TOWARDS 3D RASTER GIS: ON DEVELOPING A RASTER ENGINE FOR SPATIAL DBMS

Sisi Zlatanova[1], Pirouz Nourian[2], Romulo Gonçalves[3], Anh Vu Vo[4]
[1&2]Delft University of Technology, Delft, the Netherlands, S.Zlatanova@tudelft.nl, P.Nourian@tudelft.nl
[3]NLeSC, Amsterdam, the Netherlands, R.Goncalves@esciencecenter.nl
[4]University College Dublin, Dublin, Ireland, Anh-Vu.Vo@ucdconnect.ie

KEYWORDS: Voxel, Raster 3D, Built Environment Modelling, Urban Modelling, SQL primitives

ABSTRACT

Three-dimensional (3D) raster are simple representations, which have long been used for modelling continuous phenomena such as geological and medical objects. However, they can result in large data sets when high resolution is used, which poses challenges to algorithms for vector-to-raster conversion and data storage. This paper presents a research towards developing generic techniques and tools for efficient storage, management, and analysis of 3D spatial data types in the context of the column-store database system MonetDB. This paper focus on some issues in converting vector to raster data models in 3D spaces.

1 INTRODUCTION

Rasterization in 2D is nearly as old as computer graphics itself. It involves representation of continuous objects such as lines and curves on an inherently pixelated screen (i.e. monitor). As line rendering happens very frequently, the efficiency of these graphics algorithms will determine the latency of the whole visualization process. Therefore, these algorithms are often implemented in hardware to achieve highest performance.

In this paper, we look at the matter of rasterization from a different angle. We want to obtain rasterization of real-world objects while preserving all object’s semantics, properties as well as topological relations between the objects. Our aim is a unified 3D representation framework based on a single primitive data type called *voxel*, a volumetric pixel. Each voxel is a quantum unit of volume and has a numeric value (or values) associated with it that represents properties or independent variables of a real object or a value from a continuous field (e.g. magnetic field).

Such a representation is highly beneficial for advanced analytic procedures on digital 3D city models such as urban flow simulations (e.g. wind streams, water runoff, noise models and heat island effects), urban planning, and analysis of underground formations. Currently, digital city models can be reconstructed from massive point clouds obtained through airborne LiDAR (Light Detection and Ranging) or terrestrial scanning campaigns and reconstructed to 3D surface representations. Modelling urban objects (e.g. buildings, roads, trees) as surfaces has bottlenecks: calculating intersections and volumes, and creating cross-sections is complex.

Representing 3D urban scenes by voxels bring a number of advantages: calculating volumes is a matter of counting the number of voxels that constitute an object, 3D bisections become simple selection operations, storing volumetric spaces such as air, water and underground is possible. An additional benefit of voxel storage is the atomicity of the data type; every object is represented by only one primitive (3D cube) instead of the surface representation (i.e. points, lines and polygons).

During the last decade, many database management systems (DBMSs) have been successfully extended with support for spatial and geo-spatial applications. For instance, the OGC implementation specification "Simple feature access: SQL option", which defines basic geometry types like points and polygons, is followed by PostGIS, Oracle, MySQL, Microsoft SQL Server, and MonetDB. Using the user-defined functions (UDF) functionality of these systems can be augmented in some cases with spatial search accelerators, which operate using the abovementioned geometric data types. In other words, knowing that a series of data entries are points and inserting them as such (i.e. using a 3D point data type) we can potentially benefit from spatial indexing and geometric search operations within the database. However, contemporary DBMSs still lack advanced functionality and efficient implementations needed for analysis of big spatial data. A new data type implementation targeting 3D point clouds is under development by Oracle Spatial and Graph 12c and PostgreSQL 9.2. GRASS is one of the few GIS that provide support of voxels.

This paper reports on designing and developing a generic-purpose raster engine for spatial DBMS. We outline the quality criteria considered in the design of a raster engine and report our experiments with a laboratory tool set (developed in VB.NET, C#.NET) towards implementation of a rasterization kernel developed in C (i.e. to be integrated with MonetDB DBMS).

2 BACKGROUND

A large number of 2D raster- and vector-based applications have been developed in the last 40 years for management of spatial data: GIS (ArcGIS, QGIS, Hexagon AB), DBMS (Oracle, PostGIS), spatial extensions of CAD software (BentleyMap, AutodeskMap), image processing software (ERDAS). Despite the progress in 3D, an integrated management of 3D vector and raster is hardly possible. GRASS (http://grass.osgeo.org/) is one of the few GIS that provide support of voxels; Rasdaman (http://rasdaman.org) is an array database for storing 2D/3D raster and point data. These applications still offer limited functionality.

We envisage a high demand for a 3D information system integrating both 3D vector and 3D raster data, serving a vast range of applications. Three approaches for storage and management of data can be then distinguished:

- 3D vector data models with functions for conversion to 3D raster
- 3D raster data models with functions for conversion to 3D vector

- 3D hybrid, keeping the objects in their most appropriate representation (e.g. vector for building and raster for natural phenomena)

Each of the approaches has its advantages in certain applications. 3D vector models are often more suitable for visualization since smooth surfaces are not discretized as in the case of rasterized objects. On the other hand, 3D raster models have simpler data structure, which efficiently facilitates volumetric and neighborhood operations. Hybrid models have the potential to integrate the benefits of the two representations, if strict rules for data consistency between the two representations can be defined are introduced.

To be able to realize such systems, research and developments of both 3D vector and 3D raster domain should be integrated. A critical operation is the conversion between raster and vector. Namely, points, lines, surfaces and solids need to be converted into voxels (Figure 1) [and vice versa], while preserving the semantics of the objects.

