Geospatial Research Innovations and Development (GRID)
UNSW Built Environment, L4 R.4036, Red Centre West Wing
Sydney, NSW 2052 Australia

30 June 2022

FINAL REPORT (RG202877)

Liveable City Digital Twin: Analytics for agile decision making

Final report
(Deliverable 9)

Abdoulaye Diakite, Masoud Rahimi, Jack Barton, Michael Rigby, Kate Williams, Sisi Zlatanova

Table of Contents

List of Acronyms

Term	Description
AURIN	Australian Urban Research Infrastructure Networks
BIM	Building Information Modelling
DEM	Digital Elevation Model
CESIUM/ TerriaJS	3D Geospatial visualization platform for the Web
3DCityDB	3D City Database The database schema implements the CityGML standard with semantically rich and multi-scale urban objects facilitating complex analysis tasks, far beyond visualization. 3DCityDB extends the capabilities of PostgreSQL/PostGIS
CityGML	CityGML is an open data model and XML-based data exchange format that can be used to describe urban and landscape objects
DBMS	DataBase Management System
GeoJSON	GeoJSON is an open standard format designed for representing simple geographical features, along with their non-spatial attributes
GRID	Geospatial Research Innovation and Development lab, based at UNSW
IFC	Industry Foundation Classes, a standard file format as used in BIM software
PMS	Pavement Management System
PostgreSQL/PostGIS	PostgreSQL is an open-source relational database. PostGIS is an extension enabling geographic operability on top of the standard database
UNSW	University of New South Wales
LCC	Liverpool City Council

This project is supported by FrontierSI, AURIN, UNSW Research Infrastructure and in collaboration with NSW Spatial Services, Liverpool City Council, Data61, QLD DNRME, Astrolabe group and Spatial Vision.

Overview

A robust city modelling framework is essential if local, state and national governments are to move towards sustainable built environments and work together across complex multi-sectoral problems to drive impact on urban liveability and climate adaptability. This project addresses the lack of a publicly available, broadscale 3D digital representation of the impact of the urban landscape and building design on urban heat islands, street shading and walkability. It develops a demonstration Digital Twin which embeds analytics within a 3D city modelling framework to address critical challenges within the built environment. The project is a the first of three stages on Digital Twin:

- **Stage 1: Precinct Pilot: analytics-aided and standards-based 3D/4D Digital Twin applied to an urban liveability and climate adaptability use case for a precinct in Western Sydney, NSW.**
- Stage 2: State Demonstrator: extended, application-broad Digital Twin, exemplar using existing government spatial data frameworks to embed current solutions and data portals, including data security and licensing.
- Stage 3: National Framework: Bridge between critical spatial data portals such as AURIN Data and Data61 data.

The broad aims of the full Stage 1 are to:

- Demonstrate the application of Digital Twins (DT) as 3D City Models and real time data for decision-making;
- Prototype procedures for maintenance and update of DT content;
- Demonstrate 3D analysis;
- Clarify gaps and issues in data interoperability between national DT portals, proprietary systems and stakeholders' demand/vision;
- Create an open 3D data set that can be used for demonstration of DT and associated analysis tools.

1 Problem statement and goals

Conceptually Digital Twins have the potential to transform the design, management and performance of the built environment. Recently many institutions demonstrated a variety of DTs aimed to support exploration, visualisation and analysis of multi-dimensional data. Algorithms and tools demonstrating the analytical power of DTs and specifically 3D and real-time data streams within an integrated analytical treatment for situational awareness, are still lacking. Three generic information levels can be distinguished: data management, sense making and decision making (Figure 1).

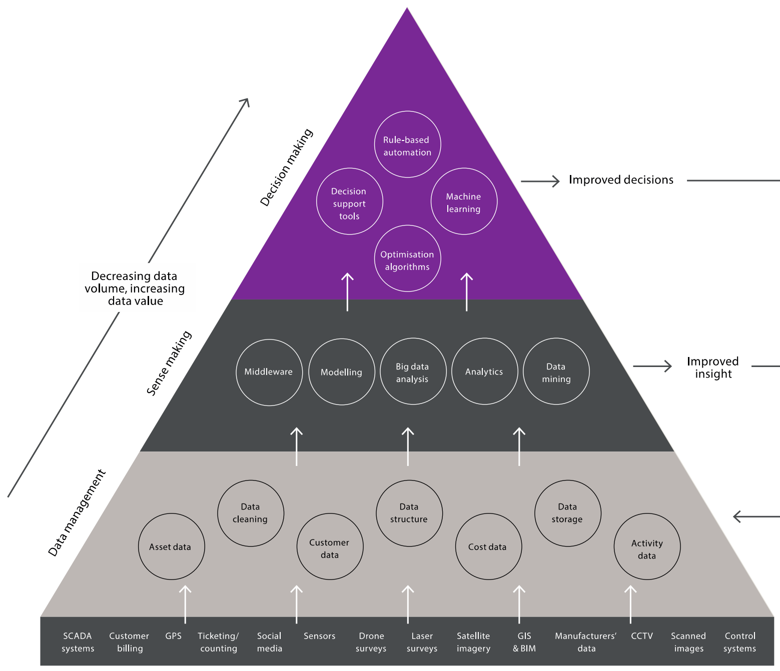


Figure 1: The information value chain: showing the connection between data and better decisions that lead to better outcomes Source: Bowers et al. 2017, cit. in Bolton et al. 2018

This project aims at the middle level, i.e. developing 'sense-making' analytics and appropriate interfaces to demonstrate 3D/4D spatial analysis and augment decision-making capacity using emerging 3D technology. The Precinct pilot stage concentrates on 3D analytics to urban micro-climate and more specifically shadowing and its effect on mobility and walkability. The goals of this project can largely be grouped as follows:

- Designing a generic data model and a spatial schema for storage of information, which utilises international standards
- Establishing procedures and methods to create the Digital Twin from heterogeneous data sets obtained from different data repositories (e.g. NSW Spatial Services, AURIN, Transport for NSW, Data.NSW, BoM, UNSW)
- Providing interfaces to querying DTs and visualising the information in different front-ends (e.g. CESIUM and QGIS)
- Preparing a set of operational algorithms, corresponding functions and operations to enable 3D and real-time data streams within an integrated analytical treatment for situational awareness.

- Developing shadowing algorithms, which will allow the analysis on shaded areas, which can support the decision-making and urban planning.
- Developing procedure for update of the information in the DT.

The project aims to illustrate benefits and reveal challenges in using Digital twins for decision making and sustainable development. Having well-structured information, supported by appropriate management and visualisation tools is expected to further demonstrate the potential of Digital Twins:

- Client applications can be substantially streamlined and maintained up-to-date with essential criteria autogenerated.
- A better understanding of the assets and the ability for the owner to assess risk objectively and in detail – resulting in lower premiums for compliant applications.
- A better evidence-base in the event of litigation or dispute.

The project has been completed through four Work Packages (WP) namely: Building DT, Use of DT, Update of DT and Industry stakeholder review. The following sections provide details on the work completed within Building DT, Use of DT and Update of DT packages.

The project test area is the Central Business District (CBD) of Liverpool city, Sydney. Data sets have been provided by NSW Spatial Services, AURIN, Geoccience Australia and Liverpool Council.

2 Building Digital Twin

The first step involves the collection and integration of the data sets in a standardised way, their storage and access management, as well as their visualization in several interfaces. We have followed an open software approach, which brings significant benefits when low cost solutions are envisaged (Li et al 2020). The core of the system architecture is a common spatial schema and a database management system (DBMS), which provides numerous benefits to the robust management of data (Li et al 2019). Figure 2 illustrates the system architecture that has been adopted for implementation, which is the basis of the whole project.

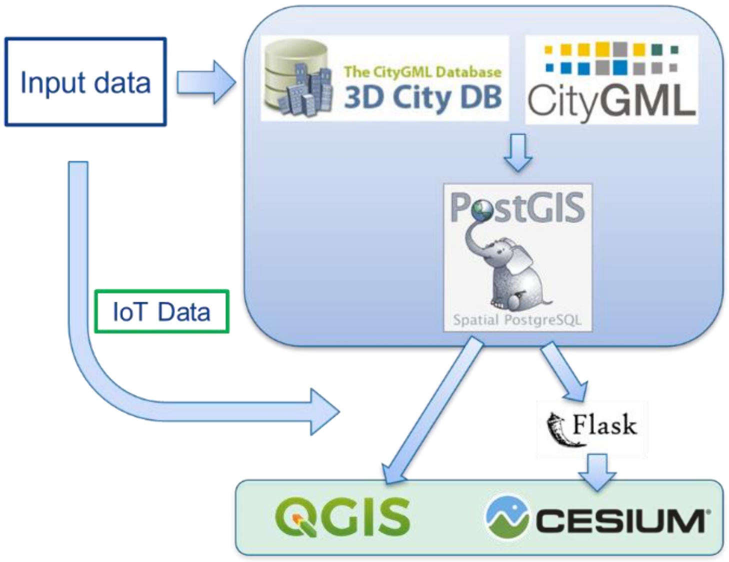


Figure 2: System architecture.

At the core of the system architecture is a 3D City Database (3DCityDB) that was chosen as a geo database to store, represent, and manage the data using a standard spatial relational database approach (Yao et al 2018). Its database schema implements the CityGML 2.0 standard (Figure 3) with semantically rich and multi-scale urban objects (Kolbe et al 2005). The 3DCityDB platform is under 3rd party development and serves as a proof of concept to demonstrate that all project information may be stored in a standardised format and attached to 3D city objects.

3DCityDB has been implemented with PostgreSQL/PostGIS, which is a freeware an opensource relational database. This approach facilitates powerful queries and analysis tools directly on the stored data at a database level and hence independently on the visualisation or editing front-end using applications, such as Cesium, QGIS or ArcGIS. It is also well supported and documented and offers convenient tools to import and export data in different formats.

QGIS and Cesium were chosen for visualisation and GIS software for complementary reasons. The choice of QGIS is motivated by its direct compatibility with Postgres/PostGIS data along with its wide range of powerful tools allowing manipulation and analysis. However, the 3D visualisation capabilities of QGIS are still very limited, which is the reason why Cesium was selected for more advanced visualisation and frontend interactions. As Cesium cannot directly communicate with the database, an API was established to connect with PostgreSQL using the Flask python library.

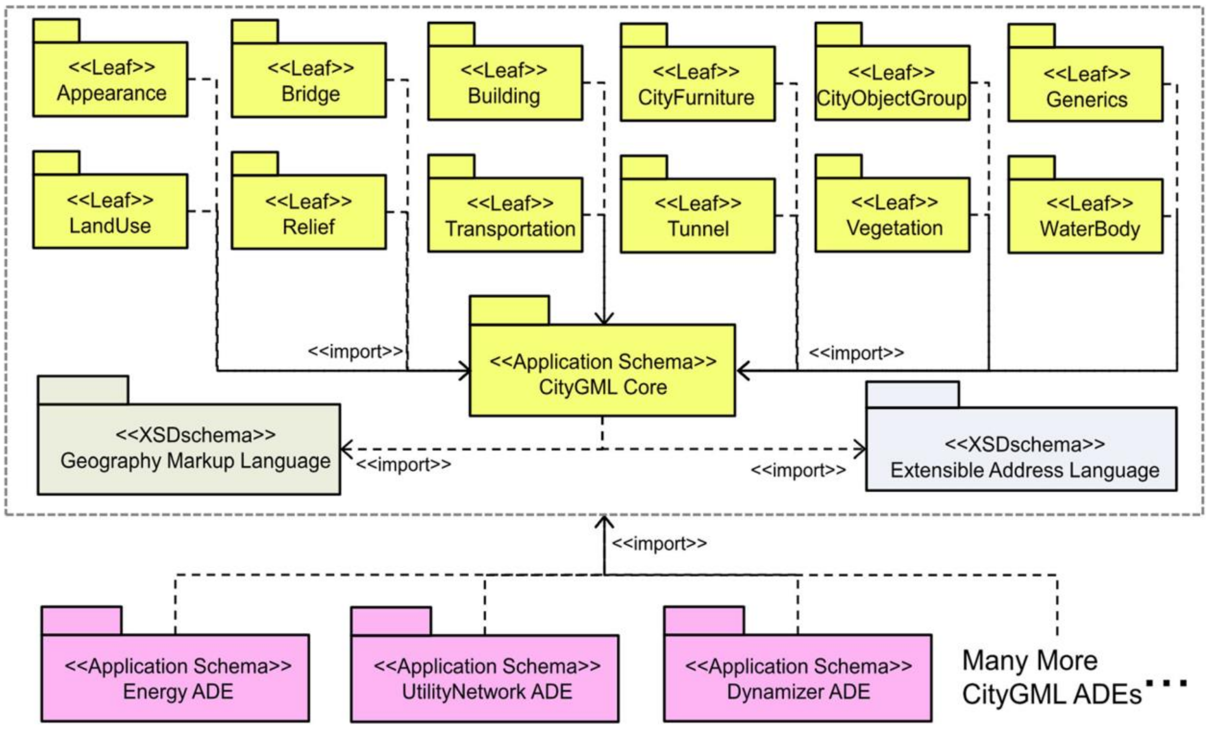


Figure 3: Overview of CityGML modules (Yao et al 2017)

2.1 Data sets and their import into the database

Several spatial datasets have been obtained for the test area of City of Liverpool, which correspond to four classes of CityGML: Building, Transportation, Relief and WaterBody (see Table 1). As these all have different sources, data structures, objects, and descriptions, a critical task for their integration and storage was mapping classes and attributes to CityGML and importing them into 3DCityDB. Through this course of work, a number of new data sets were discovered and added.

Building	**Relief**	**Transportation**	**WaterBody**
Buildings3D (Spatial Services)	**DEM (grid)** (Spatial Services)	**Railway3D** **RoadSegment3D** (Spatial Services) **PMS (2D)** **Road Cadastre (2D)** (Liverpool City Council & AURIN)	**Hydrolines3D** (Spatial Services) **Polygons (2D)** (Geoscience Australia)

Table 1: Initial input data used for the project.

As mentioned above, 3DCityDB faithfully relies on CityGML classes and attributes. For its establishment and efficiency with a relational database, a few differences need to be introduced. In the following sections, we will discuss the mapping of the obtained datasets to classes of 3DCityDB.

2.1.1 Building

Attributes matching

The Buildings3D dataset describes 3D geometries of buildings throughout the city of Liverpool (see Figure 4) that are represented in a way that is comparable to the Level of Detail (LoD) 2 of the CityGML standard. Features of the dataset contain 51 attributes in total, plus their geometric information, *geometry*. While many of these can fit into CityGML's Building class, the latter is only characterised by 34 attributes, including *geometry*.

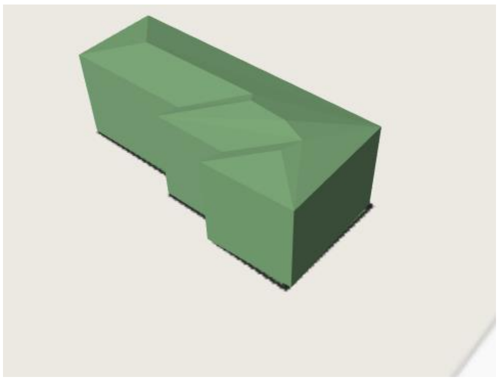


Figure 4: Example of a building from the dataset.

Direct matching between attributes of the dataset and CityGML is pretty limited however, with, only three CityGML attributes identified as being a direct match (four if we consider *id*):

Buildings3D attribute	CityGML attribute
ROOFFROM	roofType
BLDGHEIGHT	measuredHeight
Geometry	Lod2MultiSurface

The small number of direct matches must be considered with the following:

- about 20 of the matching CityGML building attributes are dedicated to other LoD representations
- attributes that do not match can still be stored within CityGML via use of the generic table, _GENERICATTRIB.

Semantic enrichment

The input dataset does not provide semantic information related to the building components. The lack of such information would result in a poor CityGML model, as one of the main strengths of the standard lies on its support of semantics. Therefore, we reconstructed the semantic information that should be associated with the 3D building dataset. As the Buildings3D dataset is a LoD2 model, the faces of the buildings need to be classified into 3 main semantic classes: RoofSurface, GroundSurface and WallSurface*.* The classification is performed using a face orientation approach in which vertical faces are considered as walls and the rest are considered as roof if they lie above the centre of gravity of the building and ground otherwise. This gives us a richer dataset where components, such as roof, can be specifically queried, which could be useful for applications such as solar exposure estimation.

Insertion to 3DCityDB tables
For a model like ours, 5 tables of 3DCityDB need to be filled:

- CITYOBJECT
- BUILDING
- CITYOBJECT_GENERICATTRIB
- THEMATIC_SURFACE
- SURFACE_GEOMETRY

The CITYOBJECT table is the main table of 3DCityDB and every feature of the model needs to be registered in it. For the rest, the *building* table is the one specific to the building classes and that will take the matched attributes (*id*, *roofType* and *measuredHeight*) but not the geometry. The latter goes to the SURFACE_GEOMETRY table where each face is stored as a separate entry (polygon) along with its specificities. The THEMATIC_SURFACE is the table that links the classified surfaces and their corresponding buildings. Finally, the CITYOBJECT_GENERICATTRIB table is where all the attributes that did not find a direct match go, while maintaining a reference to the building that they belong to (through *building_id*). Additional tables would have been considered if the Buildings3D dataset contained texture information (e.g. SURFACE_DATA, APPEARANCE, etc.)

2.1.2 Transportation

Attributes matching:
Both the roads and railways datasets correspond to the transportation model of CityGML/3DCityDB. In the standard, the transportation schema is defined by a superclass _TransformationObject which can aggregate three other classes:

- TransportationComplex,
- TrafficArea, and
- AuxiliaryTrafficArea.

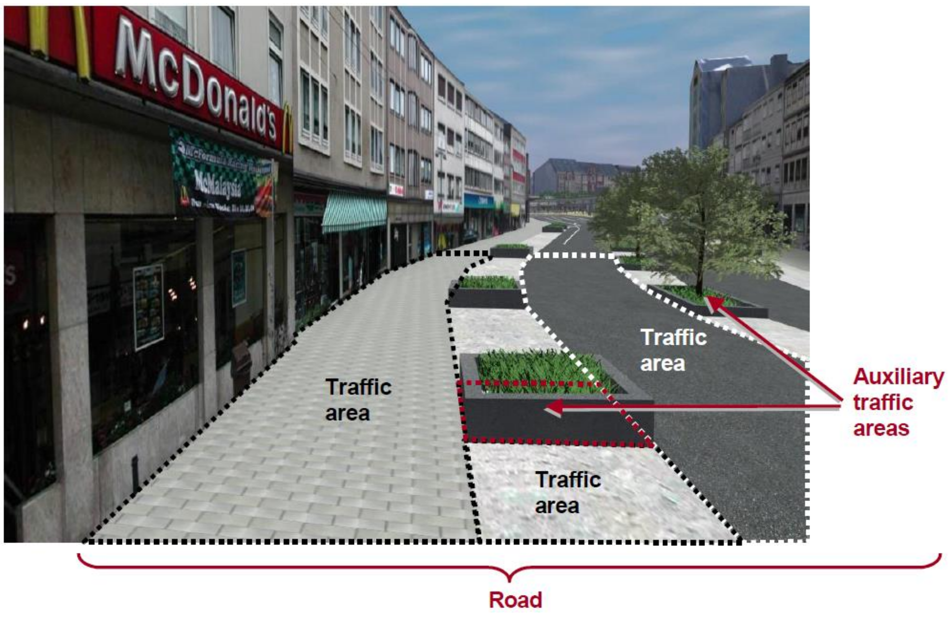


Figure 5: Representation of a TransportationComplex in LOD2 in CityGML: a road, which is the aggregation of TrafficAreas and AuxiliaryTrafficAreas (source: OGC CityGML 2.0 specifications).

TransportationComplex is the main class to represent roads, tracks, railways, squares, etc. It is composed of the parts TrafficArea and AuxiliaryTrafficArea, as illustrated in Figure 5. In our case, the roads and railway features are represented by line geometries, which means they do not provide enough details to be classified into TrafficArea and AuxiliaryTrafficArea. Therefore, we only consider the class TransportationComplex and its attributes.

A review of the input data showed that TransportationComplex has 10 attributes (13 in 3DCityDB), while both our road and railway datasets contain 35 attributes in total. Apart from the geometry and potentially the IDs, no direct matches can be identified with CityGML and we will need to rely on the use of generic attributes so as to not lose any information.

