Computers & Geosciences 70 (2014) 44–54

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Database versioning and its implementation in geoscience information systems

Hai Ha Le [a,*], Helmut Schaeben [a], Heinrich Jasper [b], Ines Görz [a]

[a] Institute for Geophysics and Geoinformatics, TU Bergakademie Freiberg, Freiberg, Germany
[b] Institute for Mathematics and Informatics, TU Bergakademie Freiberg, Freiberg, Germany

ARTICLE INFO

Article history:
Received 12 December 2013
Received in revised form
22 May 2014
Accepted 23 May 2014
Available online 5 June 2014

Keywords:
Database version manager
Modeling time
Knowledge time
Geoscience information system
Long transaction
Database workspace manager

ABSTRACT

Many different versions of geoscience data concurrently exist in a database for different geological paradigms, source data, and authors. The aim of this study is to manage these versions in a database management system. Our data include geological surfaces, which are triangulated meshes in this study. Unlike revision/version/source control systems, our data are stored in a central database without local copies. The main contributions of this study include (1) a data model with input/output/manage functions, (2) a mesh comparison function, (3) a version merging strategy, and (4) the implementation of all of the concepts in PostgreSQL and gOcad. The software has been tested using synthetic surfaces and a simple tectonic model of a deformed stratigraphic horizon.

1. Introduction

Following the success of geographic information systems (GIS), geoscience information systems (GSIS) have received attention in recent years. GSIS manage geological objects with complex geometries in at least 3-dimensional Euclidean space, mainly in the subsurface. In many nations, the subsurface is considered to be an economic resource and is managed and protected by state authorities. These authorities, mainly the Geological Surveys, use 3D geological models as a basis for their decisions. Subsurface knowledge, however, is always incomplete because geological data are only available along specific lines or surfaces such as drilling paths or seismic sections, respectively. To cope with new data from experiments, such as new wells, the models are frequently updated. Information deduced from these models is included in mine operation plans, ground water production plans, and environmental sustainability assessments. Not all decisions, however, are correct, e.g., a mining operation may cause unexpected destruction or pollution. If an insurance event occurs, one has to check whether the complication could have been predicted using available knowledge and models from the data at the time of the decision. Models can be revised when new data become available and can often predict if the original plan is too dangerous. Therefore, model versioning combined with database querying are economically relevant for subsurface management. A version management system is, thus, necessary for any GSIS.

Almost all data collection/modification processes in geosciences includes specific workflows. Each workflow involves collections of actions; some executed by computers alone, some involving human interaction with computers, and some only needing human action. The data created should be shared among a collaborative working group, but the data should not be published or shared with other groups until the process is completed. In this case, database versioning can be used to isolate groups of changes by creating a version of each workflow. Database versioning can also support the so-called “what-if” analyses by creating a new version, changing data according to artificial scenarios, using the new version for an analysis, and, when complete, remove the new version. GSIS can use database versioning to prevent “lost-update” phenomena, i.e., using obsolete data to update the data themselves. Database versioning creates a template version for each check-out action. When check-in action occurs, the database merges the template version into the main database and removes the template version.

Database versioning was studied under various names, such as long transactions in the ArcGIS product family (ESRI, 2013) and the database workspace manager in the Oracle product (Oracle, 2012, 2013). Both ArcGIS and Oracle database workspace manager are commercial products and only work with 2-dimensional objects and scalar data.

* Corresponding author. Tel.: +49 1573 8686588.
E-mail addresses: lehai@mailserver.tu-freiberg.de (H.H. Le), schaeben@tu-freiberg.de (H. Schaeben), jasper@informatik.tu-freiberg.de (H. Jasper), IGo@geo.tu-freiberg.de (I. Görz).

http://dx.doi.org/10.1016/j.cageo.2014.05.011
0098-3004/

In recent years, source code control systems have been beneficial for software developers. These systems manage (or control) all versions of a source code file in a software project. Versions are definite states of a source code which manifest important states of the development of the software. Version control systems are used to distinguish new levels of development from older ones and are characterized by unique version numbers. On a fine-grained level, revisions are used to keep track of incrementally different states of digital information. Each version can be thought of as a set of revisions of the source code file updated by a person. These systems are also known by other names, including revision control systems and version control systems. Well-known implementations are products such as CVS (2013), Git (2013), SVN (2013). Source code control systems work with text format files. They use a central storage, mainly a server computer, and many local copies, mainly in the directory of each programmer's computer. They provide an update and a commit function to synchronize between central storage and local copies. Two main functions of a source code control system are comparing and merging two text files. These systems do not work with databases or data structures representing geological objects.

