A Visualization Architecture for Collaborative Analytical and Data Provenance Activities

Aqeel Al-Naser*, Masroor Rasheed†, Duncan Irving‡ and John Brooke*
*School of Computer Science
The University of Manchester, United Kingdom
Email: aqeel.al-naser@cs.manchester.ac.uk, john.brooke@manchester.ac.uk
†School of Earth, Atmospheric and Environmental Sciences
The University of Manchester, United Kingdom
Email: masroor.rasheed@postgrad.manchester.ac.uk
‡Teradata Corp.
Email: duncan.irving@teradata.com

***Abstract*—**
When exploring noisy or visually complex data, such as seismic data from the oil and gas industry, it is often the case that algorithms cannot completely identify features of interest. Human intuition must complete the process. Given the nature of intuition, this can be a source of differing interpretations depending on the human expert; thus we do not have a single feature but multiple views of a feature. Managing multi-user and multi-version interpretations, combined with version tracking, is challenging as these interpretations are often stored as geometric objects separately from the raw data and possibly in different local machines. In this paper we combine the storage of the raw data with the storage of the interpretations produced by the visualization of features by multiple user sessions. We present case studies that illustrate our system's ability to reproduce users' amendments to the interpretations of others and the ability to retrace the history of amendments to a visual feature.

***Keywords*-geospatial visualization; data acquisition and management; provenance; data exploration; query-driven visualization;**

I. Introduction

One of the most powerful benefits that visualization brings to data analysis is the ability to harness the intuition of the user in the process of understanding the data. Human visual abilities are particularly tuned to respond to features embedded in such a space. In this paper, we consider seismic imaging data which has a natural representation, in the three dimensions of physical space, of subsurface layers and is rich of geological features such as horizons and faults.

In many cases, human intuition is supported by algorithms that help to identify and highlight features of the data. However, it can often be the case that the algorithms cannot completely identify the features of interest. Human intuition must complete the process, and given the nature of intuition this can be a source of differing interpretations depending on the human expert. This may occur in data that is noisy or visually complex. Examples of such data are found in medical imaging and in the field that is the topic of this paper, interpretation of geophysical seismic imaging data [1]. Thus we do not have a single feature but multiple interpretations of a feature. At some stage, collaborative visualization may be required for experts to discuss and reconcile these different interpretations. We also need to track the provenance of such interpretations; sometimes earlier interpretations may need to be revisited. The process can be envisaged in a similar way to the source trees created by different programmers working on a large software project and the version control systems that have arisen to manage the process of collaboration and integration. Now if the interpretations are stored as geometric objects (e.g. isosurfaces) separately from the data and possibly on different local machines, this bookkeeping becomes very complex and error prone.

In this paper we propose a novel method of creating the objects that underlie such visual interpretations, in such a way that the information contained in the interpretation is directly stored as metadata alongside the original data. In addition to the ability of users to experiment with local views of data, this provides a support from the visualization architecture for the considered results of such experiments to be stored, shared and re-used. In our proposed architecture, users' interpretations can flow in the reverse direction from the usual pipeline, back from the user's interaction with the visual presentation of the objects to the original data source of the pipeline. We utilize the increased capabilities of highly parallel databases that allow flexible indexing and optimised retrieval in response to data queries. Such databases have been created to solve problems of "big data" from the commercial world such as customer relationship management and inventory and asset control.

In this paper, we apply our methods to the problems of the interpretation of data from geoseismic surveys. This is a field that has received a great deal of attention in terms of research (e.g. [2]–[6]) as well as software development such as Avizo Earth [7], GeoProbe [8] and Petrel [9]. The data in

this field is noisy and the features to be extracted have a very complex spatial structure owing to processes of buckling, folding and fracturing [1]. This makes a purely automated approach to feature extraction very difficult to achieve. The expert interpretation is very central to the definition of the features and the considerations outlined above are of critical importance [10].

The structure of the paper is as follows. In Section II we review related work. In Section III and IV we show the abstract principles of our extension of current architectures for visualization. In Section V we describe how we implement these principles. In Section VI we present the results of our case studies with some performance measure. Section VII presents conclusions and plans for future work.

II. Background and Related Work

In this section we first give a background on the concept of a visualization pipeline and some related work. Then, we give a brief background about seismic data, the case study to which our architecture is applied.

A. The Visualization Pipeline

The visualization pipeline builds the visual objects presented to the user in the form of a data processing workflow that starts from the original data right to the rendering on the display device. This basic formulation has proved very durable and has undergone extensive elaboration since its formulation for over twenty years [11]. A basic visualization pipeline features the following modules in the order of execution: reader $\rightarrow$ geometry generator $\rightarrow$ renderer. Improvements and elaborations have been proposed to address variety of issues such as visualizing multiple datasets, visualizing large datasets and enhancing performance and efficiency.

A fuller description of data by metadata enhanced the power of visualization by allowing a more full-featured view of the data taking into account its special properties and allowing users flexibility in creating visual objects. Using metadata, users can select a region or multiple regions to process, for example this allowed Ahrens et. al. to visualize large-scale datasets using parallel data streaming [12]. In addition to regional information, a time dimension can be added to metadata, adding time control to the visualization pipeline [13]. The usefulness of metadata was further developed with the introduction of query-driven visualization [14], [15]. Query-driven pipelines require the following technologies: file indexing, a query language and a metadata processing mechanism to pass queries. For fast retrieval, indexing technologies are used, such as FastBit [16], [17] (based on compressed bitmap indexing) and hashing functions. To handle massive datasets efficiently, the visualization pipeline can be executed in parallel over multiple nodes. This was well illustrated with MapReduce [18]. Database management systems (DBMS) are also capable of parallel execution of analysis but were not widely used for the purpose of visualization. A comparison between MapReduce and DBMS was presented by Pavlo et. al. [19]; the authors suggest that DBMS has a performance advantage over MapReduce while the latter is easier to setup and to use.

