A columnar architecture for modern risk management systems

Romulo Goncalves[1], Sisi Zlatanova[2], Kostis Kyzirakos[3], Pirouz Nourian[2], Foteini Alvanaki[3], Willem van Hage[1]

[1]Netherlands eScience Center, The Netherlands
{r.goncalves,w.vanhage}@esciencecenter.nl
[2]TU Delft, The Netherlands
{s.zlatanova, p.nourian}@tudelft.nl
[3]CWI, The Netherlands
{kostis.kyzirakos,f.alvanaki}@cwi.nl

***Abstract*—3D digital city models form the basis for flow simulations (e.g. wind flow and water runoff), urban planning, under- and over- ground formation analysis, and they are very important for automated anomaly detection on man made structures. They consist of large collections of semantically rich objects which have many properties such as material and color. Such user's data structure perception is leading to complex storage schemas. The number of table relations to manage and the large data storage footprint drawbacks are then extended with the fact that not all the systems have a "real" 3D data type.**

In this work we would like to show our efforts to develop a new kind of Spatial Data Management System (SDBMS) where topological and geometric functionality for 3D raster manipulation will become part of the relational kernel and not an add-on. With it spatial analysis tailored to different use case scenarios is done on-demand and fast enough to support real-time interaction in modern risk management systems.

I. Introduction

Digital 3D city models play a crucial role in research of urban phenomena; they form the basis for flow simulations (e.g. wind streams and water runoff), analysis of underground formations and man made structures which provide crucial information for effective risk management systems.

An urban scene, represented as a 3D city model, consists of large collections of semantically rich objects which have a number of properties such as use, function, and year. They are commonly reconstructed by segmenting and triangulating a point cloud thereby creating a surface representation. Representing urban objects (e.g. buildings, roads, trees, etc.) as surfaces has drawbacks while calculating intersections and volumes, and creating cross-sections is complex. Furthermore, modeling volumetric objects, such as walls, water, and underground, requires the deployment of complex shapes [18].

Such users data structure perception is leading to complex storage schemes. The storage scheme designed for systems like Oracle Spatial, Grass, and PostGIS has limitations such as the management of many tables when the selection predicate is on the 3D city model semantics. The number of table relations to manage and the large data storage footprint drawbacks are then extended with the fact not all the systems have a "real" 3D data type. PostGIS, highly adopted in eScience projects, is a clear example.

We have tackled all these issues by re-designing the conceptual model and the storage model. For conceptual model we have adopted a voxel-based city model, a path considered novel and promising [18]. Voxels are the volumetric representation of pixels. Alongside a length and a width, voxels also have height thereby forming a cube in 3D space. Voxel storage offers a number of interesting simplifications, use cases, but also challenges. One of the major challenges is its storage and efficient handling by Spatial Database Management Systems (SDBMSs). With different semantic level of detail (e.g., LOD in CityGML [16]) models and coverage of in- and out- side empty spaces, the voxelization of an entire city will generate a massive 3D grid of voxels at different resolutions with a large number of semantic attributes attached [18].

It is clear a dense flat relational table is not ideal to store such massive 3D grid. The holy grail is an architecture which allows effective compression to reduce storage footprint, and efficient data retrieval to access only the attributes of interest at a specific resolution. Such key features is what distinguishes a column-oriented architecture from a record-oriented architecture and the reason for their efficiency on analytic workloads [5].

Despite column-oriented architectures emerge as the right candidate and the efforts to extend them for spatiotemporal analysis over large data sets [8], [6], [12], [13], their flat storage model is not yet suitable to store a large 3D city model. To do so, we extended a column-store to also support a nested column-oriented storage for 3D city models. The chosen format is Parquet [1]. It is an effective storage model for sparse data sets with a nested structure (the different LODs). Its flat columnar format fits well the column-oriented programming model.

With our contribution, spatial analysis tailored to different use case scenarios is done on demand and fast enough to be used by modern risk management systems.The adopted storage model, Parquet [1], opens doors to also exploit state-of-the-art processing technologies, such as Spark, to scale out to country size. Furthermore, the simplicity of the conceptual model gives the opportunity to use interactive front-ends borrowed from gaming for real-time interaction with the surroundings.

The remainder of the paper is as follows. Section II describes the storage strategies and their challenges. Section III presents the general architecture. Section IV shows the steps already taken to put the vision in action. The article ends with future plans in Section V and a summary in Section VI.

