the Bogus 1 data brings up the database with the selection on `Integer_3` already in place, Fig. ??.

```
{
  "name" : "Bogus 1",
  "directory" : "data/bogus/bogus_1.cdb",
  "selection": {
    "Integer_3": [20, 50]
  }
}
```

9.4 Cinema:

Cinema:Scope
is
a

form application viewer to interactively explore images in a Cinema database. Cinema:Scope can be found on the [cinema_scope GitHub](#) page. Under the readme, click on `buildpassing` to bring up the list of builds. Currently, builds are available for Linux, Windows, MacOS, and MacOS/Xcode.

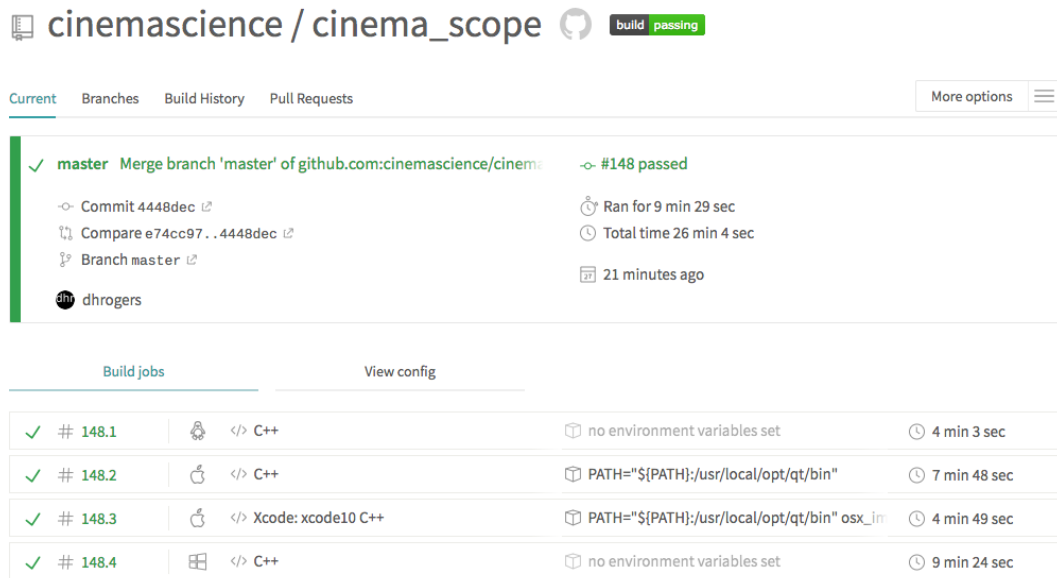
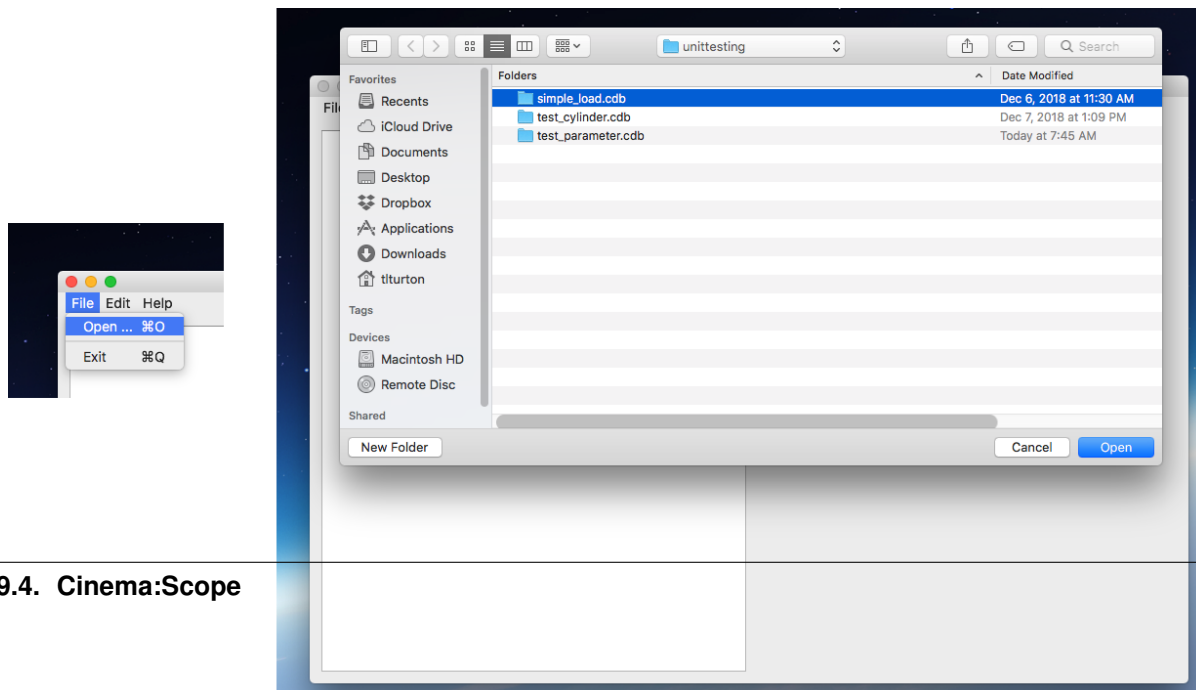