Despite such impressive development, the evolution of the visualization pipeline has not (to our knowledge) developed a reverse direction to directly link the users' understanding of the data back into the dataset. Our contention is that as the users are interacting with the data they are encoding their intuitive understanding of the features of the data that they are viewing. This intuition is often lost when the visualization session ends; it remains in the user's mental world. We wish to attempt to find ways to integrate this mental view of the data with the actual data. We can attempt to record these private mental views by recording the provenance of the visualization, i.e. the sequence of operations applied to the data to create the individual user's view. The first software to bring the concept of provenance into visualization was *VisTrails* [20], [21]. In VisTrails, the changes to the pipeline and parameter values are captured, allowing to review and compare previous versions. We share a similar aim in respect to maintaining data provenance, but our method differs in the type of metadata that is being captured from the user's visual exploration. Due to the nature of the data we deal with in this paper, fine details of the user's selection and manipulation of the objects being visualized (not only the parameter values selected, e.g. to define isosurfaces) are required to identify features. Thus, the extracted feature objects, as a result of user's visual interpretations, are explicitly saved. A categorization of provenance techniques were presented in surveys by Simmhan et al. [22] and Ikeda et al. [23].

In order to maintain users data provenance coherently with the data, we need to create data structures that contain both; thus the original data becomes progressively enhanced as the users visualize it. An architecture needs to be developed that can incorporate this enhancement in a scalable manner. Al-Naser et. al. [24] were first to introduce the concept of feature-aware parallel queries to a database in order to create a volume in real time ready for direct volume rendering. In this approach, features—which are classically represented by meshes—are stored as points tagged into the database; thus queries are "feature-aware". Their work was inspired by Brooke et al. [25] who discussed the importance of data locality in visualizing large datasets and exploited the (then) recently available programmable GPUs for direct rendering without the creation of geometric objects based on meshing. This definition of "feature-aware" differs from that used by Zhang and Zhao [26] in which an approximation is applied for time-varying mesh-based surfaces to generate multi-resolution animation models while preserving objects' features.

With the rapid advances in the capabilities of GPUs, direct volume rendering techniques such as *3D texture slicing* [27] and *GPU-based ray casting* [28] have become more efficient for interactive visualization on a large uniform grid. The latter was inspired by the introduction of shading languages such as OpenGL Shading Language (GLSL). We exploit such standard techniques in our architecture to support the primary claim in this paper; we plan to utilize other advanced techniques in future to deal with different data structures, e.g. unstructured spatial data.

B. Seismic Visualization in the Oil and Gas Industry

In this paper, we apply our method on seismic imaging data from the oil and gas industry. To acquire seismic data, acoustic waves are artificially generated on the earth's surface and reflect from subsurface geological layers and structures. Due to the variation of material properties, these waves are reflected back to the surface and their (1) amplitudes and (2) travel time are recorded via receivers [29]. This data is then processed to generate a 2D or 3D image illustrating the subsurface layers. A 2D seismic profile consists of multiple vertical *traces*. Each *trace* holds the recorded amplitudes sampled at, typically, every four milliseconds. Seismic imaging is then interpreted by a geoscientist to extract geological features such as horizons and faults. This interpretation potentially identifies hydrocarbon traps: oil or gas. Due to the continuous demand on hydrocarbon resources, geoscientists are seeking efficient visualization which can be centrally managed for greater collaboration and rapid decision making.

The SEG-Y format has been used by the industry to store seismic data since mid 1970s. SEG-Y structure consists of a *textual file header*, a *binary file header* and multiple trace records. Each trace record consists of a binary *trace header* and trace data containing multiple sample values; more details can be found in the SEG-Y Data Exchange Format (revision 1) [30].

Data in a SEG-Y file is stored sequentially and therefore retrieval of seismic data for a 3D visualization could negatively affect interactivity. For this reason, seismic visualization and interpretation applications, such as Petrel [9], offer an internal format which stores seismic data in multi-resolution bricks for fast access; this is based on the *Octreemizer* technique by Plate et. al. [2]. This has been a successful approach in visualizing very large seismic data. However, data management is still a challenge, mainly in managing multi-user interpretations and moving data between users and applications; this was confirmed to us through out working with geo-scientists from the oil and gas industry. Current seismic applications often use proprietary internal formats and also represent and store interpreted surfaces such as horizons and faults in separate objects.

III. Data-Centric Visualization Framework

In this section, we propose a data-centric visualization framework which stores users' interpretations back to the central database for reusability and knowledge sharing. For this purpose, we build our data structure on a parallel relational database management system (RDBMS). We call this *spatially registered data structure* (SRDS) (Section III-A). Since our data structure is stored in a relational database rather than in raw image files and geometric objects, we require an intermediate stage which builds in real time a volume in a format which can be directly rendered on the GPU. We call this a *feature-embedded spatial volume* (FESVo) (Section III-B).

A. Spatially Registered Data Structure (SRDS)

Our visual analysis method shifts from the classic static raw file system into a central data structure built on a relational database. This is to cater for the highly-structured relational modeling required by the integrated analytics paradigm of enterprise-scale business computing. This structure mainly features: (1) on-the-fly indexing using a hashing algorithm for direct access to data units and efficient update, (2) global spatial reference on all datasets (those of one type resides on a single table), (3) interpretation tagging which accumulate users' interpretations into the database and (4) concurrent access allowing parallel multi-threading queries from multiple users; this is described as follows.