II. Background

In this section we do a top-down description of our solution, i.e., from the conceptual model to the storage model. For the conceptual model we first identify its advantages followed by the challenges in supporting it on current SDBMSs. For the storage model we give a description on the challenges in mapping a voxel-based conceptual model into a flat and nested column-oriented storage.

A. Voxels

Our world can be represented in voxels by gridding the 3D space and specifying what each cell represents by semantically "attaching" every cell/voxel to a real world object. Storing volumetric spaces such as air, water and underground is possible.

Every object is defined by set of voxels, with set's length depending on the level of detail (LOD). The storage unit base is a 3D voxel of certain size and each voxel's characteristics e.g. type (wall, glass, roof, door, etc.), color, density, etc. is then stored as a semantic property. Such data type atomicity avoids the use of a set of multiple geometries, approach currently used in other spatial RDBMSs to store 3D city models [18].

Representing real world objects by a single geometry type (3D cube) instead of collection of polygons/polyhedron greatly simplifies a range of geometric operations: volumes and areas are calculated by simply counting the number of voxels that form an object; 3D bisections become simple selection operations; dynamic Levels-of-Detail (LOD) as objects can be resampled with larger cubes [18].

B. Storage challenges

The storage and indexing of 3D voxels linked with properties, such as voxels created to simplify a point cloud, two approaches can be considered, a homogeneous voxel grid versus a heterogeneous voxel collection. The former allows for factorization of invariant properties from the data structures, while the latter is better suited to sparse models such as a 3D city model with different LODs.

A homogeneous voxel grid is easy to define using a flat relational schema, i.e., real-world objects are formed by semantically grouping voxels together via foreign key relations and relational views. The scheme normalization is used to reduce the storage footprint at the cost of expensive spatial joins. The schema normalization storage footprint is proportional to the size of each voxel. Hence, efficient data access becomes dependent on efficient column compression techniques and effective storage of geometric empty spaces. The latter is very important because it strongly affects the data set size. If empty spaces were also materialized in the storage scheme, the storage of the whole of The Netherlands as e.g. 10 cm blocks would result in many petabytes of data.

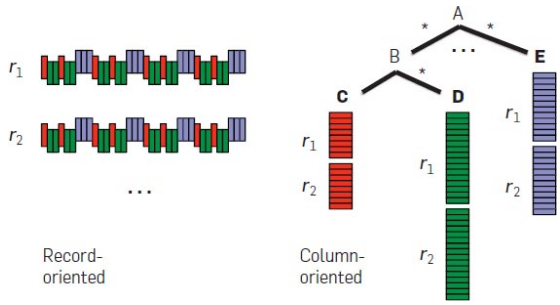

Fig. 1: *"Record-wise versus columnar representation of nested data"* [14]

A heterogeneous voxel grid poses extra challenges compared to a homogeneous voxel grid due to the preservation of the geometry semantics when converting vector to raster data. The object's semantics depends on the semantic level of detail (LOD) [18]. For example, the buildings LOD1 are *buildings*, LOD2 semantic is extended with *ground surface*, *wall surface*, and *roof surface*; LOD 3 has in addition to LOD 2 *window* and *door*; LOD4 *room*, *ceiling surface*, *interior wall surface*, *floor surface*, *closure surface*, *door*, *window*, *building furniture* and *building installation*. Hence, depending on the LOD, a voxel can have different semantic tags, e.g. (building, roof), (building, wall), (building, wall, window), etc. The LOD has a clear nested data organization and a sparse structure because of the in- and out- empty spaces. Furthermore, not all the levels in the nested structure are defined due to incompleteness or absence of vector information for a specific LOD.

C. Nested column-oriented storage

For efficient storage and data retrieval at different resolutions we embraced a column-oriented format for voxel-based 3D city models. Columnar formats have several advantages. Organization by column allows better compression, as data is more homogeneous. For large data sets the I/O is improved since it is possible to efficiently scan a subset of the columns while reading the data. Of course, better compression also reduces the bandwidth to read input data [4]. By storing together values of the same primitive type, a columnar format provides more efficient encoding and decoding.