Insertion to 3DCityDB tables:
Unlike the 3D buildings, no further processing or enrichment of the data is necessary here. The information is inserted in the database as is with the following three tables altered:

- CITYOBJECT,
- TRANSPORTATION_COMPLEX and
- CITYOBJECT_GENERICATTRIB.

The CITYOBJECT table takes entries for the same reason as explained above. Value entries to the TRANSPORTATION_COMPLEX table are dedicated to the columns *id*, *objectclass_id* (which is 44 for roads and 45 for railways) and *lod0_network* for the geometry data.

Additional consideration was taken with respect to the road dataset and its attributes such as *functionhierarchy, classsubtype,* etc. which allows to distinguish between the types of road (e.g. pathway, local road, etc.). We could thereby categorise pathways as tracks in CityGML/3DCityDB (*objectclass_id* = 43). The complete list of values for the *objectclass_id* attribute is provided in the documentation of 3DCityDB.

As done earlier with Buildings, all other attributes that did not have a direct match are stored in the CITYOBJECT_GENERICATTRIB table.

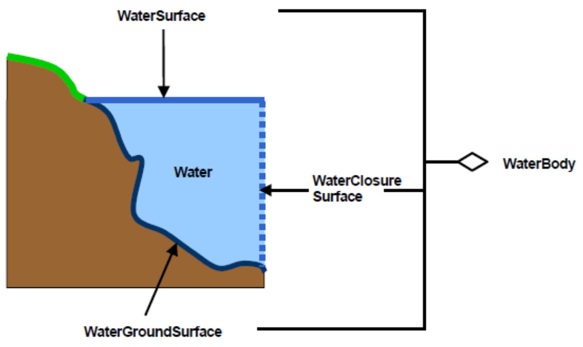


Figure 6: WaterBody according to CityGML (CityGML 2.0)

2.1.3 WaterBody

Attributes matching:
Another dataset of Liverpool provided to us describes the hydrolines traversing the city. Such kind of data corresponds best to the WaterBody class of CityGML. Water bodies can be described with a lot of detail thanks to other classes for surface information, such as WaterSurface, WaterClosureSurface or WaterGroundSurface. However, similarly to the transportation datasets, the hydrolines are only represented using the basic geometriy (LineString) of the water bodies in question. Therefore, the class Waterbody is the only one that can handle them as it provides possibilities to store MultiCurve geometries.

In terms of attributes, this scenario is similar to transportation discussed above, with 12 attributes of WaterBody not directly match with the 23 attributes of our dataset, except for *geometry*.

Insertion to 3DCityDB tables:
Here again, no further processing or enrichment of the data is necessary. The information is inserted into the database using the three default tables:

- CITYOBJECT,
- WATERBODY and
- CITYOBJECT_GENERICATTRIB.

Every water entity is registered in the CITYOBJECT table, like any other feature of the model. Then the *id*, *objectclass_id and lod0_multi_curve* columns of the WATERBODY table are updated accordingly. Again, all the attributes that did not find a direct match are stored in the CITYOBJECT_GENERICATTRIB table.

2.1.1 Relief

The datasets provided to us at the beginning of the project included the DEM of LCC, which is represented in a 1m x 1m raster grid obtained from the ELVIS platform [1]linked from the ANZLIC Foundation Spatial Data Framework [2](FSDF - Elevation and Depth). However, such raster data is not suitable for a polygon-based algorithm such as our 3D shadow estimation. Therefore, we generated a first TIN version directly from the raw DEM, as illustrated in Figure 7.

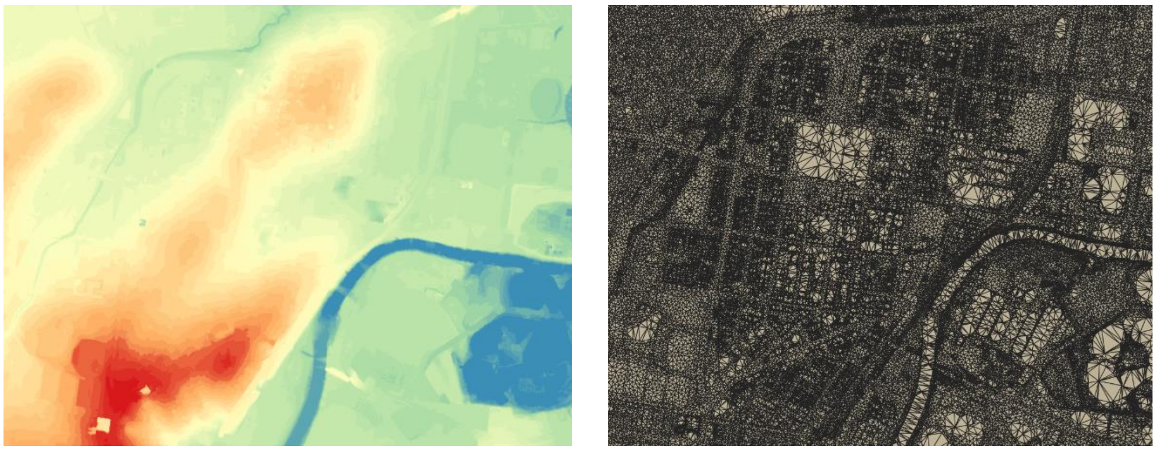


Figure 7: Coloured DEM (left - blue is the lowest elevation and red the highest), derived TIN (right).

The TIN in Figure 7 (right) considered the footprints of the buildings only. The other features that can be distinguished are due to their contrast in elevation with the rest of the terrain (e.g. between the ground and the waterbodies), but they were not explicitly considered. Despite a higher accuracy, this TIN contains approximately 5.8 million triangles and is not efficient to work with and most applications. Therefore, we decided to generate a simpler TIN that would minimise the number of polygons while including all the features of importance (buildings, roads, railways, and waterbodies). All these features are available and already stored in the DBMS of the DT. But as discussed in the next subsections, not all of them are in a ready-to-use (polygonal) form for generating the TIN.

2.1.2 IoT Sensor data

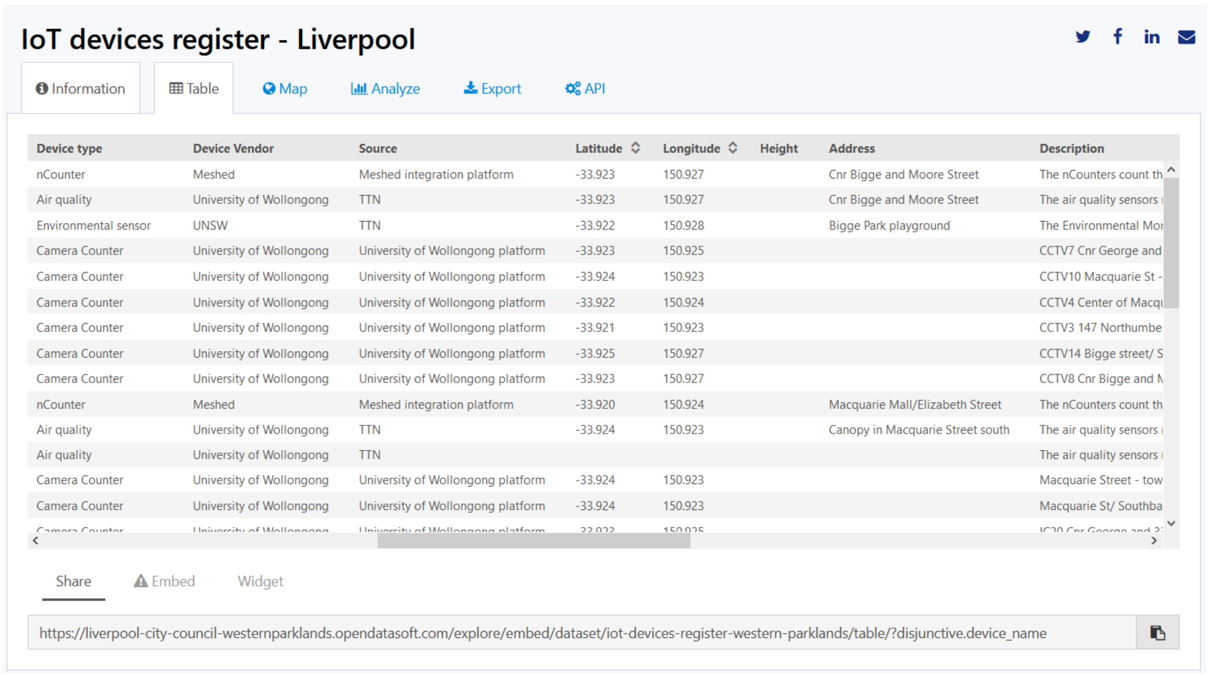
A wide range of sensors are already deployed and functional in the test site of the project (CBD of Liverpool city). As mentioned above, we were granted access to the sensor's data through the Liverpool City Council IoT and Open Data Platform API. A catalogue of 14 datasets is available, covering different themes. Two themes are of interest for this project:

- Environmental Protection and
- Roads, Parking and Transport.

The Environmental Protection theme contains datasets about air quality and other environmental information (heat stress index, ozone, brightness, etc.). The Roads, Parking and Transport theme contains datasets about people and vehicle counting, car park maps (in the CBD) and car sharing sites.

[1] https://elevation.fsdf.org.au/
[2] https://www.anzlic.gov.au/resources/foundation-spatial-data-framework

Additionally, the trajectories of people, bicycles and cars extracted from several CCTV, mounted at crossroads, were made available for the project.



Device type	Device Vendor	Source	Latitude	Longitude	Height	Address	Description
nCounter	Meshed	Meshed integration platform	-33.923	150.927		Cnr Bigge and Moore Street	The nCounters count th
Air quality	University of Wollongong	TTN	-33.923	150.927		Cnr Bigge and Moore Street	The air quality sensors
Environmental sensor	UNSW	TTN	-33.922	150.928		Bigge Park playground	The Environmental Mo
Camera Counter	University of Wollongong	University of Wollongong platform	-33.923	150.925			CCTV7 Cnr George and
Camera Counter	University of Wollongong	University of Wollongong platform	-33.924	150.923			CCTV10 Macquarie St -
Camera Counter	University of Wollongong	University of Wollongong platform	-33.922	150.924			CCTV4 Center of Macq
Camera Counter	University of Wollongong	University of Wollongong platform	-33.921	150.923			CCTV3 147 Northumbe
Camera Counter	University of Wollongong	University of Wollongong platform	-33.925	150.927			CCTV14 Bigge street/ S
Camera Counter	University of Wollongong	University of Wollongong platform	-33.923	150.927			CCTV8 Cnr Bigge and
nCounter	Meshed	Meshed integration platform	-33.920	150.924		Macquarie Mall/Elizabeth Street	The nCounters count th
Air quality	University of Wollongong	TTN	-33.924	150.923		Canopy in Macquarie Street south	The air quality sensors
Air quality	University of Wollongong	TTN					The air quality sensors
Camera Counter	University of Wollongong	University of Wollongong platform	-33.924	150.923			Macquarie Street - tow
Camera Counter	University of Wollongong	University of Wollongong platform	-33.924	150.923			Macquarie St/ Southba
Camera Counter	University of Wollongong	University of Wollongong platform	-33.923	150.925			IC20 Cnr George and 3

https://liverpool-city-council-westernparklands.opendatasoft.com/explore/embed/dataset/iot-devices-register-western-parklands/table/?disjunctive.device_name

Figure 8: Screenshot of the 'IoT devices register' table from the Liverpool City Council IoT and Open Data platform.

While all those datasets may be of interest at some point during the project, only another dataset provided under City Planning and Amenities theme and named *IoT devices register - Liverpool* was considered at this stage. It provides an exhaustive list of all the IoT sensors installed on the test site, along with their relevant metadata (id, description, device name, location, etc.), see Figure 8.

While the sensed data do not need to be explicitly stored in the 3DCityDB until necessary as this is very resource consuming), only descriptive information about the sensors and their metadata need to be integrated in our city model. This will enable management of explicit information about the sensor devices, their properties and spatial location, which will ensure direct and efficient spatial analysis with the sensed data.

Unfortunately, there is no class that provides direct compatibility with IoT sensors as features in the current version of CityGML (this is expected to change in the coming version 3.0). While they might be considered City Furniture, they are not included now. Meanwhile, one workaround that we adopted is to use the GenericCityObject class (generic city object concept), which like generic attributes, the concept allows for the storage and exchange of 3D objects that are not covered by any explicitly modelled thematic class or which require attributes not represented in CityGML.

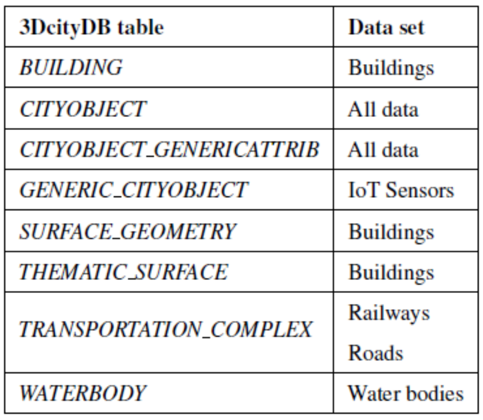
Insertion to 3DCityDB tables:
In 3DCityDB, the sensors' information is inserted in the three following tables:

- CITYOBJECT,
- GENERIC_CITYOBJECT and

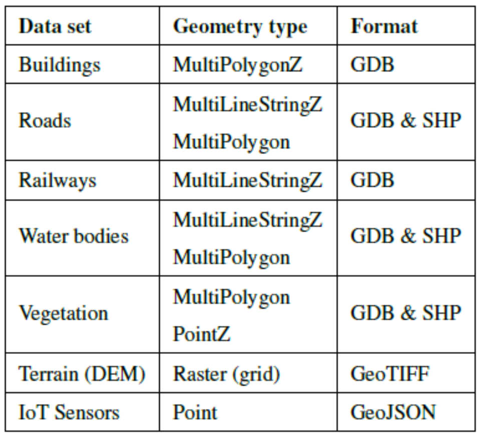
- CITYOBJECT_GENERICATTRIB.

Every sensor device gets registered in the CITYOBJECT table and is provided with a unique *cityobject_id*. The same value is used as *id* in the GENERIC_CITYOBJECT table, where an *objectclass_id* is also recorded with a Point geometry created in *lod0_other_geom* using the longitude and latitude values of the sensor. Because the height values of the sensors were missing in the datasets, an value of zero is temporarily used as the Z dimension. Finally, all the other metadata of the sensors are stored in the CITYOBJECT_GENERICATTRIB table.

The final set of 3DCityDB tables and data sets is listed in Figure 9.



3DcityDB table	Data set
BUILDING	Buildings
CITYOBJECT	All data
CITYOBJECT_GENERICATTRIB	All data
GENERIC_CITYOBJECT	IoT Sensors
SURFACE_GEOMETRY	Buildings
THEMATIC_SURFACE	Buildings
TRANSPORTATION_COMPLEX	Railways Roads
WATERBODY	Water bodies



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

Figure 9: List of 3DCityDB Tables (left) and final list of data (right)

Note: *Since the only currently available way to import spatial data into 3DCityDB is to import a CityGML model using the dedicated Importer tool. Given that none of the datasets are provided in such format, we needed to programmatically insert all the data into the database. For this purpose, we used the PyQGIS plugin, which is a Python interpreter for QGIS and simply loaded all the datasets into QGIS and ran a Python script that accessed the data and connected to an instance of 3DCityDB in a PostgreSQL (with PostGIS extension installed) for data writing. The process is easily reproducible and only requires QGIS (v3 or higher) to be installed and run the script with PyQGIS, assuming the database components are already set (PostgreSQL/PostGIS and 3DCityDB).*

2.2 Integration of objects and terrain into constrained TIN

It is well-know that data from different sources might not fit well with each other due to various issues. This is specifically true for integrating individually created data sets with the terrain (Yan et al 2019), where common issues include objects either sink ing or hovering over the terrain surface. To avoid this effect, transportation and Building objects were integrated in the triangulated surface and the text below describes the step taken.

2.2.1 Data preparation for Building footprints, Transportation and WaterBody polygons

Building footprints

The dataset describing the 3D buildings of LCC is in the form of polyhedral mesh, making it directly suitable for the TIN generation. We performed this task by simply querying the CityGML database (3DCityDB) specifying the parts of the buildings that we need. This is illustrated by the SQL query below:

```
SELECT id, geometry FROM surface_geometry WHERE parent_id IN (
  SELECT lod2_multi_surface_id FROM thematic_surface WHERE objectclass_id = 35
)
```

Here a sub-query is first invoked to generate a list of *lod2_multi_surface_id* (which correspond to the ids of the surface in the SURFACE_GEOMETRY table) from the THEMATIC_SURFACE table, while specifying only features with the *objectclass_id* 35 (corresponding to the GroundSurface class of CityGML). The geometry of the selected features is then extracted from the SURFACE_GEOMETRY table based on their *id* values (Figure 10).

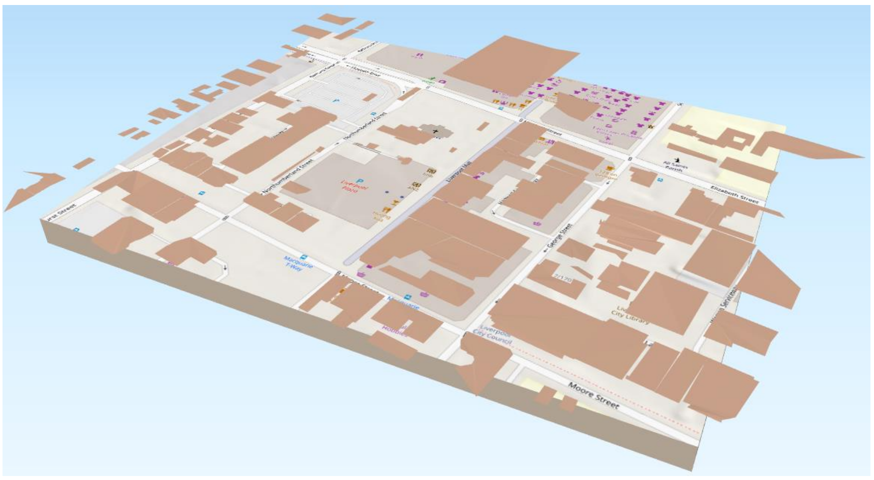


Figure 10: Few selected building footprints (elevated for visualisation purpose).

Similar queries are used to access to the other features stored in the DBMS, taking care of selecting the proper geometry tables and object classes (see report on WP1).

Road polygons

The first road dataset provided to the project was in the form of 3D lines with attributes. While this could have been included in the TIN generation, this would not have resulted in valuable information as the road surfaces, which are critical for transportation and pedestrian motions related analysis, would still be missing. We therefore looked for any polygonal dataset that would depict the roads that the LCC GIS team already had. Hence, two additional datasets were provided to the project representing the road cadastre and polygons, which were extracted from a Pavement Management System (PMS). Figure 11 illustrates these datasets and their differences.

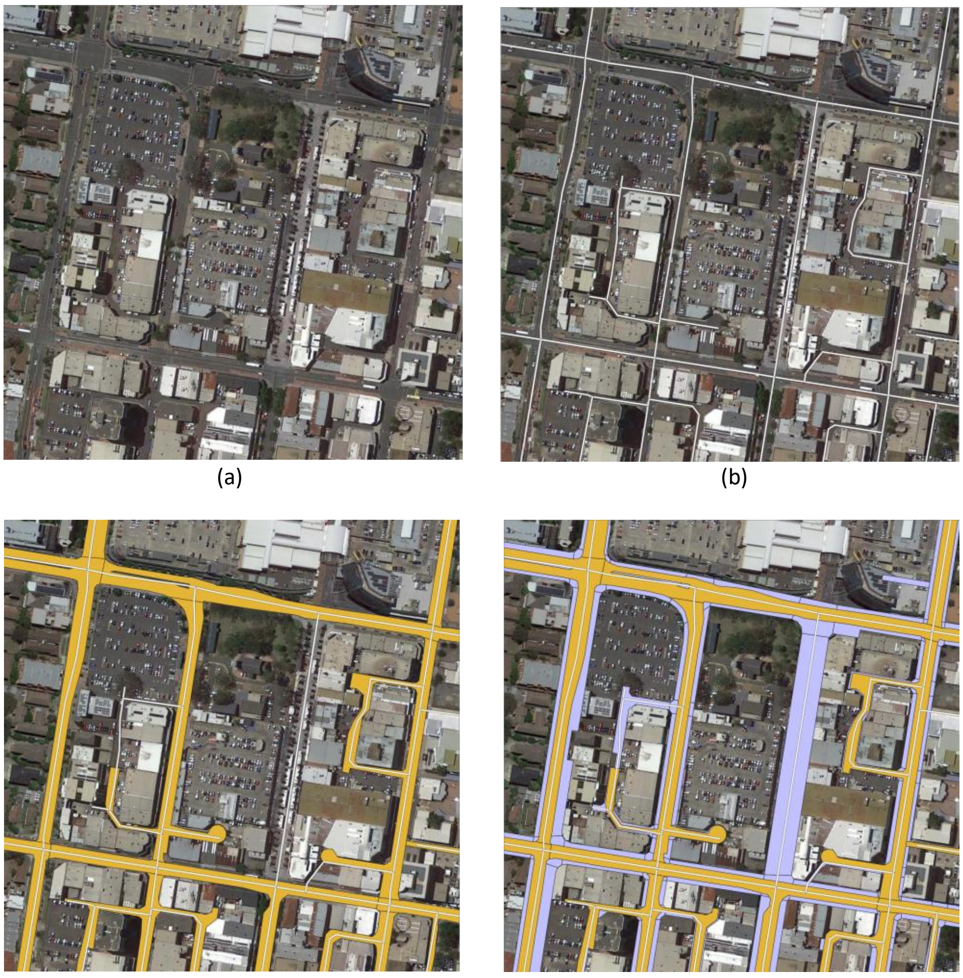


Figure 11: Different Road datasets. (a) Satellite image of a selected area in LCC. (b) 3D road line dataset (white). (c) PMS road dataset (yellow). (d) Road cadastre dataset (purple).

