

## White Papers

### Using the OpenViz Data Model to Access Large Datasets and Improve Application Performance

#### Additional resources from AVS:

- > [Implementation examples](#)
- > [Online demonstrations](#)
- > [Product Information](#)

Product Managers and Application Developers working with OpenViz have a choice between caching all of the data required for individual or group visualizations within the powerful OpenViz data model, or, periodically accessing a data source to get required data on an as-needed basis. This technical document focuses on the implications and benefits of caching a large quantity of data within the OpenViz data model.

#### OpenViz Data Model

OpenViz employs an internal data model, called Field to store and process incoming data. AVS has been employing continually refined versions of the field data model for the past fifteen years and it has been successfully deployed by thousands of customers working with every conceivable data format and size. The field data model has been fully optimized for efficiency and is specifically designed for flexibility and scalability in handling any incoming data source, whether it is a flat table, a hierarchical tree, a satellite image, a real-time feed or any other data format.

Any dataset that is provided as input to an OpenViz component is treated as if it were stored in the Field data structure. OpenViz components access input data by calling method functions that are defined in Field interfaces. Any object that implements the OpenViz Field interfaces can be used as a data source for an OpenViz component—regardless of the manner in which the data is actually stored.

A Field contains a set of nodes. Each node has a location that could be an index into an array or a 3D coordinate that describes a position in space. In addition to its location, each node can have one or more data items associated with it.

For example, consider an array that contains currency values representing a list of stock prices. Each array element is a node that has one piece of the currency data associated with it. As another example, consider a dataset that contains temperature and pressure values measured at various locations in space. Each measurement point is a node that has two data items (temperature and pressure) associated with it.

One of the greatest benefits of utilizing a single unified data model with OpenViz is that it allows the developer to fuse together multiple data sources within a single OpenViz view without having to worry about the myriad of different incoming data formats.

The Field data model is also designed to provide the software developer with fine-grain control of how much data to provide an OpenViz application. For instance, a developer may choose to initially provide only a highly aggregated view of a dataset and display it as a 3D bar chart, and, subsequently load a portion of the raw data in reaction to a user-selected aggregated value bar and display it.

Alternatively, the developer may choose to load all the raw data from the data source into OpenViz, and then use the OpenViz aggregation (analytic) and binning (data grouping) functionality to present an aggregated view of the data to the user. When the user wants to drill-down further, OpenViz can dynamically change the existing aggregated view of the data to a more refined version without having to reach the data source again. The data filtering and cropping functionality in OpenViz may also be utilized to provide a custom view of the raw data to the end user.

**Benefits and Implications of the OpenViz Field Data Model:** A significant benefit of the OpenViz Field data model is the fine-grain control the developer has in selecting the size and quantity of data to load at any given time. Choosing to load as large a quantity of data as possible at the start of a user session offers some powerful benefits:

**Data Reduction:** It is often useful to visualize only a subset of data, eliminating irrelevant information or focusing on data that is particularly relevant. In addition to allowing the developer to focus on a particular subset of data, data reduction techniques help make the use of computing resources more efficient and applications don't dedicate time and memory to process and retain data that the end user does not wish or need to visualize.

**There are many development techniques in OpenViz that support data reduction:**

1. **Crop** — Just as one crops a photograph, the OpenViz cropping components allow the developer to remove from a field any nodes which do not fall within certain coordinate boundaries.
2. **Threshold** — Thresholding allows an application to selectively nullify data in one array based on a threshold condition applied to the corresponding element in another array. This can be useful in order to focus in on a subset of data, reduce clutter and confusion within visualization, or filter out data that is clearly erroneous or irrelevant. An example would be to chart sales data excluding sales totalling less than a \$50 threshold.
3. **Downsize** — The downsize technique allows the developer to thin out a dataset by re-sampling the data at a fixed interval. This downsize component can be used to save memory and processing time on datasets that are repetitive or too large to be processed by downstream components with acceptable performance. This technique would typically retain a pattern in a data set while reducing the size needed to be processed and visualized.

**Data Mapping:** While data processing can be performed prior to bringing data into OpenViz, it is often useful to provide all of the data into an OpenViz Field structure and use OpenViz components to manipulate the data as needed for the desired visualization. OpenViz implements many of the data set operations typically performed with a database language such as SQL.

OpenViz also allows set operations on data, producing useful information about groups of records much like a Pivot Table in a spreadsheet, or a GROUP BY clause in a SQL query. In OpenViz terminology, this type of data grouping is known as binning, meaning that OpenViz creates a series of bins and distributes the records within the bins based on certain characteristics.

OpenViz allows the developer to perform arbitrary aggregation operations on the contents of each bin. For instance, an application may wish to apply the "SUM" function on a bin, indicating that the set operation is the SUM function, which simply adds up all of the values. Passing this configuration to a charting component, it is possible to generate a chart that indicates the sales totals of each salesperson. One can see how this information may be more useful than a chart that creates a bar for each and every sale.

It may seem that it would be simpler to perform these calculations on the data before creating an OpenViz field. A developer could, for example, use a GROUP BY clause to get the same total sale data that would be generated by binning. There are a number of reasons for configuring an application so that OpenViz components do some of the data manipulation; if an application offered numerous charts (such as total sales, average profits, number of sales, etc.), each user session would have to run all of the related different queries when the application was launched, or, reconnect to the database each time the user selected another chart to view. Another advantage of using OpenViz caching is that it permits multidimensional binning, meaning that bins can be defined by up to three different data values.

**Summary:** Caching data within an OpenViz field offers certain obvious advantages. It permits the fusion of data coming from multiple data sources into a single field data model. Subsequently, multiple data reduction and/or data mapping operations can be performed on the field data model without needing to connect back to the data source, a computationally- and/or time-intensive task that places a burden on network and system resources and reduces the time to insight for application users.