Operation-based revision control for geospatial data sets

Máté Cserép, Roberto Giachetta

Eötvös Loránd University

Abstract

Revision control is useful and sometimes essential tool in the maintenance of changing datasets, and the need for version control emerges in several data centric environments, such as Geographic Information Systems (GIS). However, current revision control software focus primarily on textual information as no method is available for the efficient storage of modifications on any binary content. Beside this issue of space efficiency, retrieving semantic information regarding the alterations applied between selected revisions is also an unsolved problem for most cases. In this paper we study the application of operation-based revision control on geospatial data, specifically vector features and remotely sensed raster imagery. We present an approach relying on a unified data model and operation metadata as modification deltas. The method is independent of storage solution, thus it is applicable for both files and databases, while also persisting the required semantic information about the changes. Our evaluation shows that operation-based revision control handles geospatial data more efficiently than current state of the art revision control software, including systems created especially for maintaining geospatial data.

Keywords

revision control systems, geographic information systems, remote sensing, distributed systems

1 Introduction

The evolution and spreading of data capturing methods ranging from simple GPS devices (Wang, et al., 2010) to large scale imaging equipment (including very high resolution and hyperspectral cameras, LIDAR, etc.) (Zhang, 2010) resulted in an exponential growth in the amount of spatial data maintained by companies and organizations for which cloud computing technologies and distributed systems must be utilized (Yang, et al., 2010). Also, usually multiple versions of the same data exist due to additional modification. Hence, to manage data efficiently a spatial revision control system is required which fits to the requirements raised by Geospatial Information Systems (GIS).

Revision control systems are widely used tools in software development primarily aimed on managing versions of software source code during implementation (Ruparelia, 2010). Due to source code being written in plain text format, these systems are generally usable for any text document. There

are also solutions available for text based storage of complex objects (Firmenich, et al., 2005). However these systems are not optimized for storing versions of binary data, as its format is too general for tracking changes within the binary code. Some revision control systems – for instance, Git[1] – offer limited support for binary content (Dobŏs & Steed, 2012).

In GIS several file formats exist for storing geospatial information, most formats being binary. Only a handful of data formats have textual representation, including Well-known Text (WKT), GML and GeoJSON. Although some new solutions have emerged in the revision control of geospatial data including GitSpatial[2] and GeoGig[3] (formerly GeoGit), these systems are rather concentrating on specific areas. No universal solution has been suggested with respect to all kinds of spatial data and possible operations performed on it.

Treating spatial information as binary data produces several major drawbacks compared to textual revision control. Firstly, it erases all semantic information about the applied changes, like the executed spatial operations. Secondly, even in cases when only a smaller fragment of the given geospatial dataset was modified, a single editing operation could result in the change of a larger fraction in the binary data, needlessly increasing the required storage.

The demand for revision control systems in GIS software capable to exploit the peculiarities of spatial operations arose as early as the beginning of the '90s (Easterfield, et al., 1990). Alongside the several attempts made to utilize the concept of version control in standalone GIS solutions (Wachowicz & Healey, 1994), the idea of DBMS[4]-integrated systems (Newell, et al., 1992) (Peuquet & Duan, 1995) were also researched in cooperation with the database community. However, no sufficient solution was implemented in contemporary GIS software to satisfy the previously defined requisitions of the storage space compactness and the persistence of semantic information.

In this article, we present a solution for the efficient management of geospatial data using operation-based revision control. The model is implemented as part of the *AEGIS geospatial framework* (Giachetta, 2014), a generic library for geographic and remote sensing data processing.

2 Revision control models and methods

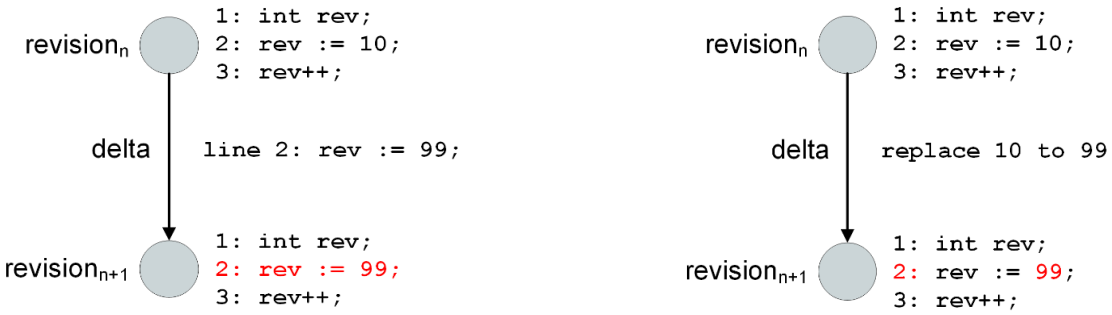
Preserving a full copy of every version of a document can easily take up a significant amount of storage space even with a relatively short version history. As a commonly used solution, most modern revision control tools spare storage space by computing and persisting only the difference (*delta*) between succeeding revisions, and only preserving the full state of a few special versions – like initial or final (*head*) revisions.