While the 3D line dataset covers the actual roads within the LCC more completely (Figure 11(b)), the cadastre also provides a generalised representation (Figure 11(d))with the PMS data offering more advanced details (e.g. road polygons with complex shapes and excluding the pavements on the sides and in-between lanes, see Figure 11(c)).It is important to note that the latter covers only areas with pavements however, missing thereby important information, such as pedestrian lanes like the Liverpool Mall located at the living heart of the city (see Figure 11 (c) and (d)). Furthermore, for the purpose of generating the TIN, details within the PMS dataset are not of great relevance and the road cadastre provides the the best trade-off as it is a polygonal representation that covers features that would need to be considered in the TIN (e.g. pavements).

***Note:** In the final version of the TIN, we ended up opting for the PMS dataset instead of the road cadastre because we needed to label polygons of the TIN according to their corresponding feature class (e.g., ground, road, building, etc.) and use the ground polygons for pedestrian related analysis. By using the cadastral roads, what we thought was an advantage (i.e., the inclusion of the pavements in the provided polygons) becomes a weakness since the pavements would also be considered as part of the road, excluding it from pedestrian* *accessible areas. The PMS polygons went however through the exact same process than what is described below for the cadastral data.*

Further pre-processing was identified necessary on the road cadastre as the data was represented in 2D and the polygons it contained carried much more vertices than they really required, as well as overlapping with the other datasets in some places. These aspects do not favour proper TIN generation because they would lead to geometric over-complexification on one hand and topological issue on the other.

To address this we started off simplifying polygons by reducing the number of points that describes them. This is a common GIS operation on lines and polygons known as simplification (or generalisation) with the Douglas-Peucker algorithm (Douglas, 1973) being the most used one. It consists of simplifying points based on a given minimum distance threshold between each point. As this approach is not adapted to our problem because our polygons are of different sizes and setting a wrong threshold could lead to heavily distorted polygons, we developed our own algorithm that simplifies points solely based on the angle that they form with their surrounding neighbours and implemented it in QGIS (python). This allowed us to remove only those points that are not bringing relevant details to the polygons' shape (e.g. multiple point on a straight line), as illustrated in Figure 12. A threshold of 5^{o} allowed us to obtain a 98.4% reduction, with a final 4724 points (Figure 12 - right) from an initial sample of 287087 points (Figure 12 - left).

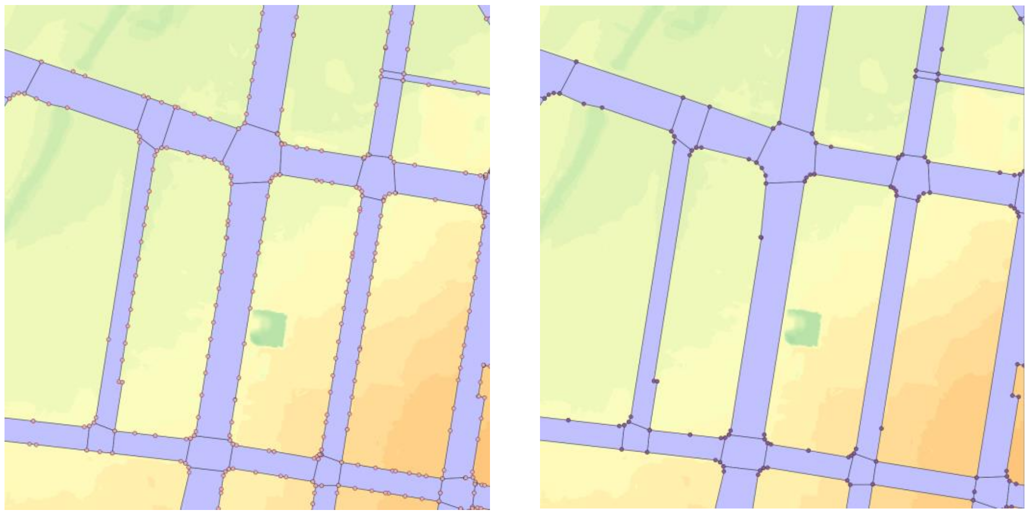


Figure 12: Road polygons simplification before (left) and after (right).

One limitation that could result from such simplification is the loss of precision in terms of height on a long road section when the 3D elevation will be recovered (z coordinate). However, this is mitigated by the high level of subdivision of the dataset which includes polygons for every road intersection.

Another pre-processing step is the removal of parts of the roads that are intersecting with other datasets. This is simply done using the 'Difference' Boolean operation in QGIS against the intersecting polygons, while on the case of the railways, the intersecting polygons were removed and considered as bridges for later enhancing the DT's database (see Figure 13).

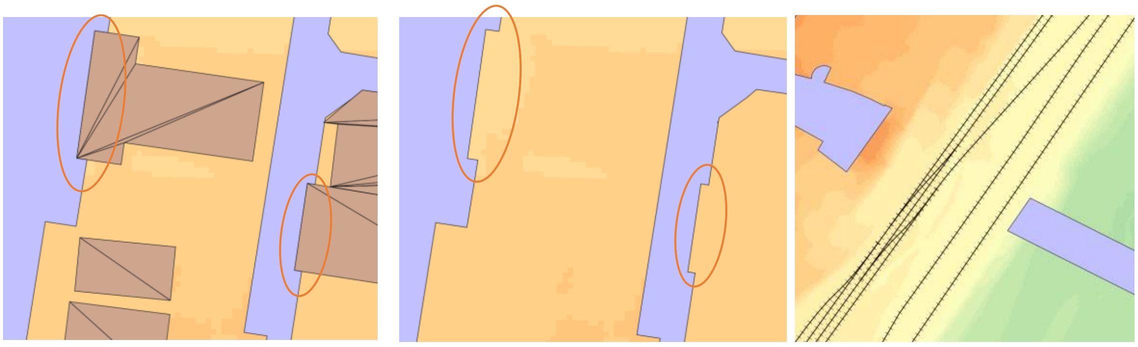


Figure 13: Intersection removal. From left to right: intersection between building footprints and road polygons; result of the difference operation; polygons crossing railways were removed.

Finally, all the polygons of the road dataset were converted to 3D by a DEM-based sampling to recover their elevation. For this again, we developed an in-house python script for QGIS as the original sampling function provided by the software can only handle point inputs rather than polygons.

Railway polygons

Similarly to the road dataset, the railways came to us in the form of 2D line geometry. However, unlike the roads, it is not necessary to seek a polygonal version as this can be fairly derived from the line since one can assume a regular width for the feature. We therefore produced polygons using the buffering tool of QGIS with a standard width of a 1435mm (4 ft $8^{1/2}$ in), as specified by the Australian Bureau of Infrastructure and Transport Research Economics (BITRE, 2019).

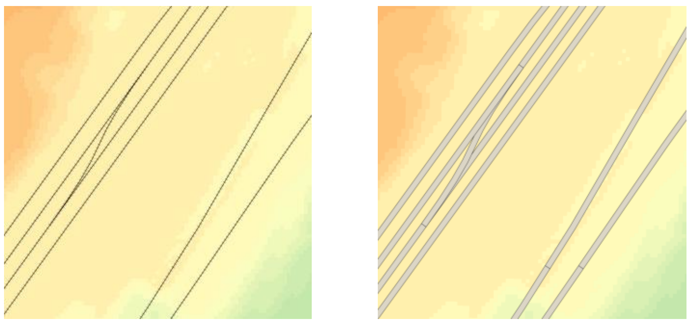


Figure 14: Buffering of the railway dataset to form polygons.

Figure 14 shows the resulting polygons, noting further pre-processing was applied to the polygons (point simplification and elevation sampling) in a way similar to the road dataset, excluding the intersection check with building footprints.

Waterbody polygons

Here again, the original data described 3D hydro-lines rather than polygons of waterbodies. The dataset describes lakes, rivers and creeks as illustrated in Figure 15, however like the roads and unlike the railways, the width of the corresponding surfaces cannot be reasonably guessed and uniformly apply to the curves.

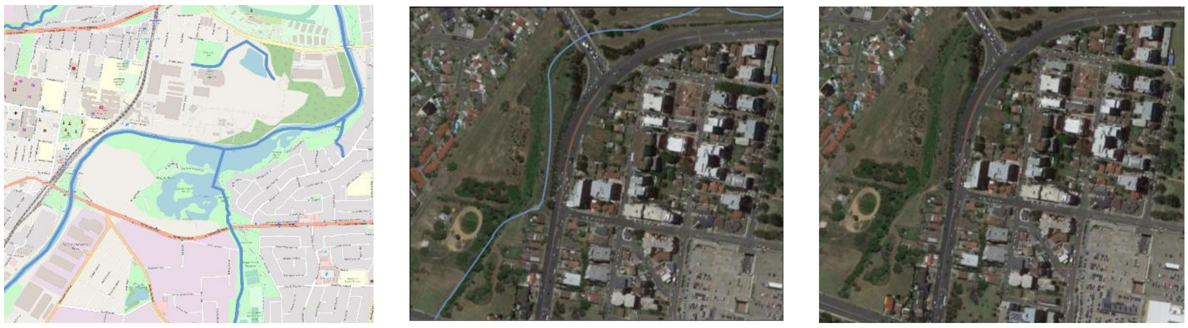


Figure 15: 3D Hydro lines dataset describe lakes (left) and creeks (middle). But the creeks may be dry (right).

After some investigations, we found through AURIN an accurate hydrology polygon dataset of the region provided by Geoscience Australia (GA) (see Figure 16). But unlike the 3D hydro lines data, creeks and other temporary waterbodies were not represented in the data.

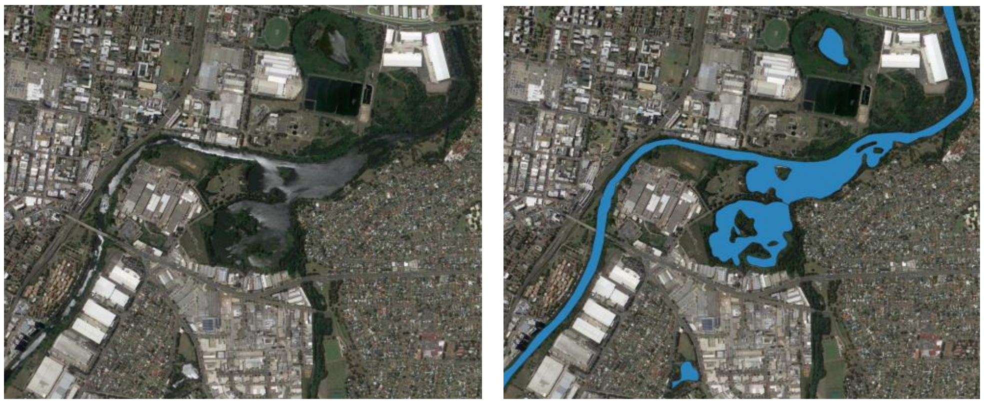


Figure 16: Accurate polygons of the waterbodies found in AURIN and provided by GA.

We therefore decided to exclude the creeks and dealt only with parts described as polygons. Here again, the pre-processing applied to the road and railway datasets was applied.

2.2.2 Generation of the integrated TIN

With all the polygons of the building footprints, roads, railways, and waterbodies ready, we proceeded to generate a TIN that integrates them all using the 3D constrained Delaunay triangulation implementation of PostGIS. For this we used the 3D boundary lines of the features as constraints for the triangulation. The input and the resulting TIN can be seen in Figure 17 and Figure 18.

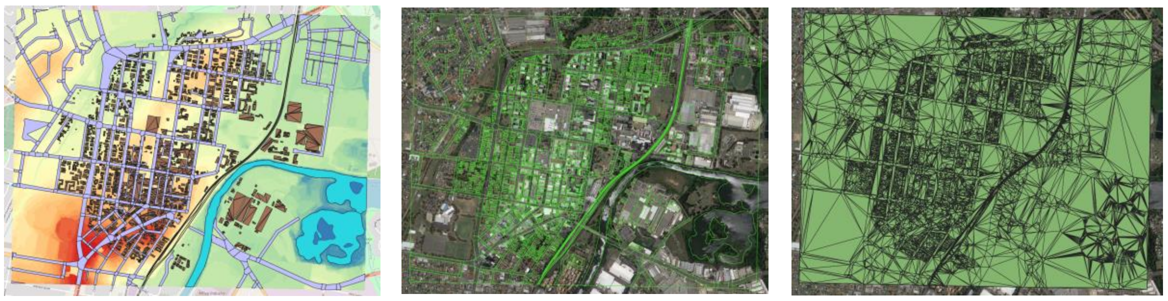


Figure 17: All the features in the selected sample area (left). The extracted 3D boundary lines of the features (middle). Generated TIN (right).

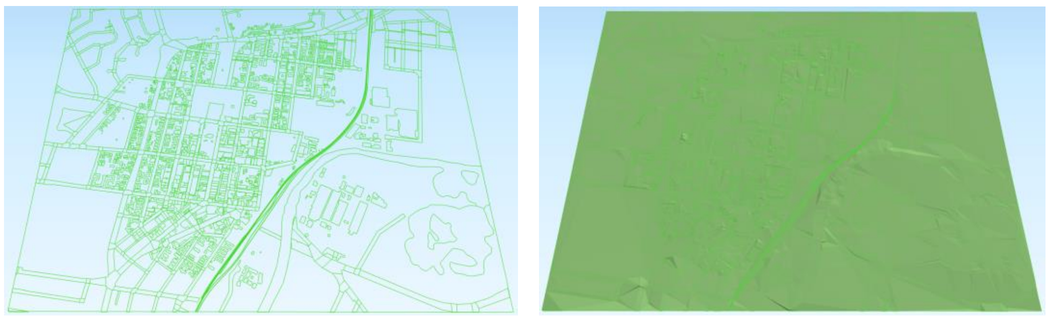


Figure 18: 3D view of the 3D boundary constraints for the TIN computation (left) and the produced TIN (right).

The resulting TIN count of 28K polygons makes it suitable for an efficient use in spatial analysis, while preserving good elevation information for all the features. Figure 19 shows two example views within the full 3D model with all features integrated together, including the new terrain.

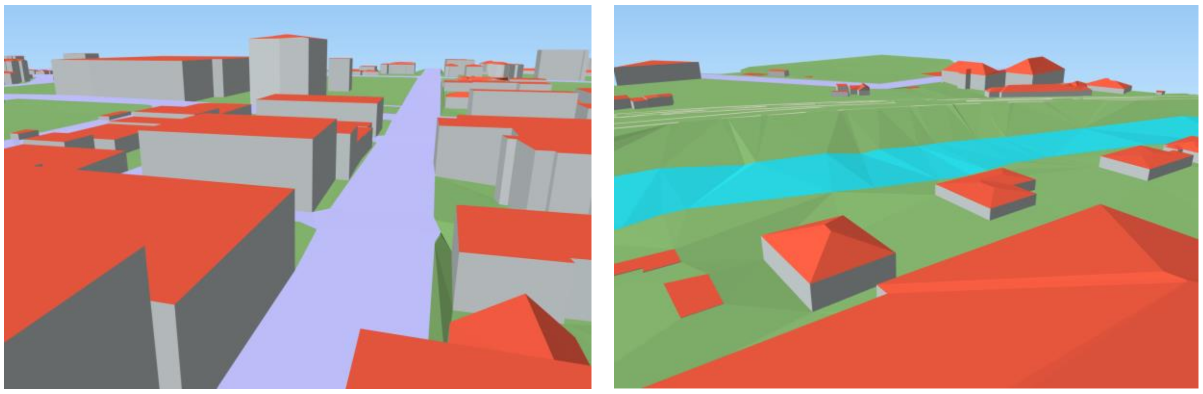


Figure 19: Views of all the features (buildings, roads, water, terrain) in QGIS.

2.3 Visualisation of data in QGIS and Cesium (D2)

The format adopted for our DT storage in PostGIS/3DCityDB enables direct access and visualisation of the data in several open-source or commercial software packages. One of them is QGIS, from which the database can be directly connected to, and features (geometry and attributes) directly read and visualized. Figure 19 illustrates the visualisation of buildings (with roofs and walls coloured according to the semantics), roads, terrain (constrained TIN) and water bodies.

Another tool that communicates with 3DCityDB is a CityGML importer/exporter tool that validates data recorded in 3DCityDB against the CityGML standard as well as importing to or exporting from the database in CityGML format. We used that tool to validate and export the recorded data, noting the GML file can also be visualised with any available CityGML viewer (a list of solutions is available here). Visualisation of the same features with FZK Viewer[3] is shown in Figure 20.

Another option to access the 3DCityDB is from a virtual platform such as Cesium. Cesium is an open source 3D visualisation platform that allows a wide range of spatial data (GIS, BIM, photogrammetry, etc.) in various formats, including CityGML, to be imported and visualised

[3] https://www.iai.kit.edu/english/1648.php

in one environment. The 3D file formats are all converted to 3D Tiles, an open specification format that optimises the streaming of large 3D data through the Cesium viewer. To access to such format, one must use the dedicated web app by creating a *Cesium ion* account (which is the proprietary version of Cesium, not to be confused with CesiumJS, the open-source version) and uploading the data for them to be converted to 3D Tiles and become accessible through a REST API. Once the tiles are created, they can be streamed to CesiumJS. This is a commonly used approach, which was also adopted for the deployment of the NSW Digital Twin[4].

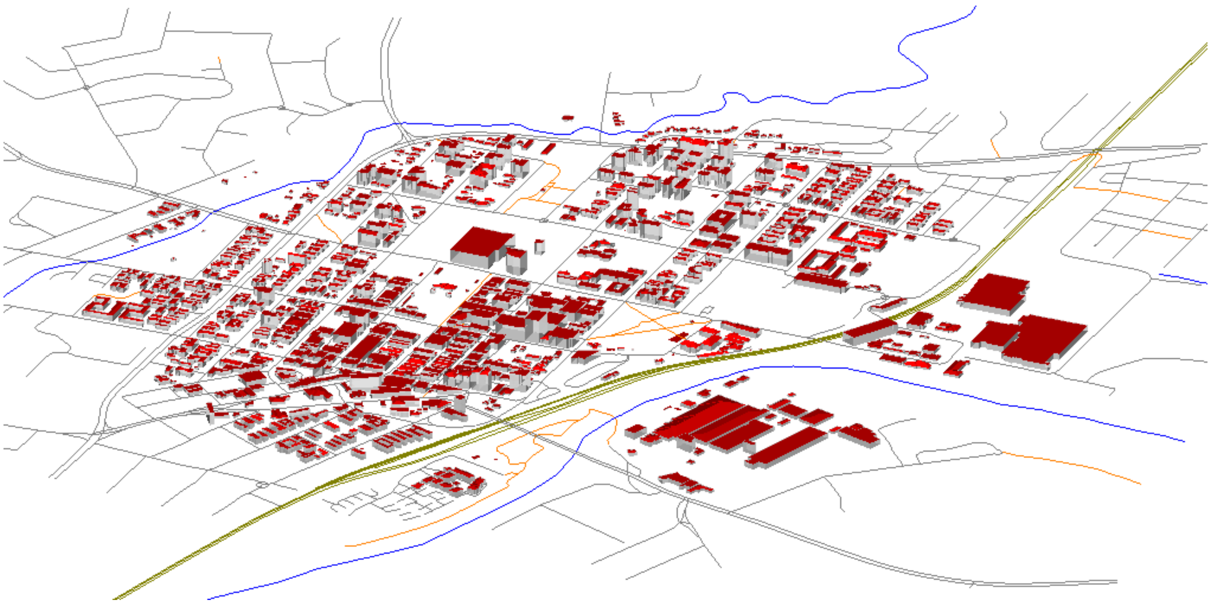


Figure 20: Liverpool CityGML digital twin extracted from 3DCityDB and visualized with the FZK Viewer.

