Report

1. UNSW Lower Campus

1.1 Raw(Original) Voxel Datasets

Raw Data	Size	#Voxels	Description
dtmbot.xyz	9.22 GB	641,624,355	A terrain with holes, in which the buildings fit
tree.xyz	906 MB	59,640,000	Tree in lower campus
bld1-54.xyz (except for 26 and 46)	3.52GB	241,613,693 in total and 4,646,418 per building	52 buildings in lower campus
be.xyz	249 MB	17,460,029	Built Environment (H13)
blockhouse.xyz	45.4 MB	3,392,202	Blockhouse (G6)
dalton.xyz	25.5 MB	1,887,512	Dalton (F12)
quadrangle.xyz	43.9MB	3,161,733	Quadrangle (E15)
roundhouse.xyz	79.9MB	6,037,174	Roundhouse (E6)
scithe.xyz	17.2MB	1,231,821	Science Theatre (F13)

Note that:

- For raw point cloud-based voxels, its resolution is 20cm. All voxels are recorded in same INTEGER coordinate with offset (336000, 6245250, 20).
- For raw BIM-based voxels, its resolution is 10 cm. Each building is in its own INTEGER coordinate with MINXYZ.
    - For be.xyz, the offset is (336300, 6245507, 25).
    - For blockhouse.xyz, the offset is (336042, 6245613, 27).
    - For dalton.xyz, the offset is (336305, 6245569, 29).
    - For quadrangle.xyz, the offset is (336409, 6245580, 31).
    - For roundhouse.xyz, the offset is (336047, 6245651, 25).
    - For scithe.xyz, the offset is (336325, 6245582, 28).

1.2 VoxelDB Database Schema Design

Four layouts serve as the main tables, and the corresponding two semantic features serve as codelist (to ensure the uniqueness and integrity of semantic information). According to the current data set, we have 5 classes of top-level objects, namely, building, Tree, Green area, road and Terrain. Here we are going to regard the top-level object as a static object, defined as Enumerated types (enum). For detail, I list the data composition as follows:

Object	Direct data sources		Indirectly generated data
	LOD1	LOD4	
building	46 buildings with exact name attribute	6 buildings with exact IFC label and name attribute	94 extruded building footprints extracted from terrain without name attribute and IFC label
tree	793 tree blocks without name attribute and IFC label	N/A	N/A

green area	N/A	N/A	120 green area blocks without name attribute and IFC label
road	N/A	N/A	115 road blocks without name attribute and IFC label
terrain	26947 terrain blocks without name attribute and IFC label	N/A	N/A

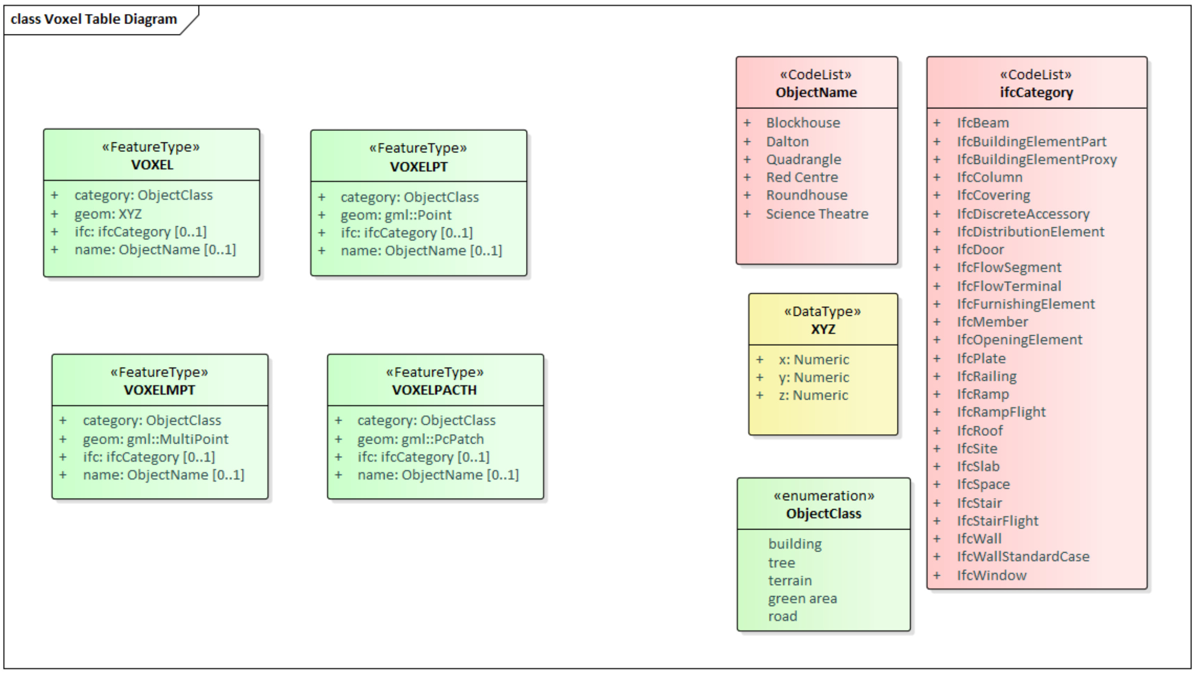


Figure 1-1. Conceptual Model of VoxelDB

1.3 Summary of Four Data Layouts in voxeldb

In this voxeldb, we consider four kinds of data layouts: flat array, point, multipoint, and pcpatch.

Data Layouts	**ARRAY**	**POINT**	**MULTIPOINT**	**PCPATCH**
Table Name	voxel	voxelpt	voxelmpt	voxelpatch
Table Size	59 GB	70 GB	568 kB	360 kB
#records	976,048,519	976,048,519	28,193	28,193
Geometry	NO	POINT	MULTIPOINT	PCPATCH
Columns	id category ifc name x y z	id category geom ifc name	id category geom ifc name	id category pa ifc name
#index	1	2	2	2
General Index	YES	YES	YES	YES
	x, y, z, category, name, ifc	category, name, ifc	category, name, ifc	category, name, ifc
	NO	YES	YES	YES

Geometry Index	N/A	point geometry	multipoint geometry	2D bounds of the patch
Index Size	74 GB	103 GB	264 kB	200 kB

1.4 Two Semantic CodeLists in voxeldb (For end-user queries)

1.4.1 ObjectName

The object name given in main table are defined by the codelist "ObjectName". The only semantic information we have so far is the building name.

Code list of *name* attribute (total 48 buildings)		
Rupert Myers-M15	International House-C6	UNSW Village-B10
Barker Street Carpark-N18	Law-F8	Blockhouse-G6
Sam Cracknell Pavilion-H8	Science Theatre-F13	Dalton-F12
Ainsworth Building-J17	Computer Science-K17	Willis Annexe-J18
UNSW Business School-E12	Quadrangle-E15	Electrical Engineering-G17
Rex Vowels Theatre-F17	Robert Webster-G14	Webster Theatres-G15
Red Centre-H13	Newton-J12	Keith Burrows Theatre-J14
Physics Theatre-K14	Old Main-K15	Chemical Sciences-F10
University Terraces-B8	Warrane College-M7	New College-L6
Shalom College-N9	Barker Apartments-N13	House at Pooh Corner-N8
Swimming Pool-B4	UNSW Fitness and Aquatic Centre-B5	Tyree Energy Technologies-H6
Goldstein Hall-D16	UNSW Hall-D14	Fig Tree Theatre-B14d
White House-C15	Roundhouse-E6	Squarehouse-E4
Colombo House-B16	Goldstein College-B17	Fig Tree Hall-B18
Philip Baxter College-D18	Basser College-D17	Old Tote-B15

1.4.2 ifcCategory

The ifc object category given in main table are defined by the codelist "ifcCategory". There are currently 26 IFC objects in our database.

Code list of *ifc* attribute (total 26 IFC objects)		
IfcBeam	IfcBuildingElementPart	IfcBuildingElementProxy
IfcColumn	IfcCovering	IfcDiscreteAccessory
IfcDistributionElement	IfcDoor	IfcFlowSegment
IfcFlowTerminal	IfcFurnishingElement	IfcMember
IfcOpeningElement	IfcPlate	IfcRailing
IfcRamp	IfcRampFlight	IfcSpace
IfcRoof	IfcSite	IfcSlab
IfcStair	IfcStairFlight	IfcWall
IfcWallStandardCase	IfcWindow	

1.5 Functionality and Simple Sample

Semantic query:

- [YES] which category? want to find 'building's? 'terrain's? Use 'category' directly!
- [YES] which building? want to find 'Red Centre'? 'Roundhouse'? Use 'name' and LIKE keyword in your query!
- [YES] which ifc object? want to find 'IfcDoor' in certain building or all buildings? Use 'ifc' directly!

- [NO] which road object? want to find “UNIVERSITY MALL”?
- [NO] which green area? want to find “quadrangle lawn”?
- [NO] which building footprint? want to find “red centre” building’s footprint in terrain?

Spatial query:

- [YES] all overall spatial queries valid on “multipoint” and “pcpatch”
- [NO] queries that do geometric splitting, like return door in “red centre” building whose height is larger than 20m? return rooms in ‘red centre’ building in level-4?

Some samples:

```
select name, ifc, category, st_astext(geom)
from voxelmpt
where category='building' and ifc is not null
limit 2;
```

	name character varying	ifc character varying	category objectclass	st_astext text
1	Blockhouse-G6	IfcBuildingElementProxy	building	MULTIPOINT Z (336046.9 6245635.6 30.4,...
2	Blockhouse-G6	IfcDoor	building	MULTIPOINT Z (336050.9 6245639 32.9,33...

Figure 1-2. Two buildings with ifc label in “voxelmpt”.

```
select name, ifc, category, st_astext(geom)
from voxelmpt
where name like 'Red Centre%' and (ifc='IfcDoor' or ifc='IfcSpace');
```

	name character varying	ifc character varying	category objectclass	st_astext text
1	Red Centre-H13	IfcSpace	building	MULTIPOINT Z (336312.7 6245547.1 4...
2	Red Centre-H13	IfcDoor	building	MULTIPOINT Z (336307.7 6245536.3 3...