Figure 1 illustrates the database schema of SRDS. Tables **SRDS_TRACE** and **SRDS** hold the datasets of seismic traces (raw data) and users' interpretations of features, respectively. The data is indexed on the combination of *(x,y)* coordinate and *source ID* (src_id). A *source ID* groups data units of one source under a unified identification (ID). The *property ID* (prop_id) field describes the type of the *property value* (prop_val) exists at an *(x,y)* location. For example, a property ID of **1** describes a *seismic trace* type data, a property ID of **2** describes a *horizon* geological feature and a property ID of **3** describes a *fault* geological feature. The *vertical distance* (z) adds a third dimension and *timestamp (ts)* allows versioning. The main difference between the two tables is the type of the *property value* (prop_val) field. The *property value* field of **SRDS_TRACE** table is of a customised binary-based type to hold trace samples; this is equivalent to 1D data. For **SRDS** table, the *property value* (prop_val) field may either represent a single measured value (e.g. porosity, permeability) or an identification to which geological body (e.g. horizon) this raw belongs; we adopt the latter at this stage. Thus, we store users' interpretations of geological features as a cloud of points; each point is in a row.

Table **SRC** serves as a source metadata table. It identifies the type of a source (e.g. raw readings, user interpretation, etc). It also determines the boundary of the dataset and

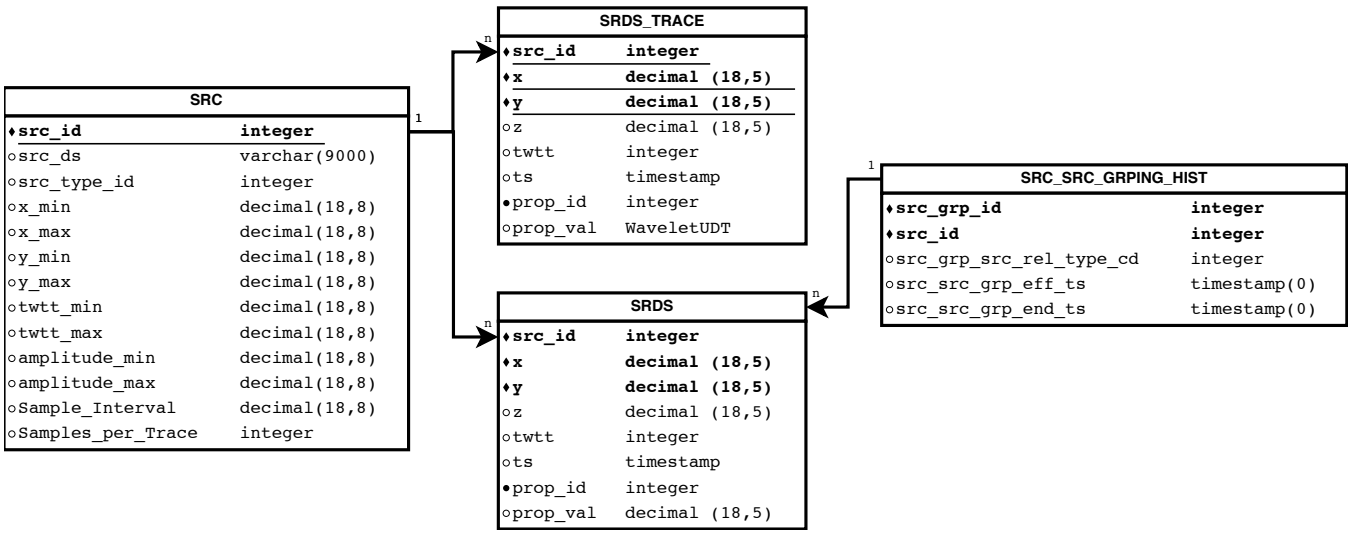

Figure 1. This diagram illustrates SRDS database schema. Abbreviations used here are as follows; **src_id:** source identification; **src_ds:** source description; **twtt:** two-way-travel time; **ts:** timestamp; **prop:** property; **rel_type_cd:** relation type code; **eff_ts:** effective timestamp; **end_ts:** end timestamp.

range of the amplitude values in the trace sample. Table **SRC_SRC_GRPING_HIST** (source-to-source grouping history) allows data provenance. The table links related interpretations and determines the relation type: e.g. *insertion* or *deletion*. The different *source ID* for each source of an interpretation and the *timestamp* field make this operation possible. In Section IV-B we discuss how we utilize the above structure allowing multi-user interpretations.

Using a hashing algorithm [31], the location of the required row can be determined through hashing functions without a construction or storage complexity. This allows retrieval and writing back from and to the database at a complexity that is proportional only to the working dataset (the size of the dataset being retrieved or written back) and not to the total size of the tables.

B. Feature-Embedded Spatial Volume (FESVo)

The indexing and parallel capabilities of the data structure (SRDS) is utilized to perform parallel queries, and an on-the-fly downsampling if a lower resolution is required, resulting in an intermediate volume which can be directly rendered on the GPU. We call this mechanism a *feature-embedded spatial volume* (FESVo). We use a standard rendering technique to visualize this volume which is the data supplied by SRDS format.

In FESVo, a loading mechanism is required to map between the different coordinate systems: (1) geographical coordinate in SRDS, (2) intermediate volume coordinate in FESVo and (3) texture coordinate in the GPU. The dimension of the intermediate volume (FESVo) is calculated based on user requests of what region to fetch and thus visualize. We explain these processes in Section IV-A.

IV. Architecture

In this section we explain how we use our framework in an architecture to visualize as well as store back users' interpretations of an exploratory spatial dataset.