The issue with such file-based approach is that, under the use of a database, export of dump files is always required before sending to Cesium ion for conversion to 3D Tiles. This is not convenient for dynamic databases where information is often updated, as one would expect for the database of city's digital twin. We therefore decided to investigate a way to directly visualize geospatial data in CesiumJS. Besides 3D Tiles, CesiumJS supports several other formats (KML, GeoJSON, OBJ, etc.) but they are all file-based once again and there is no official support of Web Feature Service (WFS). However, as a web interface based on Javascript, Cesium can send and receive HTTP requests to servers. Therefore, with the support of GeoJSON files, an ad-hoc workaround to the file issue was identified that sends the results of database queries formatted as JSON through an API to Cesium. We adopted this approach and used the Flask python library to setup a server API. Figure 21 illustrates the communication established between the client (CesiumJS) and server side containing the database (3DCityDB) and the API (Flask).

[4] https://www.spatial.nsw.gov.au/what_we_do/projects/digital_twin

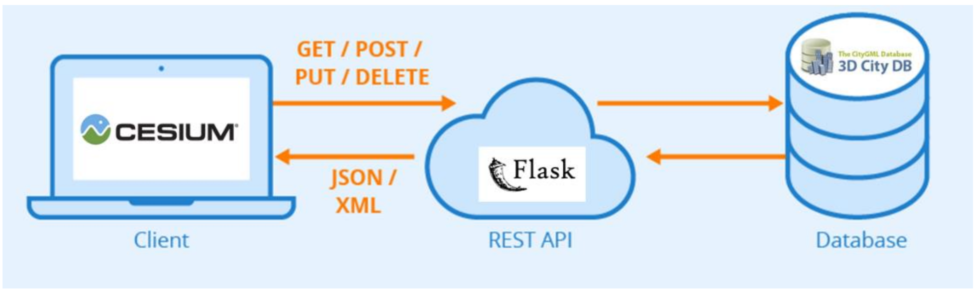
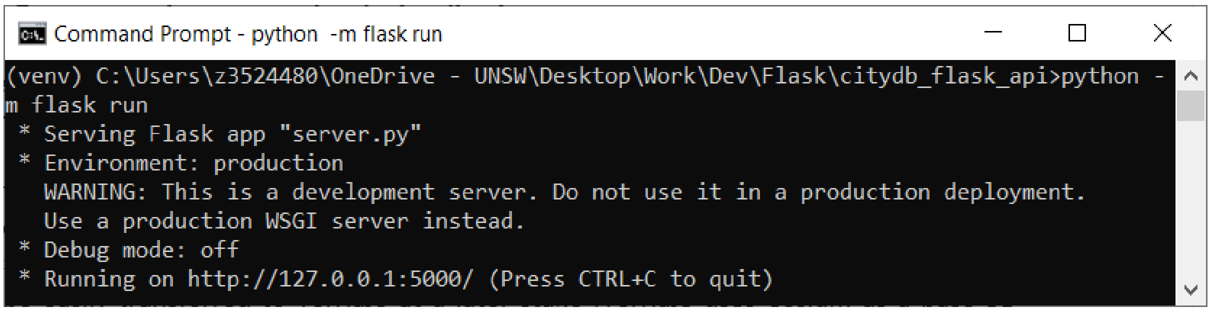


Figure 21: Diagram of the API requests established between Cesium and 3DCityDB through Flask.

The main issue to this approach is the management of the query results in terms of size. The GeoJSON support of Cesium is not optimised and files as small as 10MB can take several minutes to load. Therefore, the size of the request needs to be carefully managed to allow a smooth visualization in Cesium.

2.3.1 Implementation

Server side:
With the database already established, only the Flask API needs to be configured on the server side. The Python library (Flask) needs to be configured to connect to the database and predefined routes need to be set for specific queries. Those routes will later be the HTTP links through which the client will perform API requests to the server.



```
Command Prompt - python  -m flask run
(venv) C:\Users\z3524480\OneDrive - UNSW\Desktop\Work\Dev\Flask\citydb_flask_api>python -
m flask run
 * Serving Flask app "server.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 22: Command prompt of a configured Flask API server.

Once configured, Flask provides a command line window that describes the address from which the server data is accessible from (see Figure 22). For now, we run it only on a local server for test purposes with plans for it to be deployed on a web server at a later stage.

The queries sent to 3DCityDB through Flask should return valid GeoJSON data to be readable by Cesium. To produce such outputs, we rely on the PostGIS ST_AsGeoJSON function that formats the results of the queries. A new and handy feature of PostGIS (from v3.0.0) allows to create GeoJSON from a full database record, meaning a row containing geometry and other attributes. We thereby defined all the API routes needed to connect the UI (Cesium) to the back-end (3DCityDB) through Flask. A typical API request looks as follows:

```
http:localhost:5000/getFeatureIDs?type=Building&number=20&&bbox=150.9219,-
33.9222,150.9256,-33.9208&filterAttribute=measured_height
&filterOperator=ge&filterValue=10
```

This examples returns a list of 20 (`number=20`) *cityobject_id* of buildings (`type=Building`) stored in 3DCityDB, such that those buildings that intersect with a defined bounding box (`bbox=150.9219,-33.9222,150.9256,-33.9208`) with a height greater or equal to 10 (`filterAttribute=measured_height&filterOperator=ge &filterValue=10`). Note that localhost shall be replaced by the web server's address once established unless the server is being run on a local machine. While better ways might exist to stream bigger outputs to Cesium for a smooth visualisation still require further investigation, we adapted the queries to produce feature collections with a limited number of features (around 100 max per query), limiting their size and allowing interactive loading times.

Client side:
From the Cesium's side, a simple call to the API request needs to be performed to obtain GeoJSON data. The latter is then injected in the viewer using the GeoJsonDataSource function. The result is visualised in Figure 23 below.

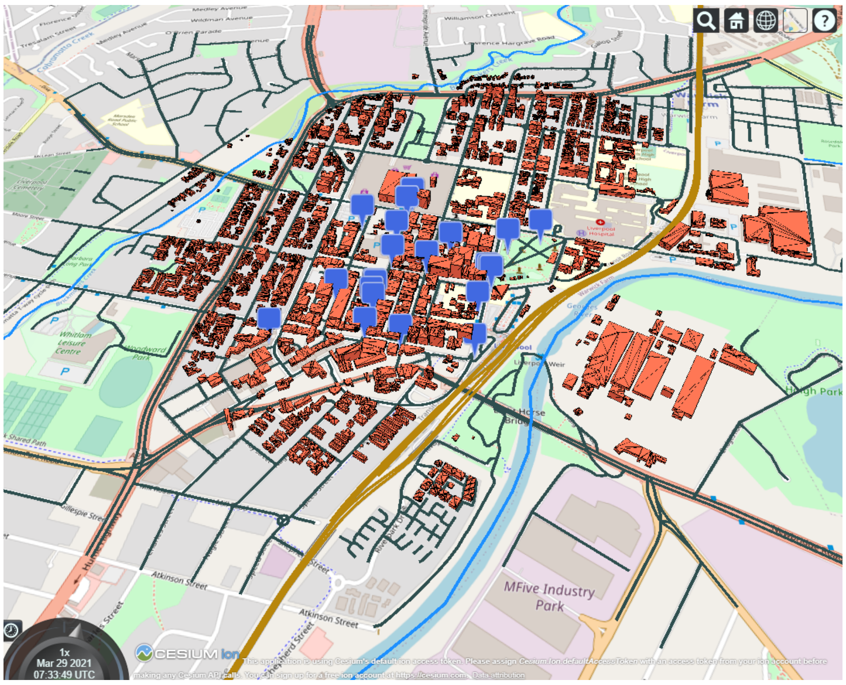
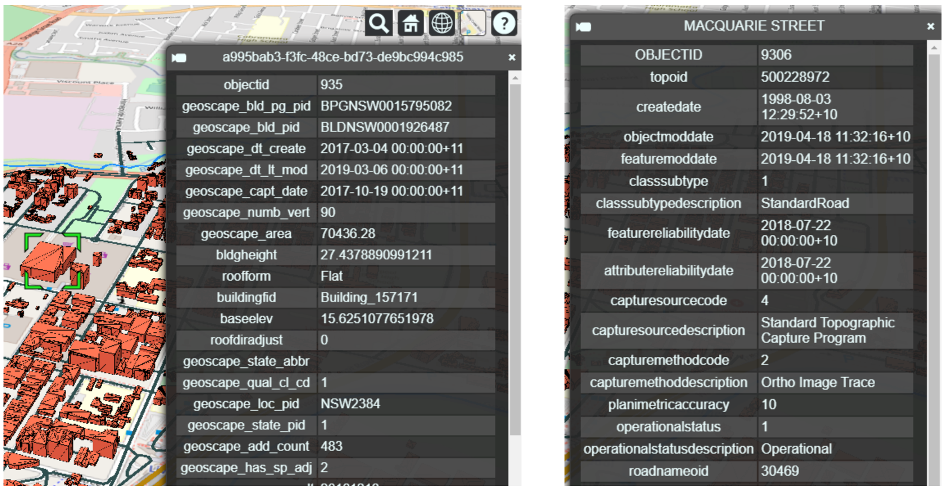


Figure 23: Digital twin of Liverpool (selected area) loaded in CesiumJS from 3DCityDB.

Every feature returned is the result of a direct query to 3DCityDB through the Flask API. Furthermore, all attributes associated to the original data are maintained and displayed when a feature is clicked on, as illustrated in Figure 24.



a995bab3-f3fc-48ce-bd73-de9bc994c985

objectid	935
geoscape_bld_pg_pid	BPGNSW0015795082
geoscape_bld_pid	BLDNSW0001926487
geoscape_dt_create	2017-03-04 00:00:00+11
geoscape_dt_lt_mod	2019-03-06 00:00:00+11
geoscape_capt_date	2017-10-19 00:00:00+11
geoscape_numb_vert	90
geoscape_area	70436.28
bldgheight	27.4378890991211
roofform	Flat
buildingfid	Building_157171
baseelev	15.6251077651978
roofdiradjust	0
geoscape_state_abbr	
geoscape_qual_cl_cd	1
geoscape_loc_pid	NSW2384
geoscape_state_pid	1
geoscape_add_count	483
geoscape_has_sp_adj	2

MACQUARIE STREET

OBJECTID	9306
topoid	500228972
createdate	1998-08-03 12:29:52+10
objectmoddate	2019-04-18 11:32:16+10
featuremoddate	2019-04-18 11:32:16+10
classsubtype	1
classsubtypedescription	StandardRoad
featurereliabilitydate	2018-07-22 00:00:00+10
attributereliabilitydate	2018-07-22 00:00:00+10
capturesourcecode	4
capturesourcedescription	Standard Topographic Capture Program
capturemethodcode	2
capturemethoddescription	Ortho Image Trace
planimetricaccuracy	10
operationalstatus	1
operationalstatusdescription	Operational
roadnameoid	30469

Figure 24: Attributes of the features loaded in CesiumJS (buildings attribute at left and road attributes at right).

2.3.2 User Interface (UI)

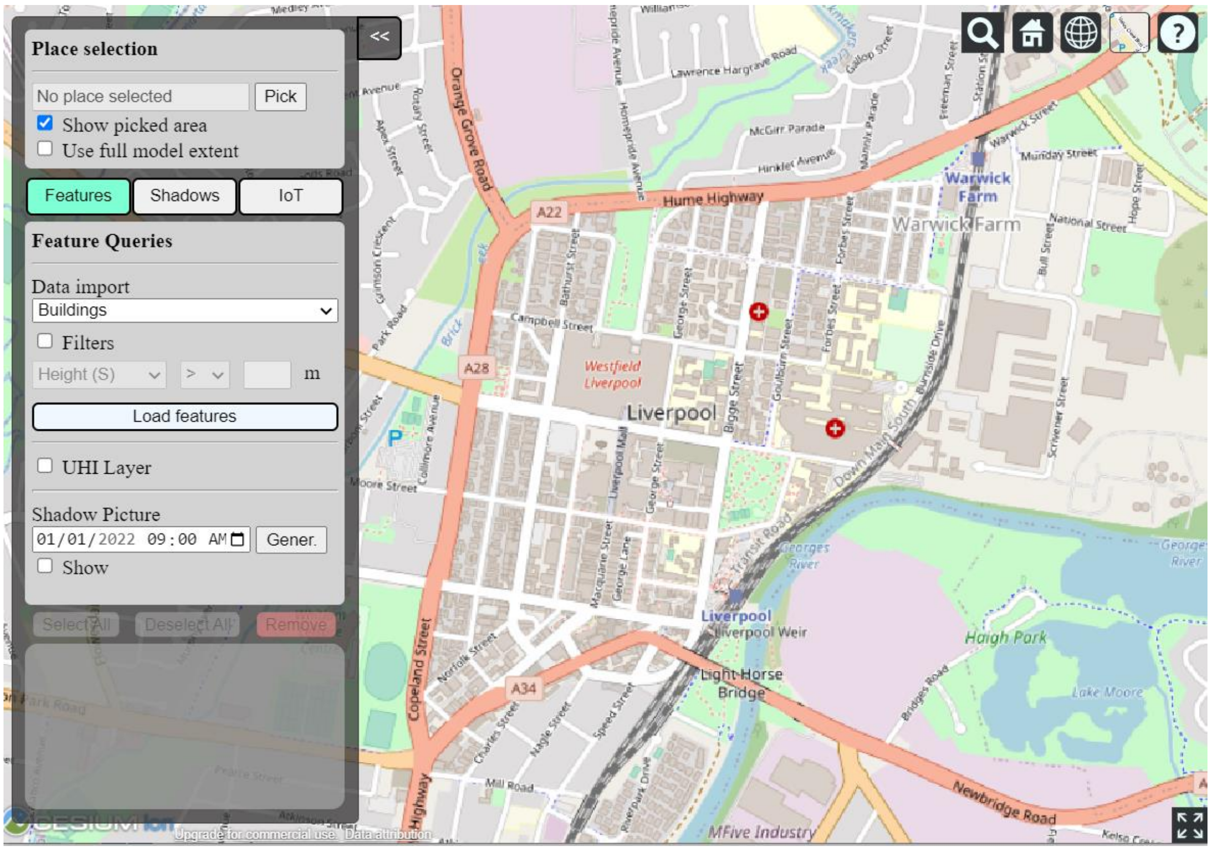


Figure 25: UI developed for the Cesium front-end.

A custom UI was developed to allow users to interact smoothly with the DT. The main interface with the menu buttons in shown in Figure 25, noting the menu is subdivided in 3 main sections, related to the different components of the DT:

- The ***Features*** tab contains the options related to queries of the features of the DT. This includes the import of stored data (buildings, roads, etc.) based on selected filters (e.g, building height, road types, etc.). It also contains some other layers related to Urban Heat Islands (UHI) and shadow analysis, that will be used and discussed in Section 3.
- The ***Shadows*** tab (discussed in Section 3) collects all the options related to the query and visualisation of shadow polygons.
- The ***IoT*** tab (also discussed in Section 3), as its name suggests, gathers options related to the queries and visualisation of sensor related feeds.

Figure 26 illustrates the result of a filtered query for building features.

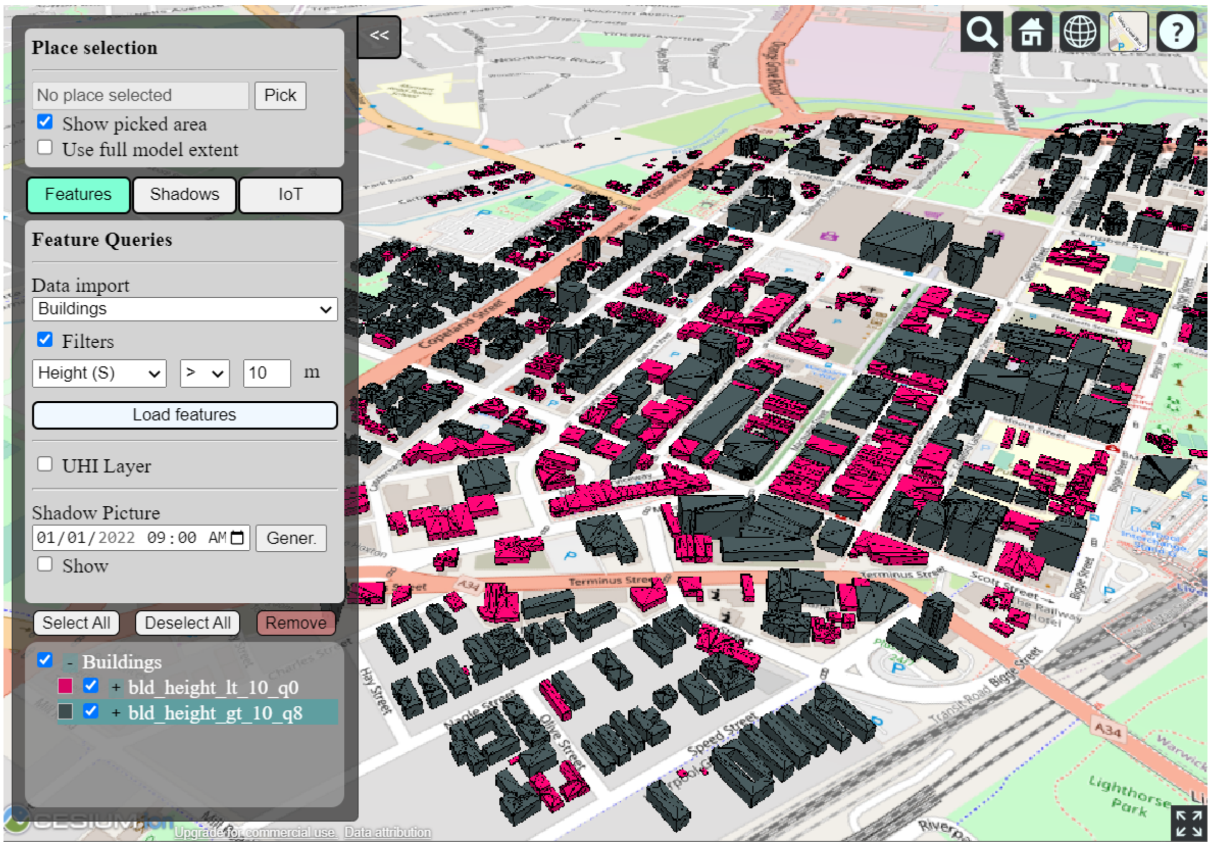


Figure 26: Filtered query of the DT buildings (red for building with less than 10m of height, and dark grey for the rest)

3 Using the Digital Twin

This section illustrates the features of the built DT database created to perform various queries, demonstrates the visualisation and analysis of the IoT sensor feeds installed in the city of Liverpool, and explores the analysis related to shadow coverage.

3.1 Algorithms to access and link sensor data to 3D geometries of DT from a DBMS

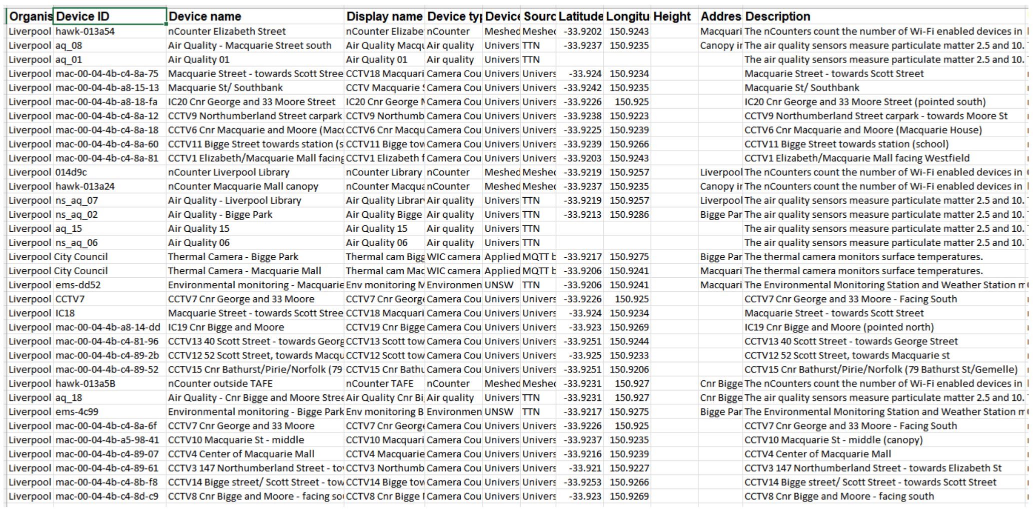
To access the sensor data, we needed to establish a link between the spatial features of the DT and the sensor feeds from the LCC IoT & Open Data Portal[5] to enable their association in visualisation and analysis. As discussed in above in Section 2.1.2, only the sensor locations and related attributes have been recorded in 3DCityDB, while the data from the sensor need to be queried directly through the API provided by LCC. Therefore, our goal here was to create new records in the DB that will help us identify all the DT features that are related to a given sensor feature. This way, when the data of a sensor is queried through the API, another query to the DB will provide the list of features that are affected by the received data.

3.1.1 DB table to link the features

To enable the linking, we needed to create a table that gives correspondence between the sensors and the spatial features. A spreadsheet of all available devices can be obtained

[5] Liverpool City Council IoT & Open Data Platform, https://liverpool-city-council-westernparklands.opendatasoft.com/

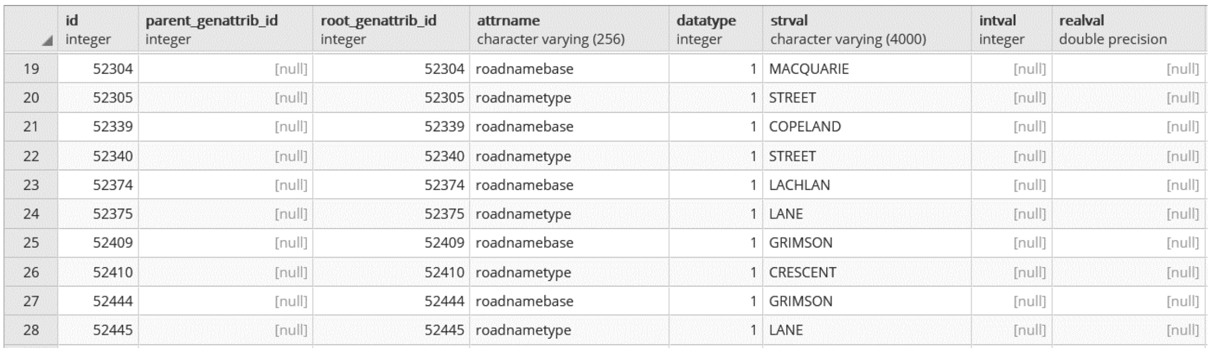
through the LCC data portal (see Figure 27), which was used initially to record the sensor locations and attributes.



Organis	Device ID	Device name	Display name	Device ty	Device	Sourc	Latitude	Longitu	Height	Addres	Description
Liverpool	hawk-013a54	nCounter Elizabeth Street	nCounter Elizabe	nCounter	Meshed	Meshe	-33.9202	150.9243		Macquari	The nCounters count the number of Wi-Fi enabled devices in
Liverpool	aq_08	Air Quality - Macquarie Street south	Air Quality Macqu	Air quality	Univers	TTN	-33.9237	150.9235		Canopy ir	The air quality sensors measure particulate matter 2.5 and 10.
Liverpool	aq_01	Air Quality 01	Air Quality 01	Air quality	Univers	TTN					The air quality sensors measure particulate matter 2.5 and 10.
Liverpool	mac-00-04-4b-c4-8a-75	Macquarie Street - towards Scott Stree	CCTV18 Macquari	Camera Cou	Univers	Univers	-33.924	150.9234			Macquarie Street - towards Scott Street
Liverpool	mac-00-04-4b-a8-15-13	Macquarie St/ Southbank	CCTV Macquarie S	Camera Cou	Univers	Univers	-33.9242	150.9235			Macquarie St/ Southbank
Liverpool	mac-00-04-4b-a8-18-fa	IC20 Cnr George and 33 Moore Street	IC20 Cnr George M	Camera Cou	Univers	Univers	-33.9226	150.925			IC20 Cnr George and 33 Moore Street (pointed south)
Liverpool	mac-00-04-4b-c4-8a-12	CCTV9 Northumberland Street carpark	CCTV9 Northumb	Camera Cou	Univers	Univers	-33.9238	150.9223			CCTV9 Northumberland Street carpark - towards Moore St
Liverpool	mac-00-04-4b-c4-8a-18	CCTV6 Cnr Macquarie and Moore (Mac	CCTV6 Cnr Macqu	Camera Cou	Univers	Univers	-33.9225	150.9239			CCTV6 Cnr Macquarie and Moore (Macquarie House)
Liverpool	mac-00-04-4b-c4-8a-60	CCTV11 Bigge Street towards station (s	CCTV11 Bigge tow	Camera Cou	Univers	Univers	-33.9239	150.9266			CCTV11 Bigge Street towards station (school)
Liverpool	mac-00-04-4b-c4-8a-81	CCTV1 Elizabeth/Macquarie Mall facing	CCTV1 Elizabeth f	Camera Cou	Univers	Univers	-33.9203	150.9243			CCTV1 Elizabeth/Macquarie Mall facing Westfield
Liverpool	014d9c	nCounter Liverpool Library	nCounter Library	nCounter	Meshed	Meshe	-33.9219	150.9257		Liverpool	The nCounters count the number of Wi-Fi enabled devices in
Liverpool	hawk-013a24	nCounter Macquarie Mall canopy	nCounter Macqua	nCounter	Meshed	Meshe	-33.9237	150.9235		Canopy ir	The nCounters count the number of Wi-Fi enabled devices in
Liverpool	ns_aq_07	Air Quality - Liverpool Library	Air Quality Librar	Air quality	Univers	TTN	-33.9219	150.9257		Liverpool	The air quality sensors measure particulate matter 2.5 and 10.
Liverpool	ns_aq_02	Air Quality - Bigge Park	Air Quality Bigge	Air quality	Univers	TTN	-33.9213	150.9286		Bigge Par	The air quality sensors measure particulate matter 2.5 and 10.
Liverpool	aq_15	Air Quality 15	Air Quality 15	Air quality	Univers	TTN					The air quality sensors measure particulate matter 2.5 and 10.
Liverpool	ns_aq_06	Air Quality 06	Air Quality 06	Air quality	Univers	TTN					The air quality sensors measure particulate matter 2.5 and 10.
Liverpool	City Council	Thermal Camera - Bigge Park	Thermal cam Bigg	WIC camera	Applied	MQTT b	-33.9217	150.9275		Bigge Par	The thermal camera monitors surface temperatures.
Liverpool	City Council	Thermal Camera - Macquarie Mall	Thermal cam Mac	WIC camera	Applied	MQTT b	-33.9206	150.9241		Macquari	The thermal camera monitors surface temperatures.
Liverpool	ems-dd52	Environmental monitoring - Macquarie	Env monitoring M	Environmen	UNSW	TTN	-33.9206	150.9241		Macquari	The Environmental Monitoring Station and Weather Station m
Liverpool	CCTV7	CCTV7 Cnr George and 33 Moore	CCTV7 Cnr Georg	Camera Cou	Univers	Univers	-33.9226	150.925			CCTV7 Cnr George and 33 Moore - Facing South
Liverpool	IC18	Macquarie Street - towards Scott Stree	CCTV18 Macquari	Camera Cou	Univers	Univers	-33.924	150.9234			Macquarie Street - towards Scott Street
Liverpool	mac-00-04-4b-a8-14-dd	IC19 Cnr Bigge and Moore	CCTV19 Cnr Bigge	Camera Cou	Univers	Univers	-33.923	150.9269			IC19 Cnr Bigge and Moore (pointed north)
Liverpool	mac-00-04-4b-c4-81-96	CCTV13 40 Scott Street - towards Georg	CCTV13 Scott tow	Camera Cou	Univers	Univers	-33.9251	150.9244			CCTV13 40 Scott Street - towards George Street
Liverpool	mac-00-04-4b-c4-89-2b	CCTV12 52 Scott Street, towards Macqu	CCTV12 Scott tow	Camera Cou	Univers	Univers	-33.925	150.9233			CCTV12 52 Scott Street, towards Macquarie st
Liverpool	mac-00-04-4b-c4-89-52	CCTV15 Cnr Bathurst/Pirie/Norfolk (79	CCTV15 Cnr Bathu	Camera Cou	Univers	Univers	-33.9251	150.9206			CCTV15 Cnr Bathurst/Pirie/Norfolk (79 Bathurst St/Gemelle)
Liverpool	hawk-013a5B	nCounter outside TAFE	nCounter TAFE	nCounter	Meshed	Meshe	-33.9231	150.927		Cnr Bigge	The nCounters count the number of Wi-Fi enabled devices in
Liverpool	aq_18	Air Quality - Cnr Bigge and Moore Stree	Air Quality Cnr Bi	Air quality	Univers	TTN	-33.9231	150.927		Cnr Bigge	The air quality sensors measure particulate matter 2.5 and 10.
Liverpool	ems-4c99	Environmental monitoring - Bigge Park	Env monitoring B	Environmen	UNSW	TTN	-33.9217	150.9275		Bigge Par	The Environmental Monitoring Station and Weather Station m
Liverpool	mac-00-04-4b-c4-8a-6f	CCTV7 Cnr George and 33 Moore	CCTV7 Cnr Georg	Camera Cou	Univers	Univers	-33.9226	150.925			CCTV7 Cnr George and 33 Moore - Facing South
Liverpool	mac-00-04-4b-a5-98-41	CCTV10 Macquarie St - middle	CCTV10 Macquari	Camera Cou	Univers	Univers	-33.9237	150.9235			CCTV10 Macquarie St - middle (canopy)
Liverpool	mac-00-04-4b-c4-89-07	CCTV4 Center of Macquarie Mall	CCTV4 Macquarie	Camera Cou	Univers	Univers	-33.9216	150.9239			CCTV4 Center of Macquarie Mall
Liverpool	mac-00-04-4b-c4-89-61	CCTV3 147 Northumberland Street - to	CCTV3 Northumb	Camera Cou	Univers	Univers	-33.921	150.9227			CCTV3 147 Northumberland Street - towards Elizabeth St
Liverpool	mac-00-04-4b-c4-8b-f8	CCTV14 Bigge street/ Scott Street - tow	CCTV14 Bigge tow	Camera Cou	Univers	Univers	-33.9253	150.9266			CCTV14 Bigge street/ Scott Street - towards Scott Street
Liverpool	mac-00-04-4b-c4-8d-c9	CCTV8 Cnr Bigge and Moore - facing so	CCTV8 Cnr Bigge I	Camera Cou	Univers	Univers	-33.923	150.9269			CCTV8 Cnr Bigge and Moore - facing south

Figure 27: IoT Device list of LCC (for counting related sensors).

We initially focussed on the sensors categorised by LCC as 'Devices, people and vehicles counting', which are directly related to the transportation features of the DT (namely the roads). A closer look at the 'Device name' and 'Description' columns provided the hints to how roads may be related to a device, while 'Address' was found to only have information recorded for few sensors. On the other hand, with the road dataset that we have in the DT, only a few attributes, such as *roadnamebase* and *roadnametype*, can be used to relate to the places described in the sensor's attributes (example shown in Figure 28),.

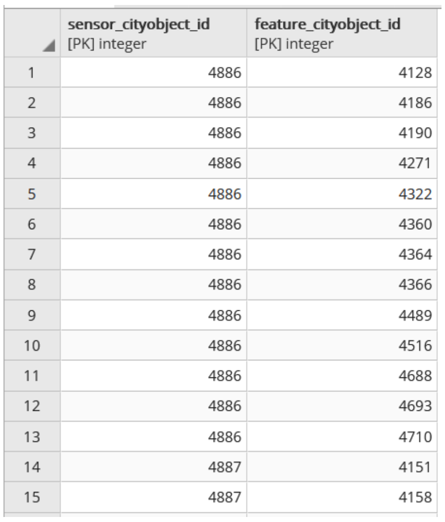


	id integer	parent_genattrib_id integer	root_genattrib_id integer	attrname character varying (256)	datatype integer	strval character varying (4000)	intval integer	realval double precision
19	52304	[null]	52304	roadnamebase	1	MACQUARIE	[null]	[null]
20	52305	[null]	52305	roadnametype	1	STREET	[null]	[null]
21	52339	[null]	52339	roadnamebase	1	COPELAND	[null]	[null]
22	52340	[null]	52340	roadnametype	1	STREET	[null]	[null]
23	52374	[null]	52374	roadnamebase	1	LACHLAN	[null]	[null]
24	52375	[null]	52375	roadnametype	1	LANE	[null]	[null]
25	52409	[null]	52409	roadnamebase	1	GRIMSON	[null]	[null]
26	52410	[null]	52410	roadnametype	1	CRESCENT	[null]	[null]
27	52444	[null]	52444	roadnamebase	1	GRIMSON	[null]	[null]
28	52445	[null]	52445	roadnametype	1	LANE	[null]	[null]

Figure 28: Attribute of road features that can be used for the linking.

With a total of 34 sensors, one could think of making the entries of the links manually. However, this can quickly become a tedious task as road features can be split into several portions (e.g. long avenues can be split where they intersect other roads), requiring checking.

Furthermore, we have deliberately discarded the approach that consists in creating a buffer around the sensor location and consider only those features that fall within this. Indeed, while one could argue that the data provided by the sensors is local (people, cars and bicycles are counted only around a sensor's location), it provides important insights on what might likely occur in other portions of the same road for reasons of contiguity and network flow. Additionally, another consideration was the fact that several CCTV counter, despite being located at crossing of several roads, would be pointed towards only one road among them. We have therefore decided mine the sensor descriptions and keep all road portions identified into them, with a special consideration for keywords such as ‘towards’, ‘facing’ or ‘Cnr’ (for corner).



	sensor_cityobject_id [PK] integer	feature_cityobject_id [PK] integer
1	4886	4128
2	4886	4186
3	4886	4190
4	4886	4271
5	4886	4322
6	4886	4360
7	4886	4364
8	4886	4366
9	4886	4489
10	4886	4516
11	4886	4688
12	4886	4693
13	4886	4710
14	4887	4151
15	4887	4158

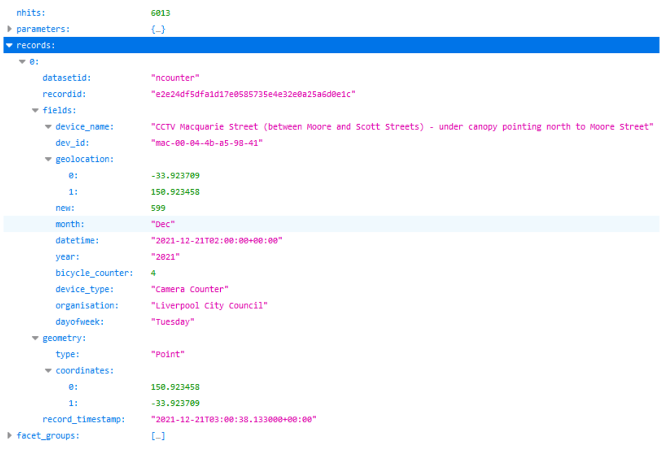
Figure 29: Database table describing links between sensors and other city features.

A python script was developed to perform the task semi-automatically (as some missing cases would require tweaking the parameters accordingly). Since there is no adapted native 3DCityDB table for our purpose, we opted for creating a simple new table with a trivial structure describing on one hand the *cityobject_id* of the sensors and on the other, those of the spatial features (Figure 29). This structure was inspired from other similar linking table of 3DCityDB (e.g. ADDRESS_TO_BUILDING, etc.) to remain coherent with standard.

3.1.2 Accessing the IoT Sensor data

The access to the IoT data goes through the API of the LCC open data platform. Based on the device ID of a sensor, let’s say ‘*mac-00-04-4b-a5-98-41*’ for example (a CCTV), we can send an HTTP request to the LCC server using the following link:

- https://data.liverpool.nsw.gov.au/api/records/1.0/search/?dataset=ncounter&q=&rows=1&sort=datetime&refine.dev_id=mac-00-04-4b-a5-98-41

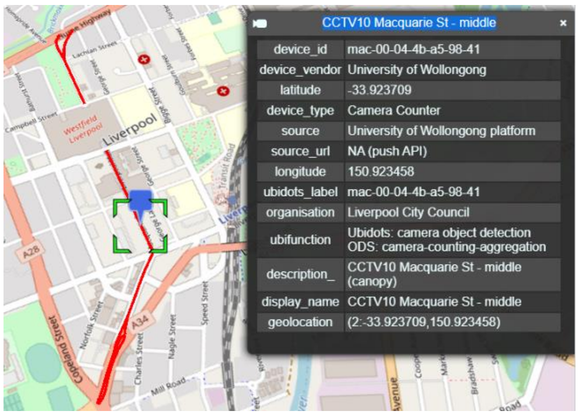


```
  nhits:                6013
▸ parameters:           {…}
▾ records:
  ▾ 0:
      datasetid:        "ncounter"
      recordid:         "e2e24df5dfa1d17e0585735e4e32e0a25a6d0e1c"
    ▾ fields:
      ▾ device_name:    "CCTV Macquarie Street (between Moore and Scott Streets) - under canopy pointing north to Moore Street"
        dev_id:         "mac-00-04-4b-a5-98-41"
      ▾ geolocation:
          0:            -33.923709
          1:            150.923458
        new:            599
        month:          "Dec"
        datetime:       "2021-12-21T02:00:00+00:00"
        year:           "2021"
        bicycle_counter: 4
        device_type:    "Camera Counter"
        organisation:   "Liverpool City Council"
        dayofweek:      "Tuesday"
    ▾ geometry:
        type:           "Point"
      ▾ coordinates:
          0:            150.923458
          1:            -33.923709
      record_timestamp: "2021-12-21T03:00:38.133000+00:00"
▸ facet_groups:         […]
```

Figure 30: JSON output of an API query to the LCC IoT & Open Data Portal.

This is a web request to query the most recent record available for the given sensor and returns as a result a JSON data (Figure 30) that can then be parsed and exploited. In Figure 30, the record named ‘new’ provides the number of pedestrians counted by the CCTV.