Hence, to store nested data structures in flat columnar format, the schema is mapped to a list of columns in such a way that records are written and read back to its original nested data structure in an efficient way. Figure 1 illustrates the record-wise versus columnar representation of nested data. In the columnar representation all the values of a nested field are stored contiguously. For example, *A.B.C* can be retrieved without reading *A.E*, *A.B.D*, etc [14].

D. Parquet

In our work we use the well known Hadoop format called Parquet [1]. It stores nested data structures in a flat columnar format using a technique outlined in the Dremel paper from Google [14]. Parquet file layout is represented in Figure 2. Its internal structure is designed for efficient data skipping during

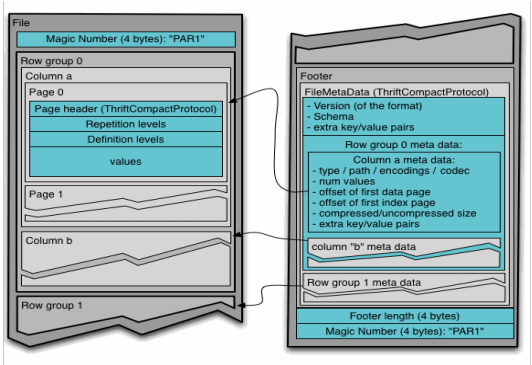

Fig. 2: Parquet file layout

query processing. The metadata stored in the file header and column-page header allows a kernel during predicate evaluation to skip data blocks and have lazy predicate evaluation over compressed data. At the same time, this metadata is used in our in-situ data access strategy, which is explained in Section III-B, to reduce the amount of data imported during query execution.

Parquet captures the record structure of each value through two integers called *repetition level* and *definition level*. During query processing they are used to fully reconstruct the nested structure [1].

Definition level. It is used to store the level of which the field is *NULL*. From 0 at the root of the schema up to the maximum level for the column. When a field is defined then all its parents are also defined. The definition level records at which level it started being null.

Repetition level. It is used to define when a new list starts in a column of values. It marks the level at which we have to create a new list for the current value.

Storing definition levels and repetition levels efficiently. For each primitive type it is necessary to store three sub columns. Due to the columnar representation the storage overhead is low. The depth of the schema defines the number of levels. For instance with 3 bits it is possible to store 7 levels of nesting. Required fields do not need definition level, and fields that are not repeated do not need repetition level.

Figure 3 represents the nested structure for a voxel-based city model. The LOD is used for the definition level. It is assumed that if a object has LOD2 semantics, it will also has LOD1 semantics, i.e., all the voxels inherit the semantics from the parent. The repetition level is the number of sub-divisions a parent voxel has. As an example, an object is semantically identified as a building in LOD1 while in LOD2 it might be composed by a set of sub-voxels to define walls, floor surface and etc.

III. A 3D RASTER SDBMS

A voxel-based 3D city model is best managed in a spatial DBMS as each voxel has a semantic relation to a real world object and various attributes (e.g. color, material, porosity, reflection properties, etc). Furthermore, a single spatial DBMS

[1]For a detailed explanation with examples we recommend the read of: https://blog.twitter.com/2013/dremel-made-simple-with-parquet

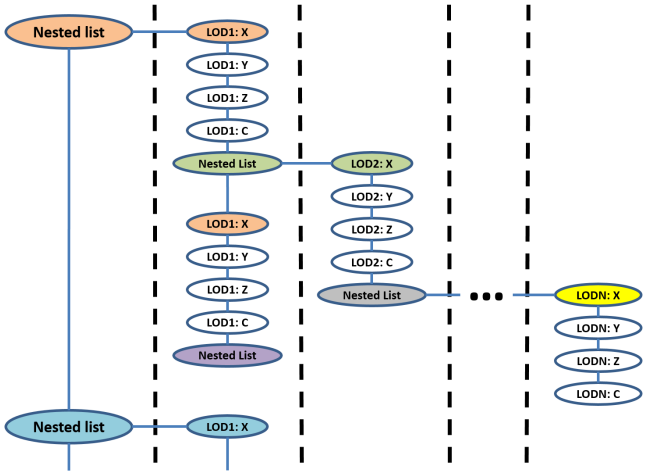

Fig. 3: LOD in Parquet

offers all functionality in one place, avoids the need for multiple software tools with associated high volume data transfer and format transformations.