Figure 1-3. door and space in ‘red centre’ building in “voxelmpt”.

```
select name, ifc, category, st_astext(PC_EnvelopeGeometry(pa))
from voxelpatch
where category='building' and ifc is not null
limit 2;
```

	name character varying	ifc character varying	category objectclass	st_astext text
1	Quadrangle-E15	IfcBeam	building	POLYGON((336410.8 6245594,336410.8 624...
2	Blockhouse-G6	IfcBuildingElementProxy	building	POLYGON((336046.9 6245621.5,336046.9 6...

Figure 1-4. Two buildings with ifc label in “voxelpatch”.

2. Pre-process -- Object Matching between Different Data Source

In this section, we did some pre-processing on raw data from multiple sources. For end users, you can skip this section.

2.1 Checking Data Info

Data	#Voxels
bld1-54	241,613,693 in total and 4,646,418 per building
tree	59,640,000
dtmbot	641,624,355
be	17,460,029
blockhouse	3,392,202
dalton	1,887,512
quadrangle	3,161,733
roundhouse	6,037,174
scithe	1,231,821
Ifcid is not null	**33,170,471**
Total	**976,048,519**

2.2 Assign Temporary classID for IFC buildings

In table “voxel” and “voxelpt”, GIS data occupied 944,110,209 rows.

At this moment, we assume the 6 IFC models with classID 57, 58, 59, 60, 61, and 62, respectively.

```
\COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\BE\classmodel.xyz' DELIMITER ' ';
UPDATE voxel SET classID=57 WHERE classid IS NULL;
\COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\BlockHouse\classmodel.xyz' DELIMITER ' ';
UPDATE voxel SET classID=58 WHERE classid IS NULL;
\COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\Dalton\classmodel.xyz' DELIMITER ' ';
UPDATE voxel SET classID=59 WHERE classid IS NULL;
\COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\Quadrangle\classmodel.xyz' DELIMITER ' ';
UPDATE voxel SET classID=60 WHERE classid IS NULL;
\COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\Roundhouse\classmodel.xyz' DELIMITER ' ';
UPDATE voxel SET classID=61 WHERE classid IS NULL;
\COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\SciThe\classmodel.xyz' DELIMITER ' ';
UPDATE voxel SET classID=62 WHERE classid IS NULL;
```

```
COPY 17460029
Time: 406199.727 ms (06:46.200)
UPDATE 17460029
Time: 758352.674 ms (12:38.353)
COPY 3392202
Time: 159865.359 ms (02:39.865)
UPDATE 3392202
Time: 541930.906 ms (09:01.931)
COPY 1887512
Time: 70745.110 ms (01:10.745)
UPDATE 1887512
Time: 441805.998 ms (07:21.806)
COPY 3161733
Time: 269617.281 ms (04:29.617)
UPDATE 3161733
Time: 553708.877 ms (09:13.709)
COPY 6037174
Time: 285602.020 ms (04:45.602)
UPDATE 6037174
Time: 629060.986 ms (10:29.061)
COPY 1231821
Time: 128098.394 ms (02:08.098)
UPDATE 1231821
Time: 529605.717 ms (08:49.606)
```

Figure 2.1. Log info for IFC data importing

2.3 Update classID for IFC buildings

For BE building, through computing bld19 and its the MAX & MIN (x,y) range in EPSG:28356 CRS, it is easy to find that they are in high probability the same building.

```sql
SELECT MAX(x)*0.1+336300 AS maxx, MIN(x)*0.1+336300 AS minx, MAX(y)*0.1+6245507 AS maxy, MIN(y)*0.1+6245507 AS miny
FROM voxel
WHERE classid=57;

SELECT MAX(x)*0.2+336000 AS maxX, MIN(x)*0.2+336000 AS minX, MAX(y)*0.2+6245250 AS maxY, MIN(y)*0.2+6245250 AS minY
FROM voxel
WHERE classID=19;
```

	maxx numeric	minx numeric	maxy numeric	miny numeric
1	336450.8	336301.6	6245552.9	6245508.6

1	336384.8	336300.8	6245552.4	6245519.4

Figure 2.2. (x,y) range for BE in EPSG:28356 CRS

Then, visualizing above two buildings in CloudCompare, it looks similar, at least in shape.

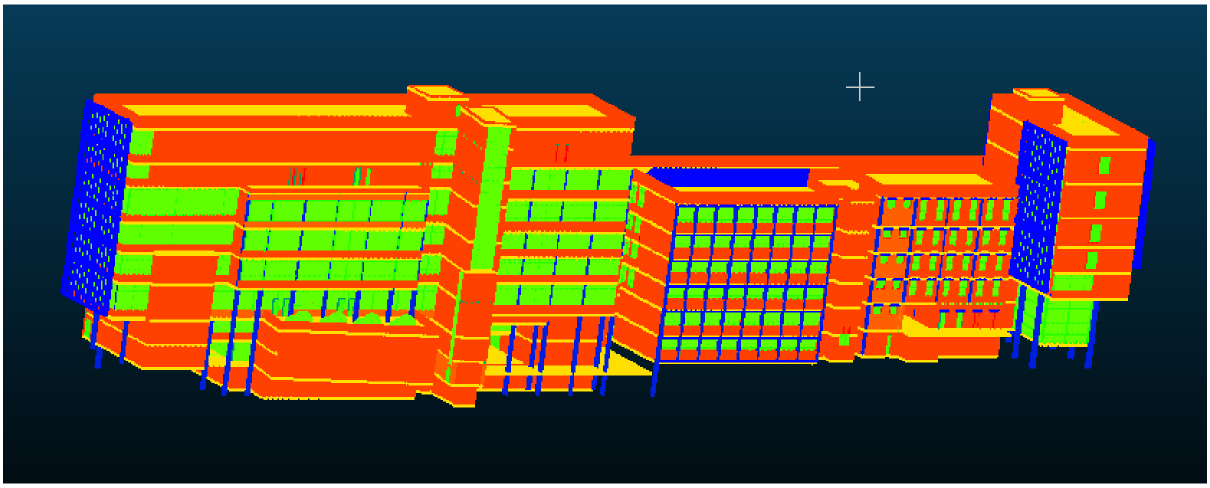


Figure 2.3. BE building in CloudCompare

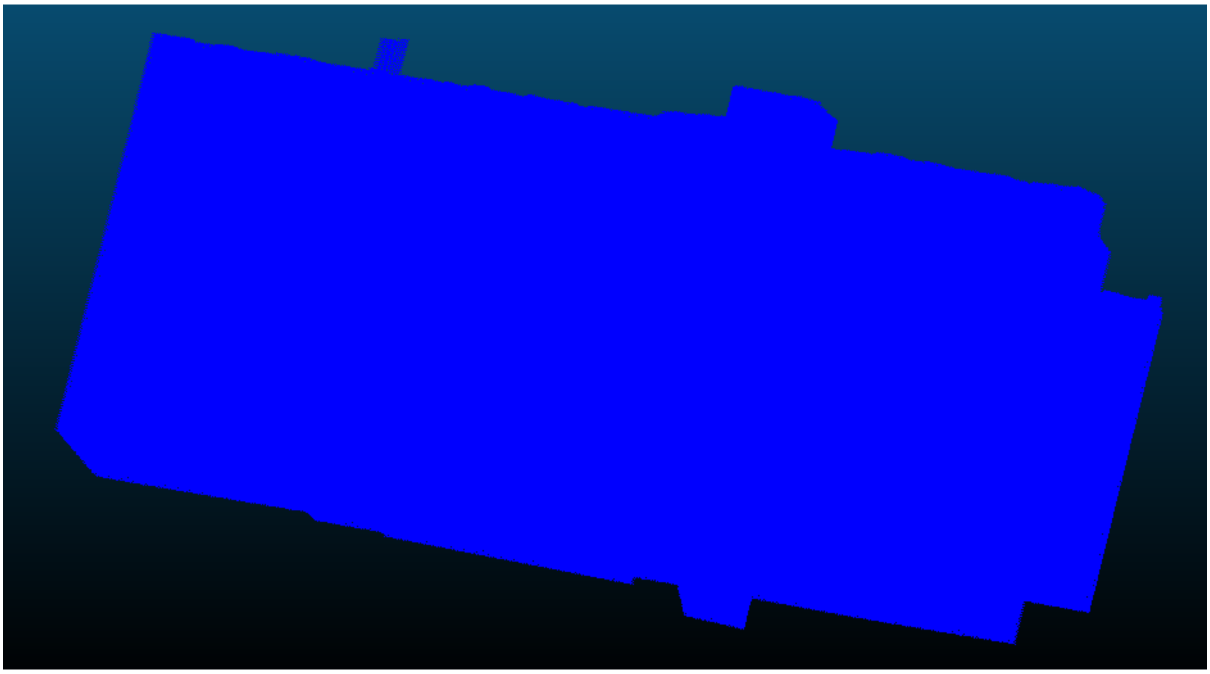


Figure 2.4. bld19 in CloudCompare

For Blockhouse, Dalton, Quadrangle, Roundhouse, SciThe buildings, calculating (x,y) range for all buildings with classID<=54. And then retrieve same range for above 5 buildings to do matching.

```sql
SELECT MAX(x)*0.1+336042 AS maxx, MIN(x)*0.1+336042 AS minx, MAX(y)*0.1+6245613 AS maxy, MIN(y)*0.1+6245613 AS miny
FROM voxel
WHERE classid=58;

SELECT MAX(x)*0.1+336305 AS maxx, MIN(x)*0.1+336305 AS minx, MAX(y)*0.1+6245569 AS maxy, MIN(y)*0.1+6245569 AS miny
FROM voxel
WHERE classid=59;

SELECT MAX(x)*0.1+336409 AS maxx, MIN(x)*0.1+336409 AS minx, MAX(y)*0.1+6245580 AS maxy, MIN(y)*0.1+6245580 AS miny
FROM voxel
WHERE classid=60;
```

```sql
SELECT MAX(x)*0.1+336047 AS maxx, MIN(x)*0.1+336047 AS minx, MAX(y)*0.1+6245651 AS maxy, MIN(y)*0.1+6245651 AS miny
FROM voxel
WHERE classid=61;

SELECT MAX(x)*0.1+336325 AS maxx, MIN(x)*0.1+336325 AS minx, MAX(y)*0.1+6245582 AS maxy, MIN(y)*0.1+6245582 AS miny
FROM voxel
WHERE classid=62;

SELECT MAX(x)*0.2+336000 AS maxX, MIN(x)*0.2+336000 AS minX, MAX(y)*0.2+6245250 AS maxY, MIN(y)*0.2+6245250 AS minY
FROM voxel
WHERE classID<=54
GROUP BY classID;
```

