3D Data Integration in the Voxel Domain

Ben Gorte[1,2], Sisi Zlatanova[1], Morakot Pilouk[3], Abdoulaye Diakite[2], Jack Barton[1]

[1]GRID, School of Built Environment, UNSW Sydney, NSW, Australia – b.gorte, s.zlatanova, jack.barton @unsw.edu.au
[2]Voxelmates, Pty Ltd, Sydney, Australia – abdou, ben @voxelmates.com.au
[3]ESRI, Redlands, USA – mpilouk@esri.com

Keywords: Voxel Operators, 3D model, Digital Twin, Valid Data Model, Data Structure, 3D Reconstruction

Abstract

The interest in 3D data integration is growing as the concept of Spatial Digital Twins gains traction among local governments and stakeholders. Spatial data is typically organized into standardised themes such as buildings, transport, and vegetation, or as ad hoc data products in either 2D or 3D dimensions. However, integrating these data sets into a cohesive 3D model presents several challenges. The foundational data sets may vary in terms of accuracy, resolution, representation, and actuality due to differences in creation standards and procedures. Consequently, inconsistencies often arise within the 3D models, making them difficult to identify and rectify. Many of these data integration issues can be traced back to the 3D vector representation. Vector 3D models exhibit significant geometric diversity, which complicates the computational operations required for examining and validating the models. In this paper, we propose a novel voxel-based approach for integrating 3D data. Voxels, analogous to pixels in 2D raster images, inherit several key properties. We employ a dedicated voxel data structure and a set of operations tailored for this purpose. First, we voxelise vector datasets, converting continuous geometric information into discrete voxel representations. Next, we merge voxel layers using a specialized voxel overlay operation. The resulting 3D model adheres to most validity and integrity requirements. To highlight the flexibility of our voxel data structure, we transformed it into point clouds for visualisation in ArcGIS Pro, enabling the addition of 3D base maps and GIS layers. We then published these datasets to Scene Services, granting public access. This allows users to easily explore the 3D scenes through a web browser, making complex geographic data more accessible. By leveraging voxels, our approach facilitates efficient and accurate 3D data integration, making it a valuable contribution to the field.

1. Introduction

In the last years, it has become a prerequisite for municipalities and institutions to have 3D models of their cities to support Digital Twins. 3D models are integrated with sensor data for environmental monitoring, traffic, mobility, or for urban planning and development. Large areas have been maintained as 3D models in many cities in across the world (e.g. Döllner et al 2006). Simplified 3D models are also freely available through geospatial vendors such as Google, Microsoft and ESRI.

However, most of these models are not readily applicable for the intended application. Frequently, either a new 3D model has to be created or an additional data have to be added to an existing one. Depending on the application, the needed 3D models may differ significantly. Some applications may require photo-realistic models, others may need object-oriented models with plenty of attributes or models with topologically correct geometries. Such considerations often lead to creating a new 3D model. Therefore, the topic of integrating data sets to create 3D city models is becoming more appealing than ever.

Reconstruction of 3D models from point clouds, images and existing data sets have been an extensive topic of investigations. In this paper, we concentrate on creating 3D models from available standard 2D and 3D data sets, which are produced and maintained by the mapping agencies or data provides. Normally such data sets are organised per theme: transportation, buildings, terrain, vegetation, water bodies. They can be accessible via Web services or a direct connection to databases or provided as files. The file formats can be diverse as well: from semantically rich models such as CityGML or IFC to the ones containing only geometry such as .obj, .stl, .ply. Accuracy and resolution can vary, i.e. buildings can be represented as 2D polygons or 3D shapes in variety of LODs. Roads might be represented by their centre lines with attribute for the width of lanes or as polygons. The geometric representation can be raster or vector. For example, the terrain is commonly maintained as raster data set, although TIN representations can be found as well. Data can be structured according to a national data standard, or vendor specific schema. As results, the available data sets are very diverse, and their fusion is always problematic.

Many papers have been published for creating 3D models from different data sets either for whole cities (Erving et al 2009) or parts of cities (e.g. Erving et al 2009, Billen et al 2015, Diakite at al 2020, La Guardia et al 2022) or buildings (Diakite and Zlatanova, 2016, Boguslawski et al 2022, Xie et al 2022). The conclusions drawn from these experiments highlight the multitude of challenges associated with the integration of 3D vector data. One significant challenge arises from the diverse nature of vector data types, including points, multipoints, polygons, multipolygons, polyhedrons, and tetrahedrons. This diversity increases the likelihood of undesirable issues such as overlapping, intersections, inclusions, and discrepancies in terrain representation.

Researchers have been actively exploring various approaches to address these challenges, including techniques for accurate 3D reconstruction (Yan et al., 2019) and methods for detecting and rectifying errors in 3D City Models (Ledoux and Meijers, 2011; Wagner et al., 2012). Additionally, efforts have been made to mitigate the variety of vector data types by using standardised database management data types (Li et al., 2019). Despite these endeavours, many errors remain difficult to rectify, posing ongoing challenges for effective 3D vector data integration.

In this paper we present our experiments on 3D data integration in the voxel domain. Voxels are equivalents to pixels in 2D raster models. Voxels have numerous advantages to vector representation. They create a 3D gridded space, each voxel has only one value, the data management is quite intuitive, and many (neighbourhood) operations can be defined. Voxels have been used extensively for processing point clouds or for modelling of continues phenomena such as geology, marine, etc., but have been rarely applied for 3D city modelling or analysis, except for indoor environments to compute navigation paths or for evacuation. (Xu et al 2018, Fichtner et al 2018, Staats et al 2018,Gorte et al 2019, Zhao et al 2022, Aleksandrov et al 2023)

In this paper we illustrate that some of the above-mentioned errors and inconsistencies can be easily avoided if the 3D data integration is performed in the voxel domain.