Fig. 9.12: Cinema:Scope uses GitHub's Travis CI. The dashboard shows current builds.

`docs/data/test_cylinder.cdb.zip`.

Open CinemaScope after it has installed. Click on File -> Open to open the directory and select the data set by highlighting `test_cylinder.cdb` (do not click into the directory) and clicking Open, Fig. 9.13. A test cylinder is viewed in Fig. 9.14.



pro-
to-
type
cross
plat-

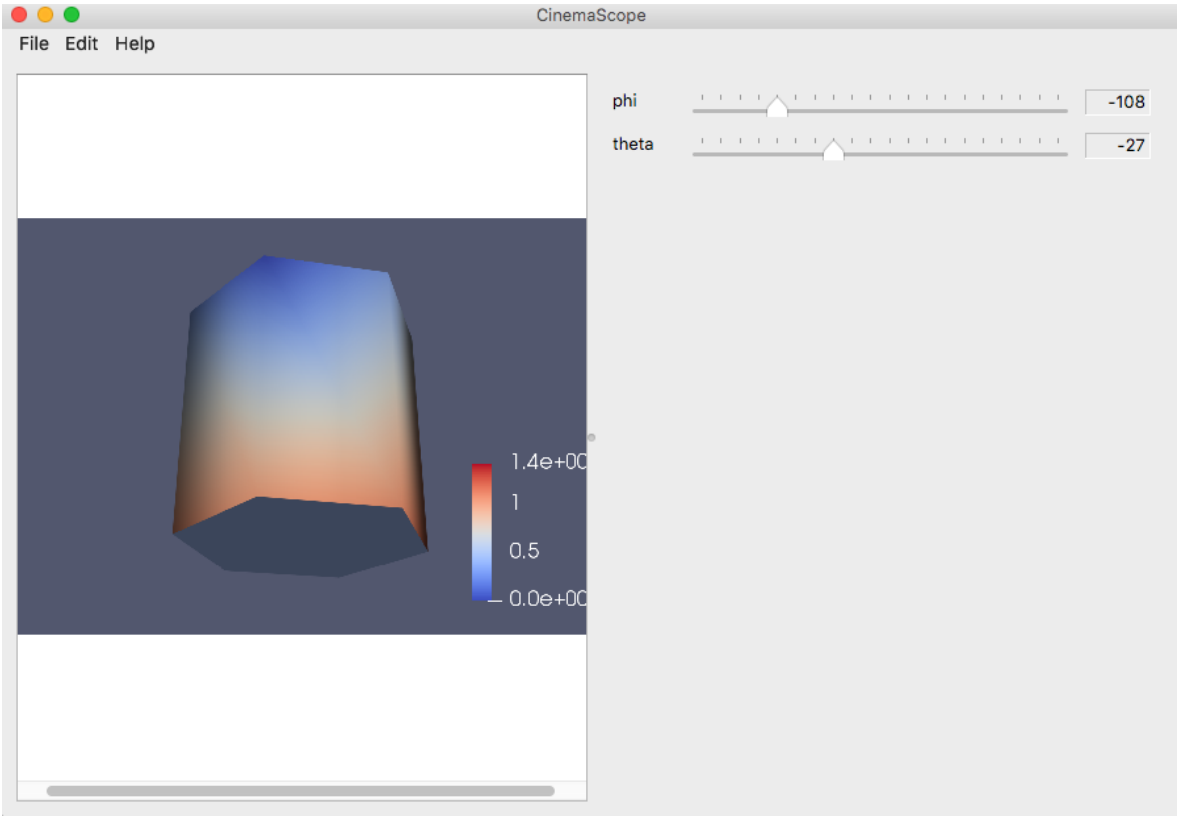
Download
the
rel-
e-
vant
build
and
in-
stall.
Test
databases
are
avail-
able
in
[https://
github.
com/
cinemascience/
cinema_
scope/
tree/
master/](https://github.com/cinemascience/cinema_scope/tree/master/)