The two main categories for producing the changeset between versions are *state-based* and *operation-based* deltas. An example for both models in case of

1 http://git-scm.com
2 http://gitspatial.com
3 http://geogig.org
4 Database management system

text documents is presented in Figure 1.

In the state-based case the general method is to decompose the document into smaller, more easily manageable parts. This is due to comparing two versions of a document as a whole would induce several difficulties when creating, applying or merging deltas. For textual documents an often applied practice is to dissolve a file into lines, ignoring the special structure (syntax and semantics) of the content. Compared to this unstructured data processing, another approach is to – partially – statically analyze the document and decompose it in a structured or semi-structured way. This latter method reduces the occurrence of possible unresolvable merge conflicts (Apel, et al., 2011), however also tailors the manageable content of the revision control system to a pre-defined format. While these techniques provide a poor storage efficiency for unstructured binary data, they can be used for tracking changes in textual geospatial information (like GitSpatial handles GeoJSON), and structured models are also feasible for specific binary file types (like GeoGig parses shapefiles).



(a) State-based *(b) Operation-based*

Figure 1 - Example of revision control models

In contrast, operation-based deltas store the actually performed operations between two succeeding versions in the revision control system, instead of the alterations in the state. While this approach also has benefits in managing textual data (Shen & Sun, 2001), it is notable for this concept to not necessarily requiring any special knowledge about the managed data format from the revision control system to create or merge changesets. As a utilization in the graphical field Chen, et al. (2011) presented a revision control system using the operation-based model for the image editing application GIMP. The core idea is the storage of graphical operations using a directed acyclic graph (*DAG*), where each vertex contains a performed editing operation. This representation is suitable for nonlinear revision control and does not require storing the state of the modified image.

Different applications of revision control may require different models. Operation-based revision control is favorable over the state-based when the following conditions are met.

- The amount of data affected by a single modification is generally large. In this case state-based deltas require large storage space, whilst the size of operation-based deltas is independent from the size of the affected

data. For example, an image operation such as histogram equalization can modify all pixels of the image, requiring the entire image to be stored again in case of the state-based model.

- The data is heterogeneous, and cannot be divided into small parts, where the modification of the individual parts can be monitored. The state-based model is mostly effective when small fragments of data are identified as location of the modification.
- The query of former revisions is performed infrequently. Frequent queries would cause performance issues, as operations need to be re-executed to present the individual versions.
- Both data and operations can be managed through a single, high level model. This is required to enable the uniform handling and storage of the operation-deltas, and the proper identification of information stored within binary content.

Due to these reasons, revision control of geospatial data is more favorable for the operation-based model. The data is heterogeneous, stored as binary content, and operations usually modify large parts of the dataset. Previous revisions are not required often (as in case of software source code), but still, revision history needs to be maintained and looked up frequently.

An example for geospatial workflow consisting of four operations can be seen in Figure 2. First, the remotely sensed image is registered for the specified reference system. Second, the image is enhanced by using histogram equalization. Third, selection of the area of interest is defined by intersection using a specified input polygon. Finally, for classification, thresholding is applied to the area.

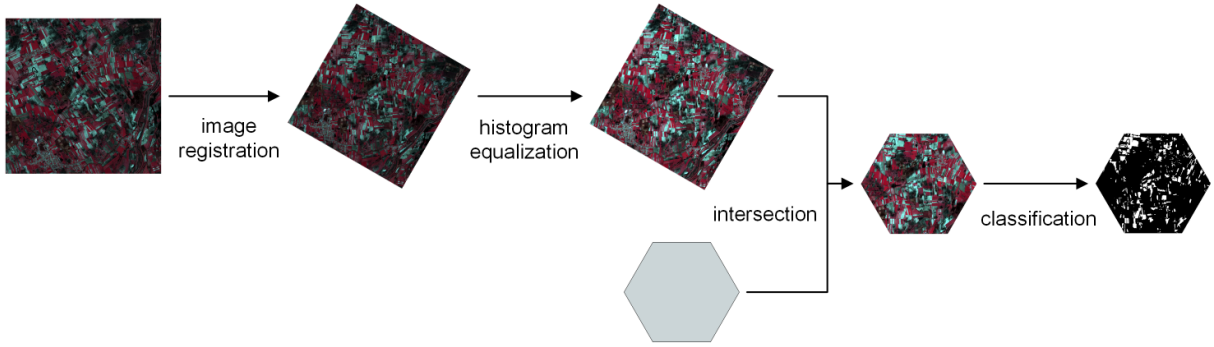


Figure 2 - Example workflow with 4 operations

3 Operation-based revision control model on geospatial data

The core concept of our solution was inspired by the idea described in (Chen et al., 2011). To create an expansively applicable model, the representation of geospatial data and operations were designed by raising the least possible requirements against them.