During the last decade many DBMSs have been successfully extended with support for spatial and geo-spatial applications. For instance the OGC implementation specification, defining basic geometry types like points and polygons, is followed in PostGIS, Oracle, MySQL, Microsoft SQL Server, and MonetDB. To implement it they use their user-defined functions (UDF) functionality augmented in some cases with spatial search accelerators. However, contemporary DBMSs still lack advanced functionality and efficient implementations needed for analysis of voxel-based models.

We might argue that Oracle Spatial, Graphs 12c, and PostgreSQL 9.2 are developing extensions to support 3D geometries, even in GIS packages, only GRASS has support for voxels, but it still stores them as flat files. The systems are still in their infancy and they offer limited functionality. Due to the complexity of their software stack, deep integration with the database engine is even further away.

A. Column-oriented architecture

For our work we have extended a modern column-store, MonetDB [9], which steps away from traditional SDBMS which are all record-oriented architectures. Through vertical partitioning of relational tables column-store significantly reduce data access. In our case, vertical partitioning is exploited to reduce the number of columns to be imported as explained in Section III-B. Such data organization improves data compression, simplifies data skipping strategies and it suits well vector processing [4].

Currently through the works [8], [6], [12], [13], MonetDB spatial features have been matured to provide core technology components for geo-spatial big data analytics. Atomic spatial types and their operations are becoming part of the relational kernel and not an add-on. All the operations are available for spatial applications through integrated environments, such as R and Python, and a SQL front-end.

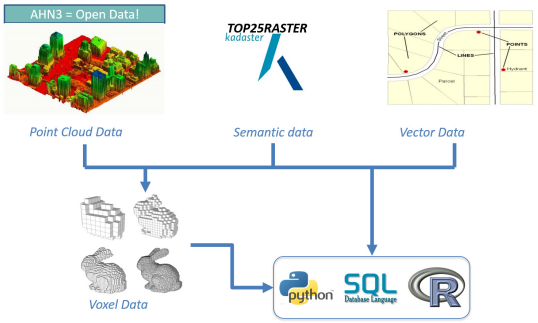

Fig. 4: MonetDB's spatially enabled architecture

Currently the system is equipped with SQL primitives for building complex spatial analysis pipelines over 3D point clouds, more concretely: geometry-based selections (rectangular query window, 3D bounding box), attribute-based selections (point intensity, RGB, multi-spectral properties), conversion to Triangulated Irregular Network (TIN) and triangulation using constrained Delaunay. It also provides the option to export the results into a pre-defined format, such as X3D, GeoJSON and LAS/LAZ format, to be loaded into visualization tools.

In the context of 3D city models, MonetDB is currently being extended with SQL operations to manipulate voxel attributes: 3D selections (contains, within, intersects); re-gridding of homogeneous voxel grids; semantic categorization; volume based aggregation; and also rendering for interactive visualization tools such as Cubiquity [2], more details in Section V-B.

Our architecture, represented in Figure 4, is an attempt to couple under the same storage descriptive spatial data, such as point clouds, vector data and 3D rasters semantically enriched. It creates the grounds to have direct and on the fly conversion to a data type tailored to the type of user interaction.

B. Dynamic data access

The need of large area coverage and up to date information to support near real-time decisions was the reason for us to explore *in-situ* data access, i.e., data is kept in its original format while scalable and distributed processing functionality is offered through a DBMS.

Our work adopts the same strategy defined in [8] where the authors presented a solution for *in-situ* data access to large NetCDF data repositories. The work stands on the shoulders of previous work called data-vaults [10]. In this article we have extended it to support Shapefiles, Parquet and LAS/LAZ file format.

The *in-situ* data access is possible due to the large amounts of metadata (data of data) existent on file formats such as Parquet and LAS/LAZ formats. Such metadata is used for effective data skipping, but also to collect data insights, e.g. summaries and samples, without having to process the entire data set.

The dynamic data loading comprises of three phases: the attachment of a file, the import of the file's content and the collection of statistics to boost query optimization. During the attachment, the file's metadata is loaded into a special DBMS catalog. At query time, such a catalog is inspected to decide whether the file has information relevant to the query. In such a case the file's content is imported into the database, otherwise, it is not.

The data import happens in two ways, if the file format has each attribute sequentially stored then the import memory maps each attribute as a column, otherwise, the data is converted and loaded into the database as temporary data. In the latter case, cache policies, such as Least Recently Used (LRU), are used for data eviction.