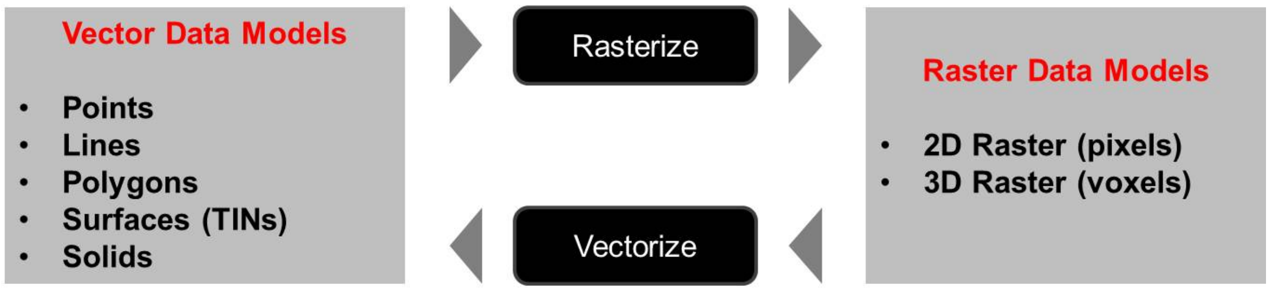


Figure 1: Data model conversions. Vectorization processes are not as well defined as rasterization processes. However, we consider this as a conceptual framework for the structuring of both our research process and the final products (the raster engine and its supported operations)

2.1 Creating 3D rasters

3D raster can be created from different sources: existing 3D vector models, 3D discrete measurements (e.g. point clouds, boreholes). The vector data can be represented according to the rules of either GIS or BIM models, i.e. they are structured data. Typically, GIS models contain simple geometries such as points, curves, surfaces and solids in a global coordinate system. BIM models allow for parametric representation (cylinder, sphere, cone), NURBS curves and surfaces, sweep shapes, etc., which can be embedded in each other. The coordinate system is mostly local and many of the objects represented with relative coordinates (e.g. doors, windows). Most of the natural phenomena and man-made objects are maintained in GIS models and only newly constructed man-made objects such as buildings, bridges, tunnels, and quays are available as BIM models. GIS models can contain different level of sematic/geometric coherence: ‘soup of polygons’ with or without semantics; structured geometry with or without object semantics.

Furthermore, some of the models can have topology. BIM models are better structured, having well-defined geometry-semantic coherence, but more complex.

2.2 Size of voxels

Modelling of objects with voxels requires a different way of representation compared to boundary (vector) representation. Voxels represent the interior of an object, while vector models represent the boundary. The boundary does not exist in raster representations. A notion of boundary between two objects is the thin surface between two voxel (see Figure 3). This implies that the size of the voxels should be smaller than the size of the actual object to be represented. Alternatively, the objects have to be exaggerated to the size of a voxel. Voxelization should comply with both the geometric accuracy and the semantics of the object:

- Semantic identification: For example, to represent walls of a house, the voxel size has to be at least 0.1m. If the wall is of no interest, the voxel size could be set even to 1.0m and all voxels will have semantic tag 'building' (Rosenfeld, 1981).
- Geometric accuracy: The voxel size depends on the shape of an objects as well as the accuracy of the measurements. For example, if a building has small corners of about 0.5m, the size of the voxel should be less than 0.5m. Similarly, higher accuracy measurements will require smaller size voxels.

Apparently, depending on the application, the size of the voxels can vary from fine (e.g. 0,1m for representing interiors) to coarse (e.g. 100m for representing geological or climate features). One uniform 3D raster would be advantageous for representing the model, but if the resolution is very fine, the size of the data can grow tremendously. In many applications (e.g. 3D raster, which contains buildings and geological objects) variable size voxels would be required.

2.3 Correct vector-raster conversion

Correct 3D vector-raster conversion requires correct representation of geometry, topology and semantics, which is specific for each model. In this paper we consider CityGML and IFC (BIM model).

Geometry: CityGML inherits the GML data types point, line, polygon and solid. This implies that real world objects represented as point, lines and polygons (surfaces) will be always smaller (in some direction) than the voxel size. If the objects have to be preserved in the resulting 3D raster, it has to be 'enlarged', to be represented with at least one voxel (Figure 3). IFC can have more complex geometries but the model used for the tests contains only surfaces.

Semantics: The semantics of the objects depends on the semantic level of detail (LOD). 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’. Depending on the LOD, a voxel can have different semantic tags, e.g. (building, roof), (building, wall), (building, wall, window), etc.

Topology: The neighborhood relations in the vector domain are represented by the boundary of the objects, which completely enclose the object. As boundary does not have an explicit representation in raster domain, the closeness of the object should be ensured by investigating the neighboring pixels/voxels. The neighboring pixels give an indication of *connectivity*. Each pixel has 4 neighboring pixels counting the shared edges and 4 neighboring pixels counting the shared vertices. Similarly, each voxel can have respectively 6 neighboring voxels considering the shared faces, 12 neighboring voxels considering shared edges or 8 neighboring voxels considering the shared vertices (Rosenfeld, 1981). This results in 4 (edges) or 8 (edges & vertices) connectivity in 2D and 6 (faces), 18 (edges & faces) or 26 (faces& edges & vertices) connectivity in 3D (Figure 2).