The paper is structured as follows: Initially, we delve into the data model and the operational methods utilized in our experiments. Subsequent sections are dedicated to the integration of four key datasets: terrain, roads, buildings, and trees, with a comprehensive explanation of the processing techniques for each dataset and their integration methodology. The final section showcases how the data is imported into ArcGIS Pro, enabling users to explore and enhance the 3D scene by incorporating base maps and additional GIS layers. Furthermore, we discuss how this data can be made publicly accessible through publishing on scene services via the ArcGIS Enterprise or ArcGIS Online platforms. The paper concludes by evaluating the benefits and limitations of employing voxels for 3D data integration, as well as their applicability in city modelling.

2. Voxel data structure

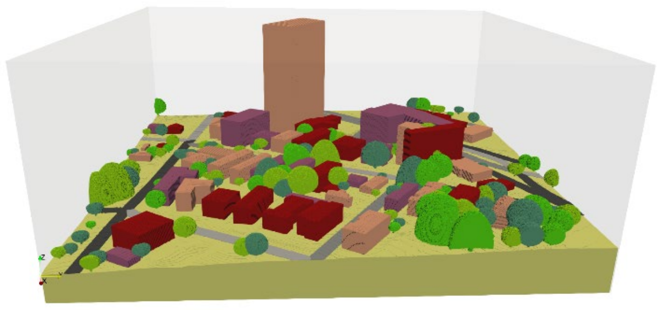
As mentioned above, voxels create a regular grid or a 3D array, in which each object is composed by a set of voxels and therefore can be seen as a solid. This allows to unify and simplify the representation of the objects, but also requires attention when voxelising data sets that are based on B-reps representations (points, lines, surfaces, and polyhedrons) (Gorte and Zlatanova 2016, Nourian et al 2019, Aleksandrov et al 2021). Fine-resolution voxels are beneficial when aiming at accuracy, but the size of the 3D raster can grow exponentially when large parts of cities are considered. The performance can be easily affected and even the computer memory might become insufficient. Therefore, data structures need to be devised to allow voxel storage on a disk.

Octrees are commonly used as a data structure to maintain the 3D regular grid, especially when many voxels have identical values as in cities (e.g. air voxels). In this paper we rely on a data structure as presented by Gorte 2023. Here, we will summarise how it works without detailing the considerations and justifications. It is an optimised octree data structure, which allows further reduction of the octree footprint. The data structure is organised in an SQLite and several generic operations are developed to work with the data structure.

2.1 Voxel scene and layer

In voxel domain, we assume all objects are volumetric. All voxels, including those inside and between solid objects, are semantically relevant. A geo-reference is assumed, which relates 'real world' coordinates (X, Y, Z) (in metres) to integer grid coordinates (x, y, z), e.g. (X, Y, Z) = (X0, Y0, Z0) + R ∗ (x, y, z). R is the resolution, or grid spacing, which is equal in X,Y,Z-directions. The range of (x, y, z) is between (0, 0, 0) and (xmax, ymax, zmax), which defines the size of the particular voxel scene.

A 3D raster can represent different spatial objects (thematic classes) or characteristics of one object. Examples of thematic classes are voxels values such as 1-buildings, 2-trees, 3-roads, 4-roofs, 5-doors, 6-windows. Building IDs (Figure 1, top) or temperature values are examples of characteristics of one object. In this paper such 3D raster will be called 'layer'. Several integrated layers are called 'scene' (Figure 1, bottom). Scene size, geo-reference and resolution are the same for all layers a specific scene.


a) A scene: integration of four layers

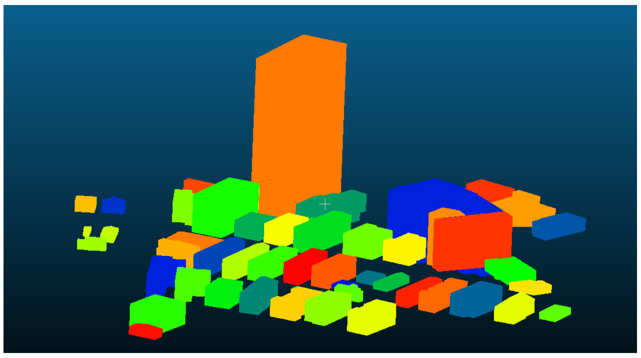

b) A building layer (each colour corresponds to a different building ID).

Figure 1: Example of a) 3D voxel scene and b) 'building' layer

The 3D raster is automatically extended into a multi-resolution voxel pyramid, following octree mechanism. This gives access to a series of resolutions, such as 0.2m, 0.4m, 0.8m, 1.6m etc. At the bottom of the pyramid, where the resolution is finest, the level (L) is 0; resolutions get coarser as the level goes up. The coordinates of voxels are the same over all levels and can be represented as (x, y, z, L). Each coordinate is converted to a key via a Key-formula, which is then linked to the voxel value (V):

$$(x, y, z, L) \longleftrightarrow \text{key (K)} \rightarrow \text{value (V)} \qquad (1)$$

2.2 Octree data structure

Here we present the 'physical' storage all the voxels of a multi-resolution voxel pyramid. We store only a subset of the pyramid in an octree data structure. In contrast to vector data types, voxels cannot be deleted or added. They can be only queried, or their values can be changed. This means that only two functions are needed OTget and OTupdate to manage the octree:

$$\text{OTget (key) -> value;} \qquad (2)$$
$$\text{OTupdate (key, value).}$$

The octree does not store all voxels. A special rule is applied, which allows to compress further repetitive voxel values of lower levels. For example, if at level L1 only two (out of eight) voxels are explicitly stored with values Va and Vb, then the other six are assumed to be V, and the values V, Va and Vb are the values for the corresponding cells at L2. This means that if a voxel (x,y,z,L) is present in the octree database, then so is its value (V).