IV. Vision in Action

Our 3D raster SDBMS emerges from efforts in providing a scalable and generic solution for eScience projects with spatio-temporal data analysis. It is a continuous work standing on the shoulders of [8], [6], [12], [13]. In this section we summarize the steps taken towards a fully functional solution. The complete system evaluation is out of the scope of this article. An extended version with such evaluation will be submitted to a referee journal.

A. Voxel data Management

A 3D raster is commonly obtained from a existing 3D vector model or 3D discrete measurements such as point clouds. The vector model contains structured data and it is represented according to the rules of either GIS or BIM models. GIS models are used for modeling natural phenomena and man-made objects while newly constructed man-made objects such as buildings, bridges, etc, typically available in BIM models.

Our work supports 3D vector-raster conversion of vector models stored in CityGML[2] and it uses the voxelization principles defined in [15]. The voxelization of surfaces and curves are a customization of the Topological Voxelization approach presented in [11] and they ensure correct representation of geometries, topology, and semantics [15].

One object at the time is voxelized and the results saved into a Parquet file. Instead of voxelizing sequentially an entire city, the voxelization is done in parallel by tiling it. For each tile a Parquet file is created. For efficient data access, a Parquet file size is kept above 1GB to maximize the length of the stored column. Such optimization, and the fact objects distribution is not uniform, the tiling is not uniform.

B. Point cloud voxelization

The voxelization library [15] additionally provides an extension for voxelization of point clouds. The methods provide an easy management of connectivity levels in the resulting voxels, they are not dependent on any external library except for primitive types and constructs, therefore, easy to integrate them into a DBMS.

The *in-situ* data access combined with efficient spatial selections allows voxelization of point cloud on demand and near real-time. It allows us to extract a series of point-clouds of certain region to determine volume differences of nature

[2]http://www.citygml.org

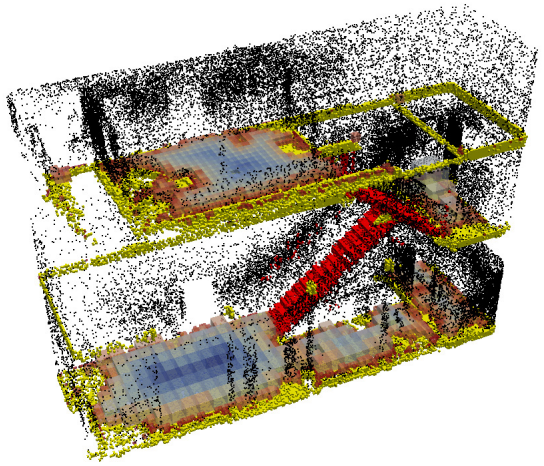

Fig. 5: Semantically enriched voxels from a point-cloud [7]

objects, or visualize quickly the impact of introducing or removing a man-made structure [15], [18].

For risk management such flexibility and efficiency allows rescue teams to study a building in a question of seconds. As an example, Figure 5 illustrates a large building of the Technical University of Delft (TUDelft). It maps the building into a series of points (red - stairs, yellow - floor and black-walls) while the voxels mapping empty spaces above the floor between objects using a color gradient, orange means objects are close by while blue means they are far away. The compact representation of the each voxel allows analysis of the possible routes and the available space to define escape trajectory routes.

C. Efficient spatial selections

For a performance profiling we have used the benchmark defined in [17]. The results are compared with the most efficient solution in [17], LAStools from Rapidlasso[3].

1) ***Setup****:* The experiments are conducted in a server with double capacity as the one used in [17]: instead of 16 Intel Xeon CPUs, it has 32 CPUs; instead of 128GB of main memory, it has 256GB; and instead of 2 x 41TB SATA in RAID 5 configuration, it has SAS (Thunderbolt) with 24 x 2TB disks. Despite the difference, input and intermediate data fits in memory. MonetDB was set to only use 16 cores. Regarding the software, we used the JUN2016 branch of MonetDB (*M*) and the latest version of LAStools (*L*).

LAStools is assisted by a DBMS (*LD*) to store each file bounding box in order to avoid the inspection of each file header. It also required us to run *lassort* and *lasindex* to boost query performance. Such pre-query stage had the same cost, around 18 hours, as a complete data import for MonetDB.