As illustrated in Figure 2, the architecture links the SRDS on a database to one or more on-the-fly created FESVo through parallel feature-aware and global spatially-referenced queries which results in a parallel streaming of data units. FESVo, on the other side, is linked to a rendering engine. Users' interpretations are stored back to SRDS. In our current work, we directly store interpretations to SRDS and rebuild the intermediate local volume (FESVo) with the newly added data; in future work we can optimize this by caching users' interpretations in FESVo then later store it into SRDS.

A. Data Loading

The following processes take a place within the architecture during data loading. A rendering engine requests a ready texture buffer to be directly rendered from FESVo based on a user request of desired datasets. Inside FESVo, a *data loader* calculates FESVo's current dimension (Section IV-A1) and performs coordinate mapping (Section IV-A2) between SRDS, FESVo and the GPU texture buffer. Upon the user's request, the *data loader* works out the data units required to build the texture buffer. For each data unit, it first checks its internal cache. The data unit, if found, is placed in the texture buffer at the computed (mapped) position. If a data unit is, otherwise, not cached, the unit is added to one of a number of queues in a load balancing manner. Each queue is associated with a thread. After completing the search in the internal cache of FESVo, the threads start, each with its queue of data units (locations) to be concurrently fetched

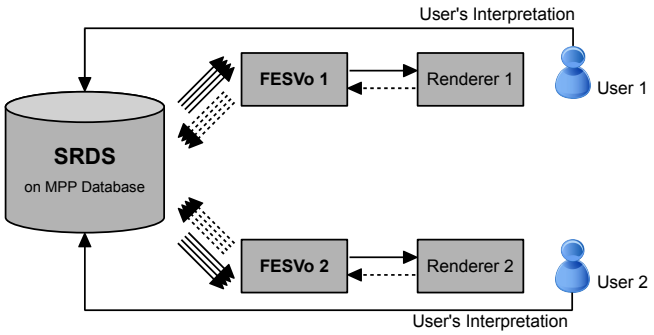

Figure 2. This is a conceptual diagram of a data-centric visual analysis architecture which consists of three loosely coupled components. The (1) spatially registered data structure (SRDS) on a centrally located database is linked to multiple on-the-fly created (2) feature-embedded spatial volume (FESVo) through parallel connections. FESVo is linked to (3) a renderer engine. Users' interpretations are directly fed back to SRDS; FESVo is then refreshed. Dashed arrows indicate SQL queries from FESVo to SRDS (0.2–0.7KB each) or requests for a texture buffer from a renderer to FESVo. Full arrows indicate data units transfer from the database to FESVo (around 4KB each), a texture buffer from FESVo to a renderer (multiple of mega bytes) or updates from users to SRDS.

from the database. Each fetched data unit is loaded to FESVo and placed in the texture buffer at the computed position. As we are using a hashing algorithm to index the dataset, data retrieval is performed at a complexity of *O(k)*, where k is the number of data units being fetched from the database; this is independent of the total size of the table (dataset).

1) Calculation of Volume Dimension: In this section, we assume that the data has been loaded in SRDS format into the database. At the launch of a visualization session, metadata of the region of interest is retrieved; a region of interest can be determined via the *source ID* in SRDS. Since datasets in SRDS are stored and retrieved through their global geographical locations, we divide the region of interest into *global cells* which have a real-world dimension. Seismic data is often regularly distributed and thus we aim to divide the region such that each *global cell* contains one spatial data unit that corresponds to one cell (voxel) in the local volume.

We calculate the dimension of the *global cells* based on the raw data; features data is neglected for this calculation. For this we use the original SEG-Y file's header (as explained in Section II-B) to indicate the dimension of the region as: *inlines* (number of traces in the X direction) $\times$ *cross-lines* (number of traces in the Y direction) $\times$ *samples per trace* (trace length). We also use the headers information to calculate the dataset width and depth by knowing the Cartesian coordinates of the four vertices that make the rectangle shape of the top view, and using the Euclidean distance between two points formula. Thus, we can find the *global cells* width and depth as follows, using the ceiling function to round up:

$$GlobalCellWidth = \lceil \frac{RegionWidth}{CrossLines} \rceil \qquad (1)$$

$$GlobalCellDepth = \lceil \frac{RegionDepth}{InLines} \rceil \qquad (2)$$

2) Coordinate Mapping and Levels-of-Detail: The architecture deals with three coordinate systems: (1) texture coordinate *(s, t, r)*—as in OpenGL, (2) local volume coordinate *(localX, localY, localZ)* per level-of-detail and (3) global geographical coordinate *(x, y, z)*.

The mapping between coordinates takes place on the top view X-Y plane. At this stage of our work, no mapping is performed on the *z-axis*; a seismic trace is fully loaded to a 1D texture location *(s, t)*. This is because the trace length of raw seismic data is, usually, fixed across one dataset and relatively small, around 200 to 2000 samples per trace; each sample is a 4-byte floating point.

As each *global cell* corresponds to one cell (voxel) in the local volume, these local cells become the highest resolution level (*LOD0*). Each subsequent level halves the dimension of its previous one; i.e. we rely on a decimation-based technique for downsampling. Since SRDS and, thus, the calculated global cells are regularly structured, a low resolution image can be obtained through direct downsampling; cells of higher levels-of-detail (lower resolution) are mapped to cells at *LOD0* based on a regular decimation. Only one resolution version (the highest) of the dataset exists and real-time mapping is performed for lower resolution levels.

3) Data Lookup: A global cell, in our seismic case, represents a subsurface dataset from a rectangular area (e.g. 12×12 meter square) in the real-world. In texture world, this is mapped to a single 1D dataset. To search inside a global cell in the database, we can choose between two modes: (1) general discovery mode and (2) specific cached mode. We maintain both modes and perform one depending on the task.