3.1 The baseline model

The data model purely assumes that all geospatial data types in the system

directly or indirectly inherit or implement a common ancestor class or interface. A well-known compliant example for the before mentioned minimalist specification is the *Simple Feature Access* (SFA) standard[5] (Herring, 2010) by the Open Geospatial Consortium (OGC)[6]. The SFA is an object-relational mapping for geospatial data, providing an abstract *geometry* as a common base for spatial objects, including collections. The standard deals only with the handling of vector data, but can be easily extended to support other features. For example, support for raster imagery can be introduced by specializing polygons to support raster data (denoted *raster polygons*, see Figure 3). By using this abstraction the source of data is irrelevant in terms of the revision control system.

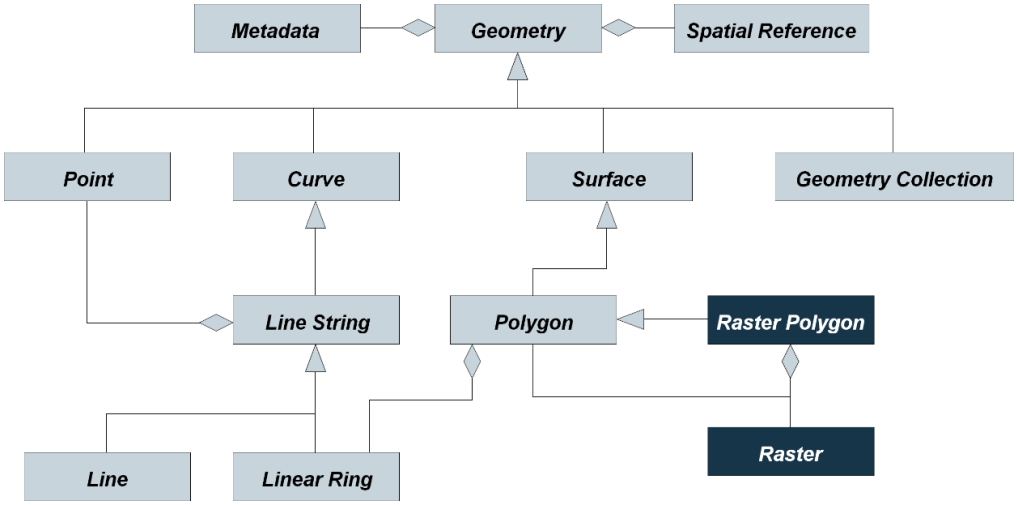


Figure 3 - The extended Simple Feature Access data model (UML notation)

Geospatial operations can be defined as a mapping between geometry objects, and can be represented as transformation objects. The objects can be described by the applied method and the values of arguments (if any). This concept is introduced in the OGC *Spatial Referencing by Coordinates* (SRC) standard (Cooper, 2010) for coordinate transformations, but can be easily generalized for all kind of geospatial operations, including raster image processing. Using this approach, only the descriptors of the operation are required to be stored as operation-deltas, which is compact data, and independent of the size of the actual transformation object.

For example, in case of the workflow presented in Section 2, histogram equalization and intersection only need the source geometries ($geom_i$) and no additional arguments. Thresholding requires the predefined threshold (th), whilst registration requires the spatial reference system identifier ($SRID$) and the geographic coordinates of the image corners ($coord_i$) in addition to the source geometries (image and intersection polygon). An operation-based revision control system does not require any specific knowledge about the previously executed spatial transformations (as described in Section 2), therefore no further requisite is necessary towards the model of operations.