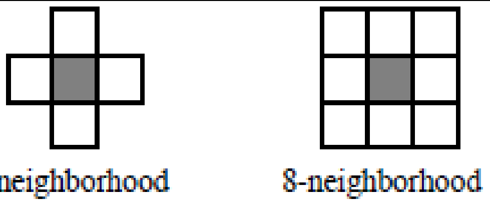
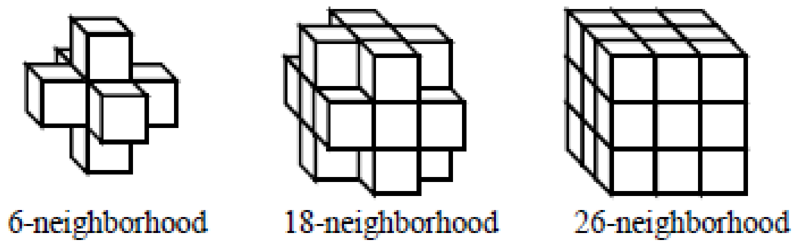
2D	Sharing Edges: 4 Neighbors Sharing Vertices: 8 Neighbors	
3D	Sharing Faces: 6 Neighbors Sharing Edges: 18 Neighbors Sharing Vertices: 26 Neighbor	

Figure 2: Adjacency and neighborhoods in raster models.
Images from Huang et al ###

The type of connectivity influences the shape of the rasterized object and can disturb the topological relationships between the objects. Figure 3 illustrates four cases in which the connectivity may lead to disconnecting or penetrating objects in 2D raster. 8-connectivity is favorable for crossing lines (a), but not for touching line and polygon (b). On the contrary, 4-connectivity is favorable for touching polygon line (d), and may lead to discontinuity of line in case of crossing lines (c).

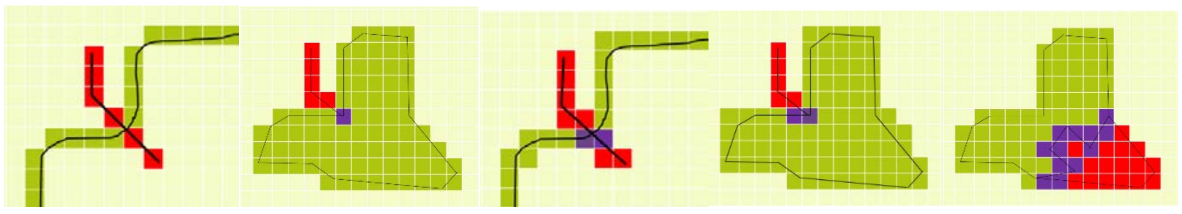


Figure 3: 2D raster: crossing 8 connected lines (a), touching 8 connected line and polygon (b), crossing 4 connected lines (c), touching 4 connected line and polygon (d and 8-connected touching polygons. The dark violet pixels in (b), (c) and (e) can lead to penetrated or disconnected objects

To deal with such cases, the *separability* of voxelization is introduced (Rosenfeld, 1981). A voxel is k-separating if there is no any path of connected voxels between ‘inside’ and ‘outside’. The rasterization of a closed line (blue pixels) on Figure 4a is 4-connected, which ensures 8-separability. In contrast, 8-connected rasterization results is 4-connectivity. The white pixels require further processing Figure 4b.

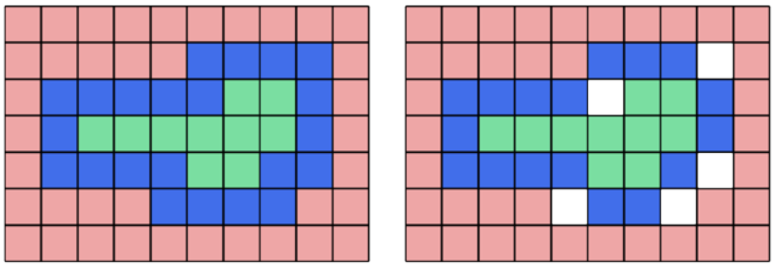


Figure 4: Separability of voxels: 4-connected voxels are also 8-separable, while 8 connected voxels are 4-separable

Semantics (tagging) identification during the rasterization process is a critical issue. The examples above, clearly illustrate the semantic identification of the pixels (i.e. ‘line’ and ‘polygon’) can result in disconnected objects or penetrated objects. If the dark violet pixel in Figure 3b is tagged as ‘line’, the line will be penetrating the ‘polygon’ object. If the dark pixels in Figure 3c are semantically identified as part of the green line, the red line is being disconnected. Similar semantic issue is observed for the touching polygons in Figure 3e. Depending on the semantic tagging the originally touching polygons might penetrate.

3 ON RASTER-VECTOR CONVERSION METHODS

As mentioned above, we envisage a 3D application to be capable of converting 3D vector and 3D data models to one another in a consistent way. The following are a few algorithms that can be used in such conversions. It is important to note that these conversions are generally irreversible because certain information is lost during a conversion. However, the importance of these methods is in their practical applications, in which this loss of information is not an important issue. For instance, many of these algorithms are developed for rendering, where the size of pixel is invisible for the human eye and incorrectness along the borders of the objects is hardly noticeable. Most of these algorithms concentrate instead on performance issues.

It might appear that voxelization is as trivial as intersecting objects with voxels; however, this naïve approach is firstly very inefficient. Furthermore, we need to ensure preservation of certain topological properties such as connectedness and separability as discussed above. We can define three quality criteria for voxelization algorithms:

- Efficient enough to be done on the fly
- Result in a 'thin' voxel collection that minimally represents voxelized features
- Provide explicit control and guarantee on preservation of topological and [thus] semantic structure of input

3.1 Rasterization approaches

Since most of the objects in 3D City Models are composed of multiple surfaces, the review is specifically on voxelization of surfaces.

Two general approaches can be distinguished: object rasterization and scan-conversion rasterization. Object rasterization concentrates on the object of interest following two steps: boundary rasterization and interior filling. The continuous rasterization scans in a given raster volume which voxels get what kind of value. All of the presented methods in these studies were based on extensions of the well-known scan-conversion method, which is well studied and commonly used in the field of 2D computer graphics.