The
mouse
wheel
can
be
used
to
in-
crease/decrease
im-
age
size.
Each
pa-
ram-

e-
ter
in
the
CDB
data.csv
will
cor-
re-
spond
to

a slider. The sliders can be used to control the parameter values. In this case, changing phi and theta rotate the cylinder, Fig. ??.

Fig.
9.14:
A
test
cylin-
der
is
view
in
Cin-
ema:Scope.



Mouse
drag
can
also
be
used
to
scroll
through
the
im-
ages.
Up/down
is
linked
by
de-
fault
to
theta
while
left/right
is
linked
to
phi.
The
linked

parameters can be modified using the Map Parameters Dialog shown in Fig. 9.15. Click Edit -> Edit Parameters to open the dialog and select the parameters from each dropdown menu. For CDBs with multiple artifacts, the Artifact dropdown menu will allow you to access each set of images.

An optional `csettings.json` file, which must be located in the `*.cdb` directory, can be used to limit the number of parameters actually linked to a slider or reorder a list of parameter columns. The `colorder` (“column order”) variable contains the list of actively linked parameters.

Fig. 9.15: Map Parameters Dialog. The following JSON snippet is an example of the `csettings.json` file:

```
{
  "colorder" : ["phi", "theta", "xValue", "yValue", "zValue", "FILE", "FILE_2"]
}
```

Parameters

Di-

9.4.1 Other Cinema viewers

log

boxAdditionally, there are other application-specific Cinema viewers that provide useful examples and inspiration for Cinema users and developers. We invite you to explore the [CinemaScience GitHub](#) for an up-to-date listing of example viewers. Cinema Viewers, in particular CinemaScope, are in active development. We invite users to join the Cinema community and contribute to the Cinema project.

to

map

the

in-

tu-

itive

mouse

con-

trols

to

spe-

cific

pa-

ram-

e-

ters

in

the

Cin-

ema

database

that

is

be-

ing

viewed

in

Cin-

ema:Scope.

Tutorial: Other Useful information

10.1 Converting Spec A to Spec D databases

It is necessary at times to convert a Spec A to a Spec D Cinema database (CDB). Spec A CDBs are available from ParaView v5.6 and earlier, from VisIt, from Ascent or you may have an older CDB that needs to be updated.

The `cinema_lib` library can be used to upgrade a Spec A CDB. Follow the instructions for downloading and installing from the [cinema_lib](#) Github page or see [cinema_lib Library](#). Included in that download is a `upgrade_cinema_database` script:

```
cinema_lib/cinema_lib/upgrade_cinema_database
```

Simply run the `upgrade_cinema_database` script giving the path to your Spec A database. This will result in the generation of a Spec D compliant `data.csv` file. Once converted, the Spec D CDB can be viewed with your favorite Cinema viewer, *CinemaScience Viewers*.

```
$ upgrade_cinema_database /path_to_database/database_name.cdb
```

10.2 A Note on Browser Security

To use the browser-based viewers, you **MUST** allow local file access. Do this in the following way, but be sure to reset these options when you are done as this allows loading of a file from any folder.