5 Also an ISO standard: ISO/IEC 19125:2004
6 `http://www.opengeospatial.org`

op_1 = (registration, $geom_1$, $SRID$, $coord_1$, …, $coord_4$)
op_2 = (hist. equalization, $geom_1$)
op_3 = (intersection, $geom_1$, $geom_2$)
op_4 = (thresholding, $geom_1$, th)

As the core structure for storing the version history the *directed acyclic revision graph* is used. In this representation each vertex symbolizes the corresponding version and contains the ordered sequence of operations performed between the current and the predecessor revision, while edges denote the semantic relationships in the version system. A possible graph for the example workflow is presented in Figure 4 with rectangles denoting geometry storage and circles denoting operation-delta storage. The initial revision contains the source geometries, whilst further revisions are computed using the specified operation.

$$revision_1 = (image, polygon)$$
$$\forall i \in 2...5: revision_i = (revision_{i-1}, op_{i-1})$$

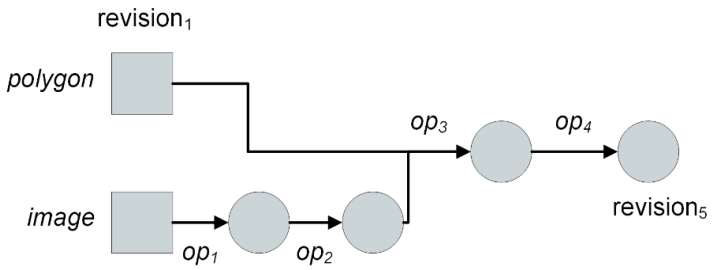


Figure 4 - Revision graph of the example workflow

3.2 Data storage

The high-level data and operation model enables the usage of all SFA compliant storage methods as source. This includes vector file formats, such as Shapefiles, GML or GeoJSON, as they can be simply converted to geometry format. The extension for handling of raster data also enables image file formats, such as GeoTIFF. Furthermore, other storage mechanisms can also be used, such as spatial databases. Thus the model is not only applicable for file based storage of revisions, but can be used on any spatial data source.

As stated in Section 3.1, operation-deltas are stored using descriptors, which can be serialized to a compact text or binary representation. Even more, the descriptors can be aggregated to the geometries as attributes. As most spatial formats enable the storage of attributes beside the geometry, these formats allow straightforward usage for storing the revision history of a geometry. For example, Shapefile stores descriptive information within a dBase file. Although specialized software is required for deserialization of the revision history, the stored geometries are readable by most geospatial software.

3.3 Centralized revision control

Our revision control system applies a natural, object-oriented approach for representation, where versions are embodied by *revision descriptor* objects. These descriptors contain meta information about the versions (e.g. the

identifier and the predecessor revision), while the changesets encompass the executed editing operations are stored and can be obtained independently for each version. This representation architecturally separates the attributes and the deltas, accelerating several revision management algorithms where the latter information is irrelevant.

The states of the spatial data managed by the revision control system may be interpreted as sets of geometrical objects. Deltas between any two versions can contain the addition of new or the deletion of existing geometry objects from the state set beside the transformation operations needed to be applied.

Using the above described revision model in linear version control systems, it becomes a straightforward task to construct the cumulative changeset between any two revisions. However, to update the working copy of a given user to a former version, the system has to revert the performed editing operations in a reversed order on the actual state. While the inverse of state-based deltas are easily constructible, operations can be irreversible. In such cases it is the task of the revision management system to support alternative approaches handling the problem. Multiple possible fallback mechanisms exist to overcome this issue, like recreating the selected revision from the other end of the version history or exceptionally storing the pre-transformed state of the affected geometry object in the original revision changeset in case of irreversible operations.

In an operation-based revision control system like the designed model, it was already shown that only the initial states of the geometries are needed to be stored, further modifications are represented with transformation objects. For efficiency reasons, fittingly selected revisions can also be snapshoted, which store the entire geometry set, so it can be retrieved without the necessity of reapplying operations. This feature makes available some balancing options in the storage space against time efficiency issues. A typical application of this method is to create a snapshot of the head revision of the main – or every – branch in a version management system, and store reverse directed inverse deltas in the changesets. In such models the fresh – significantly more often checked out – revisions can be obtained with an outstanding performance, therefore also many existing revision control tools (e.g. Subversion[7]) already use a similar methodology.