(Kaufman & Shimony 1986) and (Kaufman 1987) presented a set of algorithms serving the purpose of voxelizing a range of specific parametric primitives including 3D lines, polygons, polyhedral, cubic parametric curves, bi-cubic surfaces, circles, quadratic objects and tri-cubic solids. Voxel models resulted from the algorithms are ensured to be 26-connected. It is, however, not possible to alternate the level connectivity to 18 or 6. Computational efficiency has been rigorously optimized by keeping repetitive operations as simple as possible (i.e. use addition, subtraction instead of multiplication or division) and integer arithmetic has been utilized whenever feasible. Consequently, temporal costs of the algorithms are kept consistently lower than linear to the number of resulting voxels.

The issue of limited connectivity control has been revisited by the same research group a decade later (Cohen-or & Kaufman 1997). In that research a so called Tripod algorithm has proposed allowing changing the connectivity level to six. Nevertheless, only line was investigated. The lack of connectivity control remains a significant drawback of the assortment of algorithms.

(Huang et al. 1998) addressed the topological property of resulting raster models but not the efficiency issue. Two and three dimensional planes have been investigated in that research. The proposed solution is based on the intersection between a regular voxel grid and a buffer zone constructed around an input object. By changing the size of the buffer zone, it is possible to switch between different connectivity levels. However as pointed out by a recent study by (Laine 2013), voxel models produced by (Huang et al. 1998) are not guaranteed to be minimal. Additionally, many spatial operations such as computation of distances, intersections and constructing planes are required. The significant complexity makes the algorithm potentially inefficient and re-implementation is not straightforward.

The above methods are all object-type specific, meaning different algorithms must be used for different kinds of input primitives. As opposed to that, a generic solution can be achieved simply by creating a voxel if it is overlaid by a portion of an input object. This approach is intuitive but generally leads to conservative results (i.e. minimality is not ensured). Moreover, the temporal cost associating with this solution is usually high due to the demanding spatial operations. Utilization of graphical hardware can alleviate the issue (Chen & Fang 1998) and (Schwarz & Seidel 2010) just to name a few.

A topological method presented by (Laine 2013) is very interesting option due to its capability of voxelization generic objects while rigorously ensuring connectivity property of resulting voxel models. The method is intersection-based, which set a voxel whenever its intersection target crosses an input object. A voxel's intersection target is a spatial subset of the cubical space occupied by the voxel such as the quadrilaterals bisecting the voxel along its three axes. Intersection targets are chosen based on the desired connectivity level and the dimensionality of the input vector object. Since the proposed method is intersection-based, computational efficiency would be an issue.

In addition to the methods presented in research papers, there are a number of accessible open source programs for voxelization such as ***binvox*** (Min 2014), ***Voxelization toolkit*** (Milossramek 2013). Those tools are usually restricted to watertight meshes, unable to directly parse CityGML file or assign semantic for voxels. Due to the restrictions, we opted to a re-implementation from scratch.

3.2 Our topological voxelization algorithm

Based on the topological voxelization method devised by (Laine, 2013), we have implemented two algorithms for voxelizing 1D inputs (curves) and one for 2D inputs (surfaces). Apart from these two, we have developed a voxelization algorithm for 0D inputs (points). Full scripts of algorithms is available in (Nourian, P, Goncalves, R, Zlatanova, S, Arroyo Ahori, K, Vo, A, 2016).

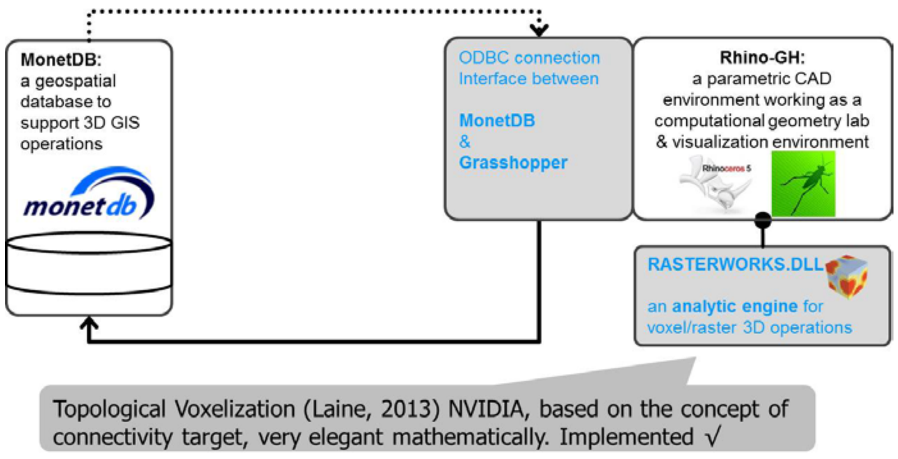

MonetDB:
a geospatial database to support 3D GIS operations
monetdb
ODBC connection Interface between MonetDB & Grasshopper
Rhino-GH:
a parametric CAD environment working as a computational geometry lab & visualization environment
Rhinoceros 5
RASTERWORKS.DLL
an analytic engine for voxel/raster 3D operations
Topological Voxelization (Laine, 2013) NVIDIA, based on the concept of connectivity target, very elegant mathematically. Implemented √

Figure 5: Vector-to-raster conversion directly in the database, avoiding unnecessary data traffic.