```
SELECT lod0_network FROM citydb.transportation_complex WHERE id IN(
    SELECT feature_cityobject_id FROM public.sensor_to_feature
    WHERE sensor_cityobject_id IN (
        SELECT cityobject_id FROM citydb.cityobject_genericattrib
        WHERE attrname = 'device_id' AND strval = 'mac-00-04-4b-a5-98-41'
    )
);
```


CCTV10 Macquarie St - middle

device_id	mac-00-04-4b-a5-98-41
device_vendor	University of Wollongong
latitude	-33.923709
device_type	Camera Counter
source	University of Wollongong platform
source_url	NA (push API)
longitude	150.923458
ubidots_label	mac-00-04-4b-a5-98-41
organisation	Liverpool City Council
ubifunction	Ubidots: camera object detection ODS: camera-counting-aggregation
description_	CCTV10 Macquarie St - middle (canopy)
display_name	CCTV10 Macquarie St - middle
geolocation	(2:-33.923709,150.923458)

Figure 31: Road features (red) related to the queried sensor (blue) and its attributes.

The script snapshot above and Figure 31 illustrate an example query of the DT to get the features related to the selected sensor. In the next phases we will see how to enhance the visualisation of the sensor data through the available viewers.

3.2 Algorithm for 3D shadowing Estimation

The next step developing the 3D model to estimate the shadowing of the 3D features and combine this with temperature distribution data for analysis. Unfortunately, there were not enough temperature sensors throughout LCC to allow such process with only two weather stations available nearby). We will discuss more about the changes that were brought to the targeted use of the DT and the related WP in the next section. In the rest of this section, we provide more details on the 3D shadow estimation process and the data preparation that was necessary to perform it.

3.2.1 Principle of the 3D shadow estimation

Shadow estimation is a common task in several urban applications (urban planning, solar exposure, building application, etc.) with common methods and tools drawn from the Computer Graphics industry that are mostly tailored to screen rendering (see Figure 32).

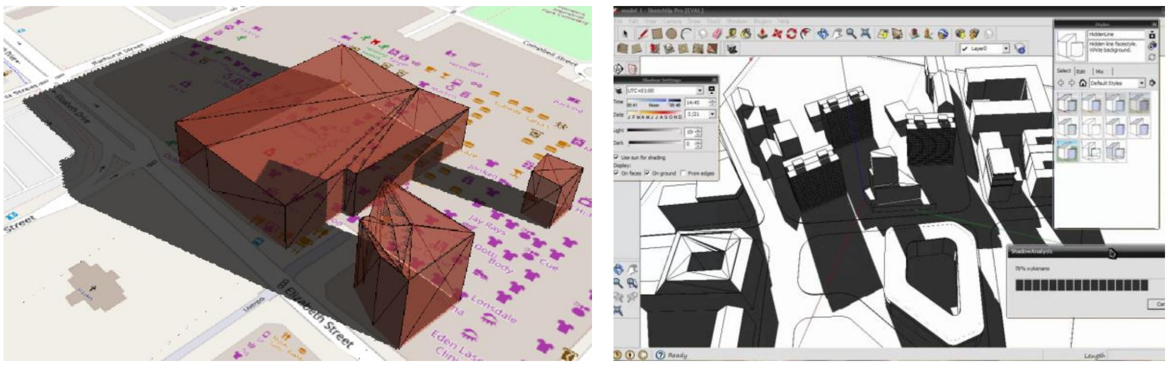


Figure 32: Shadow rendering on Cesium (left) and SketchUp (right).

While such rendering may be suitable for visual applications, it is inadequate for the types of spatial analysis that we are aiming to perform. As such estimation is based on screen rendering that is dependent of the scene's viewpoint, we developed a specific algorithm that allows the computation of 3D shadow polygons induced by city objects. As this is highly computationally intensive, a limited effort has been dedicated to approaches known as shadow volumes in the literature, which are approaches operating on the object space, i.e dealing with shadow as an actual 2D or 3D object (see Figure 33).

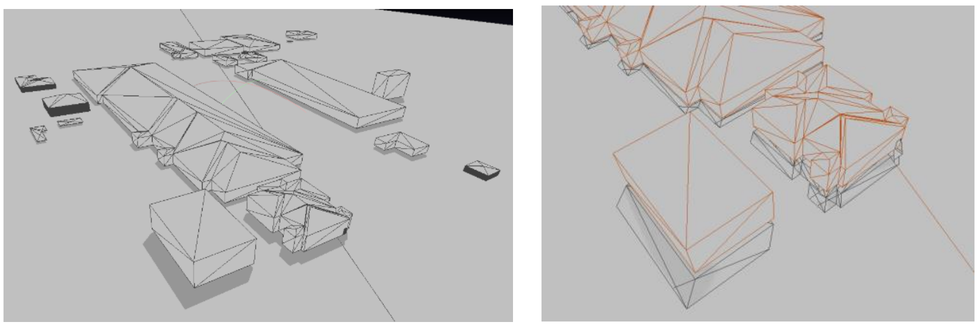


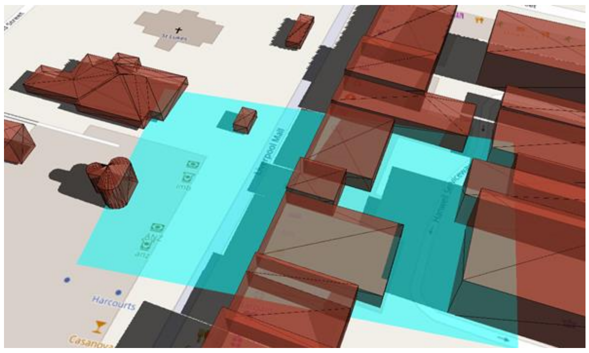
Figure 33: Shadow in the pixel space (left) vs in the object space (right).

We have adopted the unordered Shadow Volume Binary Space Partitioning (uSVBSP) method of (Chrysanthou and Slater, 1995) and improved it to mitigate the performance limitations that comes with dealing with a high number of 3D features and the implied 3D intersections. Thanks to that algorithm, we can generate the shadow as objects of the model and spatially determine its extent on the features of the Digital Twin, independently of the viewpoint of the scene. Furthermore, we can compute shadow area, which is important for a variety of spatial analysis such as estimating areas that need sun projection structures, computing paths in shaded areas in specific parts of the day or investigating causes for urban heath islands.

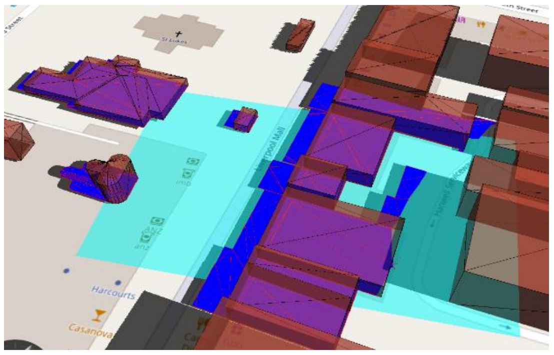
Figure 33 (right) shows an illustration of the approach implemented and considering a flat ground. However, if we want to fully exploit the 3D DT, we need to perform the shadow estimation while considering the actual terrain of model. Therefore, to complete the shadow estimation, we had to generate a reasonably accurate Triangular Irregular Network (TIN) of LCC based on the data that we had. Hence the process described in Section **Error! Reference source not found.**.

3.2.2 3D shadows on integrated TIN

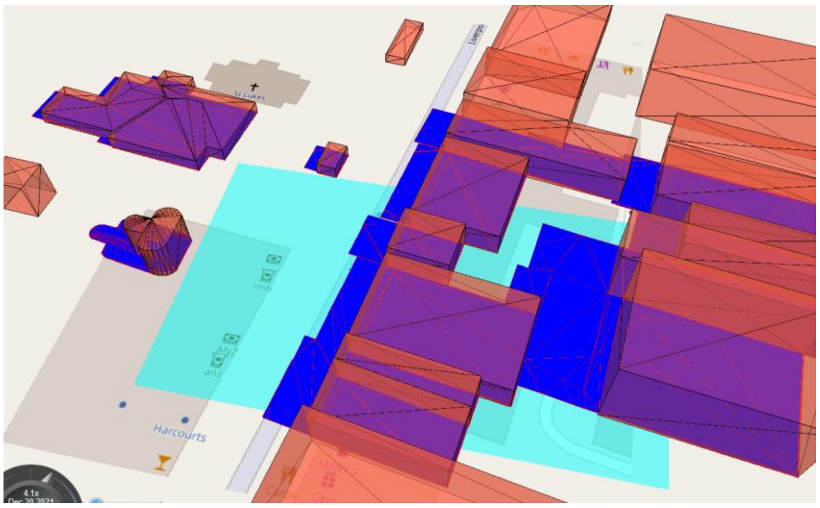
With the TIN generated, we can use it as a feature in the shadow computation. This process involve 3 main steps: (1) an area of interest is determined and all the features falling under that area are identified, (2) the geometries of the identified features are queried in the DB, collected, and sent to the shadow computation process and (3) the shadow is computed and a file (geoJSON) of the result is generated. Figure 34 illustrates an example of the process.



(a)



(b)



(c)

Figure 34: (a) Example of selected area for shadow computation in Cesium. (b) The computed shadow (dark blue) does not fit to the terrain approximation of Cesium and thereby not to Cesium's shadow. (c) When a flat terrain is used, the generated shadow appears fully.

In Figure 34, few selected buildings are displayed in Cesium JS along with the native shadow casting of Cesium. As it can be seen in Figure 34 (b), there are discrepancies between our generated shadow and the one of Cesium. The main reason for those differences is the native terrain layer of Cesium[6] provides a lower resolution (approximate resolution of 5m) than our DEM data (1m). Furthermore, a terrain exaggeration of 0.4 has been heuristically used to ensure (visually) that the features of our DT are not floating. For all those reasons, the shadow casting Cesium is less reliable. But the main advantage of our object-oriented shadow generation, beside the ability to use a better terrain estimation, is that we can manipulate the resulting shadows on any tool supporting geoJSON, such as QGIS (Figure 35: Shadow polygons imported on QGIS (left) and visualised with the building footprints (right).). This opens doors to powerful spatial analysis and visualisation as shown below (Figure 35, Figure 36, Figure 37).

[6] https://cesium.com/platform/cesium-ion/content/cesium-world-terrain/

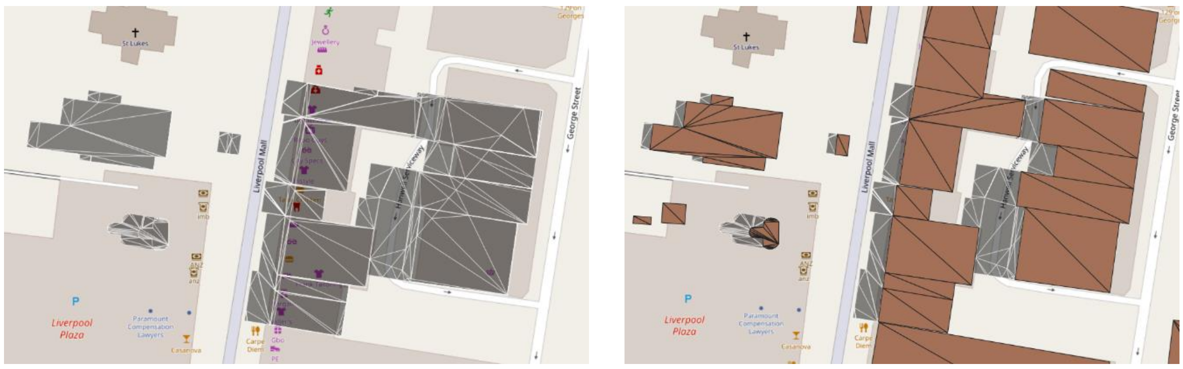


Figure 35: Shadow polygons imported on QGIS (left) and visualised with the building footprints (right).

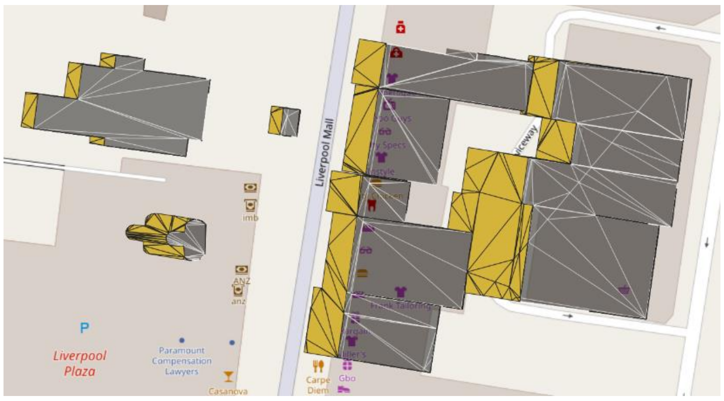


Figure 36: Spatial difference between the footprints and the shadows allowing to segregate polygons that are pure shadow.

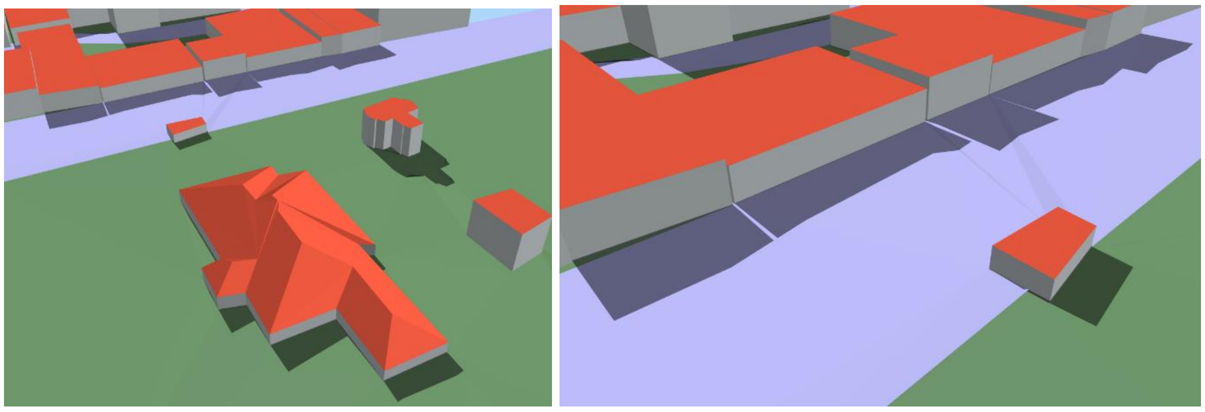


Figure 37: 3D Shadow visualised with the other features on QGIS.

3.3 3D Shadowing analysis of LCC and mitigation strategies for events and canopy planning (D5)

the goal of this use case is to perform shadow analysis on selected areas to determine their potential exposure to sun during a planned event and see how this can be mitigated (e.g., by planting new trees). To achieve this, several improvements were brought to the shadow computation algorithm and the CesiumJS user interface, through the ***Shadows*** tab and its corresponding options.

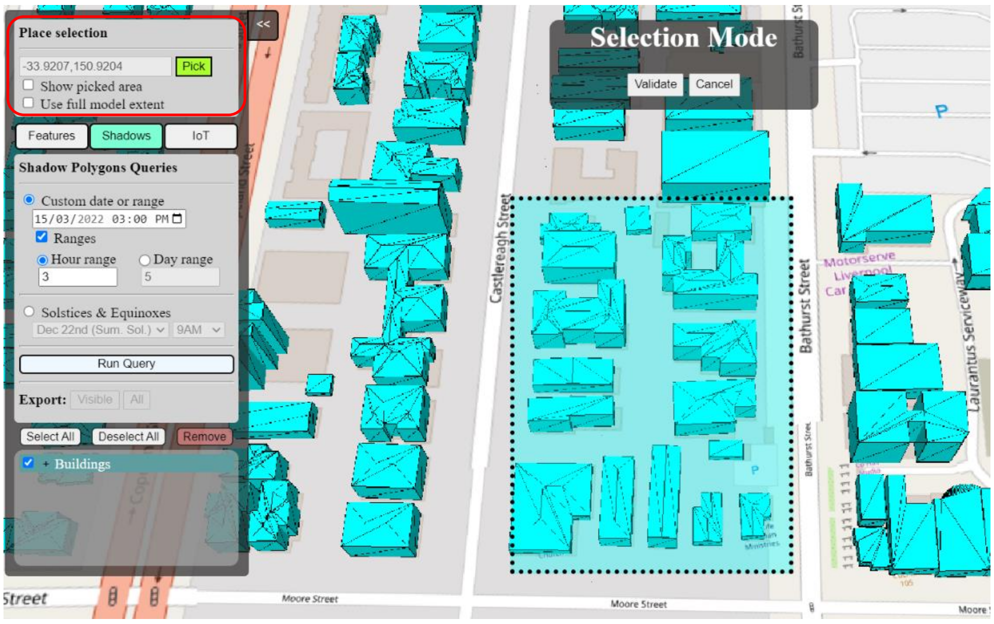


Figure 38: Place selection for shadow analysis.

The 'Place selection' part allows the user to pick a specific location around which shadow will be computed (see Figure 38). Finally, the user defines a specific date and time or a range of time or days on which the analysis will be run through. In the considered scenario, a shadow coverage between 3PM and 5PM is computed for the selected place on March 15th, and based on it, the algorithm computes, for every hour within that range 3 outputs (see Figure 39):

- the non-ground shadowed polygons (not displayed),
- the shadowed polygons on the ground (dark),
- and the sun-exposed polygons on the ground (orange).

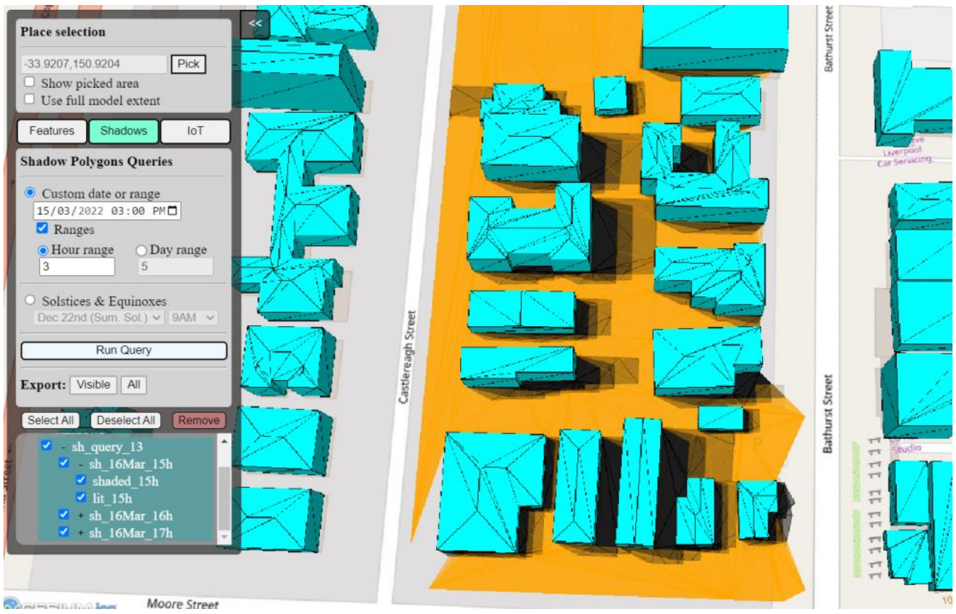


Figure 39: Ground shadow polygons (dark) and lit ground polygons (orange) for a selected date and time range.

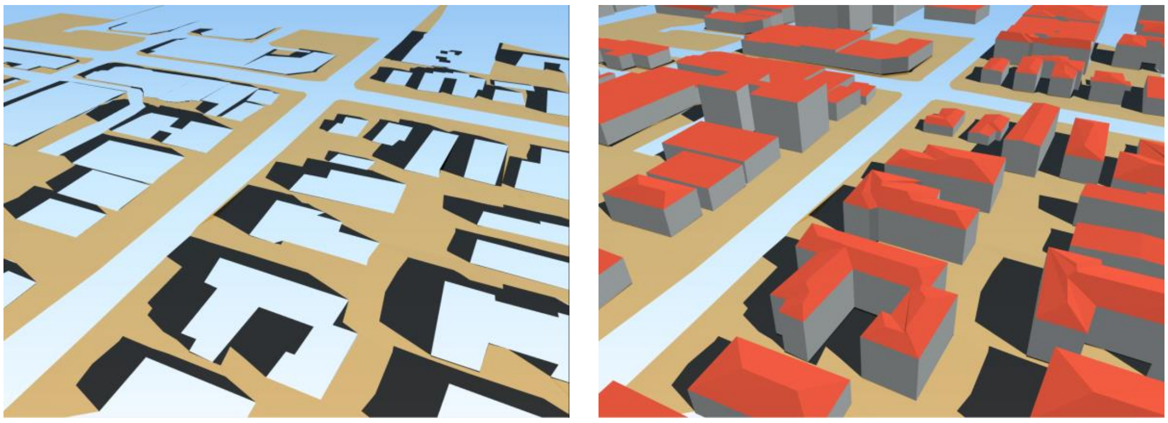


Figure 40: Shadow polygons visualised on QGIS. (Left) Shadowed and sun-exposed ground polygons. (Right) Building features causing the shadows.

This distinction in different types of outputs is made to facilitate further analysis relying on them. Furthermore, the outputs are currently saved as GeoJSON files to allow their storage, sharing and import in common GIS tools such as QGIS, as shown in Figure 40. It is visible that the road polygons are excluded from the scene for the specific analysis being run. This is made possible by the categorisation of the TIN based on the labels from the features, which helped improve the shadow analysis by allowing focus on specific features. Thereby, the shadow polygons considered specifically correspond to areas accessible to pedestrians. It remains possible to control this and include other features, when necessary. Based on the same generated data, but with additional functionalities, this use case can be extended with pedestrian routing that allows the use of the shadowed parts of the streets.

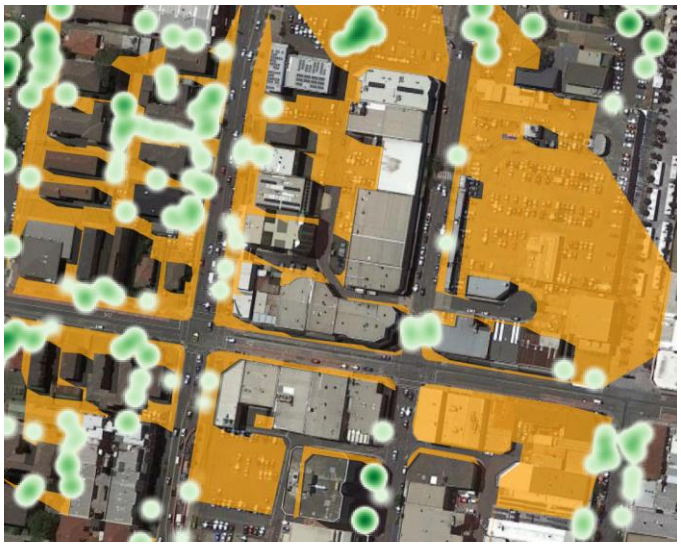


Figure 41: Unshadow areas (orange) vs. Tree coverage (green).

For the second use case, we checked the unshaded areas against the canopy/vegetation data available to us. Unfortunately, the data is just a point layer describing tree locations and with their height and elevation. Since this is not enough to guess the amount of shadow that they cast, we applied a heat map of the trees assuming an approximate area around the trees that we consider as shadowed (see Figure 41). It is visible that the studied area is under-covered in terms of trees and would benefit from more trees that would bring more shadow. Although most of the exposed zones are open parking areas and private gardens, the main streets are still not covered enough. The block at the bottom right side of the image reflects an area where no 3D building data were available.

In addition to the previous use cases, we implemented few more analysis capabilities to the DT. The first one integrates a static Urban Heat Island layer obtained from CSIRO[7]. It leverages the Landsat8 data and provides land surface temperature and urban heat island estimates for Australian capital cities. The dataset is meant to compare temperatures of the urban areas against non-urban baseline temperatures, and the colour range spanning from blue to red means a difference of temperature from 0ºC to 10ºC (see Figure 42, left).

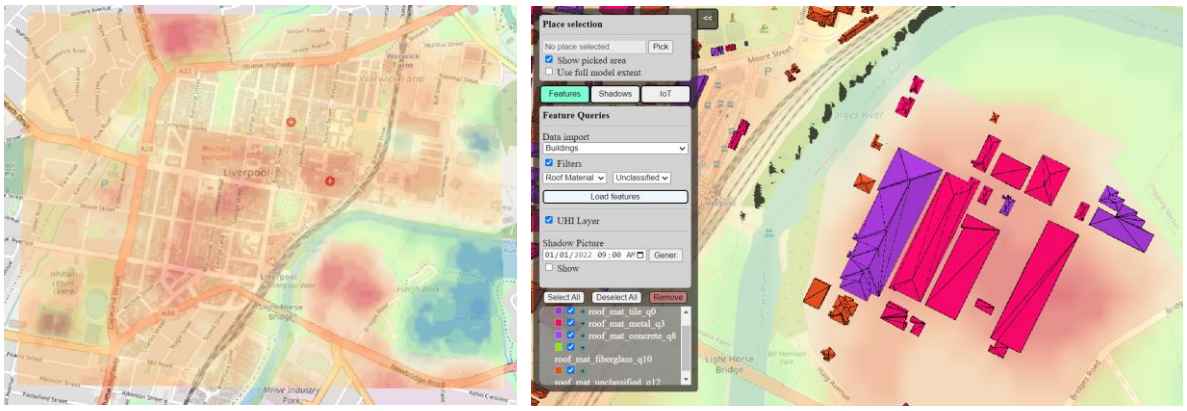
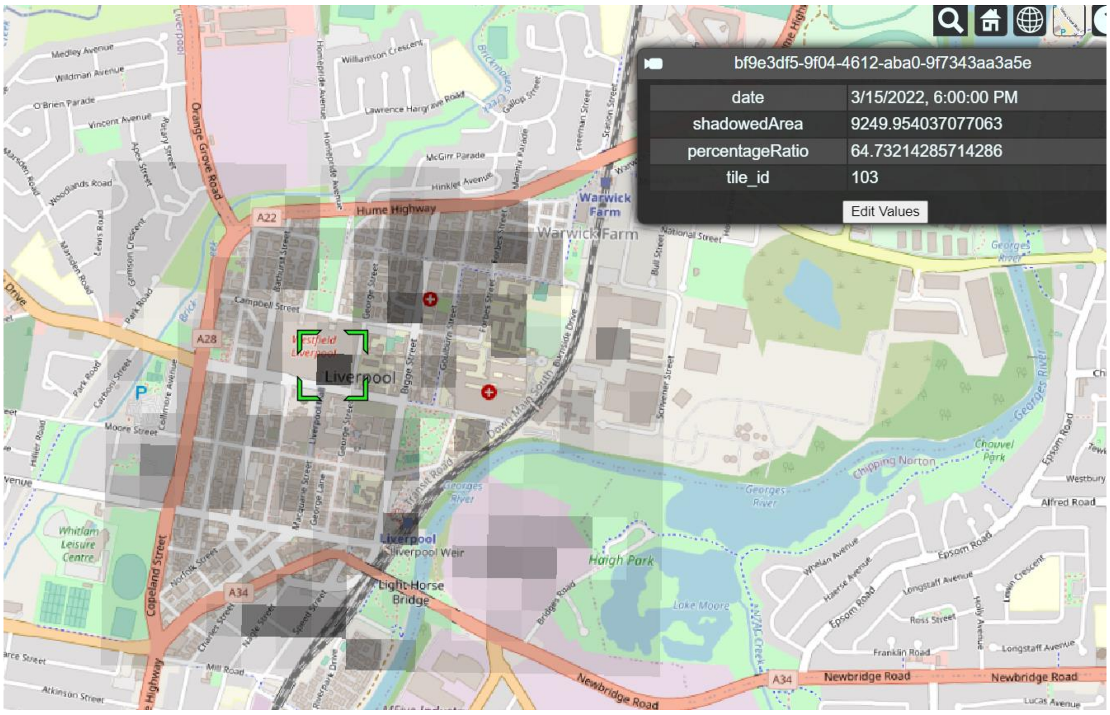


Figure 42: (Left) UHI Layer viewed in Cesium (red means +10ºC and blue means +0ºC). (Right) Querying buildings per roof type to analyse eventual impact on UHI (tile roofs in purple, metal roofs in red and unclassified roof material in orange).

Combining the UHI layer with the custom feature queries enable new insight on what may be a factor of the UHI red areas. Figure 42, right illustrates the different roof materials covering an UHI-prone area of LCC. Metal roofs (in red) are predominant, which may suggest that they contribute to increasing the temperature compared to other materials.

Finally, we have implemented what we called the **Shadow Picture** of the city. It is computed by subdividing the scene in 256 tiles and checking for each tile how much ground area it contains and how much of it is covered by shadow. This allows to produce a ratio that is then used to darken the area and give a visual insight of the level of shadow coverage. Figure 43 provides an illustration of what such shadow picture looks like for the whole area of study at 6PM on March 15th, 2022.

[7] Devereux, Drew; Caccetta, Peter (2019): Land surface temperature and urban heat island estimates for Australian capital cities, summer 2018-19. v1. CSIRO. Data Collection. https://doi.org/10.25919/5d8adf30f001e


bf9e3df5-9f04-4612-aba0-9f7343aa3a5e

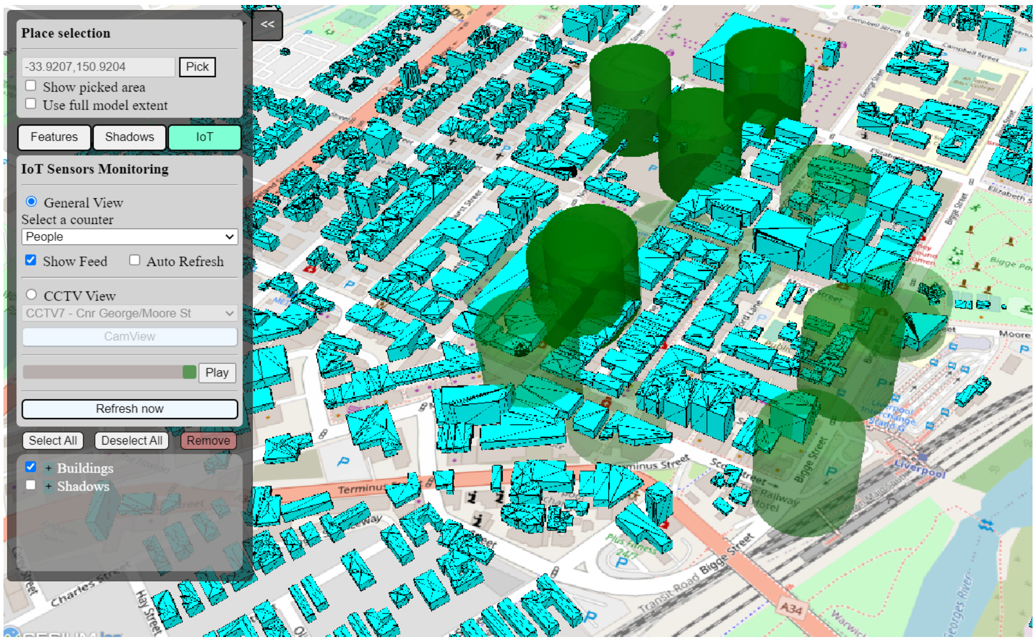
date	3/15/2022, 6:00:00 PM
shadowedArea	9249.954037077063
percentageRatio	64.73214285714286
tile_id	103

Edit Values

Figure 43: Shadow picture of the city on March 15th, 2022, at 6PM.

3.4 Interfaces for visualisation of sensor feeds and analytics (D6)