Otherwise, we have to go up the levels until we find a cell that contain (x,y,z) and take the value from there. More details can be found in Gorte 2023.

The octree is created during the voxelisation. The voxels are generated by converting points, lines, surfaces and polyhedrons to voxels at the lower octree level. The higher levels are generated automatically by aggreging voxels at low levels with same value and shifting the values to higher levels.

Several additional functions are available to support the management of the octree, which are OTfetch (K), OTstore (K,V) and OTremove (K). OTfetch extract voxels with values; OTstore replaces a voxel value with a new value or adds a voxel with value and OTremove, deletes a voxel from the octree. Conceptually the voxel is not deleted but it becomes 'empty'.

In addition to the functions to manage the octree, there are four generic operators, which are intended to help performing spatial operations.

Func (octree in, octree out, function M (x), function A (a)). Func performs mapping from the voxel values of an input dataset into the corresponding voxels of an output dataset. Func is also useful to select voxels on the basis of value (or a set of values, an interval, etc.) by specifying a mapping that yields 1 or 0 when voxels do/do not meet that criterion. A typical example could be a query 'give all roof types of the buildings'. If the input data set contains all buildings with their IDs, then the roof type can be provided by the function M (x). This is to say each ID has assigned a specific roof type. The aggregation type for the output octree is provided by function A (a). In the case of buildings and roofs it is majority voting.

OTover (octree Xin, octree Yin, octree out, function O (x,y), function A(a)). The operation OTover is similar to Func, but it takes two input datasets instead of one, and produces an output by applying a function with two parameters. An example of this function is integration of terrain layer and roads layer and indicating the overlapping voxels. Given the voxels in both data layers have two values: first data set has 'air1' and 'terrain' values and the second 'air2' and 'road' values, the function O(x,y) adds the values of the voxels as follows:
'air2'+'terrain'='terrain',
'air1'+'road'='road'.
'road'+'terrain'='road',
'air1'+'air2'='air'.

Function A plays the same role as in Func; it establishes the octree building approach.

OTfilter (octree in, octree out, kernelA, function F(k,n)) performs 3D neighbourhood operations. kernelA is a 3D array of coefficients, which can be used as a kernel in convolution operators. The function F(k,n)) perform the actual computations. An example of this function would be 'Derive the outer walls of building', assuming that these have not been stored as individual objects. The input octree has two values 'building' and 'air', the kernelA=3x3x3 and the function F (k,n) considers a neighbourhood n equal to the kernel size, computes the values and assigns 'wall' for specific kernel values. For example, in case of buildings with flat roofs, 'wall' is assigned when the value is less than 18.

OTprofile. Similar to Function OTprofile traverses the two-dimensional footprint of a voxel dataset, collects the column at each (x,y) position and passes in as an array of voxels into a user-specified profile-analysis function P(p). The analysis function yields a new (or modified) column, which is to be stored in an output octree dataset. This allows, for example, to encode the highest voxels of buildings into a dedicated roof value. If the bottom layer of each building is known to be flat, the operation can construct floors inside buildings by turning building voxels at pre-defined heights (e.g. multiples of 2.8m) above the lowest building voxel into 'floor' voxels. This function is also used during the voxelisation of solids to fill out voxels between a footprint and a roofing surface, given that the connection between them is only vertical.

OTras (octree in, 3D raster out, boundB, resL, dataT). OTras reads an octree dataset into a 3D array, which is effectively an octree-to-raster conversion. OTras handles additional parameters, allowing to specify a bounding box (BoundB), a resolution level (ResL), and an output data type (byte, short, int) (dataT). The resulting 3D array can then be manipulated directly in the memory.

3. Use case and data sets

This experiment is performed on the data sets available for the project Liveable City Digital Twin for the City of Liverpool, Australia (Diakite et al 2022). The data sets were obtained from NSW Spatial Services, Liverpool council and the company GeoScape. As can be observed in Table 1, most of the data set are provided in ESRI shape (.shp) or Geodatabase (.gdb) file format with various attributes.

Many of the data sets are 3D but there are also some 2D such as roads, vegetation and water bodies. The DTM is available as 2D raster with 1x1m resolution and height attribute. Buildings are available as LOD2, but without semantic distinction between walls and roof. The vegetation data set contains X,Y,Z coordinates of the footprint of individual trees and a height attribute.

Data set	Geometry type	Format
Buildings	MultiPolygonZ	GDB
Roads	MultiLineStringZ MultiPolygon	GDB & SHP
Railways	MultiLineStringZ	GDB
Water bodies	MultiLineStringZ MultiPolygon	GDB & SHP
Vegetation	MultiPolygon PointZ	GDB & SHP
Terrain (DEM)	Raster (grid)	GeoTIFF
IoT Sensors	Point	GeoJSON

Table 1: Data sets within the Liveable City Digital Twin, Liverpool (Diakite et al, 2022)

Diakite et al 2022 presented a workflow for 3D integration and importing the data in 3DCityDB (Kutzner et al 2020). All data sets were processed accordingly, including the creation of 3D constrained Triangular Irregular Network (TIN) for the terrain, considering the corresponding surface objects and the footprints of buildings and trees. The followed procedure resembles the steps presented in Yan et al 2019.

4. 3D data integration process

Four data sets, i.e. terrain, buildings, trees, and roads, are used in the voxelisation experiment. The voxel resolution is 0.2m and the selected test site is 280x300x128m (1400x1500x640 voxels). The applied voxelisation procedures for surfaces and solids are presented in Gorte and Zlatanova 2016. The voxelisation of lines is performed in two steps with the help of 2D raster (see road centrelines below).