In the first mode, we have no knowledge in advance about the exact coordinates of the data units; thus, it is a discovery mode. To query the database for a dataset which lies in a global cell, we explicitly query every possible location (*x*,*y*) with a minimum step (e.g. 1 meter). The reason why explicit values of *x* and *y* are provided in the query is to perform hash-based point-to-point queries and avoid a full table scan by the database. Due to the massive size of seismic datasets and because we place all raw seismic datasets in a single table for multi-datasets access, we always attempt to avoid a full table scan which leads to a performance proportional to the table size.

In the specific mode, we pre-scan the tables for the required region and dataset source(s) and then cache all the *(x,y)* coordinates, using a sorted table. Starting with the texture coordinate we need to load, the mapped geographical coordinate is calculated: *(xInitial, yInitial)*. As all valid data coordinates are cached on the client, we can efficiently look for a point *(xTrue, yTrue)* which lies on the location of the current global cell. Having a valid and explicit coordinate, a point-to-point query is executed per required global cell.

This mode overall performs at a complexity of $O(k)$, where k is the number of data units returned; this is regardless of the table size and number of datasets in the table.

B. Multi-user Input with History Tracking

Using the structure explained in Section III-A, multiple users can interact by adding or changing others' interpretations while maintaining data provenance. For a user to insert some interpretations as an extension to another user's work, we do the following. We create a new entry in the grouping table linking the user's source ID to the source ID of the original interpretation to which the extension is applied. In this entry we insert a timestamp and the relation type of this grouping which is *insertion* in this case, since the user is inserting a new interpretation. Then, we insert the points which form the user's new interpretation into the features table with his/her user ID and the earlier timestamp inserted in the grouping table. In the case of deleting a previously created interpretation, the relation type would be *deletion* instead and we insert the points which the user wants to delete in the features table with his/her ID and the grouping timestamp. By doing so, we accumulate users' interpretations and do not physically delete but tag as deleted so users can roll back chronologically.

To retrieve a geological feature which involved multiple users in interpretation, we query the database such that we add points of an *insertion* relation type and subtract points of *deletion* relation type. Such points can be identified via the *source ID* and *timestamp*, linked to the *grouping* table. A pseudo query is presented in Section V-B. We can roll back and visualize how a feature was interpreted chronologically by controlling the *timestamp* in the query.

V. Implementation

At the current stage, referring to Figure 2, SRDS is implemented on a Teradata database virtually running on a 64-bit Windows Server 2003 and both FESVo and the renderer engine are deployed on laptop and desktop machines equipped with graphics cards of 256MB to 1GB of memory. We use standard SQL for queries and data updates.

In addition, the implementation provides functionalities including the following:

1) Selection of a seismic survey and available interpreted geological features based on user IDs.
2) Setting parameters prior to loading such as local texture size and number of parallel threads.
3) Visualization of the selected datasets using a basic texture mapping rendering technique.
4) Amendment to a horizon object collaboratively.
5) The ability to go back in history to view how a feature object was formed by different users.

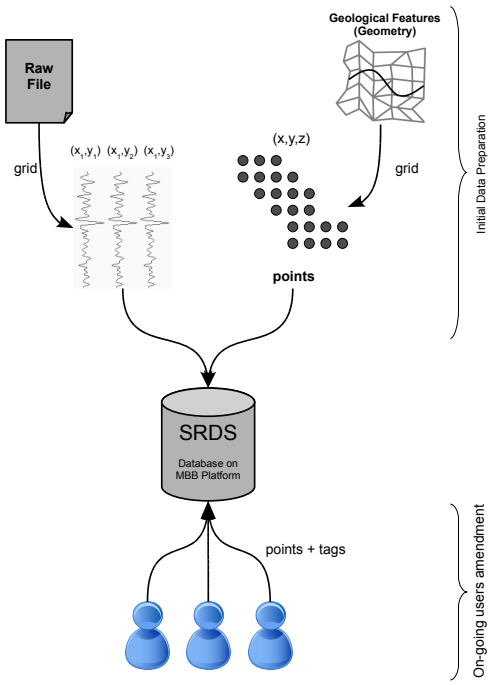

Figure 3. Seismic datasets on the database is initially prepared from SEG-Y files for raw data and geometry files for geological features. Amendment to feature objects (addition/deletion) is later updated directly from users to the database.

A. Data Input into SRDS

In our case, the data is initially prepared from SEG-Y files and geometry as illustrated in Figure 3. We start from post-stack 3D seismic conventional files (SEG-Y format) and extract *traces*, which are 1D vertical subsurface readings of amplitude values. The trace data is loaded into the database tagged with its geographical location, which is extracted from the trace header. Geological features, which were previously interpreted by users, are obtained in the form of geometry. This is converted into an *(x,y,z)* cloud of points and loaded into the database. Then, the on-going users' amendments to the features are directly stored in the same format, as a cloud of points with proper tagging.

B. Data Query

Referring to Section IV-B, we retrieve a geological feature, which was interpreted by multiple users, by performing the following pseudo query, where *ts* means *timestamp*.

```
SELECT points from SRDS
 WHERE source_id = <baseline>

UNION
 SELECT points from SRDS
 JOIN grouping_table
  ON source_id
   AND ts
```

```
    AND relation_type = INSERTION

EXCEPT
 SELECT points from SRDS
 JOIN grouping_table
  ON source_id
   AND ts
   AND relation_type = DELETION
```

In this query, the *baseline* is the original interpretation which was first imported from an external source. We control the history tracking by manipulating the *timestamp* value.

C. Rendering Engine

At this stage, we adopt a back-to-front *textured slice mapping* rendering technique [27] along with a shader program, using OpenGL Shader Language (GLSL). Two texture objects (buffers) exist at any time: one for seismic raw data (volumetric datasets) and the other one is for all geological features.