Data types tailored for geologic applications need to describe the geometry and the properties of geologic objects. One widely-used data structure is the triangulated mesh, which represents geologic surfaces, for example, the boundaries of geologic objects. Triangulated mesh is a discrete data model using a vector-based representation of a surface created using irregularly distributed nodes with 3D (*XYZ*) coordinates. The nodes are arranged in a network of non-overlapping triangles, which create one surface object. Because the triangulated mesh is flexible in mesh resolution and adaption to the geologic data, it is used to subdivide a modeling domain into parts representing geologic bodies. It then forms the basis for body and property modeling. Because the triangulated mesh is the most prominent data structure for 3D geologic modeling, we will concentrate on this data structure.

This paper presents a data model combined with functions to manage versioned data in a database system. We present algorithms to compare and merge two triangulated meshes. The method is implemented in PostgreSQL (2013) and a gOcad plug-in as a client module (GOCAD, 2013).

In the next section, we review related work. The algorithms are described in Section 3. In Section 4, the implementation and the testing of the software are presented. Finally, Section 5 is devoted to the discussion and conclusions.

2. Related work

GSIS and 3D GIS have received a lot of attention from researchers (Apel, 2004; Breunig and Zlatanova, 2011; Gabriel et al., 2012; GST, 2013; Le, 2013; Le et al., 2013). These studies did not address version and revision management. The ArcGIS product family is one of the most famous commercial GIS systems. ArcGIS Server (including ArcSDE) enables end-users to manage enterprise geodatabases in database management systems, such as Oracle, DB2, and PostgreSQL, and to perform many useful server-side functions. Versioned data are managed in ArcGIS Server through closed-code ArcSDE, whose detailed design and algorithms are not published. In addition to GIS, the Oracle database workspace manager enables application developers and database administrators to manage multiple versions of data in the same database. Versions of table row values are grouped in different workspaces. End-users are permitted to create new versions of data while maintaining a copy of the old data. Catalog data and functions are placed in the DBMS_WM package with a closed code. Oracle's approach uses the view mechanism and triggers in the database to make versioning transparent to the user of the "production data" (Oracle, 2013). Our solution is tailored to geoscience applications where researchers can compare versions, e.g., in scientific studies. We use the more familiar terms revision and version instead of the terms workspace and version as used in the Oracle database workspace manager.

Im et al. (2012) proposed a framework to manage versions of RDF (Resource Description Framework) data in relational databases. This framework stores the original version and the deltas between two consecutive versions. To improve the performance of queries on a version, some sequences of deltas are aggregated and redundantly stored and are called Aggregated Deltas. This framework manages versions of a specific data structure, i.e., RDF triples, and a version cannot be further changed when its child version exists. Our method manages arbitrary tables, and versions can be changed after its child version was created.

Concurrent Versions System (CVS), Subversion (SVN), and source code management (Git) are versioned document management systems, in which documents are mainly text files. The common framework is server-based versioned file management combined with local concurrent copies. The most important functions are comparing two documents and merging (submitted) versions. Our method follows these systems by comparing two geological objects and merging versions of data.

Our comparison of two geological objects includes the mesh comparison, which can be performed by a nearest neighbor search, range search, and point location. Our algorithm is simpler because it is defined in 3-dimensional Euclidean space (low dimensionality) using Euclidean distance, and we only need to find a coincident vertex. Hundreds of publications from researchers in a number of fields are related to nearest neighbor searching, range searching and point location issues (Clarkson, 2006; Dhanabal and Chandramathi, 2011). In general, a searching algorithm includes two phases: preprocessing and querying. In the preprocessing phase, some data structures are used to accelerate the query phase. Famous among these data structures are, for example, octree, *k–d* tree, AABB tree, locality-sensitive hashing and many modifications of these (Bentley, 1975; CGAL, 2013; Datar et al., 2004; Liaw et al., 2010; Samet, 1994; Zatloukal et al., 2002). Until now, to the best of our knowledge, for a point set in 3-dimensional Euclidean space and Euclidean distance, the best solutions have, on average, the complexity of $O(\log(n))$ querying time (for each query), while preprocessing time and space complexities are $O(n\ \log(n))$. Our algorithm, which uses the *k–d* tree, queries in time $O(\log(n))$ in the worst case (for each query), preprocesses in time $O(n\ \log(n))$, and uses space $O(n)$.

3. Methodology

3.1. Terminology

In this paper, we use terms that are defined as follows.

A *version* (or *database version*) is a user-defined name in the database to define a working area. Versions in a database are organized as a tree, where the root of the tree is the predefined version, named "DEFAULT", and all other versions are its descendants. Working on a version by an end-user/application does not disturb any others' versions (isolation). Each database working session of an end-user/application is on a specific version, named *working version* of this session.

A *revision* is a group of changes. It is determined by an increasing integer number. We will use term revision as an integer and a group of changes. End-users/applications must not necessarily know about revisions. A revision can, however, be assigned to a name to manage a snapshot by end-users/applications. A version associates with some revisions, but a revision associates with only one version. The *revision*