After the voxelisation octree has five levels: L0 - 0.2m voxel resolution, L1 - 0.4m, L2 - 0.8m, L3 - 1.60 and L4 - 3.2m cell resolution. To visualise the resulting voxel layers, the function OTres is always used. To create triangles for rendering, a dedicated function creates cubes (and triangles) to represent voxels and prepare .obj files. This function also takes into consideration only visible sides of the cubes, i.e. the internal cubes are not rendered. We have used MeshLab for visualisation of the .obj files.

4.1 Terrain

As mentioned above, the input data set is raster of 1m. It is resampled to 0.2m using bilinear interpolation (instead of triangulation), which is a 2D raster operation. Then for each point (x,y) of the raster the value z is stored and the key is computed for each pixel. Consequently, the function OTstore builds the octree. At this point the voxels represent only a terrain surface. Using the function OTprofile, the terrain voxels are ’extended’ to go to -2m from the lowest point of the Digital Terrain Model (DTM). Figure 2 illustrates the resulting DTM.

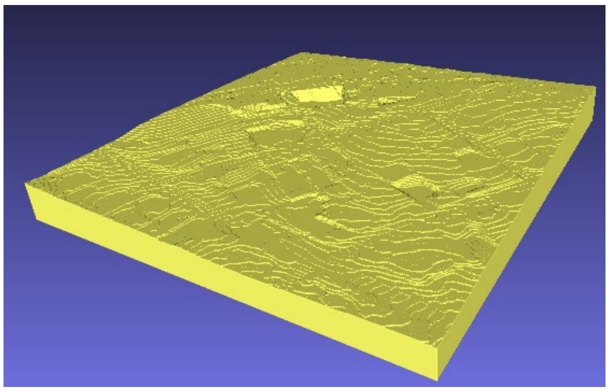


Figure 2: Voxel DTM (20cm voxel resolution)

4.2 Roads

The data set, which contains the centrelines (MultilineStringZ) and the number of lanes as an attribute, is used. There is one MultiLineStringZ for each centreline. The (X,Y)-s of the centre lines are exactly 1m apart. The steps are as follows:

- the MultiLineStrings are rasterised into a 20cm roads raster (2D), forming 8-connected raster lines, using the segments-IDs as pixel values.
- width of roads is chosen according to the lane count of each segment. One line is considered approximately 2m. Pixels in the (x,y) raster that are near to the roads become road, and receive the ID of the nearest road as the pixel value. This is done by 2D distance transform: roads raster is input and the output are two rasters A and B, in which each non-road pixel receives (A) the distance to the nearest road and (B) the ID of the nearest road. A second 2D distance transform provides again two raster A and B, but the B is the value of the nearest road pixel (which is the z-coordinate).
- the lane count allows to select how many pixels of both sides of the line should become road. Line count is 1,2 and 3 in this case, it is selected that 1 lane corresponds to 10 voxels, 2 to 15 and 3 to 20 voxels. Then a raster road map is generated. Z value of the new road voxels can be added in two ways: 1) rasterise the segments into another 2D raster, by interpolating Z between consecutive vector points in the MultiLineStringZ or 2) overlay the rasterized (X,Y)’s with the 20cm grid DTM. In this paper, the second approach is used. Function OTover integrates the two layers of roads and terrain, following the overlay principle as described above, i.e. ‘road’+’terrain’=’road’ (Figure 3). In this case the roads are considered surfaces and have a thickness of one voxel.

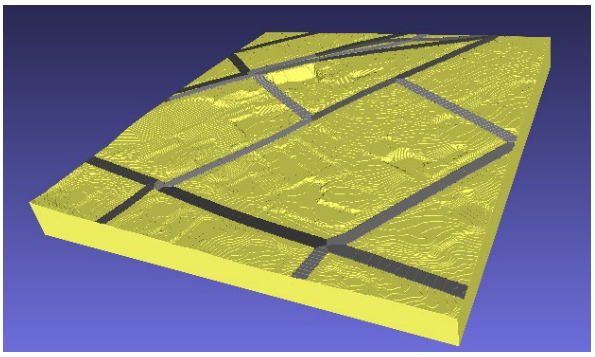


Figure 3: Roads integrated in the DTM (20cm voxel resolution). Grey shading represents different road IDs.

4.3 Trees

Trees are represented with their 3D coordinates and a height attribute. However, we do not use the original Z-coordinate. The tree points are ‘projected’ on the DTM and the resulting z is considered. Then the 3D shapes of the trees are generated with the help of five parameters as in Figure 4.

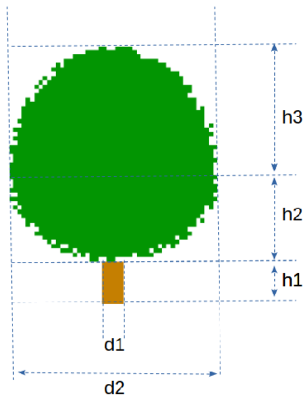


Figure 4: Tree parameters to compute a 3D shape of a tree.

Tree types of trees are created with respect to the height of the tree as follows (Figure 4): trees up to 10 m (r1=0.5, r2=6, h1=2, h2=5 , h3=3) trees with height 10m to 20m (r1=1, r2=10, h1=2, h2=8, h3=10 and trees between 20m and 30m (r1=1, r2=10, h1=3, h2=5, h3=15). These values are randomly selected without considering species or other biological characteristics. They indeed can be related to the types of species. Figure 5 displays the resulting trees integrated with the DTM and the roads.