VI. Results and Case Studies

The following are case studies illustrating the proposed architecture. In these cases, we assume that the feature extraction process, which often involves human interactions with some automations, are provided from an external source. These cases were performed by geoscience literate postgraduate students and senior staff. To them, these cases are simple tasks that may take part in their interpretation work flow. We selected these tasks only for the purpose of demonstrating the functionality of our architecture, such that provenance of users' interpretations is maintained using a two-way visualization pipeline with a central relational database. In the following, we define a case then explain how technically it is achieved on our architecture.

A. Case 1: Horizon Time Shifting

In this case study, the user can adjust a horizon by shifting its two-way-travel time (TWTT). Graphically, this *time* is the z axis of an early-stage seismic imaging data; it is later converted into real depth. The process of time shifting a horizon can be done in several ways in respect to selecting where the shift is applied. In our implementation, we allow the user to select the following:

1) the horizon to which the shift is applied
2) a seed point
3) a shift value (+/-) (e.g. 50 milliseconds)
4) a diameter value to which the shifting is applied (e.g. 400 meters)

After setting these parameters, we start a *deletion* type grouping in SRDS linked to the original interpretation source ID and tagged with this user ID and a current timestamp. Then, all points lying within the selected diameter are inserted into the database in parallel threads, tagged with the user ID and the timestamp. Next, we end the *deletion* type grouping and start an *insertion* type grouping in SRDS with the same user ID but a new current timestamp. Then, all points laying within the selected diameter are inserted into the database in parallel threads, with a new *time* value calculated in respect to the original value (this calculation is performed on the database) and tagged with the user ID and the new timestamp. Then we end the grouping. Thus, user amendment is saved while the original interpretation is maintained centrally with the original dataset. These steps are illustrated in Figure 4.

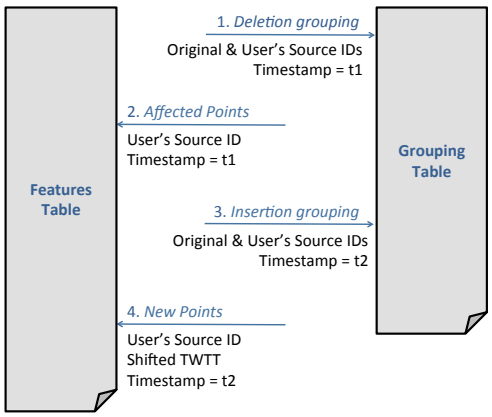

Figure 4. This figure shows the steps taken in interaction with the database for a user to shift a previously interpreted horizon.

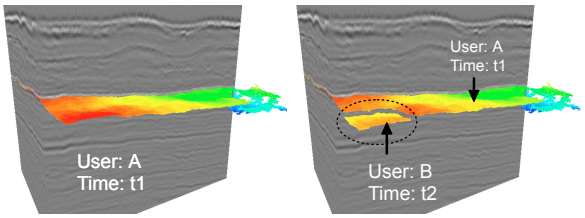

Figure 5. The left screenshot shows an interpreted horizon by user A at time t1. The right screenshot shows a partially shifted horizon by user B at time t2.

As described in Section V-B, using an SQL statement which links the features table with the grouping table, we can query the latest form of the feature; see Figure 5. At this stage, we do not cache users' amendments into FESVo. Thus, we clear FESVo and reload it.

In current seismic visualization applications, any small adjustment to a feature object means a new whole object to be saved. In our case, we just apply the process on the affected area and then build the form of the current version.

B. Case 2: Deletion of an Interpreted Object with History Tracking

In this case study, we assume that two users have added into an existing interpretation of a horizon from a particular source. A senior (more expert) user later visualized both

interpretations and decided that one is more accurate than the other and therefore wanted to delete the less accurate interpretation.

The expert user can select a session with a data *insertion* tag to delete. As in Case 1, we start a *deletion* type grouping in SRDS tagged with this user ID and a current timestamp. Then, a single update query containing the user ID and timestamp of the session to be deleted is executed. This results in re-inserting the points of this session but tagged with the expert user's ID and a timestamp of the created *deletion* type grouping. We then end this grouping.

As in Case 1, we refresh FESVo and reload the latest version of interpretations which includes the original (previously existed) version and the additional interpretation by the more accurate user; see Figure 6. In current seismic visualization applications, we would face a similar issue as pointed in Case 1.

As we record a timestamp when starting a grouping between different interpretation sources, we can go back in history to visualize earlier versions. In this case study, as well as the first one, users can get a list of previous interpretations. Upon selecting one, the user has a choice of either visualizing only the selected version or an accumulation of the previous sessions until the selected one. The timestamp of the selected version is the key of the SQL statement explained in Section V-B. In current seismic visualization applications, users extract a complete version of their interpretations (usually in files) and thus it is challenging to walk through subversions by different users.

C. Performance Measure

Our current aim is to achieve data provenance with an acceptable performance. Our tests and cases were performed on laptop and desktop machines equipped with graphics cards of 256MB to 1GB of memory. The database was running virtually on a 64-bit Windows Server 2003. The total size of the tables, which mainly include seismic imaging datasets and users' interpretations, were around 35GB.

Over a local area network (LAN), we were able to initially load the seismic traces at a level of detail with a lower resolution of one seismic volume in around 7 to 8 seconds; this level had a size of 38MB. The feature object (horizon) was loaded in around 1 to 2 second(s); both traces and the feature object forms the output is illustrated in Figures 5 or 6. The loading process used 4 threads concurrently. Each SQL command is a multi-statement request consists of a maximum of 16 point-to-point queries. Each query is a fetch request of a macro previously set up to query a single trace or a set of feature points against a unique geographical location *(x,y)*. It appeared to us that the fetching time from SRDS is less than fetching a SEGY file located remotely in a network storage; see Table I.