3.3 Rasterizing test environment

We focus on vector-raster conversion algorithms in a laboratory setup as schematized in Figure 5. The system consists of MonetDB, Rhinoceros® and Grasshopper© [1]. MonetDB (https://www.monetdb.org/) is intended for storage and management of integrated 3D vector and 3D raster representation. Rhinoceros is a CAD environment and is intended for query and visualization of vector/raster data. Grasshopper is used as a laboratory workbench for testing the algorithms. The rasterization algorithms are organized in a raster engine (RASTERWORKS.DLL), which is currently working in this architecture as a library of methods for rasterization, in combination with several utility functions. The connection between the MonetDB[2] DBMS and Rhino is established by ODBC (the standard programming API for accessing DBMS). The chosen algorithms have been implemented in C#.NET and VB.NET. Source codes are available on GitHub:

1. https://github.com/Pirouz-Nourian/TUD_RasterWorks_V_0.1
2. https://github.com/NLeSC/geospatial-voxels (Nourian, P, Goncalves, R, Zlatanova, S, Arroyo Ahori, K, Vo, A, 2016)

3.4 Rasterizing 3D indoor models

We have worked with two types of models: CityGML of one building at the TU Delft campus and a building model provided by Bentley systems. We investigated two approaches for voxelization, one as including empty space voxels with a colour code as black, visualizing empty space (Figure 6). Including empty voxels can be very problematic to produce larger datasets, e.g. a model of a whole city.

[1] http://www.grasshopper3d.com/
[2] https://www.monetdb.org/Home

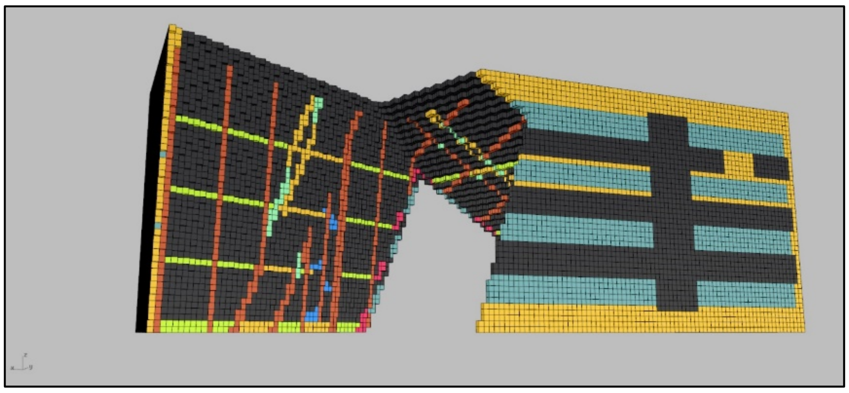


Figure 6: A building on the campus of TU Delft, rasterized from a CityGML model

The second building called ‘the Bentley building’ was used to experiment if all type of objects indoor/outdoor can be voxelized. The experiments show that the proper choice of voxel size can be critical for some objects. Some objects might not be representable with inappropriately sized voxels, for instance the wind turbine in this building disappears or becomes disjoint when choosing a coarse resolution (Figure 7, Figure 8). We suspect that this issue might be resolved by taking into account the importance of the object.

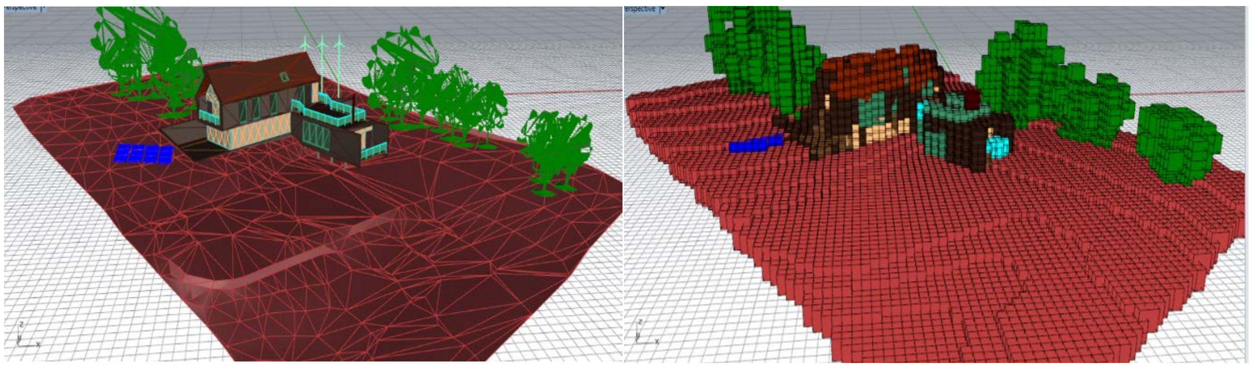


Figure 7: IFC building model imported as OBJ generated from original BIM model, corrected and coloured to represent objects of different semantics.

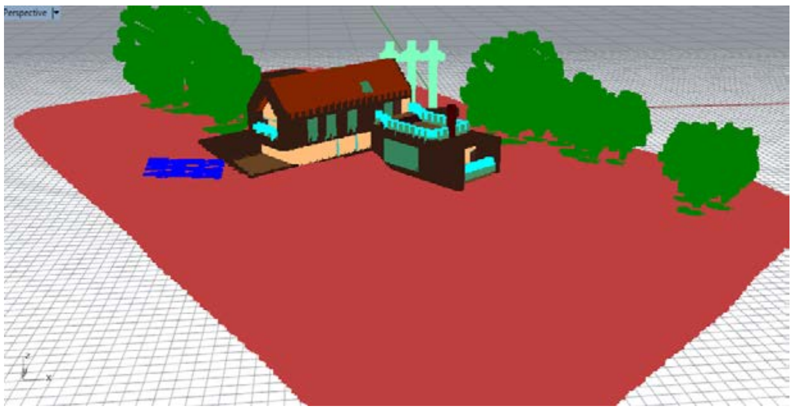


Figure 8: Topological Rasterization with Semantics from a BIM model [0.1 by 0.1 by 0.1 M]

3.5 Rasterizing CityGML LOD2 models (GIS)

In voxelizing CityGML LOD2 models (Figure 9, Figure 10), the main issues are defining a proper schema both for raster 3D and voxels in order to keep semantic information such as the ones modeled in a CityGML. Both of these issues and also the matter of ensuring uniqueness of voxels (avoiding duplicates when joining multiple voxel collections) requires further investigation.