We developed a visualisation interface for the IoT sensors available in area of study as a third use case. The **IoT** tab of the UI is dedicated to those functionalities. While the data and some dashboards can be found on the LCC data portal, visualising the feed on a 3D map gives a better spatial context to the information.


Place selection

-33.9207,150.9204 Pick

☐ Show picked area

☐ Use full model extent

Features | Shadows | IoT

IoT Sensors Monitoring

● General View

Select a counter: People

☑ Show Feed ☐ Auto Refresh

○ CCTV View

CCTV7 - Cnr George/Moore St

CamView

Play

Refresh now

Select All | Deselect All | Remove

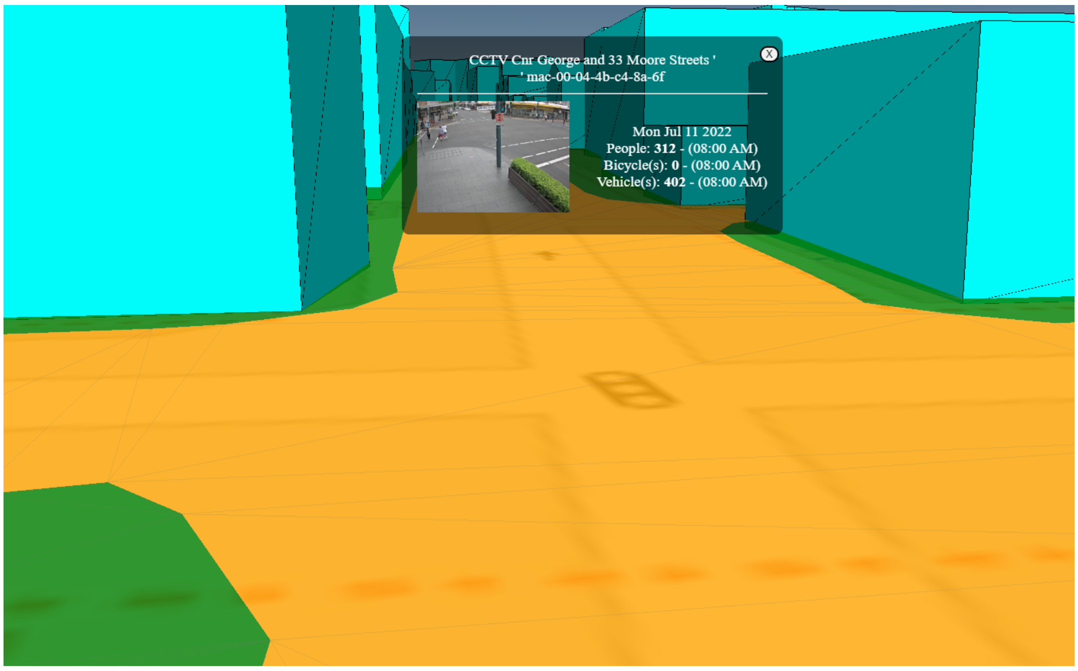
☑ Buildings

☐ Shadows

Figure 44: Visualisation of the feed from the people (green) counting devices.

Figure 44 shows the feed of few counting devices running at the time of the visualisation.

The user can select the type of counting device to visualise (people, bicycle, vehicle, or all of them simultaneously) and the latest feeds available will be queried and the scene updated accordingly. Options to hide or display the feed are also proposed, as well as the possibility to enable/disable the live updating (which occurs every 10 min, according to the pace of the sensors). We have chosen to use cylinders of the same radius since the sensors' roles are localised and provide only 1-dimensional value. The colour opacity and height of the cylinder are used to express the quantity counted (more opaque colours and higher means more counted entities). Finally, the records obtained from the API queries are collected and displayed as attributes of the visualised feeds (see Figure 45).



CCTV Cnr George and 33 Moore Streets '
' mac-00-04-4b-c4-8a-6f

Mon Jul 11 2022
People: **312** - (08:00 AM)
Bicycle(s): **0** - (08:00 AM)
Vehicle(s): **402** - (08:00 AM)

Figure 45: Simulated camera view of a selected CCTV device with the road and pavement coloured according to people (green) and vehicle (orange) feeds.

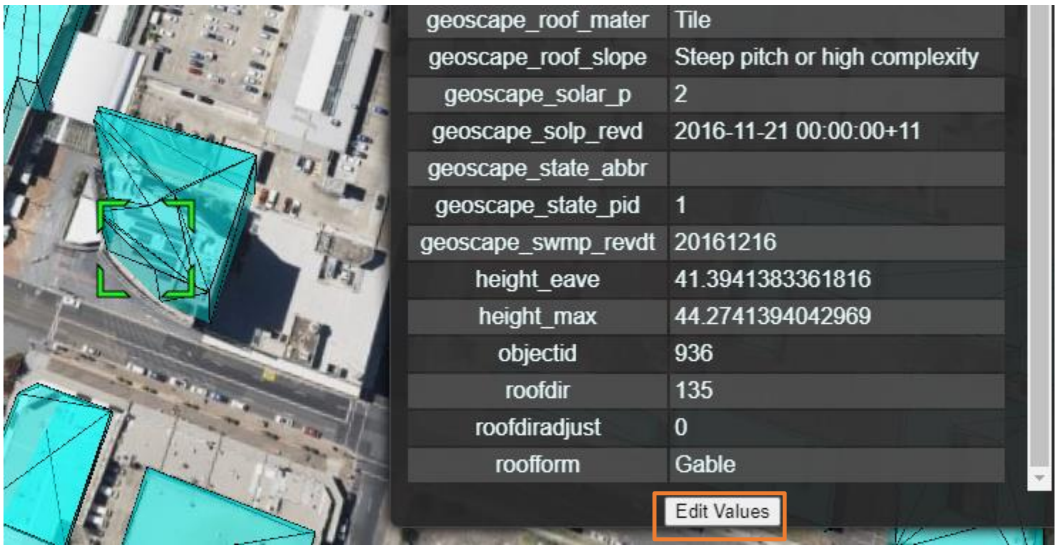
Furthermore, to leverage the 3D capabilities of the DT, we also implemented a simulated camera view of selected CCTV to offer a contextual view of the pedestrian, bicycle and vehicle flows at a street level (see Figure 45). Grounds (green) and road polygons (orange) are thereby coloured respectively according to the received pedestrian and vehicle feeds. A real picture from the simulated CCTVs along with corresponding feed values is also provided for reference.

4 Updating the Digital Twin

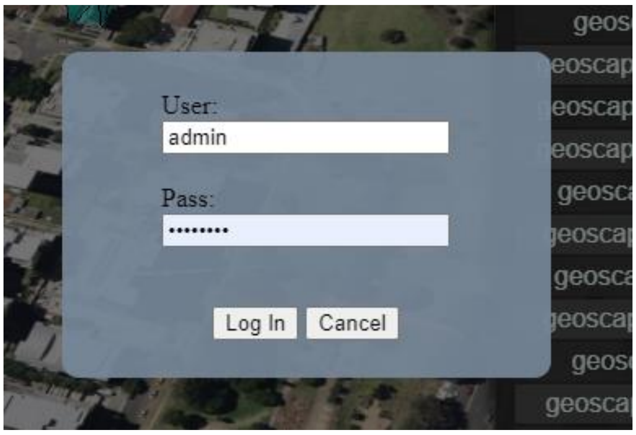
In Section 2 we explained the proper adaptation and storage and visualisation of initially collected spatial data in a database (DB), forming the digital twin of the city of Liverpool. Section 3 comprised the implementation of several use cases leveraging the stored spatial data. This Section concentrates on the semantic labelling and update of attributes.

4.1 A procedure for improving semantic labelling on selected DT

The intention is to allow authorised users to directly edit semantic and attribute information of the digital twin through the viewer / user interface (UI). The established connection between the UI and the DB allows live interaction with the model, ensuring real-time information update. This has the advantage to allow the correction of detected error directly through the front-end side of the tool, with visual support, rather than having to do it from the back-end side.



geoscape_roof_mater	Tile
geoscape_roof_slope	Steep pitch or high complexity
geoscape_solar_p	2
geoscape_solp_revd	2016-11-21 00:00:00+11
geoscape_state_abbr	
geoscape_state_pid	1
geoscape_swmp_revdt	20161216
height_eave	41.3941383361816
height_max	44.2741394042969
objectid	936
roofdir	135
roofdiradjust	0
roofform	Gable



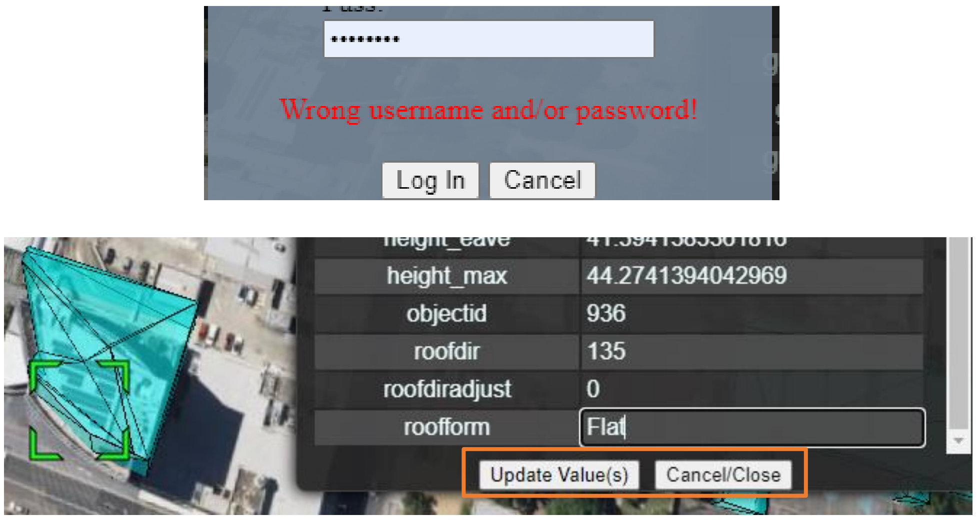
User: admin

Pass: ••••••••

Figure 46: The 'Edit Values' button attached to the attribute table of a selected building (top) and the connection request it prompts (bottom).

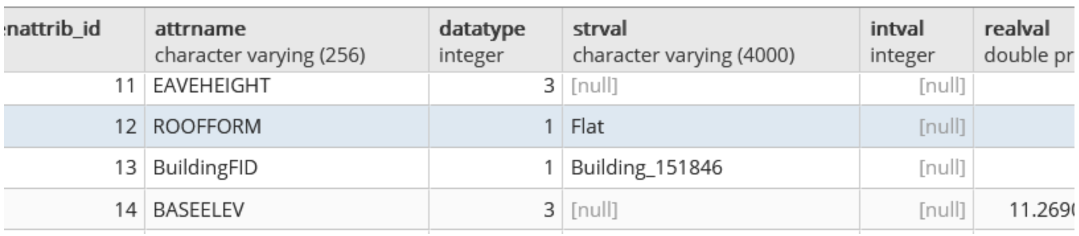
The implemented functionalities are accessible through a button attached to the attribute table of the any selected feature (Figure 46, top). Clicking on the 'Edit Values' button will prompt a connection request, if no successful log in has been performed before, during the session (Figure 46, bottom). Obviously, as the UI is supposed to be widely accessible, this is to constraint and ensure some security on the possibility to edit the data.

When the login fails, an error message is prompted to let the user know (Figure 47, top). After a successful login, it becomes possible for the user to edit the values of the attributes. Figure 47 (bottom) shows the update of the '*roofform*' attribute that was changed from Gable to Flat (for illustration purpose). Other attributes can also be modified, and the Update button will send the new values to the DB. The process can also be cancelled, and no change will be flowed to the DB.


Wrong username and/or password!

height_eave	41.3941385301816
height_max	44.2741394042969
objectid	936
roofdir	135
roofdiradjust	0
roofform	Flat

Figure 47: Error message prompted when login fails (top) and new options enabled when successful (bottom).



nattrib_id	attrname character varying (256)	datatype integer	strval character varying (4000)	intval integer	realval double pr
11	EAVEHEIGHT	3	[null]	[null]	
12	ROOFFORM	1	Flat	[null]	
13	BuildingFID	1	Building_151846	[null]	
14	BASEELEV	3	[null]	[null]	11.269

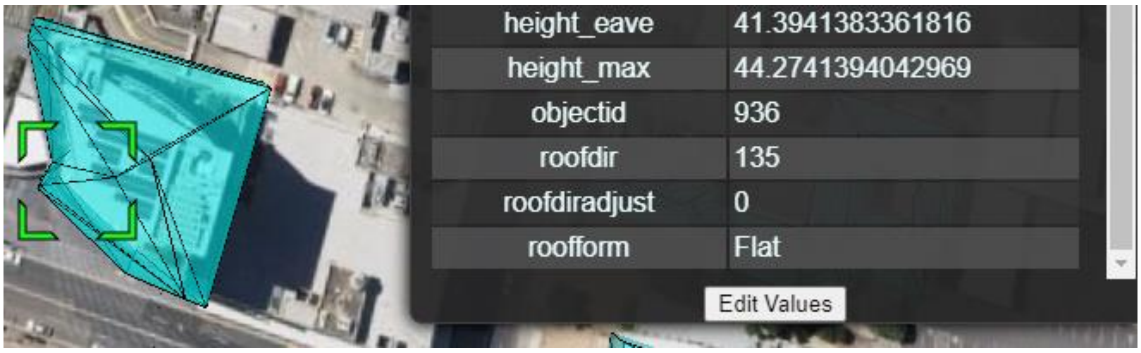
Figure 48: If all the conditions are met, the update if propagated to the DB.

Few extra constraints are set on the process to ensure validity of the data. Firstly, the user cannot edit the property names because this may affect the schema of the original data. Furthermore, some critical values, such as the **cityobject_id** property, cannot be edited as well, since they are primary keys to the features in the DB. Finally, the user has to make sure that the data types of the edited properties are respected otherwise the changes will fail (e.g., a property expecting a number should not be provided with a text, etc.). On the validation of the updates by the user, those additional checks are performed and if successful, the changes are propagated to the DB (Figure 48).

4.2 Visualization of the updated attributes via the interface (D8)

The newly changed properties of the DT features need to be reflected in the UI. While reloading the model could be the most straightforward way to do this, (since the newly updated DB would then be queried again), this would not be an efficient solution as the ongoing scene of the user would be lost and they would have to restart from scratch. Instead, we update the properties of the entities loaded in the scene in parallel to the DB update process.

Figure 49 shows the displayed attributes of the updated feature. This has the advantage to seamlessly affect only the entities that are changed without requiring any scene reload. However, among the users with running DT instances at the time of the update, only the one who performed the changes will be able to see it through the UI. The other current users will only get access to the changes when instantiating a new session, while any newly joining user will be able to see it directly as well.



height_eave	41.3941383361816
height_max	44.2741394042969
objectid	936
roofdir	135
roofdiradjust	0
roofform	Flat

Figure 49: Visualisation of the updated attribute on the UI.

5 Discussion on Transferability, Reusability, and Maintainability

A successful DT should be able to support the main characteristics of software quality, which includes the *transferability*, *reusability*, and *maintainability*. Transferability generally responds to the question of transferring the provided logic to other environments with different settings and whether this can be supported. Reusability of the product evaluates if different components of the product work as modules and can be modified, built upon, or re-applied elsewhere. It is one of the core concepts in *FAIR Principles for Research Software* and should be considered to ensure the quality of deliverables. Eventually, maintainability ensures that the final platform can be extended faster while reducing the costs and efforts of maintenance. These characteristics matter specifically in the DT domain as it allows each project to be the cornerstone of the other, letting the experiences flow and grow over the stack of projects and helping the projects to save time and budget significantly.

To respond, several approaches can be employed. A possible workaround is to create abstraction between the physical model, the logic, and the data. In this project, we employed Object-Relational Mapping (ORM) to realize this abstraction. In what follows, first the logic and benefits of ORM are discussed and then, technical details of the implementation of the approach are presented.

5.1 Object-Relational Mapping as a Solution

ORM is a mechanism that makes it possible to address, access, and manipulate objects from databases with a layer of abstraction. This means that the logic can employ the database objects without having to consider how those objects interact with the data sources (Figure 50).

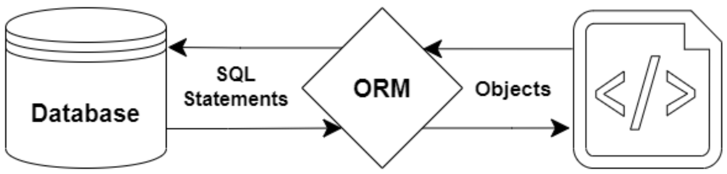


Figure 50: Object Relation Mappers acts as a mediator to give a level of abstraction betwen database and the code logic.

One of the key benefits of employing ORM is that it lets programmers maintain a consistent view of objects over time, even as the sources that deliver them change, together with the sinks that receive them and the applications that access them. Based on the abstraction, ORM manages the mapping details between a set of objects and underlying relational databases, XML repositories, or other data sources and sinks, while simultaneously hiding the often-changing details of related interfaces from developers and the code they create.
ORM can also hide and encapsulate changes in the data source itself, so that when data sources or their APIs change, only ORM needs to change to keep up—not the logic that uses ORM to insulate themselves from this kind of effort. This capacity helps developers take advantage of new classes as they become available and makes it easy to extend ORM-based applications. In many cases, ORM changes can incorporate new technology and capability without requiring changes to the code for related applications. This project uses SQLAlchemy as an open-source and prominent toolkit and ORM in the Python environment.

5.2 Using ORM in the Backend Stack

This project employs the following backend stacks to support the requirements of the project while ensuring the quality characteristics:

Compartment	**Technology + Version**
Web framework	Flask (2.1.2)
ORM	SQLAlchemy (1.4.37 + package Flask-SQLAlchemy 2.5.1)
Programming language	Python 3.8.10
Database	Postgres (12.11) + PostGIS (3.2)
Deploying server	HTTP Web Server: Apache/2.4.41 - Port 80: for UI - Port 443: for Flask server OS: Ubuntu 20.04.4 LTS

Table 2: The backend stacks in LCDT.

This means that the ORM is at the interaction level between the flask server and RDBMS (Figure 51).

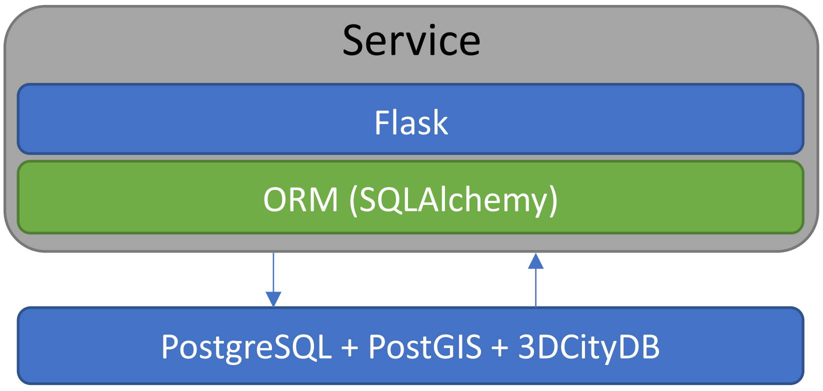
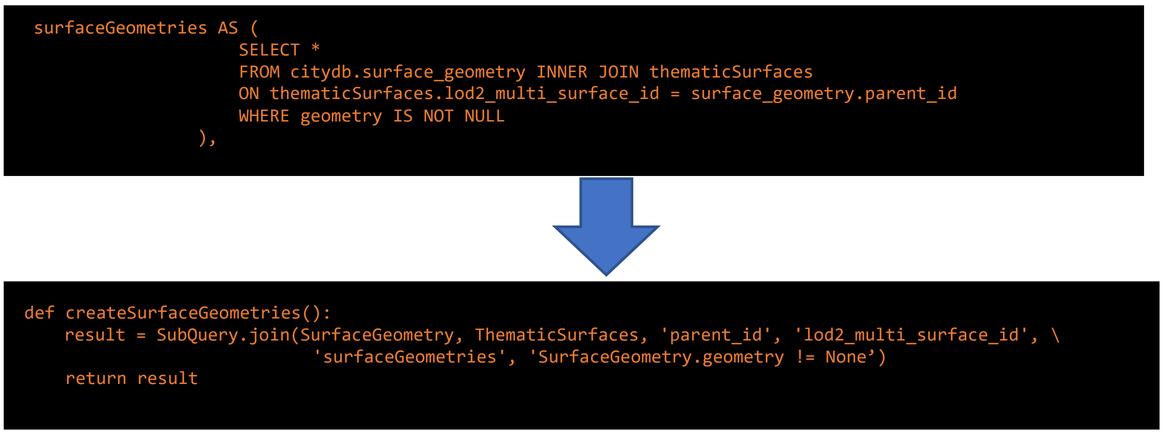


Figure 51: The database and service layer architecture.

5.3 Employing SQLAlchemy

SQLAlchemy provides a generalized interface for creating and executing database-agnostic code without needing to write SQL statements. The abstraction capability allows the usage of various RDBMSs in the background – e.g., Postgres, Oracle, or SQLite – while the logic stays independent from the database layer. This facilitates the transferability of the logic to other contexts. This also helps tackles maintainability issues using traditional inline SQL queries that bring readability difficulties are hard to debug. Figure 52 shows how a sample inline SQL command can be translated into a maintainable and easy to read python method. The capability also allows the automation of any SQL commands, specifically the commonly used *create*, *read*, *update*, and *delete* operations.