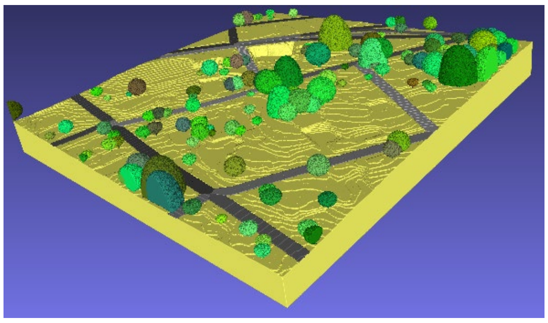


Figure 5: Trees and roads integrated in the DTM (20cm voxel resolution)

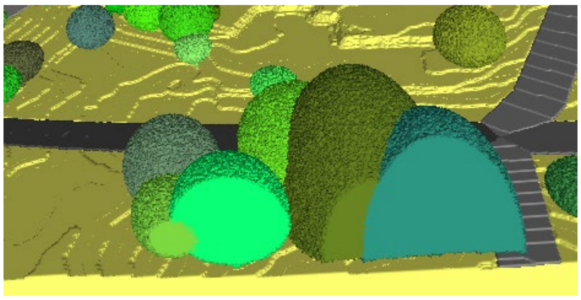


Figure 6: 'Non-overlapping' trees (20 cm voxel resolution). Colours are automatically generated and correspond to tree IDs

The rule that mixed voxels are not allowed is valid also for tree. When the trees are created the voxels in the tree layer are checked by the function OTover whether they are already 'occupied' by another tree. If this is the case the newly created tree is created only on the 'free' of trees voxels. Figure 6 illustrates the results of this rule. Only the first created tree has a complete crown. All neighbouring trees are somewhat wrapped around it.

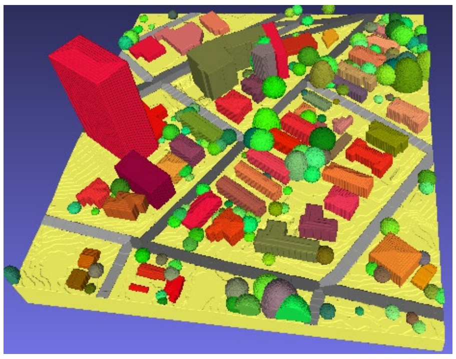
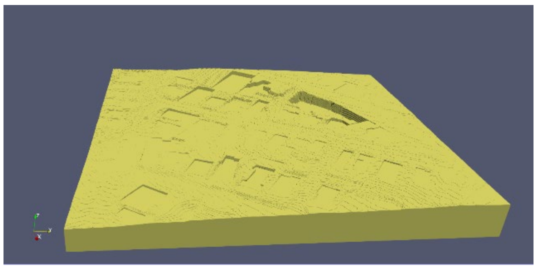


Figure 7: Final integrated 3D scene in L1 (40cm voxel resolution)

4.4 Buildings

Buildings are represented in the input vector database as water-tight solids, i.e. triangulated objects. The function OTtri (not explained here) voxelises triangle by triangle, in such a way that every (x,y,) in the footprint of a triangle is represented by a voxel. Another more intuitive explanation could be that if a triangle edge 'intersects' a voxel that the voxel is assigned value 'building'. The z-value of each voxel is interpolated from the z-values of triangle points. This is independent of the orientation (steep or flat) of the triangle. As this operation is performed in x,y, vertical triangles are omitted, i.e. the result is roofs and footprints. Every column in the voxel space that is a building has non-zero value at two z-positions, at the bottom and the top of building. OTprofile is applied to fill 'building' values to all voxels between footprints and roofs.

The buildings are recorded in a 'building' layer, which is the overlayed with the DTM. Like the overlay between terrain and roads, 'building' value is leading and takes over when mixed voxels are detected. The function used in OTover (Figure 8). The mixed voxels can be all voxels' values in the existing scene: DTM trees, air and roads. In the selected data sets mixed voxels were not detect between buildings and roads.


a) resulting DTM after overlay with buildings

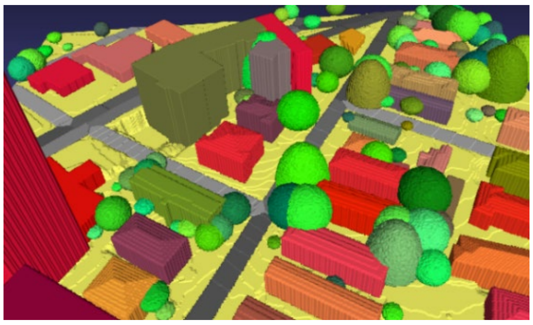
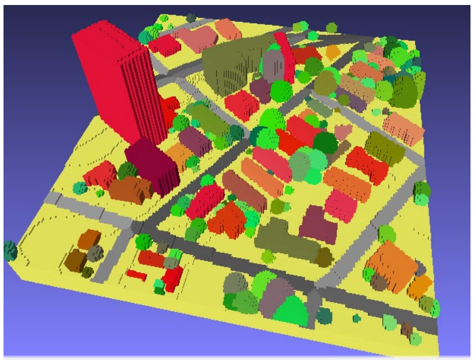

b) overlay between buildings and trees

Figure 8: The results of voxel overlay between a) DTM and buildings and b) buildings and trees. In both cases the function OTover assigns 'building' value to mixed voxels. (20cm voxel resolution).