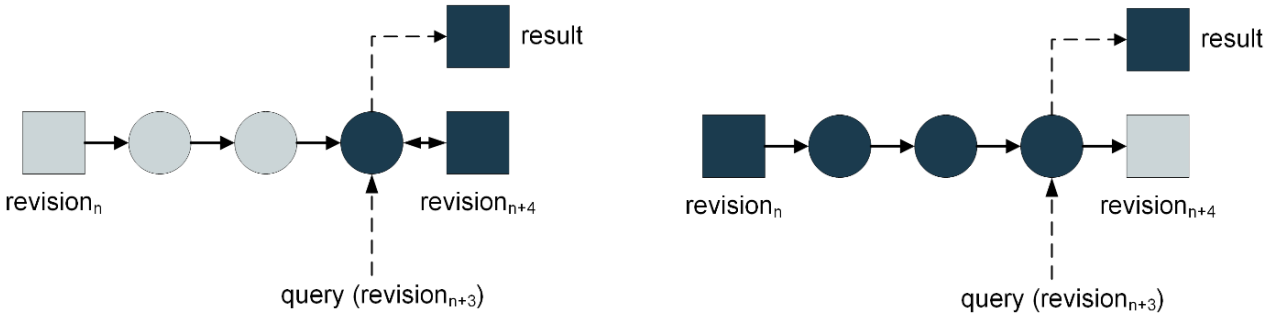
An example query is presented in Figure 5. Assuming that $revision_n$ is stored as snapshot, $revision_{n+4}$ is the head revision, and additional revisions using operation deltas, the query method for $revision_{n+3}$ depends on the reversibility of the final operation. If the operation is reversible, the revision can be reconstructed using a reverse delta from the head revision (Figure 5a). Otherwise, the revision is reconstructed from $revision_n$ by execution of multiple operations (Figure 5b).

This model also complies with the nonlinear revision control paradigm (branching and merging) by enabling multiple revision descriptors to reference a common predecessor. With this extension, branching itself becomes a simple task. However, there are multiple issues left with merging. For example, the

7 http://subversion.tigris.org

aggregated changesets might contain not only a single, but multiple editing operations per branches for a geometry object – the atomic unit in the model. Therefore the corresponding order of the alterations must be defined to resolve the conflict, for which task the involvement of the user is generally unavoidable.

The support of merging also requires some slight modifications of the revision descriptor objects. As a descriptor declares a single other version as its predecessor (and the changeset defines the modifications between these two revisions), the information about which other version(s) were merged into the current revision is lost. Therefore the identifier of all the merged versions shall be stored in every revision descriptor as this meta information might be important or helpful to the human users.



(a) Query using reverse delta *(b) Query using forward deltas*

Figure 5 - Revision query in the operation-based model

3.4 Distributed revision control

As cloud computing and distributed data storage became a generally widespread and popular research topic in recent times, most third generation version control tools (O'Sullivan, 2009) – like Git or Mercurial[8] – also abandoned the centralized paradigm to implement distributed data management and revision control instead. This concept ensures a higher availability of service and better distribution of processing and load (Reuter, et al., 1996).

The presented revision control model also supports this paradigm. A higher level software layer can be created above the model, which is responsible for the network communication and cooperation between repositories. The synchronization of the individual repositories can be performed by one of the following methods.

- *Data based synchronization* of deltas, snapshots and head revision. As in the state-based model, this method requires transferring large loads of data, and replacement of remote content to the new content. Update of the remote repository can be performed quickly.
- *Operation based synchronization*. This method requires only the transfer of operation-deltas, thus is more applicable in low bandwidth environments. Based on the operation deltas, the remote repository can rebuild any snapshot and the head revision. Therefore this method requires additional execution of operations.

8 http://mercurial.selenic.com

The benefit of this approach is that the rebuilding of revisions is not needed to be performed immediately. The construction can be delayed until the next query of the repository.

The method of synchronization is not required to be determined beforehand. Both methods can be supported by the system, and based on current environmental conditions, the appropriate method or even a hybrid approach (e.g. transfer of the head revision, but reconstruction of the snapshots) may be chosen automatically by the system with each synchronization.

Special forms of repositories can also be formed, which are only used as remote sources, such as the “bare” repository in case of Git. When using operation based synchronization, no rebuilding is required, as the revisions are reconstructed anyway by the client repository. This approach saves both storage space and execution time.

3.5 Implementation considerations

To prove the usability of the presented revision control model the implementation was carried out as part of the AEGIS geospatial framework as the system satisfies all requirements described in Section 3.1, including a general, abstract data model based on SFA and operation support which provides operation metadata (Giachetta, 2015). The implementation of both the AEGIS and the revision control system was carried out using the *.NET/Mono Frameworks* to utilize the possibilities and the simple usage of this object-oriented development platform.

As it was stated, the physical storage of the version history is independent from the revision control model. The various implementations of the storage component do not contain any specific knowledge about the inner architecture of the revision system, but they are solely responsible for storing and returning the requested data regarding a particular physical storage (e.g. a file system or a spatial database). Despite the specific inner data representation of the system, the sample implementation of the revision control model presented in the current section is easily connectible via the AEGIS framework, allowing the usage of widespread geospatial formats and services.