```
  surfaceGeometries AS (
                    SELECT *
                    FROM citydb.surface_geometry INNER JOIN thematicSurfaces
                    ON thematicSurfaces.lod2_multi_surface_id = surface_geometry.parent_id
                    WHERE geometry IS NOT NULL
                ),
```

```
def createSurfaceGeometries():
    result = SubQuery.join(SurfaceGeometry, ThematicSurfaces, 'parent_id', 'lod2_multi_surface_id', \
                           'surfaceGeometries', 'SurfaceGeometry.geometry != None’)
    return result
```

Figure 52: Translation of SQL inline commands to pythonic methods.

5.4 Defining the Objects

The first step in employing the ORM is the definition of the objects. With the implementation of the 3DCityDB schema, we follow the *Database-First* approach that allows us to use a legacy database in the ORM application. Thus, we first need to define each of the database entities as objects within the code to enable further manipulation/analysis on top of them. To do this the following objects were defined considering the features, their corresponding hierarchal structure and data types:

- *Building*
- *ThematicSurface*
- *SurfaceGeometry*
- *TransportationComplex*
- *Road*
- *CityObject*
- *CityObjectGenericAttribute*
- *GenericCityObjec*
- *Sensor*
- *Tin34kWLabel*
- *WaterBody*

These objects were then used in the processes and analytical operations, such as those described in Section 2.1.

5.5 Basic Operations and Helper Methods over ORM

To facilitate and support effective queries on database objects, we implemented a range of operations over SQLAlchemy. This included basic SQL operations, from insert/delete and updates, to ordering features, joining and filtering results, bounding box intersections, JSON aggregations and crosstab queries. All spatial operations, such as collection, transformation, intersection and GeoJSON generation, are supported (Figure 53).

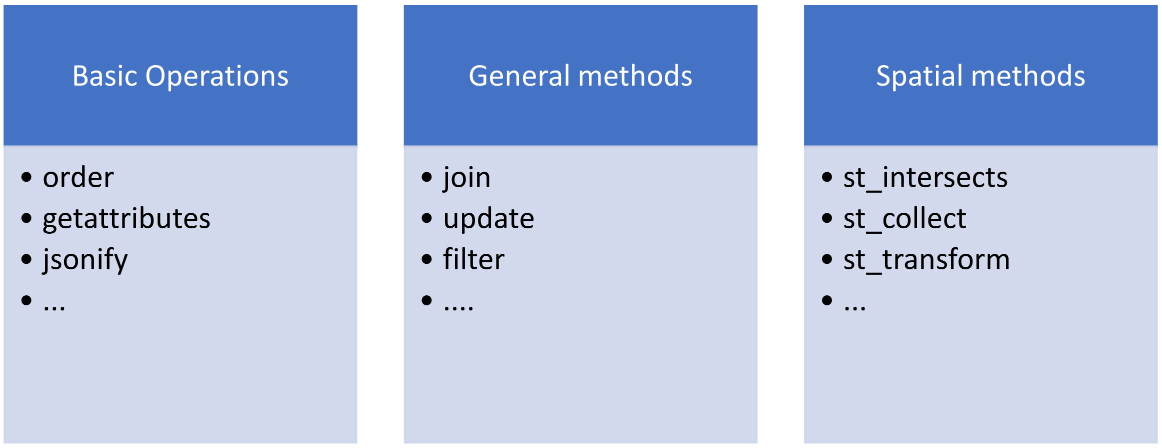


Figure 53: Various type of operations and helper methods supported by LCDT ORM realization.

As these methods are based on basic SQLAlchemy methods and properties, they are independent from the physical model, allowing the abstraction of the logic from the actual database technology. This allows the implemented methods to be used in the code logic to support the analytical requirements of the LCDT.

6 Conclusions

This project is a part of the research and development agenda of leading geospatial institutions such as FrontierSI, AURIN, Data61, UNSW and Spatial Services towards maturing the concept of spatial Digital Twins to support urban decision-making. This project concentrated on three important aspects of Digital Twin, i.e. integration of data, 3D analytics and maintenance/update. The project also demonstrated the use of the 3D international standard CityGML and specifically its Postgress/PostGIS implementation 3DCityDB. The Digital twin covered the central part of the City of Liverpool, but it was sufficient to obtain a glimpse of all challenges. It is an excellent first step in building extended, application-broader Digital Twin, at a second stage, help to clarify critical spatial data portals, standards, software and interfaces.

This project demonstrated the challenges in building spatial Digital Twins for large scale urban applications. It is also an illustration of a functioning digital twin using existing data and open-source technologies. Such an implementation can represent the contemporary environment across a range of detail and provide access to IoT data feeds directly via API, approaching a real-time representation.

The information is structured according to an international open standard CityGML and its database schema implementation 3DCityDB for the free ware database management Postgres/PostGIS. As mentioned previously, the use of spatial DBMS allows to 1) organise all data in a standardised way, 2) perform many spatial queries at a database level using the DBMS functionality and 3) access and query the data from a different software. We have shown examples with QGIS and Cesium. In a similar way the data can be accessed from ArcGIS, Revit and so forth for a range of tasks, such as urban planning and architeture.

Specifically interesting for a large group of users is the visualisation and interaction in an 3D immersive environment via accessible web-based tools, i.e. Cesium. This means that local governments and citizens are not exposed to great technical hurdles, nether they need to follow specialised training when the system is deployed. Notably, the formats employed in this demonstration are congruent with W3C and OGC standards, such as XMLbased CityGML, GeoJSON and REST API. Having constructed this extensible foundation using web-enabled open-source resources, the potential for access, intuitive operation, sharing and update is maximised, while extended facilities are also available for implementation, such as facilitating secure/authorised access.

During the data procurement process, however, collecting all needed data was difficult, even with formal data sharing agreements. For instance, any existing BIM data was usually kept with 3rd parties and unavailable. Similarly, pedestrian counters on video surveillance cameras was reduced to a simple number/count and visualised on an image covering the vield of view of the video camera. This placed many burdens on further analysis into identifying pedestrian direction and linking it to other parameters such as weather conditions. In the future, interoperability with proprietary models will require multipartite agreement on, and compliance with exchange standards to best ensure downstream value add when public data is feeding commercial implementations.

The project also clearly revealed that efficiency, accuracy and detail need to be considered in order to create a convincing and trustworthy analogue of the original environment that is as easy to use as possible. In terms of scalability, levels of details is critical and increasing levels of detail must be accommodated. The Digital twin needs to work with incomplete data inputs as well and allow to re-create and fill gaps. The ability to develop more complex datasets from known source data such as street and rail centerlines, building footprints and derived pedestrian walkways is an efficient way to employ proxy data if needed, and presents to opportunity to ingest higher quality data as they become available.

The topologically correctness and validity of data cannot be avoided and will always require several additional steps. Data are maintained by different institutions for different purposes and the procedures for digitalisation will result in a range of differences in feature delineation, accuracy, detail and precision. A typical example is the integration of above ground features such as Buildings, Ttansportation, WaterBodies, Vegetation with terrain. In this project we applied a constrained triangulation assuming the features have a higher accuracy compared to the obtained digital terrain. As discussed, a number of cleaning operations had to be performed to achieve topologically correct fusion of data.

This project has also developed a shadow casting algorithm, which allows to compute shaded areas for certain time intervals, which were also stored in the database. The 3D terrain is important for impactful 3D analysis like modelling shadow-fall from one building to another, the coverage of tree canopy shading. When combined with dynamic pedestrian data, the interactions between several factors taking into account slope, shade, time of day, season and the general built environmental context can provide a powerful tool for decision-making.

This project provided many tools for automatic processing, modelling and analysis. The automated approach for deriving integrated and managing 3D environments plays a critical role in supporting higher quality analysis in the subsequent phases of user-defined workflows, especially when users are not expects in 3D geospatial applications.

An important feature of this DT is the ability to ingest many diverse types of urban data into a coherent environment around government management and in turn, output a variety of 'views' appropriate for the aims of the end-users. In a next phase, the results of this project should be tested with both expert and non-expert users to elicit user feedback regarding ease of use and suggestions for future development.

Regardless of the purpose of a DT, it is important that organisations producing or collecting data ensure their durability and interoperability as much as possible. While this is a significant research topic on its own, we believe that open standards are the right way to go for future-proofing today's data for the DTs of tomorrow. They also remain the current best practice in terms of interoperability through avoidance of unharmonized data schemas and commercial/proprietary lock-ins, while remaining compatible with new technological advances (e.g., API, web support, etc.).

This project provided first practical insight to building spatial Digital Twins. In future research we will concentrate to connecting to new national programs such Digital Atlas and ANZLIC principles for DTs, the mechanisms and the value of connecting to other DTs in an ecosystem or Digital treads, examine and recommend technologies, standards and procedures to support urban planning and decision making.

References

Cesium GS, I., 2022. CesiumJS. 3D geospatial visualization for the web. cesium.com/platform/cesiumjs (26 April 2022).

Chrysanthou, Y. and M. Slater, 1995, Shadow Volume BSP Trees for Computation of Shadows in Dynamic Scenes, I3D '95: Proceedings of the 1995 symposium on interactive 3D graphics, pp. 45-50.

QGIS Development Team, 2022. QGIS Geographic Information System. QGIS Association.

Kutzner, T., Chaturvedi, K., Kolbe, T. H., 2020. CityGML 3.0: New functions open up new applications. PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science, 88(1), 43–61.

Kolbe, T. H., Groeger, G., Plumer, L., 2005. Citygml: Interoperable access to 3d city models. Geo-information for disaster management, Springer, 883–899.

Li, W., S. Zlatanova, J. Yan, A. Diakite, and 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, 2019

Li, W., S. Zlatanova, A. A. Diakite, M. Aleksandrov and J. Yan, 2020, Towards Integrating Heterogeneous Data: A Spatial DBMS Solution from a CRC-LCL Project in Australia, ISPRS Int. J. Geo-Inf. 2020, 9(2), 63

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, 147–154

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.

Chue Hong, Neil P., Katz, Daniel S., Barker, Michelle, Lamprecht, Anna-Lena, Martinez, Carlos, Psomopoulos, Fotis E., Harrow, Jen, Castro, Leyla Jael, Gruenpeter, Morane, Martinez, Paula Andrea, Honeyman, Tom, Struck, Alexander, Lee, Allen, Loewe, Axel, van Werkhoven, Ben, Jones, Catherine, Garijo, Daniel, Plomp, Esther, Genova, Francoise, … RDA FAIR4RS WG. (2022). FAIR Principles for Research Software (FAIR4RS Principles) (1.0). https://doi.org/10.15497/RDA00068.