Figure 8 illustrates the voxels of L0 in the octree. As mentioned previously when creating the octree, all levels are automatically created and available for examination or analysis. The rougher levels result in a much smaller 3D raster and might be considered for performing analysis on very large territories, e.g. urban heat island, climate analysis etc. These levels can be also used as LODs for visualisations to speed up the navigation and exploration of the voxel scenes. Figure 9 illustrates the roughest levels L3 and L4, which have significantly low number of vertices compared to L1 (Figure 7), which has 2,798,125 vertices.


a) L3- voxel resolution 1.60m (177,483 vertices)

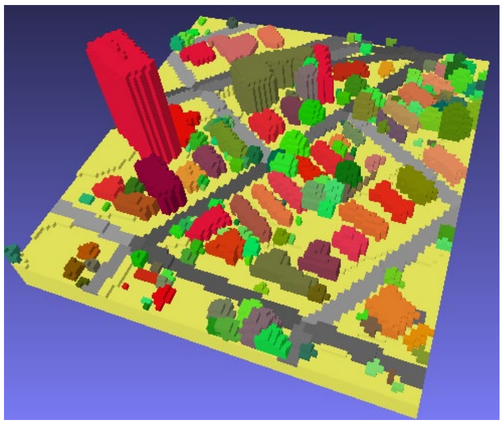

b) octree L4, voxel resolution 3.20m (45,215 vertices)

Figure 9: Examples of the roughest levels L3 and L4, which corresponds to cells 1,60m (a) and 3.20m (b)

5. Large voxel data sets

The octree data structure has been specifically designed for processing very large dataset (Gorte 2023). Processing large voxel models has always been challenging. Either the voxel scene does not fit in the computer memory or when it (just) fits the processing has been time consuming. It is also well known that accessing voxels by searching in a file is slower than by addressing an array in memory. The developed octree data structure allows to process large data sets with a very good performance.

The same procedure as explained above has been completed for an area of 3200x3200x140m (16000x16000x704 voxels). Figure 10 displays the 3D model.

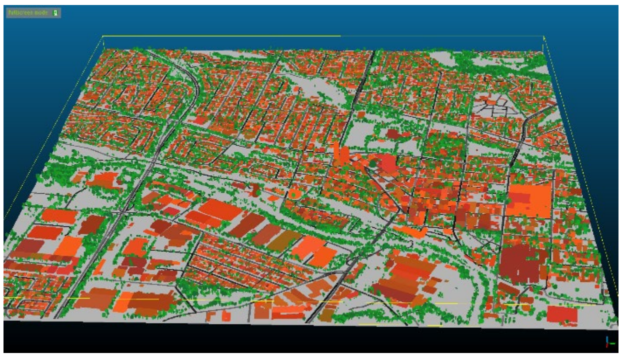


Figure 10: 3Draster model of Liverpool on an area of 3200 x 3200 x 140m

6. Import to ArcGIS Platform

ArcGIS platform was chosen to demonstrate the versatility of the integrated voxel dataset due to its world-wide adoption in GIS community. Primarily, ArcGIS Voxel Layer seemed to be a 3D layer type suitable for the purpose. However, supported data formats for importing into a Voxel Layer are still limited at the time of this experiment. Therefore, a workflow based on point clouds was developed for the purpose. We generated point clouds from our integrated voxel data into the .las format using Web Mercator coordinate system. ArcGIS Pro can directly import .las file into 3D scene. Its internal spatial index and on-the-fly resampling allow 3D rendering with high frame rate (Figure 11).

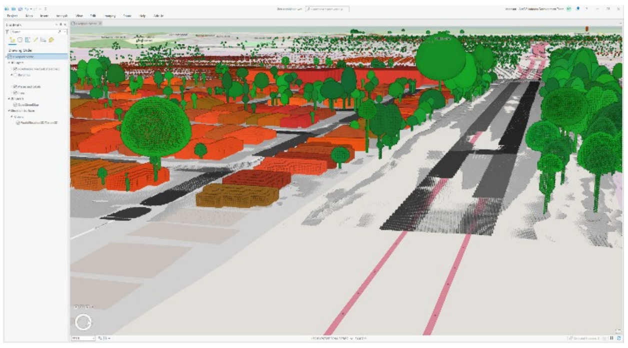


Figure 11: Integrated voxel data imported as a point cloud layer into ArcGIS Pro

2D and 3D base maps can be added to the 3D scene where transparency can be set allowing overlaying and comparison of the data (Figure 12 and Figure 13)

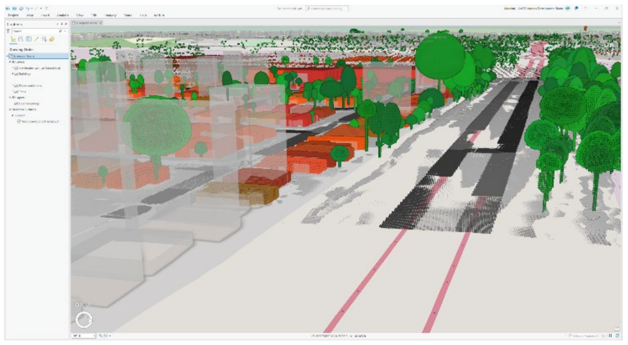


Figure 12: Integrated voxel data with OpenStreet 3D base map in transparent in ArcGIS Pro.

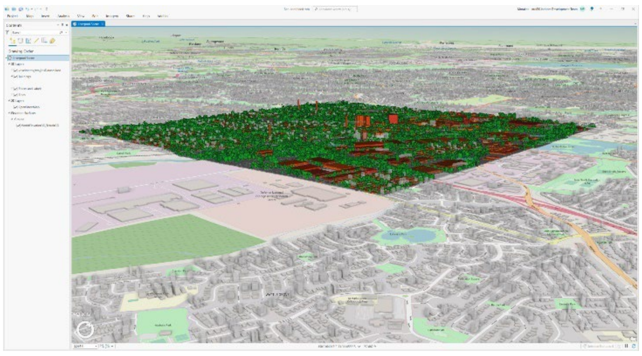


Figure 13: Full extent of the integrated voxel data with Openstreet 3D base map in ArcGIS Pro.