By experiment, we found that the number of threads to run concurrently is preferred to be multiple of the database's parallel units, known by Teradata[1] as *Access Module Processor* (AMP) [32]. The database used in our tests has four AMPs. Thus, as shows on the graph of Figure 7, the throughput becomes stable when using four threads or more. This would only be the case when performing the queries on a fast connection, as is the case with the test of Figure 7. When using a slower connection, more threads, up to a limit, would boosts the overall throughput as the number of threads overcomes the slowness in connections.

Data Source	Fetching Time
Local Drive	2.6s
NAS 1	38.4s
NAS 2	76.7s
SRDS	29.0s

Table I
THE TABLE SHOWS THE FETCHING TIME FOR A 155MB-DATASET FROM FOUR DIFFERENT SOURCES: USING A CONVENTIONAL SEGY FILE ON A LOCAL DRIVE AND NETWORK ATTACHED STORAGES ON TWO DIFFERENT LOCATIONS (NAS1 AND NAS2), AND USING OUR ARCHITECTURE WITH SRDS WHICH WAS PHYSICALLY LOCATED NEAR NAS2.

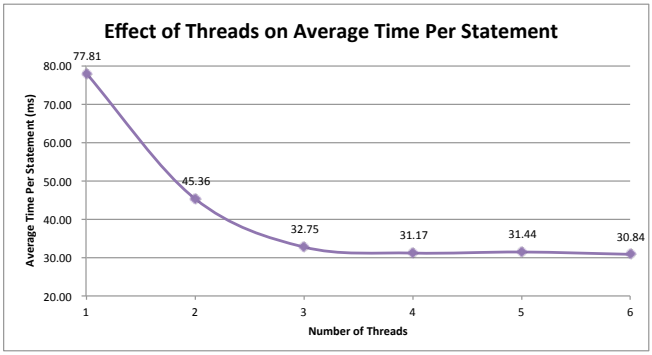

Figure 7. The number of threads to run concurrently is preferred to be multiple of the database's parallel modules. Using a high speed connection and a database with 4 parallel modules, the throughput becomes stable after 4 concurrent threads. This test was using the general query mode as explained in Section V-B.

VII. Conclusion and Future Work

In this paper we have demonstrated a proof of concept of our data-centric approach to data provenance in feature-rich visualization, applying this to seismic imaging data as a case study. Our method represents and accumulate users' interpretations of geological features as metadata and combine it with the raw seismic data into one storage. We link this to a renderer through a loading mechanism and also allow users' amendments of interpretations to flow back as new metadata to the data storage. In this paper, we have presented case studies that illustrated the system's ability of allowing users to amend others' interpretations and trace the history of amendments.

[1]Teradata: a data warehousing company (www.teradata.com)

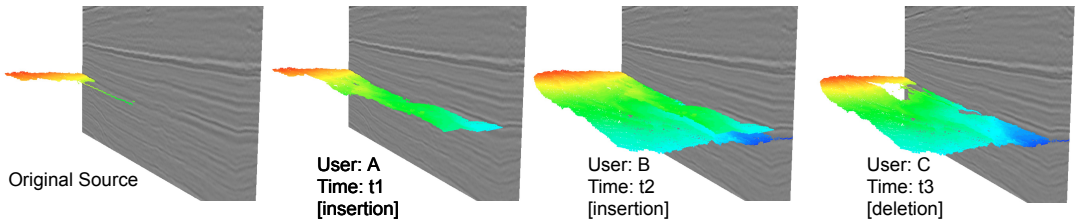

Figure 6. From left, the second screenshot shows some interpretation added to the original one (first one from left) by User A at time t1. The third screenshot shows more contribution by User B at time t2. The fourth screenshot, an expert user (User C) decided to delete the interpretation by User A due to, for example, lack of accuracy. The interpretation of User A is in fact not deleted but tagged as deleted. Users, therefore, can go back in history and visualize previous versions of interpretations.

Our plan for the future is to integrate our architecture with feature extraction methods. In the case studies of this paper, we assumed that the feature extraction process are provided from an external source. As a future work, we plan to integrate feature extraction techniques, such as the work presented by Höllt et. al. [6], into our architecture. Also, we plan to extensively evaluate our architecture with users.

In addition, it is vital to test our methods on massive datasets due to the nature and demand of the oil and gas industry. At this stage, we have tested our method on a data of around 35GB in size; this includes seismic imaging datasets and multiple users' interpretations. Our methods are currently working with a relational database that is capable of supporting four concurrent threads without performance degradation due to time slicing. As discussed in the background section, parallel databases have shown reliable performance results. Based on this, we plan to scale our implementation into massive parallel processing databases to support massive datasets at a high performance.

Finally, in this paper we have considered only seismic imaging datasets. However, our method can potentially be extended to support other types of data, such as oceanographic. We plan to explore this in future.

References

[1] E. Robein, *Seismic Imaging: A Review of the Techniques, their Principles, Merits and Limitations*. EAGE Publications bv, 2010.

[2] J. Plate, M. Tirtasana, R. Carmona, and B. Fröhlich, "Octreemizer: a hierarchical approach for interactive roaming through very large volumes," in *Data Visualisation*. Eurographics Association, 2002, pp. 53–60.

[3] L. Castanie, B. Levy, and F. Bosquet, "VolumeExplorer: Roaming Large Volumes to Couple Visualization and Data Processing for Oil and Gas Exploration," in *IEEE Visualization*, vol. im. Ieee, 2005, pp. 247–254.