- **Firefox (preferred)**

- In the browser search bar, enter `about:config`
- Set `privacy.file_unique_origin` to `false`
- Set `security.fileuri.strict_origin_policy` to `false`

- **Safari**

- Safari->Preferences->Advanced->Show Develop menu in menu bar

- Safari->Develop->Disable Local File Restrictions (on)
- **Chrome**
 - open **chrome** with the option `--disable-web-security`
 - **Mac example:** `open -na "Google Chrome" cinema_view.html --args --user-data-dir="<path/to/repo>" --disable-web-security`

The [CinemaScience GitHub](#) page and the [CinemaScience website](#) are useful sources for more information and ideas.

Cinema Publications

If using Cinema in your work, please cite the original publication introducing Cinema.

James Ahrens, Sébastien Jourdain, Patrick O’Leary, John Patchett, David H. Rogers, and Mark Petersen. *An image-based approach to extreme scale in situ visualization and analysis*. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC ‘14). IEEE Press, Piscataway, NJ, USA, 424-434, 2014. DOI:10.1109/SC.2014.40

If using Cinema:Debye-Scherrer in your work, please cite the following publication introducing Cinema:Debye-Scherrer.

Sven C. Vogel, Chris M. Biwer, David H. Rogers, James P. Ahrens, Robert E. Hackenberg, Drew Onken and Jianzhong Zhang. *Interactive visualization of multi-dataset Rietveld analyses using Cinema:Debye-Scherrer*. J. Appl. Crystallogr. Vol. 51, 2018. DOI:10.1107/S1600576718003989

Other publications leveraging Cinema databases or using Cinema-based analysis and visualization approaches:

Pascal Grosset, Christopher Biwer, Jesus Pulido, Arvind Mohan, Ayan Biswas, John Patchett, Terece Turton, David Rogers, Daniel Livescu, James Ahrens. *Foresight: Analysis That Matters for Data Reduction*. 2020 IEEE 10th Symposium on Large Data Analysis and Visualization (LDAV) (2020). DOI:10.1109/SC41405.2020.00087

Petar Hristov, Gunther Weber, Hamish Carr, Oliver Rübel, James Ahrens. *Data Parallel Hypersweeps for In Situ Topological Analysis*. 2020 IEEE 10th Symposium on Large Data Analysis and Visualization (LDAV) (2020). DOI:10.1109/LDAV51489.2020.00008

Jonas Lukasczyk, Christoph Garth, Matthew Larsen, Wito Engelke, Ingrid Hotz, David Rogers, James Ahrens, Ross Maciejewski. *Cinema Darkroom: A Deferred Rendering Framework for Large-Scale Datasets*. 2020 IEEE 10th Symposium on Large Data Analysis and Visualization (LDAV) (2020). DOI:10.1109/LDAV51489.2020.00011

Daniel Orban, Divya Banesh, Cameron Tauxe, Christopher M. Biwer, Ayan Biswas, Ramon Saavedra, Christine Sweeney, Richard L. Sandberg, C. A. Bolme, James Ahrens and David Rogers. *CinemaBandit: a visualization application for beamline science demonstrated on XFEL shock physics experiments*. Journal of Synchrotron Radiation (2020) 27, 1-10. DOI:10.1107/S1600577519014322

Terece L. Turton, Divya Banesh, Trinity Overmyer, Benjamin H. Sims, David H. Rogers. *Enabling*

Domain Expertise in Scientific Visualization With CinemaScience. IEEE Computer Graphics and Applications (2020) 40:1 90-98. DOI:10.1109/MCG.2019.2954171