4 Experimental results

The practical applicability of the designed and implemented version control framework can be measured through comparing the developed software with existing revision control tools in the research field. In the interest of quantifying the result of the comparison, storage efficiency, the most significant attribute of revision management systems was chosen and examined in Section 4.1.

As the version history for a project increases in size, the recreation of an arbitrary revision with a satisfactory performance in speed could meet obstacles, as the cause of this issue and possible solutions were discussed in Section 2 and Section 3.3. In Section 4.2 we measure and analyze this other substantial aspect of our version control tool for different scenarios.

4.1 Storage efficiency

Three existing version management tools were selected as the basis of the benchmark: Subversion, Git and GeoGig. The former two software are probably the most used revision control tools in the world (O'Sullivan, 2009) for general purposes, while GeoGig is the only other known and formerly[9] actively developed software specifically for tracking changes in (binary) spatial data.

The tested data format was the Shapefile, as it is a widely used binary storage format for vector-graphic information supported by all four compared software. This type of binary data is usually handled by general delta producing algorithms in most classical revision control tools.

Most version control tools used in production – like Subversion and Git – compute not only the deltas instead of storing the full snapshots of the data, they also apply lossless compression to further reduce the required storage space. In order to avoid any remarkable distortion in the results, a similar method[10] had been implemented in the prototype tool created with AEGIS. Since Git uses a l*oose object format* (Chacon, 2009), the storage space allocated by Git was determined after the repacking of the objects was enforced manually into *packfiles*.

Two test were carried out to measure storage efficiency. In the first one a repository of all four revision control tools were populated with 1.000 revisions of randomly generated subsequent states of a single Shapefile, while the editing operations evaluated between the versions were also provided for our prototype operation-based system. The experimental results of this test are shown in Figure 6a, where for reference purposes the storage space requirement without any revision control tool applied was also indicated, exceeding 33 gigabytes. The outcome of the analysis clearly shows that while Subversion and Git required about 3-5 percent of the unversioned storage space, the AEGIS prototype demanded a much more limited amount of space, around 0.2 percent. The results of the GeoGig testing might be a surprise, hence its repository occupied more than 10 percent of the storage space of the full snapshots of all versions altogether. The explanation for this result might be the loose, superfluous and redundant inner data representation of GeoGig. This unfortunate aspect of the software was significantly improved compared to previous versions, however the tool is still in a beta phase and the progress of development slowed down recently.

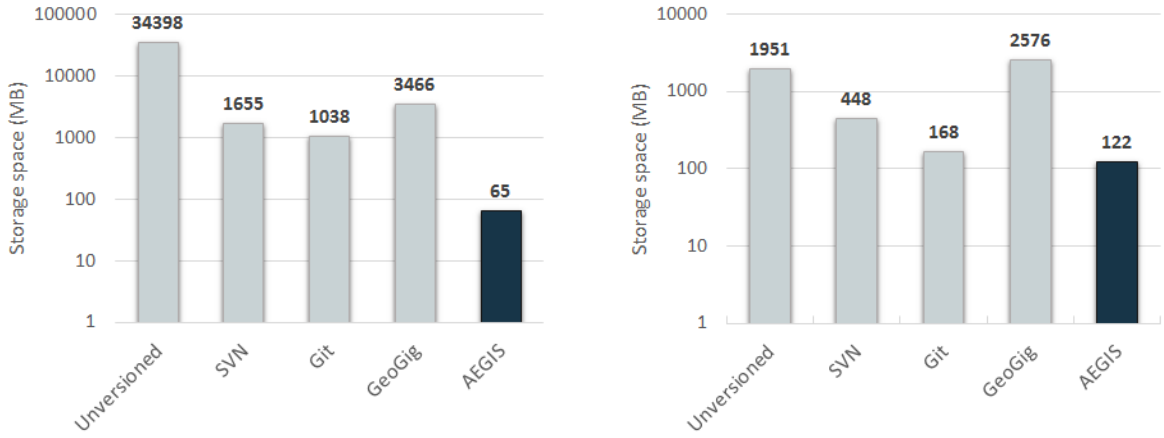
In order to evaluate the storage requirement efficiency in a real-world scenario, the second test case was based on the available history of the OpenStreetMap datasets about Hungary in Shapefile format[11]. Altogether 11 revisions were used covering over a year of timespan. The evaluation of the second test case is displayed on Figure 6b, revealing similar results to the previous test on generated data. While the gathered and stored semantic information regarding

9 The development of GeoGig was unfortunately suspended a few months ago.

10 As both SVN and Git uses the *DEFLATE* procedure for compression purposes, we used the *LZMA*, since both are based based on the algorithm described by Lempel and Ziv (1977).