Building Name	Range in (x,y) EPSG:28356 CRS				
Blockhouse		maxx numeric	minx numeric	maxy numeric	miny numeric
	1	336126.3	336043.0	6245650.8	6245614.9
Dalton		maxx numeric	minx numeric	maxy numeric	miny numeric
	1	336332.4	336306.0	6245643.1	6245570.7
Quadrangle		maxx numeric	minx numeric	maxy numeric	miny numeric
	1	336501.5	336410.1	6245626.4	6245581.7
Roundhouse		maxx numeric	minx numeric	maxy numeric	miny numeric
	1	336125.8	336048.0	6245749.7	6245652.9
SciThe		maxx numeric	minx numeric	maxy numeric	miny numeric
	1	336380.3	336326.9	6245633.4	6245583.5

	maxx numeric	minx numeric	maxy numeric	miny numeric
1	336447.4	336385.0	6245404.8	6245321.4
2	336135.8	336085.8	6245789.2	6245746.2
3	336483.0	336222.0	6245811.4	6245730.0
4	336525.6	336450.0	6245403.8	6245313.4
5	336207.4	336125.8	6245668.0	6245607.2
6	336121.4	336045.4	6245647.8	6245623.2
7	336156.8	336122.0	6245573.2	6245534.8
8	336380.0	336338.8	6245634.0	6245597.2
9	336331.6	336308.2	6245646.6	6245575.4
10	336516.6	336462.6	6245500.2	6245411.6
11	336555.4	336506.8	6245495.6	6245383.4
12	336397.4	336285.2	6245684.4	6245643.2
13	336538.8	336394.4	6245675.4	6245573.6
14	336572.2	336480.4	6245639.2	6245541.0
15	336519.4	336509.4	6245578.0	6245568.6
16	336473.0	336368.4	6245592.4	6245559.2
17	336449.6	336429.2	6245605.2	6245582.2
18	336449.8	336382.6	6245545.4	6245510.0
19	336384.8	336300.8	6245552.4	6245519.4
20	336314.8	336293.6	6245539.2	6245474.2
21	336403.6	336376.6	6245519.2	6245499.8
22	336404.6	336374.8	6245501.2	6245470.6
23	336458.4	336307.6	6245522.8	6245441.4

24	336190.6	336134.6	6245698.0	6245669.8
25	336309.4	336219.6	6245641.8	6245589.8
26	336205.2	336146.6	6245822.8	6245754.6
27	336106.6	336062.2	6245439.8	6245374.0
28	336093.8	336048.8	6245513.4	6245444.4
29	336225.6	336146.4	6245397.6	6245353.4
30	336348.4	336227.4	6245383.6	6245340.6
31	336366.8	336352.2	6245366.6	6245358.2
32	336364.8	336357.0	6245353.6	6245345.6
33	336148.8	336112.4	6245398.8	6245364.2
34	336169.4	336144.0	6245740.4	6245705.6
35	336049.2	336000.0	6245846.2	6245800.0
36	336138.8	336051.8	6245839.8	6245794.4
37	336119.8	336032.4	6245585.4	6245511.4
38	336517.2	336478.0	6245706.4	6245667.0
39	336228.4	336199.6	6245746.2	6245709.6
40	336460.6	336359.0	6245730.4	6245677.2
41	336450.8	336426.6	6245776.0	6245754.6
42	336448.0	336432.2	6245729.4	6245713.0
43	336130.4	336051.8	6245740.6	6245660.2
44	336040.2	336009.8	6245735.2	6245698.6
45	336529.6	336503.0	6245600.2	6245577.0
46	336527.4	336489.6	6245780.4	6245726.8
47	336563.8	336529.4	6245774.8	6245720.2
48	336594.4	336562.0	6245770.6	6245713.6
49	336586.2	336553.2	6245709.6	6245653.4
50	336554.6	336521.6	6245716.8	6245662.2
51	336481.0	336458.6	6245771.8	6245754.6
52	336236.6	336225.0	6245779.6	6245771.0

Figure 2.5. (x,y) range for all 52 building

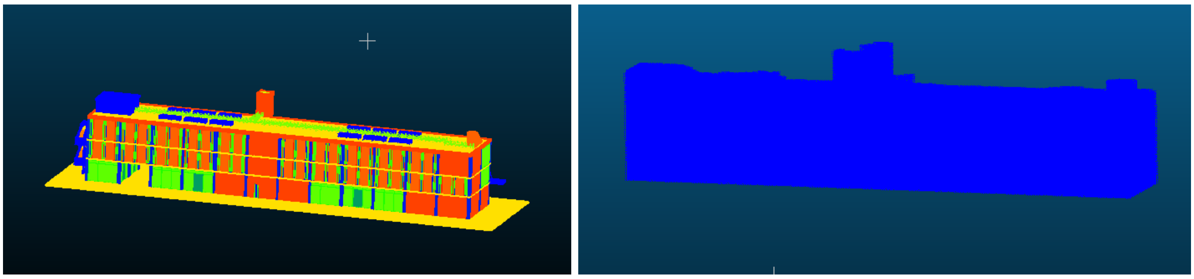


Figure 2.6. Blockhouse

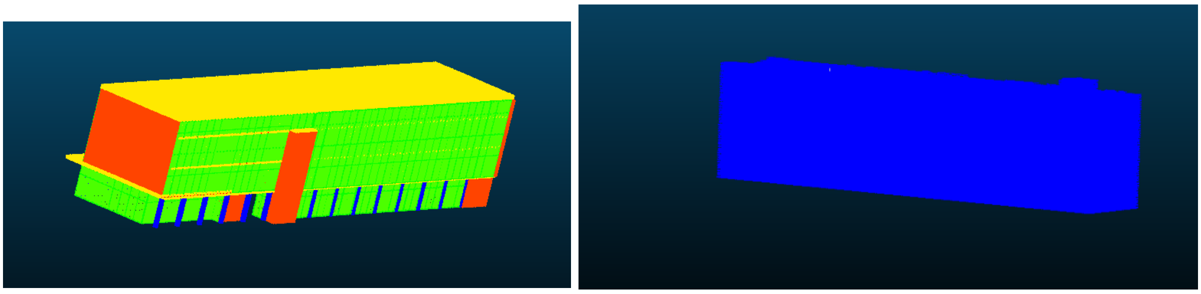


Figure 2.7. Dalton

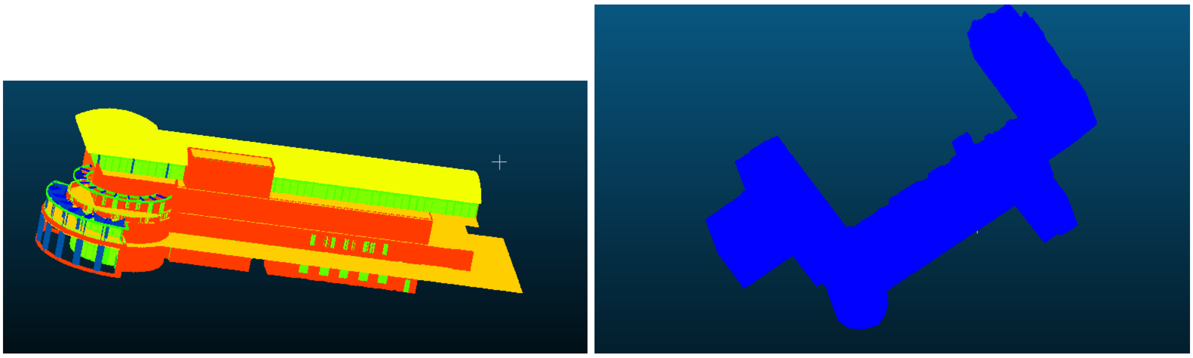


Figure 2.8. Quadrangle

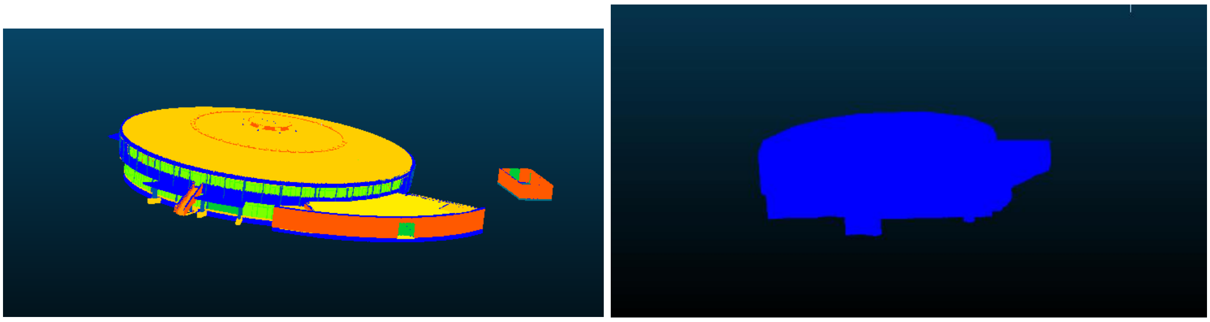


Figure 2.9. Roundhouse

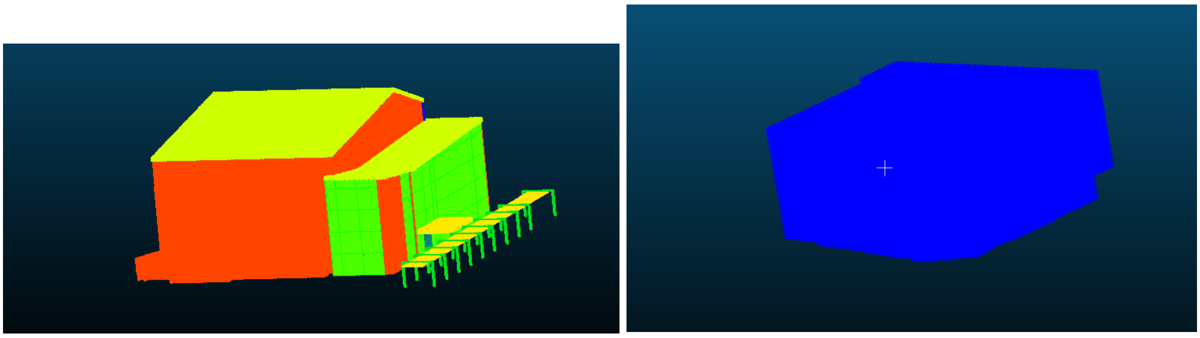


Figure 2.10. Science Theatre

In summary, the corresponding bld are list in below table:

Id	Name	Num	Old classID	New classID
Bld19	Built Environment	H13	57	19
Bld6	Blockhouse	G6	58	6
Bld9	Dalton	F12	59	9
Bld13	Quadrangle	E15	60	13
Bld44	Roundhouse	E6	61	44
Bld8	Science Theatre	F13	62	8

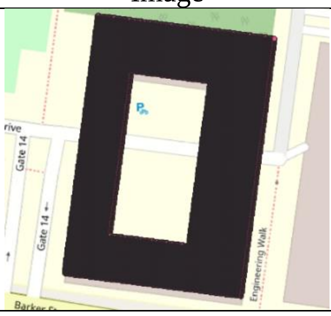
Update the 6 buildings:

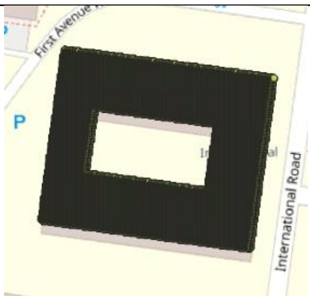
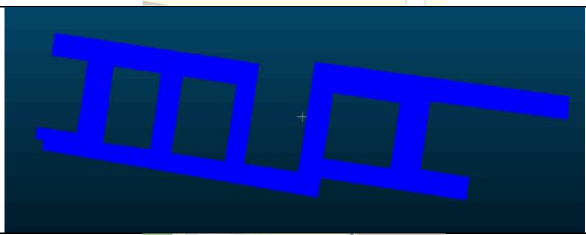
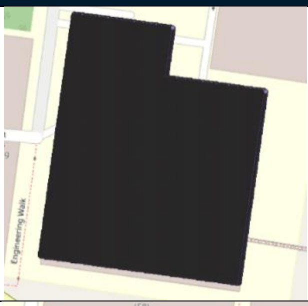
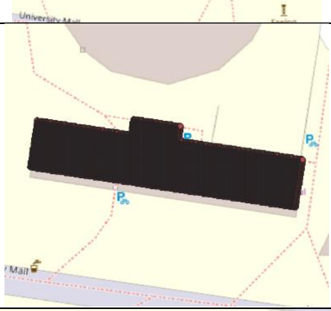
```sql
UPDATE voxel SET classID=19 WHERE classid=57;
UPDATE voxel SET classID=6 WHERE classid=58;
UPDATE voxel SET classID=9 WHERE classid=59;
UPDATE voxel SET classID=13 WHERE classid=60;
UPDATE voxel SET classID=44 WHERE classid=61;
UPDATE voxel SET classID=8 WHERE classid=62;
```

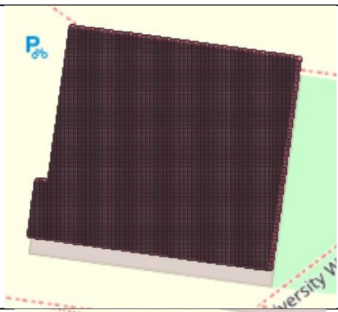
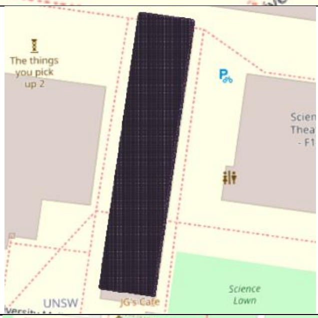
Time: 1h 12m 19s

2.4 Assign Name for Each Building in Lower Campus

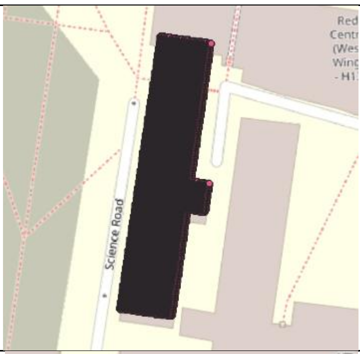
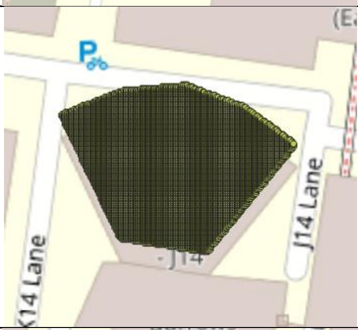
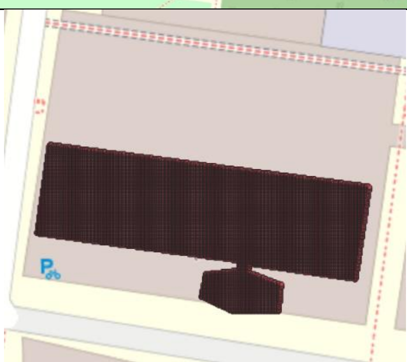
List of building name and ID

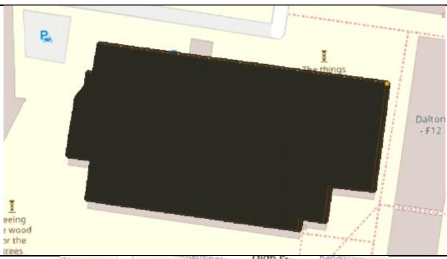
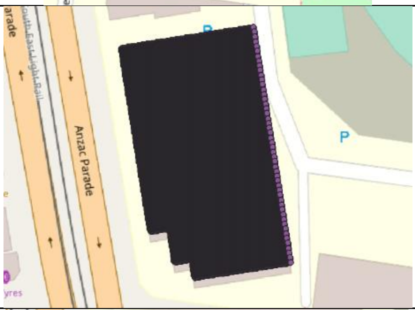
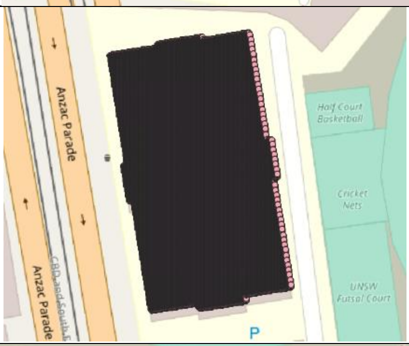
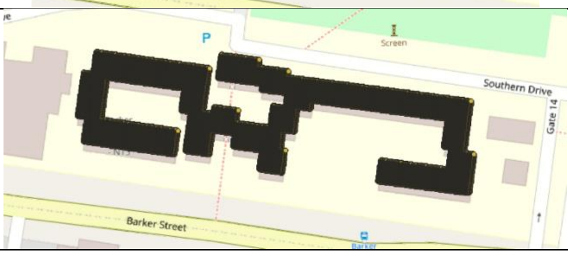
num	Image	Name	ID
bld1		Rupert Myers	M15

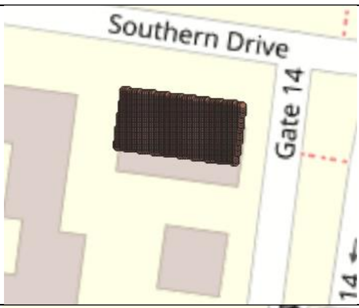
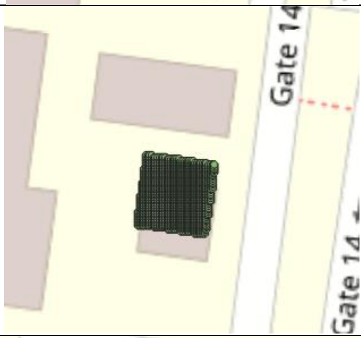
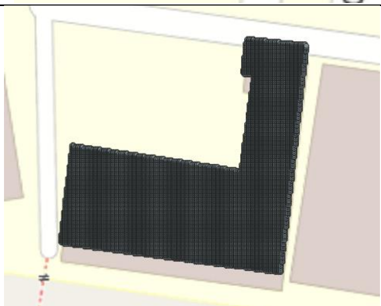
bld2		International House	C6
bld3		UNSW Village	B10
bld4		Barker Street Carpark	N18
bld5		Law	F8
bld6		Blockhouse	G6
bld7		Sam Cracknell Pavilion	H8

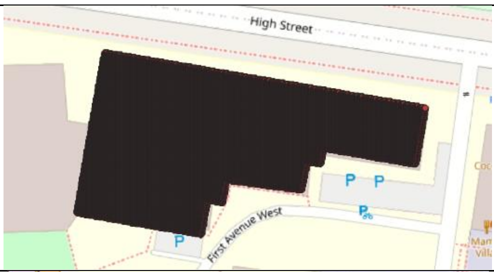
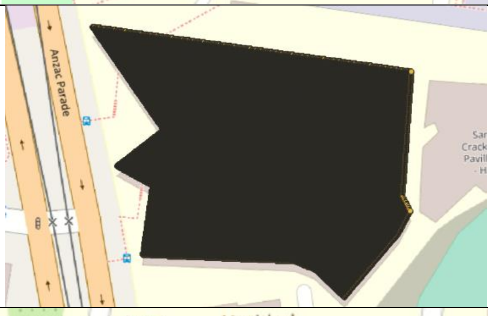
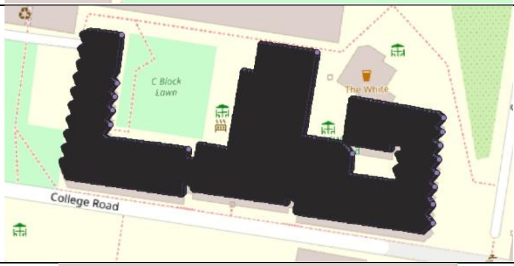
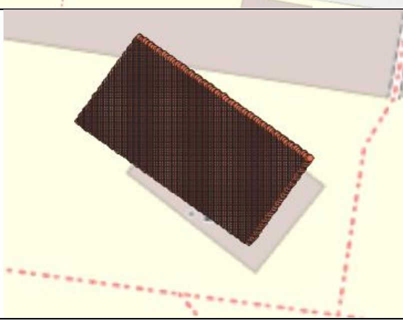
bld8		Science Theatre	F13
bld9		Dalton	F12
bld10		Ainsworth Building Computer Science	J17 K17
bld11		Willis Annexe	J18
bld12		UNSW Business School	E12
bld13		Quadrangle	E15