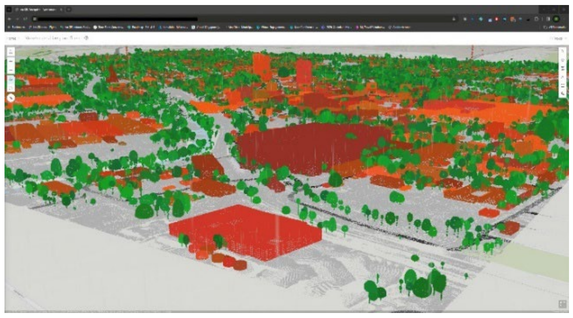


Figure 14: 3D scene of the integrated voxel data visualised in a web browser. A weather effect (rain) was added.

To make the 3D scene with the integrated voxel data available for public access, it can be simply published as a scene service to either ArcGIS Enterprise or ArcGIS Online. The publisher can decide whether the scene service is available to authenticated users or accessible publicly (Figure 14).

7. Discussion

It should be noted that the two most important aspects that must be considered in the data integration process are: connectivity of objects and resolving mixed voxels. As stated previously, voxel objects are considered volumetric. However, many objects may have parts that are smaller than the selected voxel resolution or are represented as lines and surfaces in vector data sets. In such cases the objects can get enlarged to preserve the connectivity. Applying different connectivity strategies 6 (faces), 18 (faces and edges) or 26 (faces, edges, and vertices) might play a critical role in some voxelisation algorithms (Nourian et al 2016).

'Mixed' are voxels which have obtained two or three values from different layers after the overlay. In the example above, we had mixed voxels from DTM and buildings, DTM and roads, and buildings and trees. These mixed voxels were easily resolved using intuitive sematic rules, but in many cases additional investigation might be needed. A typical example is mixed voxels, detected after overlaying building and roads. Where is the overlay coming from: the voxel resolution or then incorrect initial data sets?

Similar cases can appear during the voxelisation within one voxel layer. The tree layer above illustrates many intersecting trees. The strategy to have one value per voxel was achieved by keeping the value of the 'first-in' tree. This simplified approach might be not ideal for trees, which normally can penetrate each other, but is still plausible. A more sophisticated trees overlay would be needed when the volume of the individual tree crowns must be considered, e.g. for simulations (Xu et al 2022). Intersections within one layer between other objects like building, would require an inspection of the original data sets or refining the voxel size.

A completely different situation appears, when the overlay creates 'cavities' between objects, i.e. the voxels between two objects have a value 'air'. Most illustrative example is a building footprint that doesn't touch the DTM, i.e. the building (could be also a tree or a road) 'flies' above the DTM. These issues would also require specific procedures: filling the cavity either with 'building' or 'DTM' values. Which value would be used will depend largely on the accuracy of the data sets. If the building data set is considered more accurate, the cavity could be filled with 'DTM' values. The 'filling' procedures can readily be performed with the operator OTprofile.

As already mentioned, the accuracy of the data sets may significantly influence the data integration process. In our case we assumed the buildings are more accurate than the DTM and therefore the DTM have been reshaped. It might happen that DTM is the more reliable data set and the footprints of the building must be modified to fit the DTM. Note, the z coordinate of trees was not considered in this experiment. But the tree data set might be more correct (if each tree location is surveyed) and the z-coordinate should be preserved.

We will continue to work on the octree data structure and the concept of layers. In this paper the layers contain only one theme (building, transportation, etc). Having the IDs as a value for the voxels allows to maintain a rich set attributes in separate tables. However, many objects contain other objects. Buildings consists of rooms and corridors. Rooms contain furniture, windows, and doors. If attributes need to be stored per each nested object, it seems unavoidable to create dedicated layers for doors, windows, etc. A 3D scene of a building with its interior will be created by overlay of all layers: rooms, walls, doors, windows, furniture, etc. This might bring some advantages as to selectively creating building indoors with, e.g. only floor, only rooms and doors, or only windows. But it will lead to a repetition of voxel storage because each voxel will be represented in several layers under a different classification. Further investigations and conceptualisations are needed in this direction.

Another interesting aspect that needs further research is the multiresolution voxel pyramid. We have mentioned that the levels are created automatically and can be used for visualisation and analysis. However, it has not been tested whether a direct voxelisation with e.g. 0.80, 1.60 and 3.20m resolution would yield the same results as the automatic octree generation. Furthermore, the current procedure does not ensure connectivity of individual objects. There is no strategy yet how to keep (semantically) important objects. For example, whether it is possible and how to preserve indoor walls, when going to L3 (1,60m) and L4 (3.20m).

We believe a voxel 3D data integration has a lot of potential. Our demonstration showcases a simple workflow for importing and visualising voxel data within ArcGIS Pro's 3D environment, followed by seamless publication for online access and interactive viewing via web browsers. Such integration may unlock diverse options, including environmental analysis, temperature and air quality monitoring, hazard dispersion simulations, and comprehensive analysis of both underground and above-ground features such as CO2 storage, geothermal heating, and construction projects.

The vector databases can be kept as they are, and the 3D voxel scenes can be created for computations, simulations, and predictions in specific spatial Digital Twins. Another approach would be to retain the voxel layers alongside the vector datasets, thereby extending the utility of voxel data across broader applications.

8. Conclusion

In this paper we have presented an approach for voxel-based 3D data integration. The data sets are voxelised, processed to create 3D layers, organised in an octree data structure and finally the layers are overlayed in a 3D scene. Once the data are voxelised, only a few operations are needed to integrate them to a valid 3D model in which each voxel has only one value. All data are in one environment represent by the very simple data type (voxel), which allows to perform robust matrix operations, instead of vector geometric computations. Intersections and inclusions can then be resolved investigating the semantics and devising rules for assigning correct values. The voxels are organised in an octree data structure with five levels of different resolution, which opens opportunities for visualisation of large voxel data sets. Approaches for fast rendering will be investigated in near future.