2) ***Data****:* Massive 3D discrete measurements have been obtained through airborne LiDAR (Light Detection and Ranging) or terrestrial scanning campaigns. As an example, the height map of the Netherlands, the *Actueel Hoogtebestand Nederland 2* (AHN2)[4] which is stored and distributed in more than 60,000 LAZ files, contains 640 billion points. The data is only composed by X, Y and Z coordinates, i.e., no extra benefit for column-oriented architectures compared to record-oriented architectures.

3) ***Queries****:* The queries are a series of small (*c*), large (*l*) space selections using simple (*S*) or complex (*C*) polygons. Table I's first line has which type of query and all other lines has the hot-run execution times for LAStools (*L*), LAStools combined with a DBMS (*LD*) and MonetDB (*M*). Due to space constraints, we have omitted all the queries for which it was not possible to obtain result in less than 1000 seconds or if it was 10 times worse.

4) ***Results****:* The 18 hours reported for pre-query data loading and preparation are automatically reduced thanks to our *in-situ* data access approach, i.e., only data relevant for the queries is imported using the file's metadata loaded during the attachment phase. For query performance, only for very small selections or simple polygons, LAStools combined with a DBMS, performs better than our solution. For all other queries with exception of query 24 and 27, our solution outperforms LAStools. It is important to notice that selections or aggregations on other attributes would be hard to express for the file-based solution and would required the inspection of all files' header. Furthermore, the numbers obtained by our open-source solution are comparable with the best commercial solution with customized hardware [17]. Overall our solution stands as the best DBMS solution.

D. Query execution

The used column-store, MonetDB, has operator-at-the-time paradigm with output of each operator being materialized before being passed as an argument to the next operator. The feature simplifies the integration of a nested column-oriented format, such as Parquet, since it allows us to un-nest a column and materialize it when needed.

One of the advantages of columnar processing is late materialization, i.e., tuples are re-constructed as late as possible in the query plan. It allows column-oriented architecture to have a low memory footprint during query execution, especially during the filtering phase. The extended operators, independently if they are processing nested or unested data, continue to support late materialization.

[3]http://rapidlasso.com

[4]http://www.ahn.nl

TABLE I: Efficient Spatial Selections Over A Massive Point Cloud Data Set

Typ	**sS**	**sS**	**sS**	**sS**	**sS**	**sC**	**sS**	**sC**	**sC**	**sS**	**sS**	**sS**	**lC**	**lC**	**sS**	**sS**	**sS**	**lS**	**lS**	**lS**	**lS**	**lS**	**lS**	**lC**	**lC**
Q	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	21	22	23	24	26	27	28	29
LD	.07	.16	.07	.16	.7	1.52	0.55	3.72	2.34	.08	.05	.26	412	102	.49	.04	0.32	2.28	142.0	313	234	282	.13	x	x
L	.9	.87	.78	.92	33.2	32.8	32.29	36.2	34.8	.88	.84	1.03	829	424	1.39	.75	1.18	20.74	x	828	x	x	923	x	x
M	.2	.41	.22	.5	.36	.66	.25	.69	.54	.24	.24	.44	24.9	23.6	.41	.08	.73	1.5	99.9	35.9	363*	17.2	.16	224	108

The scan- and aggregation operators were modified to be aware of the repetition level. The definition level is only used by projection operators to un-nest the data and do tuple re-construction. The tuple re-construction happens in the presence of a blocking operator, or a result constructor, or when it needs to combine nested data with flat data.

V. Future Plans

Once fully operational, we will study thoroughly the robustness of the proposed conceptual model and the efficiency our storage model using in-house projects. At the same time, we will design support for horizontal scalability and for interactive visualization of voxel-based 3D city models.

A. Horizontal scalability

With voxels stored in Parquet, our current work aligns with on going advances for large scale spatial processing in the cloud. As future work we intend to explore the possibility of 3D data manipulation of large scale voxel-based 3D city models using GeoSpark [3]. GeoSpark was built to efficiently exploit the internals of Spark[5]. It extends Resilient Distributed Datasets (RDDs) to form Spatial RDDs (SRDDs). For efficient data parallelism it efficiently partitions SRDD data elements across machines and introduces novel parallelized spatial geometric operations.