[4] J. C.-R. Lin and C. Hall, "Multiple oil and gas volumetric data visualization with GPU programming," *Proceedings of SPIE*, vol. 6495, pp. 64 950U–64 950U–8, 2007.

[5] D. Patel, O. y. Sture, H. Hauser, C. Giertsen, and M. Eduard Gröller, "Knowledge-assisted visualization of seismic data," *Computers & Graphics*, vol. 33, no. 5, pp. 585–596, Oct. 2009.

[6] T. Höllt, J. Beyer, F. Gschwantner, P. Muigg, H. Doleisch, G. Heinemann, and M. Hadwiger, "Interactive seismic interpretation with piecewise global energy minimization," in *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, Hong Kong, Mar. 2011, pp. 59–66.

[7] Visualization Sciences Group, "Avizo Earth," http://www.vsg3d.com/avizo/earth.

[8] Halliburton, "GeoProbe Volume Interpretation Software," http://www.halliburton.com/ps/Default.aspx?navid=220\&pageid=842.

[9] Schlumberger, "Petrel Seismic to Simulation Software," http://www.slb.com/services/software/geo/petrel.aspx.

[10] M. Bacon, R. Simm, and T. Redshaw, *3-D Seismic Interpretation*. Cambridge University Press, 2003.

[11] K. Moreland, "A Survey of Visualization Pipelines," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 19, no. 3, pp. 367–378, Mar. 2013.

[12] J. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. Law, and M. Papka, "Large-scale data visualization using parallel data streaming," *IEEE Computer Graphics and Applications*, vol. 21, no. 4, pp. 34–41, 2001.

[13] J. Biddiscombe, B. Geveci, K. Martin, K. Moreland, and D. Thompson, "Time dependent processing in a parallel pipeline architecture." *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1376–1383, 2007.

[14] K. Stockinger, J. Shalf, K. Wu, and E. Bethel, "Query-Driven Visualization of Large Data Sets," in *Visualization, 2005. VIS 05. IEEE*. IEEE, 2005, pp. 167–174.

[15] L. J. Gosink, J. C. Anderson, E. W. Bethel, and K. I. Joy, "Query-driven visualization of time-varying adaptive mesh refinement data." *IEEE transactions on visualization and computer graphics*, vol. 14, no. 6, pp. 1715 –1722, 2008.

[16] K. Wu, "FastBit: an efficient indexing technology for accelerating data-intensive science," *Journal of Physics: Conference Series*, vol. 16, pp. 556–560, Jan. 2005.

[17] K. Wu, S. Ahern, E. W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. Geddes, J. Gu, H. Hagen, B. Hamann, W. Koegler, J. Lauret, J. Meredith, P. Messmer, E. Otoo, V. Perevoztchikov, A. Poskanzer, Prabhat, O. Rübel, A. Shoshani, A. Sim, K. Stockinger, G. Weber, and W.-M. Zhang, "FastBit: interactively searching massive data," *Journal of Physics: Conference Series*, vol. 180, p. 012053, Jul. 2009.

[18] H. Vo, J. Bronson, B. Summa, J. Comba, J. Freire, B. Howe, V. Pascucci, and C. Silva, "Parallel Visualization on Large Clusters using Map Reduce," in *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*. IEEE, 2011, pp. 81–88.

[19] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 165–178.

[20] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo, "VisTrails: Enabling Interactive Multiple-View Visualizations," in *Visualization, 2005. VIS 05. IEEE*. IEEE, 2005, pp. 135–142.

[21] C. E. Scheidegger, H. Vo, D. Koop, J. Freire, and C. T. Silva, "Querying and creating visualizations by analogy." *IEEE transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1560–1567, 2007.

[22] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *SIGMOD Rec.*, vol. 34, no. 3, pp. 31–36, Sep. 2005. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1084805.1084812

[23] R. Ikeda and J. Widom, "Data Lineage: A Survey," Stanford University, Technical Report, 2009. [Online]. Available: http://ilpubs.stanford.edu:8090/918/

[24] A. Al-Naser, M. Rasheed, J. Brooke, and D. Irving, "Enabling Visualization of Massive Datasets Through MPP Database Architecture," in *Theory and Practice of Computer Graphics*, H. Carr and I. Grimstead, Eds. Eurographics Association, 2011, pp. 109–112.

[25] J. M. Brooke, J. Marsh, S. Pettifer, and L. S. Sastry, "The importance of locality in the visualization of large datasets," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 2, pp. 195–205, Feb. 2007.

[26] S. Zhang and J. Zhao, "Feature Aware Multiresolution Animation Models Generation," *Journal of Multimedia*, vol. 5, no. 6, pp. 622–628, Dec. 2010.

[27] T. Mcreynolds and S. Hui, *Volume Visualization with Texture*. SIGGRAPH, 1997, ch. 13, pp. 144–153.

[28] M. Hadwiger, P. Ljung, C. R. Salama, and T. Ropinski, "Advanced Illumination Techniques for GPU Volume Raycasting," in *ACM SIGGRAPH Courses Program*. New York, NY, USA: ACM, 2009, pp. 1–166.

[29] C. Ma and J. Rokne, *3D Seismic Volume Visualization*. Norwell, MA, USA: Springer Netherlands, 2004, vol. VI, ch. 13, pp. 241–262.

[30] Society of Exploration Geophysicists, "SEG Y rev 1 Data Exchange Format," Tech. Rep. May, 2002. [Online]. Available: http://www.seg.org/SEGportalWEBproject/prod/SEG-Publications/Pub-Technical-Standards/Documents/seg_y_rev1.pdf

[31] S. K. Rahimi and F. S. Haug, *Query Optimization*. Wiley, 2010.

[32] C. Ballinger, "The Teradata Scalability Story," Teradata Corporation, Tech. Rep., 2009.