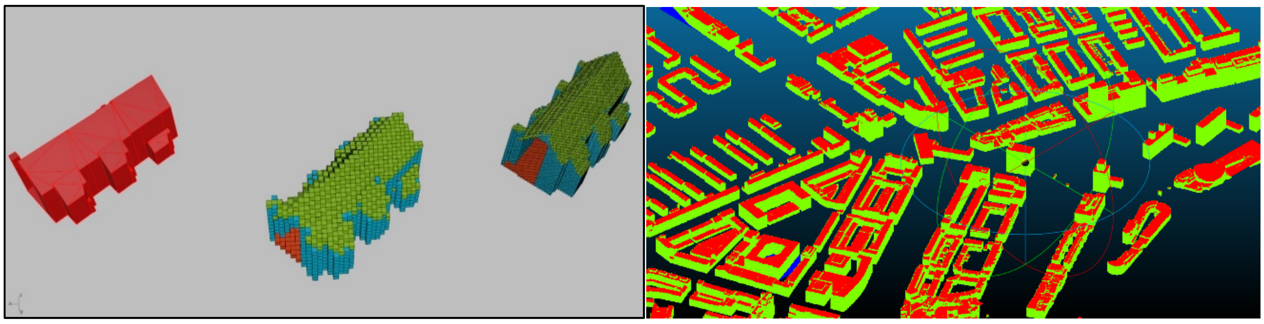


Figure 9: LOD2 building model from CItyGML models of Rotterdam (Rubroek), rasterized with 40 cm and 1 meter resolutions, colors encode semantic properties roof, wall, and ground floor as green, blue and red respectively.

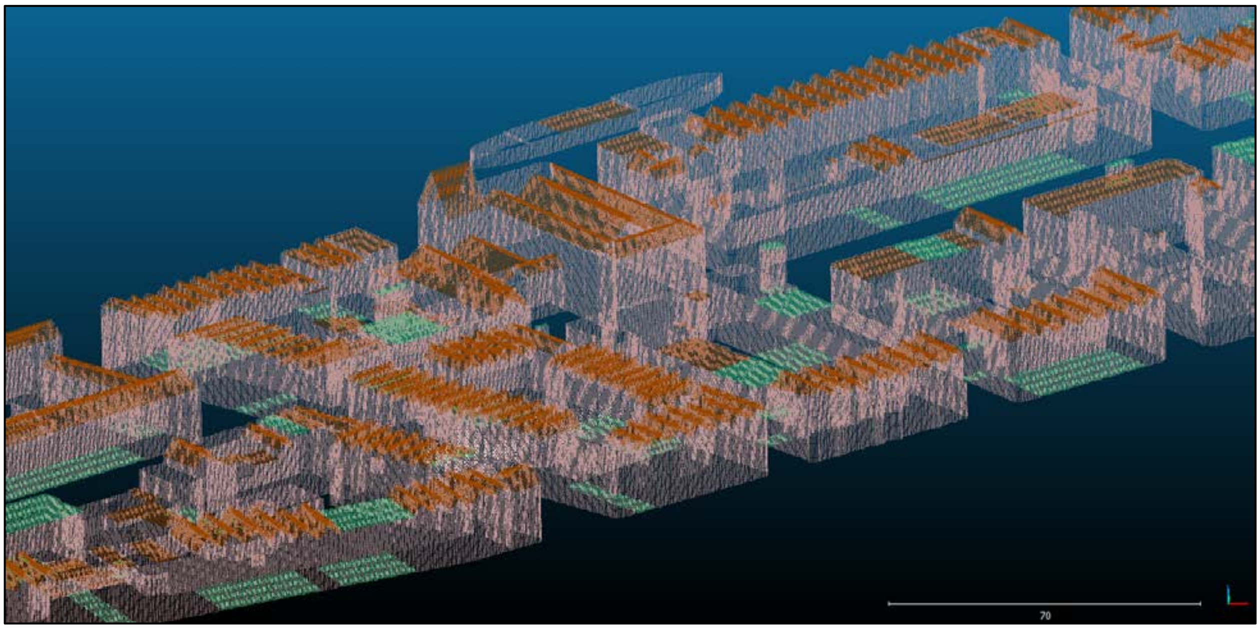


Figure 10: Rotterdam CityGML, Noorderiland, 0.2x0.2x0.2; zoomed in.

3.6 Loading in MonetDB

Using the described rasterization algorithms, we have loaded a collection of approximately 60 million voxels into MonetDB for testing. The voxels have been stored initially as tuples of [double x, double y, double z, double numeric attribute, integer semantic tag] and later as tuples of [integer ID, double x, double y, double z, byte R, byte G, byte B] in which IDs correspond to the objects voxelized and RGB colours encode semantic attributes of the CityGML files of Rotterdam.

The load of 60 million voxels takes 30 seconds using bulk loading from a comma-separated-values (CSV) file. MonetDB is a columnar-oriented relational DBMS, each column is stored sequentially in an independent file. Hence, if the coordinates are stored as doubles the disk footprint for X, Y, and Z is 1.34 GB. If instead of doubles we use decimals due to the small coordinate range, i.e., the coordinates are saved into a 4 byte-integer and the scale is stored as a table property, the disk footprint for X, Y, and Z is 686 MB.