GeoSpark is still in an early development stage and it only supports few geometries (point, rectangle, and polygon), two spatial indexes (R-Tree and Quad-Tree). On top of that, it also supports spatial queries, e.g., range queries, K nearest neighbor (KNN) queries, and join queries on large-scale spatial datasets. Its major advantage is the fact it is built on top of Spark. By using Spark infra-structure, Hadoop friendly file formats such as Parquet can be directly ingested and used for large spatial analysis. In addition to our single-server mode using MonetDB, we will also provide cluster-mode using Spark (both providing integrated R and Python environments).

B. Interactive visualization

For interactive visualization we plan to explore Cubiquity [2] as an extension of Unreal Engine 4[6]. Cubiquity is a voxel engine written in C++ and released under the terms of the MIT license. It allows the creation of volumetric (voxel-based) environments which can be dynamically modified, i.e., it enables dynamic digging, building, and destruction.

Cubiquity is a flexible and powerful voxel engine, e. g., create terrains with caves or defined environments built from millions of colored cubes. It supports both smooth terrain and colored cubes type environments, multiple volumes which can exist in transform hierarchies and direct voxel access for implementing procedural generation.

[5]https://spark.apache.org/

[6]https://github.com/volumesoffun/cubiquity-for-unreal-engine

VI. Summary

In this work we have presented an architecture for a 3D column-oriented raster DBMS and so far our efforts on its implementation. The uniqueness of our solution stands on the combination of a novel concept model for 3D city models, a voxel-based one, with a efficient nested column-oriented format to explore the 3D city model at different levels of detail.

It is designed to iteratively load data from different sources and where topological and geometric functionality for 3D raster manipulation is part of the relational kernel and not an add-on. It is the first DBMS based solution with in-situ access to spatial data repositories and on demand voxelization. With it, spatial analysis tailored to different use case scenarios is done on demand and fast enough to be used by modern risk management systems where real-time interaction is a key feature.

References

[1] https://parquet.apache.org/.

[2] https://bitbucket.org/volumesoffun/cubiquity.

[3] http://geospark.datasyslab.org/.

[4] D. J. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the ACM SIGMOD*, 2006.

[5] D. J. Abadi, S. Madden, and N. Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the ACM SIGMOD*, 2008.

[6] F. Alvanaki, R. Goncalves, M. Ivanova, and et al. GIS navigation boosted by column stores. *PVLDB*, 2015.

[7] F. Fichtner. Semantic enrichment of as point cloud based on a octree for multi-storey path finding. *MSc Thesis, TUDelft*, 2016.

[8] R. Goncalves, M. Ivanova, F. Alvanaki, J. Maassen, K. Kyzirakos, O. Martinez-Rubi, and H. Muhleisen. A round table for multi-disciplinary research on geospatial and climate data. *IEEE e-Science*, 2015.

[9] S. Idreos, F. Groffen, N. Nes, S. Manegold, and et al. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Engineering Bulletin*, 2012.

[10] M. Ivanova, Y. Kargin, and et al. Data Vaults: A Database Welcome to Scientific File Repositories. SSDBM, 2013.

[11] S. Laine. A topological approach to voxelization. *Eurographics Symposium on Rendering*, 2013.

[12] O. Martinez-Rubi and et al. Benchmarking and improving point cloud data management in monetdb. *SIGSPATIAL Special*, 2015.

[13] O. Martinez-Rubi, M. L. Kersten, R. Goncalves, and M. Ivanova. A column-store meets the point cloud. *FOSS4G-Europe*, 2014.

[14] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive analysis of web-scale datasets. *VLDB'10*, pages 330–339, 2010.

[15] P. Nourian, R. Gonçalves, and et al. Voxelization algorithms for geospatial applications: Computational methods for voxelating spatial datasets of 3d city models containing 3d surface, curve and point data models. *MethodsX*, pages 69 – 86, 2016.

[16] A. Stadler and T. H. Kolbe. SPATIO-SEMANTIC COHERENCE IN THE INTEGRATION OF 3D CITY MODELS. *ISPRS Archives – Volume XXXVI-2/C43*, 2007.

[17] P. van Oosterom, O. Martinez-Rubi, and et al. Massive point cloud data management: design, implementation and execution of a point cloud benchmark. *Computer Graphics*, 2015.

[18] S. Zlatanova, P. Nourian, R. Gonçalves, and A. V. Vo. Towards 3d raster gis: on developing a raster engine for spatial dbms. *ISPRS WG IV/2 Workshop: Global Geospatial Information and High Resolution Global Land Cover/Land Use Mapping*, 2016.