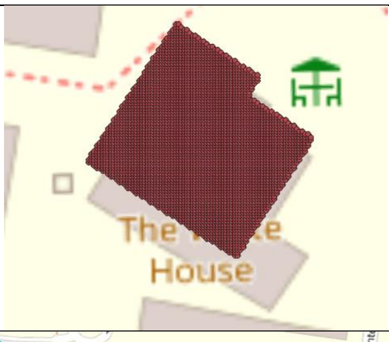
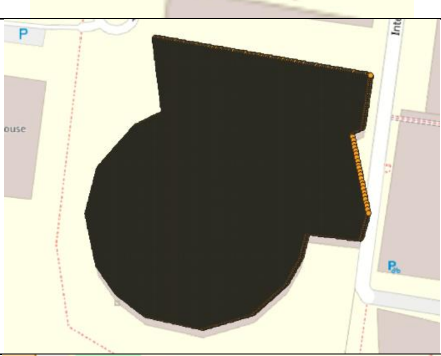
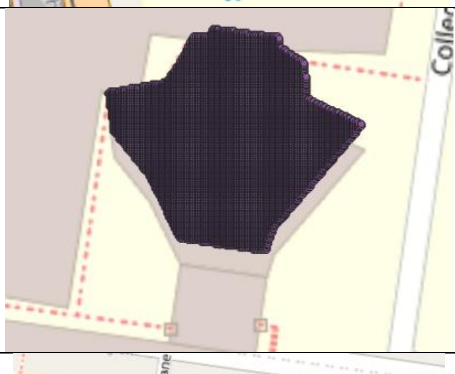
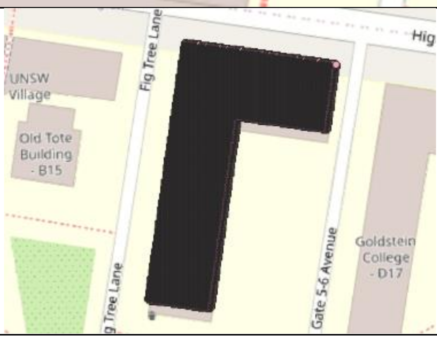
bld14		Electrical Engineering	G17
bld15		Rex Vowels Theatre	F17
bld16		Robert Webster	G14
bld17		Webster Theatres	G15
bld18		Red Centre	H13
bld19		Red Centre	H13

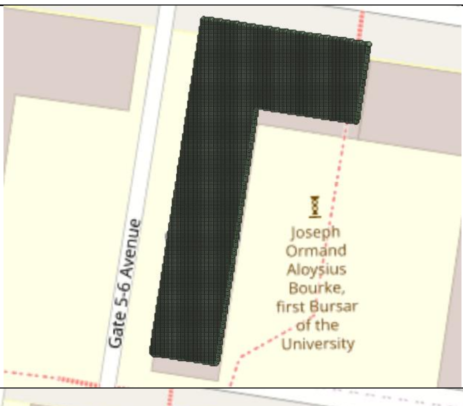
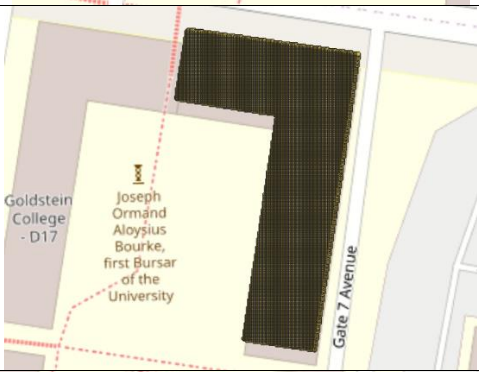
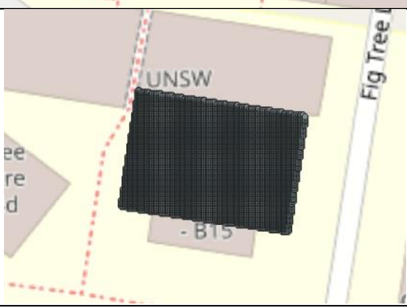
bld20		Newton	J12
bld21		Keith Burrows Theatre	J14
bld22		Physics Theatre	K14
bld23		Old Main	K15
bld24		N/A	N/A

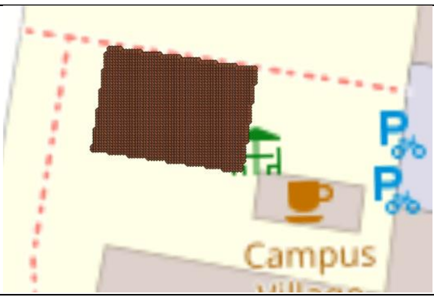
bld25		Chemical Sciences	F10
bld27		University Terraces	B8
bld28		Warrane College	M7
bld29		New College	L6
bld30		Shalom College	N9
bld31		Barker Apartments	N13

bld32		N/A	N/A
bld33		N/A	N/A
bld34		House at Pooh Corner	N8
bld35		N/A	N/A
bld36		Swimming Pool	B4

bld37		UNSW Fitness and Aquatic Centre	B5
bld38		Tyree Energy Technologies	H6
bld39		Goldstein Hall	D16
bld40		N/A	N/A
bld41		UNSW Hall	D14
bld42		Fig Tree Theatre	B14d

bld43		N/A White House	N/A C15
bld44		Roundhouse	E6
bld45		Squarehouse	E4
bld47		Rex Vowels Theatre	F17
bld48		Colombo House	B16

bld49		Goldstein College	B17
bld50		Fig Tree Hall	B18
bld51		Philip Baxter College	D18
bld52		Basser College	D17
bld53		Old Tote	N/A B15