Daniel Orban, Ayan Biswas, James Ahrens and David Rogers. Drag and Track: A Direct Manipulation Interface for Contextualizing Data Instances within a Continuous Parameter Space. IEEE Trans Vis Comput Graph. (2019) 25:1, 256-266. DOI:10.1109/TVCG.2018.2865051

Robin G. C. Maack, David H. Rogers, Hans Hagen, Christina Gillmann. *Exploring Cinema Databases using multi-dimensional Image Measures*. Leipzig Symposium on Visualization in Application. (2019)

Divya Banesh, Mark Petersen, Joanne Wendelberger, James Ahrens, and Bernd Hamann. *Comparison of piecewise linear change point detection with traditional analytical methods for ocean and climate data*. Environ Earth Sci (2019) 78: 623. DOI:10.1007/s12665-019-8636-y

Jonas Lukaszcyk, Eric Kinner, James Ahrens, Heike Leitte, and Christoph Garth. *VOIDGA: A View-Approximation Oriented Image Database Generation Approach*. Leipzig Symposium on Visualization In Applications (LEVIA). Leipzig, Germany, 2018. DOI:10.1109/LDAV.2018.8739204

Divya Banesh, Joanne Wendelberger, Mark Petersen, James P. Ahrens, and Bernd Hamann. *Change Point Detection for Ocean Eddy Analysis*. Workshop on Visualisation in Environmental Sciences (EnvirVis), K. Rink, D. Zeckzer, R. Bujack and S. Janicke, eds., The Eurographics Association, 2018. DOI:10.2312/envirvis.20181134

Roxana Bujack, David H. Rogers, and James Paul Ahrens. *Reducing Occlusion in Cinema Databases through Feature-Centric Visualizations*. 2018 IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV) Berlin, Germany, 2018.

Vignesh Adhinarayanan, Wu-Chun Feng, David H. Rogers, James P. Ahrens, and Scott Pakin. *Characterizing and Modeling Power and Energy for Extreme-Scale In-Situ Visualization*. 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS) Orlando, FL, 2017, pp. 978-987. DOI:10.1109/IPDPS.2017.113

Divya Banesh, Joseph A. Schoonover, James P. Ahrens, Bernd Hamann. *Extracting, Visualizing and Tracking Mesoscale Ocean Eddies in Two-dimensional Image Sequences Using Contours and Moments*. In Workshop on Visualisation in Environmental Sciences (EnvirVis), K. Rink, A. Middel, D. Zeckzer, and R. Bujack, eds., The Eurographics Association, 2017. DOI:10.2312/envirvis.20171103

Jonathan Woodring, James P. Ahrens, John Patchett, Cameron Tauxe, and David H. Rogers. *High-dimensional Scientific Data Exploration via Cinema*. 2017 IEEE Workshop on Data Systems for Interactive Analysis (DSIA). pp. 1-5, 2017. DOI:10.1109/DSIA.2017.8339086

Anne S. Berres, Terece L. Turton, Mark Petersen, David H. Rogers, James P. Ahrens. *Video Compression for Ocean Simulation Image Databases*. In Workshop on Visualisation in Environmental Sciences (EnvirVis), K. Rink, A. Middel, D. Zeckzer, and R. Bujack, eds., The Eurographics Association, 2017. DOI:10.2312/envirvis.20171104

Patrick O’Leary, James Ahrens, Sébastien Jourdain, Scott Wittenburg, David H. Rogers, and Mark Petersen. *Cinema image-based in situ analysis and visualization of MPAS-ocean simulations*. Parallel Comput. ,55, C, p 43-48, 2016. DOI:10.1016/j.parco.2015.10.005

Vignesh Adhinarayanan, Wu-Chun Feng, Jonathan Woodring, David H. Rogers and James P. Ahrens. *On the Greenness of In-Situ and Post-Processing Visualization Pipelines*. 2015 IEEE International Parallel and Distributed Processing Symposium Workshop Hyderabad, 2015, pp. 880-887. DOI:10.1109/IPDPSW.2015.132

CHAPTER 12

Indices and tables

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