

Cinema Database
Specification "B" (Composite)
v1.0, rev1.0

David DeMarle (Kitware)
David Rogers (LANL)
John Patchett (LANL)
Berk Geveci (Kitware)

LA-UR-15-20572

Cinema *Composite* Database Specification

David DeMarle^{**}, David H. Rogers^{*}, John Patchett^{*} and Berk Geveci^{**}

^{*}Los Alamos National Laboratory

^{**}Kitware, Inc.

November 1, 2015

v1.0

1 Overview

Extreme scale scientific simulations are leading a charge to exascale computation, and data analytics runs the risk of being a bottleneck to scientific discovery. Due to power and I/O constraints, we expect in situ visualization and analysis will be a critical component of these workflows. Options for extreme scale data analysis are often presented as a stark contrast: write large files to disk for interactive, exploratory analysis, or perform in situ analysis to save detailed data about phenomena that a scientist knows about in advance. We present a novel framework for a third option—a highly interactive, image-based approach that promotes exploration of simulation results, and is easily accessed through extensions to widely used open source tools. This in situ approach supports interactive exploration of a wide range of results, while still significantly reducing data movement and storage.

More information about the overall design of Cinema is available in the paper, *An Image-based Approach to Extreme Scale In Situ Visualization and Analysis* [1], which is available at the following link:

<https://datascience.lanl.gov/data/papers/SC14.pdf>

A Cinema database is a collection of data that supports this image-based approach to interactive data exploration. It is a set of images and associated metadata, and is defined and an example given in the following sections.

1.1 Use Cases

A Cinema Database supports the following three use cases:

1. Searching/querying of meta-data and samples. Samples can be searched purely on metadata, on image content, on position, on time, or on a combination of all of these.
2. Interactive visualization of sets of samples.
3. Playing interactive visualizations, allowing the user on/off control of elements in the visualization.

1.2 Cinema is Implementation Agnostic

The Cinema Database is implementation agnostic. This database specification separates the metadata description of a set of images from the implementation of how these images are stored. In particular, if the images for a specific instance of a database are stored on disk, the design of the directory structure, metadata files, and image filenames on disk is entirely up to the person writing the data. Instead, this specification expects a database of URIs that maps metadata to data products required by the specification.

2 The Cinema *Composite* Specification

A cinema database is a set of precomputed visualization samples that can be queried and interactively viewed. This document describes release v1.0 of the Cinema *Composite* Database, in which image constituents are stored instead of pre-rendered images. The constituent images are inputs to a deferred rendering algorithm which allows the user to control which objects are in view and how each is colored. A separate document [2] specifies a *Simple* Cinema Database, which contain a fixed set of pre-rendered images for a fixed set of combinations of visualization operations.

Cinema is independent of any one specific visualization platform. Any system that can save images while varying a defined set of parameters can produce *Simple* Cinema Databases. Any system that can do the above and produce depth and other component images of isolated objects in the scene can produce *Composite* Cinema Databases.

A Cinema *Composite* database is either a set of images of type *Simple* or type *Composite*. As noted before, the specification for Simple Databases can be found in [2].

Dimensions of freedom for databases:

- Composable or non-composable
- Number of composable elements
- Dynamic or fixed colormap
- Camera position

A *Composite* database is a collection of results sampled by a set of visualization parameters. Visualization parameters can be created for any control that the user might have in a traditional visualization session. Examples include isosurfaces, slice planes and viewpoint. Cinema viewing applications give the user a control for each parameter setting and display the corresponding precomputed results, as if the user had made the same choices in a traditional visualization tool.

Examples of typical parameters in a cinema database are:

- **Time.** Time varying data can be sampled at arbitrary points along the temporal domain.
- **Camera positions.** In a static camera the position and orientation is fixed. In a spherical camera the position varies over a set of positions centered around a chosen focal point. More complicated camera tracks are possible. There is sufficient information in the cinema database to determine the world space camera position for each image.
- Zero or more **operators**, such as clipping plane and isocontour samples along their respective ranges. Each result is sampled and saved in isolation from all others with nothing else visible in the scene. In a viewing application, the user can choose any number of these samples and see them rendered together with correct occlusion culling.
- **Color** components as described next.

In a Cinema workflow, the application producing a Cinema Database creates a set of image constituents for each sample in the parameter space. These image constituents can be:

- **Depth** image. This encodes a depth value for every pixel, relative to the camera. Required for compositing.
- **Luminance** image. This encodes a rendered shading brightness for each pixel. Required for lighting to be included in the final rendering
- **Color** image. This encodes a standard RGB value for each pixel - the result of rendering the viewpoint from the camera.
- **Value** image. This encodes an arbitrary array value associated with the data that is visualized at each pixel. Required if the final rendering is to be recolored.

It is important to note that not all image constituents must be present. Because Cinema viewing applications combine these components together to produce pictures on demand for the user, the final result is dependent upon what image constituents are available. Examples include:

- If a *depth* image is available, compositing is possible.
- If a *float* image is available, the final rendering can be recolored.
- If a *luminance* image is available, the final rendering can include a lighting effect.
- If only a *color* image is available, the final rendering cannot be recolored.