11 The shapefiles were downloaded from the publicly accessible history available on the www.geofabrik.de website.

the editions is a secure benefit of the operation-based model, the vantage of AEGIS in space requirement is evidently relaxed in this case. The reason is that only a few revisions were analyzed in quantity and a significant part of the modifications were additions of new data as the OpenStreetMap dataset about Hungary increased by over 60% in size over the last year.



(a) Generated dataset *(b) OpenStreetMap dataset*

Figure 6 - Storage efficiency comparison of revision control tools

This section proved that an operation-based approach can not only preserve crucial semantic information about the editing operations, but even a prototype level implementation of a geospatial revision control system can comply a better performance in storage space requirement than the general binary delta algorithms applied by the version management systems in production.

4.2 Computation performance

Alongside the advantages emphasized at the end of the previous section, the potential drawbacks regarding the computational efficiency of the operation-based architecture must also be examined, because in this model the changesets contain sequences of transformations instead of state modifications. Since the evaluation of certain spatial operations can be quite time consuming and these transformations have to be executed each time when recreating a revision from the state of another one, it might cause a remarkable effect on the performance of such a system. The problem can be diminished with proper implementation decisions, for instance storing the head state of the branches and persisting reverse deltas in the changesets ensures that the fresh revisions are always easily producible by the framework. Moreover, automatically saving snapshots by a declared heuristic can also guarantee a maximum limit of revision recreation time at a cost on storage space. However, as stated in Section 3.3, not all operations are reversible.

As a practical demonstration of our before mentioned theory we created three repositories with our prototype tool, containing the same spatial version history of a few hundreds revisions. The changesets attached to each version primarily consisted of time expensive transformation operations. The three repositories applied different models related to the changesets: the first one used forward

deltas, the second one applied reverse deltas and the last one also worked with reverse deltas, but persisted a snapshot of the actual state at every 20th revision on each branch. Figure 7 shows the recorded average checkout times for the initial, the head and randomized revisions in the repositories, presenting the performance advantage of reverse deltas with snapshots applied[12].

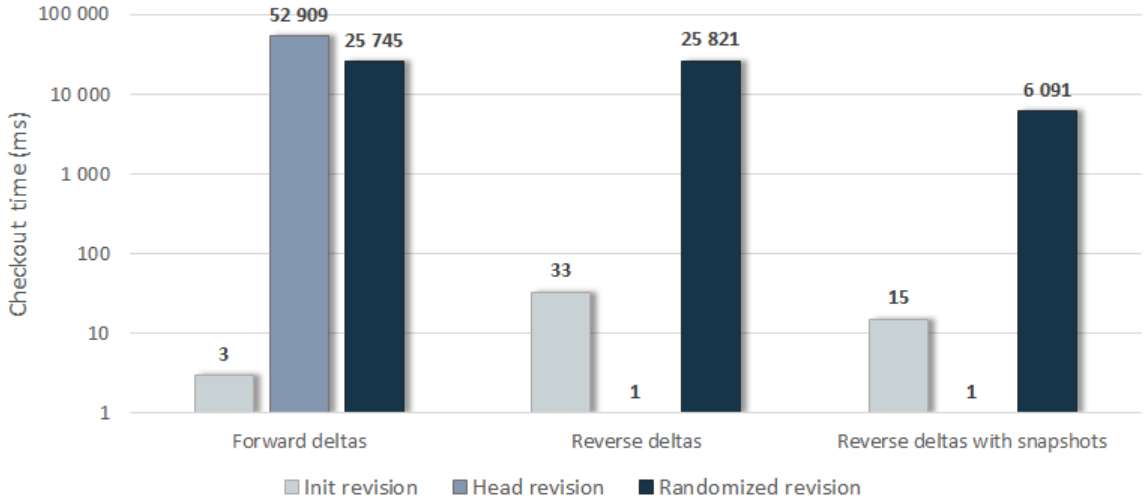


Figure 7 - Speed performance comparison of different methods

5 Conclusion and future work

In this article we presented an approach to geospatial data management using operation-based geospatial revision control model. As demonstrated, using a high level data model the method may support both vector and raster data and a variety of storage solutions. We argued that despite the inevitable computational overhead of this architecture, considering the peculiarities of the geographical information systems, this model is capable to deliver formerly unobtainable benefits on the field of revision control of spatial data, like persisting the semantic information of modifier operations. Our article presented a detailed design on that concept enabling not only linear control, but also branching and tagging, and usage of distributed repositories. Evaluation has shown, that our approach is capable of matching and even excelling efficiency of currently available solutions both in terms of storage space and computation performance.