bld54		N/A	N/A

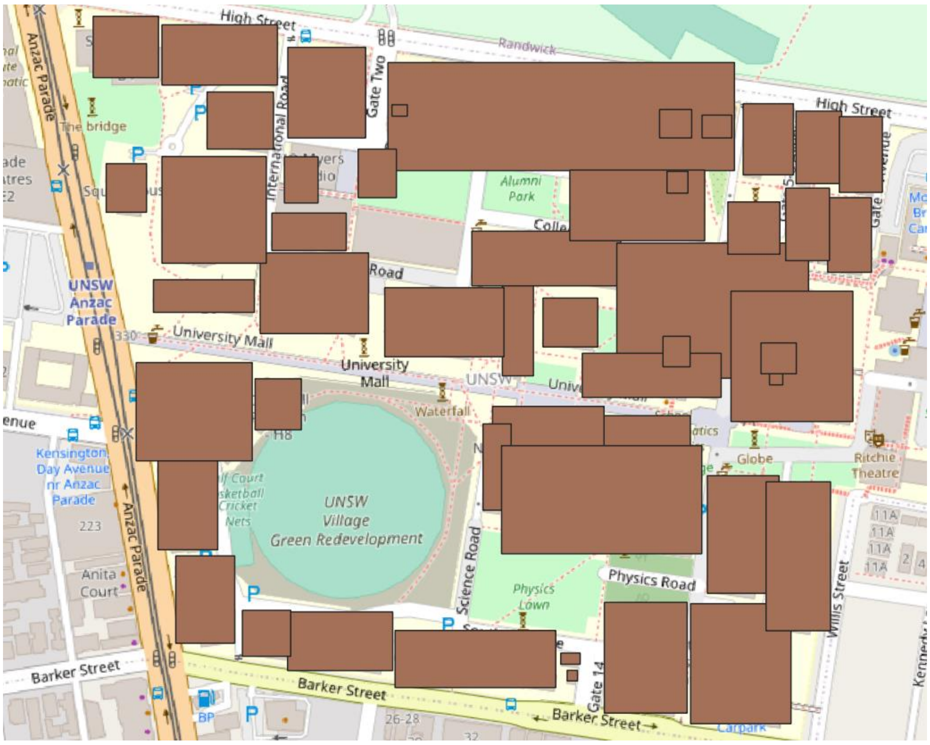


Figure 2.11. 2D bounds for all buildings

Figure 2.12. 2D bounds VS. building with IFC attribute

Figure 2.13. 2D bounds VS. building without IFC attribute

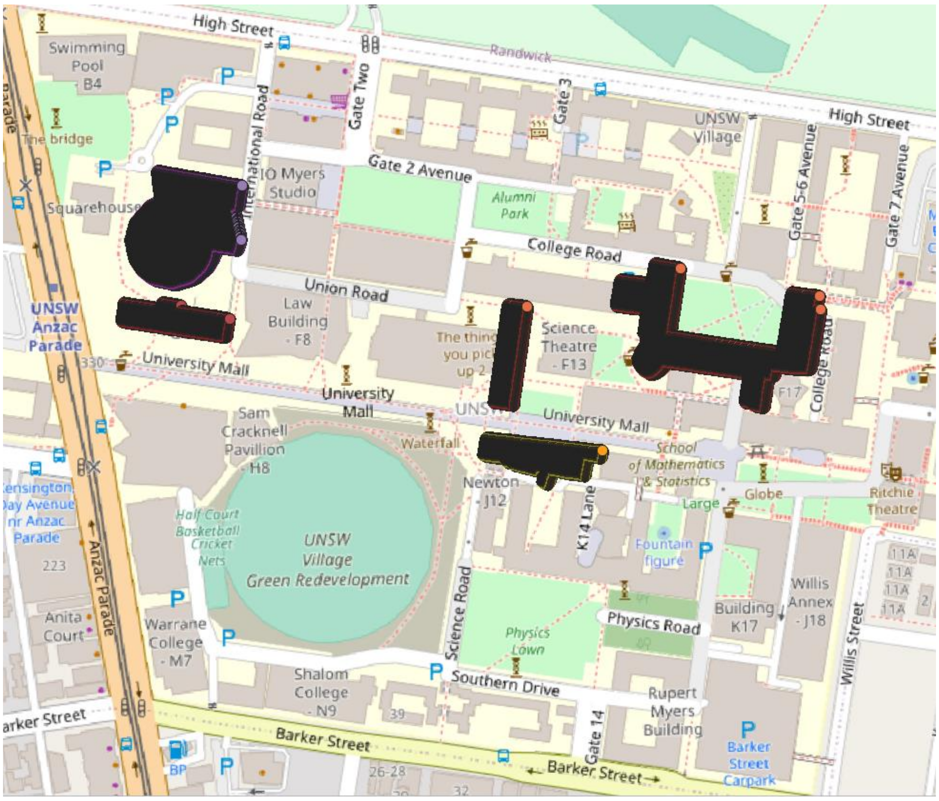


Figure 2.13. OpenStreetMap VS. building without IFC attribute

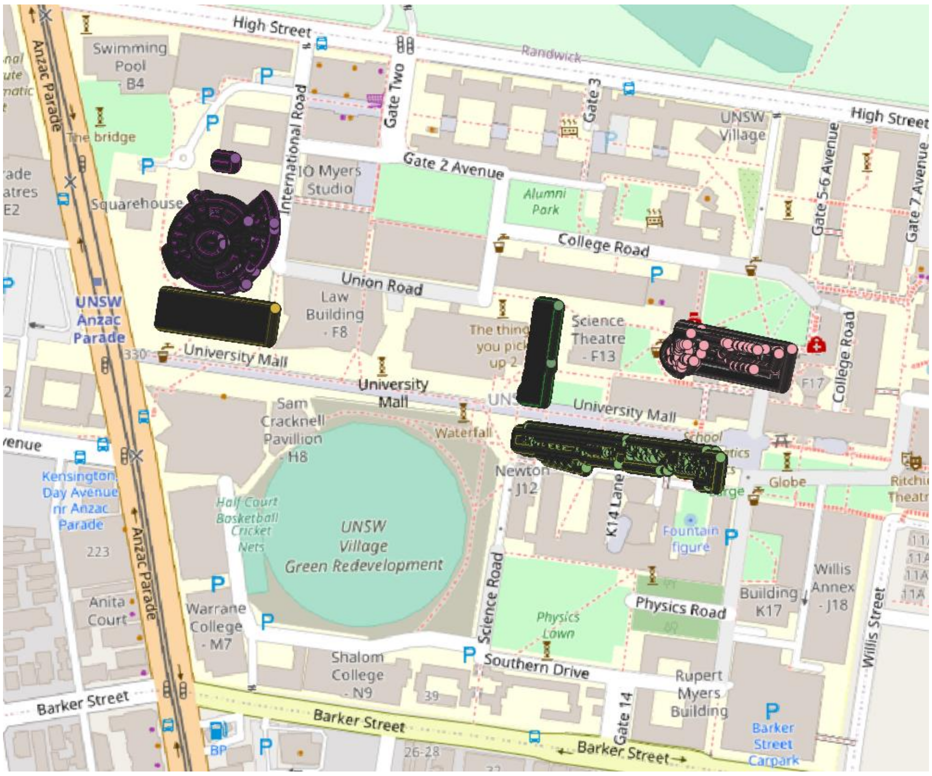


Figure 2.13. OpenStreetMap VS. building with IFC attribute

3. Create Flat ARRAY Table

First of all we consider the simplest data layout – flat table for voxel data.

It is easy to regard each (x,y,z) coordinate along with its semantic information as one record in database table. To avoid further complex geometry translation and scaling operation for end user, we choose sacrifice storage space for the convenience of spatial queries. The data type for “x”, “y” and “z” column will be set to “NUMERIC(8,1)” due to the precision of coordinates will not beyond 8.

3.1 Create Table

```sql
DROP TABLE IF EXISTS voxelflat CASCADE;
CREATE TABLE voxelflat
(
    id serial PRIMARY KEY,
    x NUMERIC(8,1) NOT NULL,
    y NUMERIC(8,1) NOT NULL,
    z NUMERIC(8,1) NOT NULL,
    classID INTEGER,
    ifcID INTEGER
);
```

Query returned successfully in 290 msec.

3.2 Import Data

We directly recalculate the coordinates (x,y,z) from the data in table “voxel”. Due to the scale and offset compositions contain seven pairs, the vast majority of voxels is in 20cm and (336000,6245250,20) offset. In contrast, the other six pairs belong to special voxel with IFC feature. We need to separate to handle them one by one.

```sql
INSERT INTO voxelflat(x,y,z,classID,ifcID)
SELECT x*0.2+336000, y*0.2+6245250, z*0.2+20, classID, ifcID FROM voxel
WHERE ifcID IS NULL;
INSERT INTO voxelflat(x,y,z,classID,ifcID)
SELECT x*0.1+336300, y*0.1+6245507, z*0.1+25, classID, ifcID FROM voxel
WHERE classID=19 AND ifcID IS NOT NULL;
INSERT INTO voxelflat(x,y,z,classID,ifcID)
SELECT x*0.1+336042, y*0.1+6245613, z*0.1+27, classID, ifcID FROM voxel
WHERE classID=6 AND ifcID IS NOT NULL;
INSERT INTO voxelflat(x,y,z,classID,ifcID)
SELECT x*0.1+336305, y*0.1+6245569, z*0.1+29, classID, ifcID FROM voxel
WHERE classID=9 AND ifcID IS NOT NULL;
INSERT INTO voxelflat(x,y,z,classID,ifcID)
SELECT x*0.1+336409, y*0.1+6245580, z*0.1+31, classID, ifcID FROM voxel
WHERE classID=13 AND ifcID IS NOT NULL;
INSERT INTO voxelflat(x,y,z,classID,ifcID)
SELECT x*0.1+336047, y*0.1+6245651, z*0.1+25, classID, ifcID FROM voxel
WHERE classID=44 AND ifcID IS NOT NULL;
INSERT INTO voxelflat(x,y,z,classID,ifcID)
SELECT x*0.1+336325, y*0.1+6245582, z*0.1+28, classID, ifcID FROM voxel
WHERE classID=8 AND ifcID IS NOT NULL;
```

Query returned successfully in 2 h 30 min 50 secs.

After finishing filling in the voxel data in table “voxelflat”, we next go to replace old “voxel” with “voxelflat”.

```sql
DROP TABLE voxel;
ALTER TABLE voxelflat RENAME TO voxel;
```

3.3 Build Index

In order to speed up the queries involving spatial or semantic retrievals, we build default B-tree indices on “x”, “y”, “z”, and two semantic columns “classID” and “ifcID”.

```sql
DROP INDEX IF EXISTS idx_voxel CASCADE;
CREATE INDEX idx_voxel ON voxel(x, y, z, classID, ifcID);
```

Query returned successfully in 1 h 27 min 31 secs.

	id integer	x numeric (8,1)	y numeric (8,1)	z numeric (8,1)	classid integer	ifcid integer
1	571255464	336434.0	6245891.4	31.6	56	[null]
2	571255465	336434.0	6245891.4	31.8	56	[null]
3	571255466	336434.0	6245891.6	20.2	56	[null]
4	571255467	336434.0	6245891.6	20.4	56	[null]
5	571255468	336434.0	6245891.6	20.6	56	[null]

Figure . Example of flat ARRAY table.

4. Conversion to POINT geometry

In this section, we consider the second data layout, which stores each voxel as a geometry POINT including (x,y,z) by using PostGIS Geometry Constructors – “ST_MakePoint”.

Spatial indices are one of the greatest assets of PostGIS. Here we use the generic index structure (GIST) for geometry column, and continue use B-tree index for other two semantic columns. While you can create a b-tree index on a geomtery object (point, region, etc) it can only actually be used for equality as ordering comparisons like > are generally meaningless for such objects. A GiST index is required to support more complex and general comparisons like "contains", "intersects", etc. In a nutshell: B-Tree indexes perform better, but GiST indexes are more flexible.

4.1 Create Table

```
CREATE EXTENSION IF NOT EXISTS POSTGIS;
DROP TABLE IF EXISTS voxelpt CASCADE;
CREATE TABLE voxelpt
(
    id serial PRIMARY KEY,
    classID INTEGER,
    ifcID INTEGER,
    geom geometry(POINTZ,28356)
);
```

Query returned successfully in 4 secs 592 msec.

4.2 Import Data

```
INSERT INTO voxelpt(classID, ifcID, geom) SELECT classID, ifcID, ST_SetSRID (ST_MakePoin
t(x,y,z), 28356) FROM voxel AS VALUES;
```

Query returned successfully in 2 h 21 min 52 secs.

4.3 Build Index

```
DROP INDEX IF EXISTS idx_voxelpt CASCADE;
DROP INDEX IF EXISTS geom_ voxelpt CASCADE;
CREATE INDEX idx_voxelpt ON voxelpt(classID, ifcID);
CREATE INDEX geom_voxelpt ON voxelpt USING GIST (geom);
```

Query returned successfully in 6 h 33 min 18 secs.

	id integer	classid integer	ifcid integer	geom geometry
1	2	56	[null]	01010000A0C46E000000000000C88814419A9999D980D357419...
2	3	56	[null]	01010000A0C46E000000000000C88814419A9999D980D35741C...
3	4	56	[null]	01010000A0C46E000000000000C8881441666666E680D357413...
4	5	56	[null]	01010000A0C46E000000000000C8881441666666E680D357416...
5	6	56	[null]	01010000A0C46E000000000000C8881441666666E680D357419...

Figure Example for POINT geometry table

5. Conversion to MULTIPOINT geometry

To enable the use of geometrical functions from PostGIS, POINTs are transformed into MULTIPOINT type, which is a collection of POINTs.

5.1 Create table

```
DROP TABLE IF EXISTS voxelmpt CASCADE;
CREATE TABLE voxelmpt
(
    id serial PRIMARY KEY,
    classID INTEGER,
    ifcID INTEGER,
    geom geometry(MULTIPOINTZ,28356)
);
```

Query returned successfully in 272 msec.