In addition to the sampled data, a cinema database contains metadata that describes what the samples are, how they are stored and how they can be used. It must:

- identify the specific format (version number and content type) of the store,
- enumerate the entire set of parameters that are explorable in the database,
- map parameter value combinations to storage locations,
- define the type (depth image, color image, value rendered image, ...) of each image stored in the database,
- define the storage format (jpeg encoded buffer, png encoded buffer, raw float binary buffer, text file, etc...) of each image stored in the database,
- describe relationships between parameters.

Relationships between parameters are included because in a visualization session some parameters are constrained by others. A scene with two unrelated objects in it could have entirely different sets of arrays for each one of them and thus the choice of colors to colormap by depends on the choice of object displayed. Relationships turn otherwise order independent options into a graph of parent child relationships.

Parameter constraints and the necessity to record depth information along with standard color images are the distinguishing features between cinema composite databases and cinema simple databases.

3 Outline of a Cinema Composite Database schema in JSON data format

We now describe a JSON file schema developed to hold the metadata for a cinema database. The JSON file serves to impose structure on the collection of results.

3.1 Header information

This is required database header information, and values must be as defined below.

```
"type" : composite,
"version" : "0.0",
"metadata": {
  "type": "composite-image-stack",
  "store_type": <one of ["FS" or "SFS"]>
},
```

3.2 name_pattern

For FileStore type databases, this string maps parameter values to filenames for samples. It must be a path relative to the location of the `database.json` file.

```
"name_pattern" : <valid file path with substitutions>
Ex: "name_pattern" = "{time}/{phi}/{theta}/image.png" (several subdirectories)
    "name_pattern" = "{time}_{phi}_{theta}_image.png" (all images in one directory)
```

The filename extension (.png, .jpg, etc) determines the default type of data of items held in the store. Items in the store with an argument that has a "types" entry ignore the extension and use their own. For example a 'depth' type image is not well suited to a '.png' file, so for depth images cinema is free to use a different format, such as '.im'.

Because of the nature of parameter relationships, the pattern is only unique up to the point of constrained arguments. For constrained arguments the actual file path begins with that described in the name_pattern string. Directory and filenames for dependent parameters are then appended using the convention of "name=value/" directory names.

3.3 arguments and associations

The rest of the file defines the visualization parameters. Here we define the set of parameters, or 'arguments' and the set of valid values for each one. Arguments have the following form:

3.3.1 arguments

```
<string>: {
  "default": <value>,
  "label": <string>,
  "type": <one of ["range", "list", option]>,
  "values": [ list of unique values ],
  <optionally types: [ list of one of [rgb, depth] for each value] >
}
```

The types entry defines what image constituent the resulting sample is.

3.3.2 associations

```
<name1> : {
  <name2> : [list of permissible values for name2 that enable name1],
}
```

4 Example

This example is based on the above JSON schema outline.

```
{
  "arguments": {
    "color": {
      "default": "white",
      "isfield": "yes",
      "label": "color",
      "type": "hidden",
      "types": [
        "rgb",
        "depth",
        "rgb"
      ],
      "values": [
        "white",
        "depth",
        "red"
      ]
    },
    "color2": {
      "default": "white",
```

```

    "isfield": "yes",
    "label": "color2",
    "type": "hidden",
    "types": [
        "rgb",
        "depth",
        "rgb"
    ],
    "values": [
        "white",
        "depth",
        "red"
    ]
},
"contour": {
    "default": 0.26539633572101595,
    "islayer": "yes",
    "label": "contour",
    "type": "option",
    "values": [
        0.26539633572101595,
        0.3826981753110885,
        0.5000000149011612,
        0.6173018544912339,
        0.7346036940813064
    ]
},
"object": {
    "default": "contour",
    "islayer": "yes",
    "label": "object",
    "type": "option",
    "values": [
        "contour",
        "slice"
    ]
},
"phi": {
    "default": 0,
    "label": "phi",
    "type": "range",
    "values": [
        0,
        50,
        100,
        150
    ]
},
"slice": {
    "default": 0.1,
    "islayer": "yes",
    "label": "slice",
    "type": "option",
    "values": [

```

```

        0.1,
        0.3,
        0.5,
        0.7,
        0.9
    ]
},
"theta": {
    "default": -180,
    "label": "theta",
    "type": "range",
    "values": [
        -180,
        -130,
        -80,
        -30,
        20,
        70,
        120,
        170
    ]
}
},
"associations": {
    "color": {
        "contour": [
            0.26539633572101595,
            0.3826981753110885,
            0.5000000149011612,
            0.6173018544912339,
            0.7346036940813064
        ]
    },
    "color2": {
        "slice": [
            0.1,
            0.3,
            0.5,
            0.7,
            0.9
        ]
    },
    "contour": {
        "object": "contour"
    },
    "slice": {
        "object": "slice"
    }
}
},
"metadata": {
    "type": "parametric-image-stack"
},
"name_pattern": "{phi}_{theta}_{object}.jpg"
}

```

References

- [1] 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, pages 424–434, Piscataway, NJ, USA, 2014. IEEE Press.
- [2] David Rogers, James Ahrens, and John Patchett. Cinema simple database specification. Technical Report LA-UR-15-20572, Los Alamos National Laboratory, January 2015.