Future work includes the development of a production ready revision control tool by further performance improvement of the prototype implementation. The thorough examination of complex snapshot creation heuristics is also required to find convenient and balanced solutions for snapshot management. The application of revision control in distributed data management is another future research topic, as AEGIS is capable of distributed data processing using the MapReduce paradigm (Giachetta, 2015). Revision control may be utilized for different purposes in distributed environment, such as data backup, recovery and transfer (Vrable, Savage, and Voelker, 2009).

12 In a production environment naturally a better heuristic should be chosen, for instance regarding not only the quantity, but also the complexity of the changesets.

References

Apel, S., Liebig, J., Brandl, B., Lengauer, C., & Kästner, C. (2011). Semistructured merge: Rethinking merge in revision control systems. *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)* (pp. 190-200). New York: ACM.

Chacon, S. (2009). *Pro Git* (1st ed.). Berkley, CA, USA: Apress.

Chen, H.-T., Wei, L.-Y., & Chang, C.-F. (2011). Nonlinear revision control for images. *ACM Trans. Graph., 30*(4), 105-115.

Cooper, P. (Ed.). (2010). *The OpenGIS Abstract Specification - Topic 2: Spatial referencing by coordinates* (version 4.0 ed.). Wayland, MA, USA: Open Geospatial Consortium.

Dobŏs, J., & Steed, A. (2012). 3D Revision Control Framework. *Proceedings of the 17th International Conference on 3D Web Technology (Web3D '12)* (pp. 121-129). New York: ACM.

Easterfield, M. E., Newell, R. G., & Theriault, D. G. (1990). Version Management in GIS - Applications and Techniques. *Proceedings of the European Conference on Geographical Information Systems (EGIS 1990)* (pp. 1-8). Amsterdam: Elsevier.

Firmenich, B., Koch, C., Richter, T., & Beer, D. G. (2005). Versioning structured object sets using text based Version Control Systems. *Proceedings of the CIB W78's 22nd International Conference on Information Technology in Construction* (pp. 105-112). Dresden: Technische Universität Dresden.

Giachetta, R. (2014). AEGIS - A state-of-the-art spatio-temporal framework for education and research. *OSGeo Journal, 13*, 68-77.

Giachetta, R. (2015). A framework for processing large scale geospatial and remote sensing data in MapReduce environment. *Computers & Graphics*.

Herring, J. R. (Ed.). (2010). *OpenGIS Implementation Standard for Geographic Information: Simple Feature Access – Common Architecture* (version 1.2.1 ed.). Wayland, MA, USA: Open Geospatial Consortium.

Newell, R. G., Theriault, D., & Easterfield, M. (1992). Temporal GIS - Modeling the evolution of spatial data in time. *Computers & Geosciences*, 427-433.

O'Sullivan, B. (2009). Making sense of revision-control systems. *Communications of the ACM, 52*(9), 56-62.

Peuquet, D. J., & Duan, N. (1995). Peuquet, D. J.; Duan, N. *International journal of geographical information systems, 9*(1), 7-24.

Reuter, J., Hänßgen, S. U., Hunt, J. J., & Tichy, W. F. (1996). Distributed revision control via the World Wide Web. *Lecture Notes in Computer Science, 1167*, 166-174.

Ruparelia, N. B. (2010). The History of Version Control. *SIGSOFT Softw. Eng. Notes, 35*(1), 5-9.

Shen, H., & Sun, C. (2001). Operation-based revision control systems. *Proceedings of the 3rd Annual International Workshop on Collaborative Editing Systems.* New York: ACM.

Vrable, M., Savage, S., & Voelker, G. M. (2009). Cumulus: Filesystem backup to the cloud. *Trans. Storage, 5*(4), 14:1-14:28.

Wachowicz, M., & Healey, R. G. (1994). Towards temporality in GIS. *Innovations*

in GIS, 1, 105-115.
Wang, J., Ghosh, R., & Das, S. (2010). A survey on sensor localization. *Journal of Control Theory and Applications, 8*(1), 2-11.
Winter, S., & Frank, A. (2000). Topology in Raster and Vector Representation. *GeoInformatica, 4*(1), 35-65.
Yang, C., Raskin, R., Goodchild, M., & Gahegan, M. (2010). Geospatial Cyberinfrastructure: Past, present and future. *Computers, Environment and Urban Systems, 34*(4), 264-277.
Zhang, J. (2010). Multi-source remote sensing data fusion: status and trends. *International Journal of Image and Data Fusion, 1*(1), 5-24.
Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory, 23*(3), 337-343.