5.2 Partition Principles

Rule-1: For general building objects without IFC features, each building is one Multipoint.

Rule-2: For building objects with IFC objects, we regard each IFC object as one Multipoint.

Rule-3: For tree and dtmbot, we combine GIS dataset to decide the patch size.

5.3 Data Generation

5.3.1 Building without IFC

First, for general building voxels without IFC semantic information, we straightforward collect all POINTs in "voxelpt" with same classID into one MULTIPOINT geometry. We have tried to generate multipoint geometry including each building, but due to geometry size limit

```
DO $$
DECLARE
    f record;
BEGIN
    FOR idx in 1..54
    LOOP
        IF idx = 26 OR idx = 46 THEN
            raise notice 'The buidling % could not be found', idx;
        ELSE
            INSERT INTO voxelmpt(classID, geom)
            VALUES (idx, ST_Collect(ARRAY(SELECT geom FROM voxelpt WHERE ifcID IS NULL AND classID=idx)));
        END IF;
    END LOOP;
END;
$$

```

Query returned successfully in 101 min 50 secs.

5.3.2 Building with IFC

Next, for building objects with IFC features, each partition is a collection of IFC voxels with same ifcID.

```
DO $$
DECLARE
    f record;
BEGIN
    FOR f in SELECT DISTINCT classID, ifcID
        FROM voxelpt
    LOOP
        INSERT INTO voxelmpt(classID, ifcID, geom)
        VALUES (f.classID, f.ifcID, ST_Collect(ARRAY(SELECT geom FROM voxelpt WHERE ifcID IS NOT NULL AND classID=f.classID AND ifcID=f.ifcID)));
    END LOOP;
END;
$$
```

Query returned successfully in 9 min 31 secs.

5.3.3 Tree

Last, for tree, there is another tree data set. It consists of points, which represent the trunk of the tree. So using this point as a center (total 1345 POINTs) and assuming an horizontal radius (3-4m) we can try to partition the trees. Note that, above strategy may be not suitable to our tree voxels since the interception range of Voxel data and GIS data is different, and the location is offset.

```
pg_dump -U postgres -h 149.171.16.253 -p 5432 -t tree crc_lcl_proj | psql -h 149.171.16.253 -p 5433 -U postgres -d voxeldb
pg_dump -U postgres -h 149.171.16.253 -p 5432 -t terrain crc_lcl_proj | psql -h 149.171.16.253 -p 5433 -U postgres -d voxeldb
```

PS D:\Program Files\PostgreSQL\12\bin> .\pg_dump.exe -U postgres -h 149.171.16.253 -p 5432 -t terrain crc_lcl_proj | .\psql.exe -h 149.171.16.253 -p 5433 -U postgres -d voxeldb

A straightforward method is calculating the buffer of each GIS tree point with radius=4m via PostGIS function "ST_Buffer" (https://postgis.net/docs/ST_Buffer.html). Assign tree voxels an ifcID=30 and name='tree'.

```
DO $$
BEGIN
    FOR idx in 1..1345
    LOOP
        INSERT INTO voxelmpt(classID, ifcID, geom)
        VALUES (55, 30, ST_Collect(ARRAY(
            SELECT V.geom
            FROM voxelpt V
            JOIN tree T ON ST_WITHIN(V.geom, ST_Buffer(T.geom, 4, 'quad_segs=8'))

            WHERE V.classid=55 AND T.id=idx)));
        DELETE FROM voxelmpt WHERE classID=55 AND ifcID=30 AND geom IS NULL;
    END LOOP;
END;
$$
```

Query returned successfully in 122 min 57 secs.   ***793 records***

5.3.4 Dtmbot

Actually, we have to create terrain objects with respect to the surface objects as paths, gardens, roads. Jinjin has these vector non-overlapping polygons. You can use them to find 'all voxels of the dtm in a specific polygon' and assign the semantic of the polygon. Then the dtm will be not partitioned randomly but according to the surface objects.

```
pg_dump -U postgres -h 149.171.16.253 -p 5432 -t lawn crc_lcl_proj | psql -h 149.171.16.253 -p 5433 -U postgres -d voxeldb
pg_dump -U postgres -h 149.171.16.253 -p 5432 -t road crc_lcl_proj | psql -h 149.171.16.253 -p 5433 -U postgres -d voxeldb
pg_dump -U postgres -h 149.171.16.253 -p 5432 -t building crc_lcl_proj | psql -h 149.171.16.253 -p 5433 -U postgres -d voxeldb
```

For Windows OS, backup the table "lawn" and "road" through PgAdmin with "plain" and "UTF8" in .sql file. After that, restoring these two tables through psql command:

```
\i C:/Users/z5039792/Documents/Vox3DMod/data/lawn.sql
\i C:/Users/z5039792/Documents/Vox3DMod/data/road.sql
\i C:/Users/z5039792/Documents/Vox3DMod/data/building.sql
```

We consider the following objects on the terrain that will become the partition rules.

- Road
- Greenarea
- Building footprint

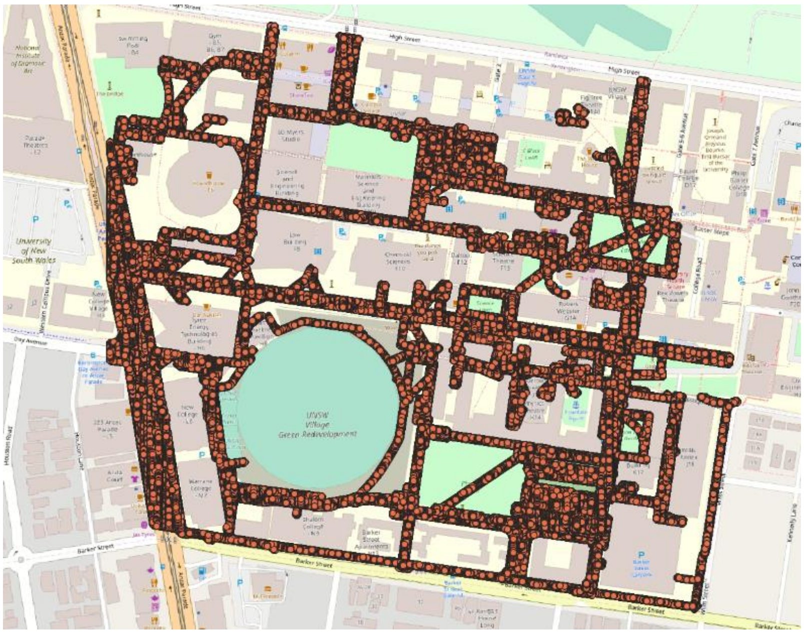
5.3.4.1 Road Surface

For "road", there are 118 polygons, we aim to group dtmbot voxels into such 118 polygons. Here, we can utilize function "ST_WITHIN" (https://postgis.net/docs/ST_Within.html) to decide which set of voxelx are located in which polygon. Besides, we need to allocate one semantic label for this kind of "road" object. Due to few knowledge of road in UNSW campus (only FME id), we decide to assign "road" semantic as a near IFC semantic, that is, we temporarily set up 27 as ifcID value for "road" object.

```
INSERT INTO ifcclass(ifcid, name) VALUES (27, 'road');
DO $$
BEGIN
    FOR idx in 1..118
    LOOP
        INSERT INTO voxelmpt(classID, ifcID, geom)
        VALUES (56, 27, ST_Collect(ARRAY(
            SELECT V.geom
            FROM voxelpt V
            JOIN road R ON ST_WITHIN(V.geom, R.geom)
            WHERE V.classid=56 AND R.id=idx)));
        DELETE FROM voxelmpt WHERE classID=56 AND ifcID=27 AND geom IS NULL;
    END LOOP;
END;
$$

```

Query returned successfully in 759 min 35 secs.   ***115 records***



5.3.4.2 Greenarea Surface

For “greenarea”, there are 120 polygons, we aim to group dtmbot voxels into such 120 polygons. Same as “road”, we decide to assign “greenarea” semantic as a near IFC semantic, that is, we temporarily set up 28 as ifcID value for “greenarea” object.

```
INSERT INTO ifcclass(ifcid, name) VALUES (28, 'greenarea');
DO $$
BEGIN
    FOR idx in 1..120
    LOOP
        INSERT INTO voxelmpt(classID, ifcID, geom)
        VALUES (56, 28, ST_Collect(ARRAY(
            SELECT V.geom
            FROM voxelpt V
            JOIN greenareas G ON ST_WITHIN(V.geom, G.geom)
            WHERE V.classid=56 AND G.gid=idx)));
        DELETE FROM voxelmpt WHERE classID=56 AND ifcID=28 AND geom IS NULL;
    END LOOP;
END;
$$

```

Query returned successfully in 89 min 37 secs. ***120 records***

5.3.4.3 Building Surface

For “building”, there are 121 polygons, we aim to group dtmbot voxels into such 121 polygons. Different from “road” and “greenarea”, we have already had multiple source building semantic information (e.g., point cloud, BIM). It is better to set up the corresponding ID. The challenge is we have no idea about point cloud building (such as name).

One option is to build a hierarchical data model for the whole voxel dataset. Following links are for reference:

- https://coderwall.com/p/whf3-a/hierarchical-data-in-postgres
- https://www.cybertec-postgresql.com/en/postgresql-speeding-up-recursive-queries-and-hierarchic-data/
- https://www.pinnsg.com/modeling-hierarchical-data-postgres/

Another option is we ignore semantic information and only consider how to store terrain slice according to the footprint of the building. Thus, giving all terrain belong to building surface a same ifcid = 29 (“building”).

```
INSERT INTO ifcclass(ifcid, name) VALUES (29, 'building');
DO $$
BEGIN
    FOR idx in 1..121
    LOOP
       INSERT INTO voxelmpt(classID, ifcID, geom)
        VALUES (56, 29, ST_Collect(ARRAY(
            SELECT V.geom
            FROM voxelpt V
            JOIN building B ON ST_WITHIN(V.geom, B.geom)
            WHERE V.classid=56 AND B.id=idx)));
        DELETE FROM voxelmpt WHERE classID=56 AND ifcID=29 AND geom IS NULL;
    END LOOP;
END;
$$
```

Query returned successfully in 437 min 47 secs. ***94 records***

5.3.4.4 Others

For other dtmbot voxels, we can directly ignore them due to no semantic information.