Column-oriented architectures allow us to have efficient compression by using column-specific compression algorithm such as delta compression (https://en.wikipedia.org/wiki/Delta_encoding). With delta compression MonetDB reduces the size of X, Y, and Z from 686 MB to 171 MB.

The use of compression not only reduces the disk footprint to store the coordinates but it also improves query response. Such improvements are due to lazy decompression during query execution, i.e., a predicate is evaluated against the compressed data which improves I/O and speeds up data skipping. A performance profile is however not covered in this article. Such study will be published in a future publication.

We have performed tests with a set of 15 queries. Queries 1-7 extract voxels in a given MINMAX box, queries 8-11 are sematic selection (i.e. according to a semantic criterion) and 12-15 are update statements. The following are examples of the three groups of queries

Query 2: Select the whole set of voxels
```
DROP TABLE voxels_within_box;
CREATE TABLE voxels_within_box (x double, y double, z double, m double, s int);
INSERT INTO voxels_within_box SELECT * FROM Rubroek_voxels;
```

Query 10: Select all ‘ground’ voxels
```
DROP TABLE semantic_collection;
CREATE TABLE semantic_collection(x double, y double, z double, m double, s int);
INSERT INTO semantic_collection SELECT * FROM Rubroek_voxels WHERE s=3;
```

Query 15: Create a ‘floor’ voxels by assigning a value relative to their relative height (two versions, one in a loop)
```
UPDATE Rubroek_voxels SET m= ROUND(Z/3,0)
```

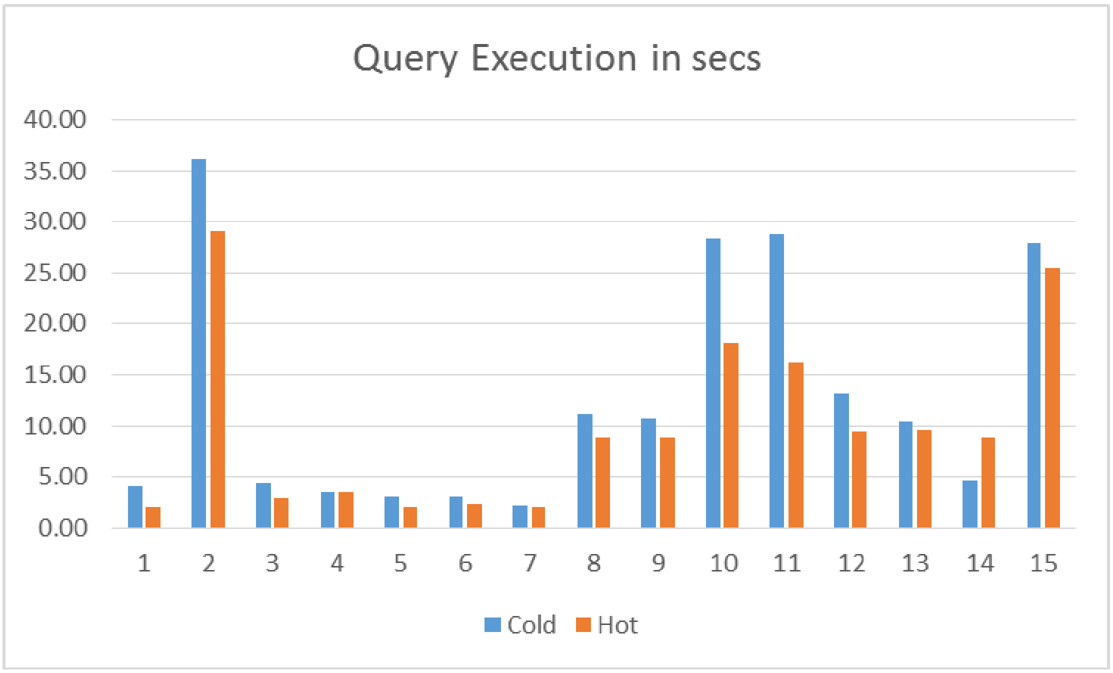


Figure 11: Execution of the queries in seconds.

The results (Figure 11) are in the range of 15 seconds with exception of some ‘expletive’ queries as extracting all voxels (query 2) or performing some computations in de update statement (query 15). It is also apparent that after several executions of one query the results improve, i.e. the result given in red (hot).

4 DISCUSSION

This paper presented some of our research and development of an extended raster engine for voxelization of semantically rich 3D city models. The rasterization engine is one of the components needed for realizing a raster 3D GIS. We have developed algorithms, which are able to rasterize 3D CityGML and IFC models. However, many issues are still open.

4.1 Some bottlenecks

The main bottleneck in the current process is a data parser, for instance in reading from CityGML files. We can currently only read the lowest level of semantics from CityGML files as we have to use a file format converter (FME) which can produce the file formats (.dwg or .obj) that Rhino can import. In this process of conversion, the higher levels of semantic get lost. Reliable data convertors have to be developed, which can keep all semantics and properties of objects.

The current working prototype of the voxelizer engine in C is significantly slower than the C# version. Although this does not sound logical, the problem is that the OBJ file parser we have implemented now reads the whole OBJ file of an urban neighborhood as one big mesh. This results in a very big bounding box and thereby an enormous number of voxels to be iterated on. The solution to this problem is in fact tightly related to the problem of semantics: when parsing a CityGML file we need to be able to read objects one by one building. Our OBJ parser is still naïve and rather simplistic and can be easily jammed by unexpected features in an OBJ file. Rasterizing large data set such as CityGML models of Rotterdam is currently done by rasterizing neighborhood models one by one. As the models do not have overlaps it might seem to be a good approach. However, we have not yet implemented any procedure to prevent creation of duplicate voxels. We envisage that by implementing a special type of spatial indexing this issue can be prevented. This case is closely related to the matter of standardization of a data model as for 3D raster data models. We regard this matter as one of the most important unsolved problems that requires further investigation.