References

Aleksandrov, M., A. Diakité, J. Yan, W. Li, and S. Zlatanova, 2019, System architecture for management of BIM, 3D GIS and sensor data, *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., IV-4/W9*, 3–10, 2019.

Aleksandrov, M., S. Zlatanova, and D. J. Heslop, 2021, Voxelisation Algorithms and Data Structures: A Review, *Sensors*, 2021 21(24), 8241

Aleksandrov, M., S. Zlatanova, D. J. Heslop and A. Diakite, 2023, BIM-based connectivity graph and voxels classification for pedestrian-hazard interaction*, Journal of Spatial Science*, DOI: 10.1080/14498596.2023.2281923

Billen, R., A-F. Cutting-Decelle, C. Métral, G. Falquet, S. Zlatanova, and O. Marina, 2015, Challenges of Semantic 3D City Models: A Contribution of the COST Research Action TU0801, *3D Printing: Breakthroughs in Research and Practice, Chapter 16,* pp. 296 – 305

Boguslawski, P., S. Zlatanova, D. Gotlib, M. Wyszomirski, M. Gnat, P. Grzempowski, 2022, 3D building interior modelling for navigation in emergency response applications, *International Journal of Applied Earth Observation and Geoinformation*, Vol 114, November 2022, 103066

Diakite, A.A, L. Ng, J. Barton, M. Rigby, K. Williams, S. Barr, and S. Zlatanova, 2022, Liveable City Digital Twin: a pilot project for the city of Liverpool (NSW, Australia), *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., X-4/W2-2022*, 45–52

Diakité, A. A., and S. Zlatanova, 2016, Valid Space Description in BIM for 3D Indoor Navigation, *International Journal of 3-D Information Modeling*, Vol 5(3), pp 1-17

Döllner, J., Kolbe, T. H., Liecke, F., Sgouros, T. and Teichmann, K., 2006. The Virtual 3D City Model of Berlin - *Managing, Integrating and Communicating Complex Urban Information, 25th International Symposium on Urban Data Management* UDMS 2006 in Aalborg, Denmark, p12.

Erving, A., P. Rönnholm, M. Nuikka, 2009, Data integration from different sources to create 3D virtual model, *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XXXVIII-5/W1*, 55, p7.

Fichtner, F. W., A.A. Diakité, S. Zlatanova and R. Voûte, 2018, Semantic enrichment of octree structured point clouds for multi-story 3D pathfinding, *Transactions in GIS*, 22(1), pp. 233-248

Gorte, B., 2023, Analysis of very large voxel data sets, *International Journal of Applied Earth Observation and Geoinformation*, Vol. 119, May 2023, 103316

Gorte, B., S. Zlatanova, 2016, Rasterization and voxelization of 2-d and 3-d space partitioning, *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLI-B4, 283-288*

Gorte, B., S. Zlatanova, F. Fadli, 2019, Navigation in indoor voxel models. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences IV-2/W5,* 279-283

La Guardia, M., Koeva, M., D'Ippolito, F., and Karam, S., 2022 3D data integration for we-based open source WebGL interactive visualisation, *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLVIII-4/W4-2022*, 89–94.

Ledoux, H., M. Meijers, 2011, Topologically consistent 3D city models obtained by extrusion*, International Journal of Geographical Information Science*, Vol 25(4), pp 557-574

Li, W., S. Zlatanova, J. Yan, A. Diakite, M. Aleksandrov, 2019, A geo-database solution for the management and analysis of building model with multi-source data fusion. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLII-4/W20*, 55–63

Nourian, P., R. Gonzales, S. Zlatanova, K. Arroyo Ohori, A.V. Vo, 2016, Voxelization Algorithms for Geospatial Applications, *MethodsX*, Vol. 3, pp. 69-86,

Staats, B.R., A.A. Diakité, R.L. Voûte and S. Zlatanova, 2018, Detection of doors in a voxel model, derived from a point cloud and its scanner trajectory, to improve the segmentation of the walkable space, 2018, *International Journal of Urban Sciences*, pp.369-390

Wagner, D., M. Wewetzer, J. Bogdahn, N. Alam, M. Pries and V. Coors, 2012, Geometric-Semantical Consistency Validation of CityGML Models, in LNGC: *Progress and New Trends in 3D Geoinformation Sciences,* pp 171-192

Xie, R., S. Zlatanova, J. Lee, M. Aleksandrov, 2023, A motion-based conceptual space model to support 3D evacuation simulation in indoor environments, *ISPRS Int. J. Geo-Inf.* 2023, 12(12), 494

Xie, R., S. Zlatanova and J. Lee, 2022, 3D indoor environments in pedestrian evacuation simulations, *Automation in Construction*, 144 (12), 104593

Xu, H., C. C. Wang, X. Shen, and S. Zlatanova, 2022 Evaluating the performance of high level-of-detail tree models in microclimate simulation, *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., X-4/W3-2022*, 277–284

Xu, W., L. Liu, S. Zlatanova, W. Penard and Q. Xiong, 2018, A pedestrian tracking algorithm using grid-based indoor model, *Automation in Construction*, vol. 92. August 2018, pp. 173-187

Yan, J., S. Zlatanova, M. Aleksandrov, A. Diakite. C. Pettit, 2019, Integration of 3D Objects and Terrain for 3D Modelling Supporting the Digital Twin. ISPRS *Annals of Photogrammetry, Remote Sens and Spatial Inf. Sci. 2019 IV-4/W8*, 147–15

Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., Adolphi, T., Kolbe, T. H., 2018. 3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. *Open Geospatial Data, Software and Standards*, 3(1), 1–26.

Zhao, J., Q. Xu, S. Zlatanova, L. Liu, C. Ye and T. Feng, 2022, Weighted octree-based 3D indoor pathfinding for multiple locomotion types, *International Journal of Applied Earth Observation and Geoinformation*, 112 (8), 102900