4.2 Future research

The main issues that need further investigation are related to resolution of semantic issues shown in test results, namely conflicting semantics and standardization of a voxel datatype to be capable of serving a multitude of environmental modelling application such that semantic properties of objects can be preserved seamlessly. Ingestion of large vector data models can be problematic, as it will currently result in many duplicates of voxels.

Proper 3D raster data models have to be investigated, developed and tested in order to deal with this issue in a systematic manner. Voxel data type is atomic and needs to be defined in reference to a voxel collection as 3D raster. In order to standardize a 3D raster data model, we need to investigate and test more case studies in real-world environmental modelling/simulation scenarios and improving both algorithms and data models.

We currently present voxels in absolute geographical coordinates; however, this might have negative effects when dealing with big data. In addition, the main unanswered question is whether a 3D raster should be defined using the coordinates of every single voxel in the collection or integers referring to their (i, j, k) entries in a 3D array or only Boolean values corresponding to (i, j, k) entries. Each of these methods has their specific advantages and disadvantages. In order to answer this important question; more research and development have to be undertaken in collaboration with partners specialized in both data science and environmental modelling.

5 ACKNOWLEDGEMENTS

The work reported here has been funded by NLeSC Big Data Analytics in the Geo-Spatial Domain project (code: 027.013.703). We would like to thank our colleagues Oscar Martinez his constructive suggestions and Liu Liu for his precious guidance in dealing with CityGML files.

6 REFERENCES

Akio, Doi, and Akio Koide, 1991. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems* , 74(1), pp. 214-224.

Boekhorst, M., 2007. *Method for determining the bounding voxelisation of a 3D polygon.* USA, Patent No. US Patent 7,173,616, 2(12).

Borrmann, A. & Rank, E., 2009. Specification and implementation of directional operators in a 3D spatial query language for building information models. *Advanced Engineering Informatics,* 23(1), p. 32–44.

Cohen-OR D., Kaufman A., 1995. Fundamentals of surface voxelization. *Graph. Models Image Process,* 57(6), pp. 453-461.

Cohen-OR D., Kaufman A., 1997. 3D line voxelization and connectivity control. *Computer Grpahics and Applications IEEE* , 17(6), pp. 80-87.

Huang, J, Yage R, Filippov V and Kurzion Y, 1998. An Accmate Method for Voxetiting Polygon Meshes. In: *IEEE Symposium on Volume Visualization (Cat. No.989EX300).* s.l.:IEEE, p. 119–126.

Kaufman, A. & Shimony, E., 1987. *3D scan-conversion algorithms for voxel-based graphics..* New York, ACM Press, p. 45–75.

Kaufman, A., 1987. *Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes.* New York, ACM Press, pp. 171-179.

Laine, S., 2013. A Topological Approach to Voxelization. *Computer Graphics Forum,* 32(4), p. 77–86.

Lorensen, William E., and Harvey E. Cline, 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM Siggraph Computer Graphics,* 21(4).

Nourian, P, Goncalves, R, Zlatanova, S, Arroyo Ahori, K, Vo, A, 2016. Voxelization Algorithms for Geospatial Applications. *Methods X,* Volume 3, p. 69–86.

Rosenfeld, A., 1981. Three-Dimensional Digital Topology. *Information and control,* Volume 50, pp. 119-127.

Schwarz M., Seidel H. P., 2010. Fast paralell surface and solid voxelization on GPUs. *ACM trans. Graph.* , 29(6), pp. 179:1-179:10.

Varadhan G., Krishnan S., Kim Y.J., Diggavi S. Manocha D., n.d. *Effeicient max-norm distance computation and reliable voxelization.* s.l., s.n., pp. 116-126.

Wang, S.W. & Kaufman, A.E., 1993. *Volume sampled voxelization of geometric primitives.* s.l., IEEE Comput. Soc. Press, p. 78–84.

Xie, X., Liu, X. & Lin, Y., 2009. The investigation of data voxelization for a three-dimensional volumetric display system. *Journal of Optics A: Pure and Applied Optics,* 11(4), p. 45707.

Contact:

Assoc. Prof. Dr. Sisi Zlatanova
Delft University of Technology (TU Delft)
Faculty of Architecture and the Built Environment
Department of Urbanism, 3D Geoinformation
Julianalaan 134
2628 BL, Delft
The Netherlands
Phone: +31 15 278 2714
E-mail: s.zlatanova@tudelft.nl

Pirouz Nourian, PhD Candidate
Instructor in Design Informatics and Geomatics
Delft University of Technology
Faculty of Architecture and the Built Environment
Department of Urbanism & Department of Architectural Engineering + Technology
Julianalaan 134
2628 BL Delft, PO Box 5043
2600 GA Delft
The Netherlands
Phone: +31 15 27884094
E--mail: p.nourian@tudelft.nl

Dr. Romulo Gonçalves
eScience Research Engineer
Netherlands eScience Center
Science Park 140
1098 XG Amsterdam
The Netherlands
Phone: +31 20 460 4770
E-mail: r.goncalves@esciencecenter.nl

Anh-Vu Vo, PhD Candidate
University College Dublin
School of Civil, Structural and Environmental Engineering
Urban Modelling Group
G67 Newstead Block B
Newstead, Belfield, Dublin 4
Ireland
Phone: +353 1 716 3232
E-mail: anh-vu.vo@ucdconnect.ie