5.4 Build Index

```sql
DROP INDEX IF EXISTS idx_voxelmpt CASCADE;
DROP INDEX IF EXISTS geom_voxelmpt CASCADE;
CREATE INDEX idx_voxelmpt ON voxelmpt(classID, ifcID);
CREATE INDEX geom_voxelmpt ON voxelmpt USING GIST (geom);
```

Query returned successfully in 38 secs 31 msec.

5.5 Summary

	id integer	classid integer	ifcid integer	st_astext text
1	57	44	19	MULTIPOINT Z (336098.5 6245700.4 28.7,336098.6 62457...
2	60	19	12	
3	62	19	8	MULTIPOINT Z (336307.7 6245536.3 36.4,336307.7 62455...
4	63	19	13	
5	93	6	27	MULTIPOINT Z (336077.6 6245640.5 36.2,336077.6 62456...

Figure 5-1. Sample of “voxelmpt” table.

6. Conversion to PCPATCH

6.1 Create table

```
CREATE EXTENSION IF NOT EXISTS pointcloud;
CREATE EXTENSION IF NOT EXISTS pointcloud_postgis;
DELETE FROM pointcloud_formats;
INSERT INTO pointcloud_formats (pcid, srid, schema) VALUES (1, 28356,
'<?xml version="1.0" encoding="UTF-8"?>
<pc:PointCloudSchema xmlns:pc="http://pointcloud.org/schemas/PC/1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <pc:dimension>
    <pc:position>1</pc:position>
    <pc:size>4</pc:size>
    <pc:description>X coordinate as a float.</pc:description>
    <pc:name>X</pc:name>
    <pc:interpretation>double</pc:interpretation>
  </pc:dimension>
  <pc:dimension>
    <pc:position>2</pc:position>
    <pc:size>4</pc:size>
    <pc:description>Y coordinate as a float.</pc:description>
    <pc:name>Y</pc:name>
    <pc:interpretation>double</pc:interpretation>
  </pc:dimension>
  <pc:dimension>
    <pc:position>3</pc:position>
    <pc:size>4</pc:size>
    <pc:description>Z coordinate as a float.</pc:description>
    <pc:name>Z</pc:name>
    <pc:interpretation>double</pc:interpretation>
  </pc:dimension>
  <pc:metadata>
    <Metadata name="compression">dimensional</Metadata>
  </pc:metadata>
</pc:PointCloudSchema>');
DROP TABLE IF EXISTS voxelpatch CASCADE;
CREATE TABLE voxelpatch (
  id SERIAL PRIMARY KEY,
  classID INTEGER,
  ifcID INTEGER,
  pa PCPATCH(1)
);

```

Query returned successfully in 637 msec.

6.2 Partition Principles

Same as MULTIPOINT partition:

- Rule-1: For general building objects without IFC features, each building is one Multipoint.
- Rule-2: For building objects with IFC objects, we regard each IFC object as one Multipoint.
- Rule-3: For tree and dtmbot, we combine GIS dataset to decide the patch size.

6.3 Data Generation

6.3.1 Building without IFC

```
DO $$
```

```
BEGIN
    FOR idx in 1..54
    LOOP
        IF idx = 26 OR idx = 46 THEN
            raise notice 'The buidling % could not be found', idx;
        ELSE
            INSERT INTO voxelpatch(classID, pa)
            VALUES (idx, PC_Patch(ARRAY(SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM voxel WHERE classID=idx AND ifcID IS NULL)));
        END IF;
    END LOOP;
END;
$$
```

Query returned successfully in 353 min 50 secs.

6.3.2 Building with IFC

```
DO $$
DECLARE
    f record;
BEGIN
    FOR f in SELECT DISTINCT classID, ifcID
        FROM voxel WHERE ifcID IS NOT NULL
    LOOP
        INSERT INTO voxelpatch(classID, ifcID, pa)
        VALUES (f.classID, f.ifcID, PC_Patch(ARRAY(SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM voxel WHERE ifcID IS NOT NULL AND classID=f.classID  AND ifcID=f.ifcID)));
    END LOOP;
END;
$$
```

Query returned successfully in 522 min 2 secs.

6.3.3 Tree

For “tree” and “dtmbot” objects, we tried to follow the way we did in previous section, but it failed. Since it is impossible to convert MULTIPOINT to PCPATCH directly, we consider breaking up MULTIPOINT geometry and generate a set of POINT, which can further casted into PCPOINT object through “pcpoint::geometry” .

```
DO $$
DECLARE
    f record;
BEGIN
    FOR f in SELECT geom
        FROM voxelmpt WHERE classID=55 AND ifcID=30 AND geom IS NOT NULL
    LOOP
        INSERT INTO voxelpatch(classID, ifcID, pa)
        VALUES (55, 30, PC_Patch(ARRAY(
            SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM (
                SELECT ST_X((ST_DumpPoints(f.geom)).geom) AS x,
                ST_Y((ST_DumpPoints(f.geom)).geom) AS y,
                ST_Z((ST_DumpPoints(f.geom)).geom) AS z
                ) AS g
            )));
    END LOOP;
END;
$$

```

Query returned successfully in 26 secs 737 msec. ***793 records***

6.3.4 Dtmbot

7.3.4.1 Road Surface

```
DO $$
DECLARE
    f record;
BEGIN
    FOR f in SELECT geom
        FROM voxelmpt WHERE classID=56 AND ifcID=27 AND geom IS NOT NULL
    LOOP
        INSERT INTO voxelpatch(classID, ifcID, pa)
        VALUES (56, 27, PC_Patch(ARRAY(
            SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM (
                SELECT ST_X((ST_DumpPoints(f.geom)).geom) AS x,
                ST_Y((ST_DumpPoints(f.geom)).geom) AS y,
                ST_Z((ST_DumpPoints(f.geom)).geom) AS z
                ) AS g
            )));
    END LOOP;
END;
$$

```

Query returned successfully in 4 min 39 secs. ***115 records***

6.3.4.2 Greenarea Surface

```
DO $$
DECLARE
    f record;
BEGIN
    FOR f in SELECT geom
        FROM voxelmpt WHERE classID=56 AND ifcID=28 AND geom IS NOT NULL
    LOOP
        INSERT INTO voxelpatch(classID, ifcID, pa)
        VALUES (56, 28, PC_Patch(ARRAY(
            SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM (
                SELECT ST_X((ST_DumpPoints(f.geom)).geom) AS x,
                ST_Y((ST_DumpPoints(f.geom)).geom) AS y,
                ST_Z((ST_DumpPoints(f.geom)).geom) AS z
                ) AS g
            )));
    END LOOP;
END;
$$
```

Query returned successfully in 5 min 5 secs. ***120 records***

6.3.4.3 Building Surface

```
DO $$
DECLARE
    f record;
BEGIN
    FOR f in SELECT geom
        FROM voxelmpt WHERE classID=56 AND ifcID=29 AND geom IS NOT NULL
    LOOP
        INSERT INTO voxelpatch(classID, ifcID, pa)
        VALUES (56, 29, PC_Patch(ARRAY(
            SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM (
```

```
                    SELECT ST_X((ST_DumpPoints(f.geom)).geom) AS x,
                    ST_Y((ST_DumpPoints(f.geom)).geom) AS y,
                    ST_Z((ST_DumpPoints(f.geom)).geom) AS z
                    ) AS g
                ));
    END LOOP;
END;
$$

```

Query returned successfully in 11 min 33 secs. ***94 records***

6.4 Build Index

```
DROP INDEX IF EXISTS idx_voxelpatch CASCADE;
DROP INDEX IF EXISTS geom_voxelpatch CASCADE;
CREATE INDEX idx_voxelpatch ON voxelpatch(classID, ifcID);
CREATE INDEX geom_voxelpatch ON voxelpatch USING GIST(PC_EnvelopeGeometry(pa));
```

Query returned successfully in 735 msec.

6.5 Summary

	id integer	classid integer	ifcid integer	pa pcpatch
1	1	1	[null]	01010000000100000056AD6900031941010078DAECC5CB55D5501400D...
2	2	2	[null]	0101000000010000000193C2200038D6A000078DAECC5C791D45014004...
3	3	3	[null]	
4	4	4	[null]	
5	5	5	[null]	

Figure 6-1. Sample of “voxelpatch” table.

7. Align the IFC with the Extrusion Buildings (NOT READY)

The alignment can be according to the footprint from the 2D map. In general, the orientation of the footprint should be correct, but the shape and size might differ.

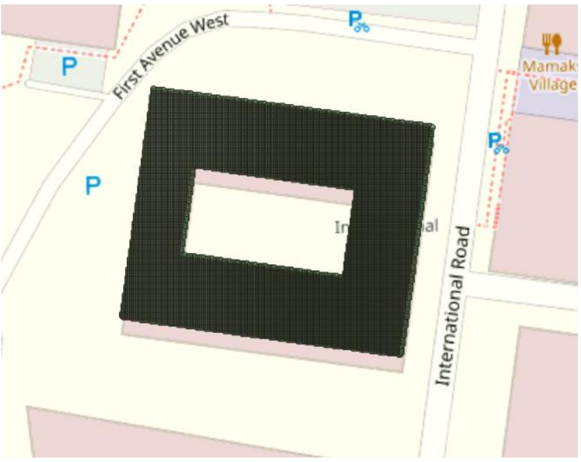
What has been given as one footprint on the 2D map might have been two buildings and vice versa… two footprints have been drawn for one building. This might happen with buildings like Red Centre (check OSM map: https://www.openstreetmap.org/#map=19/-33.91822/151.22950). What is on the maps in not wrong: the footprint represents the outline of the building that ‘steps’ on the terrain. If there is a tunnel or a sky bridge or an overhanging part … they are not represented on the map. Therefore when we have extruded building (as all these 50+ buildings with flat roofs) , some discrepancies in the shape may occur.

I suggest you measure the offset in QGIS with respect to the footprint of the map. What I will do I would print all extruded buildings in groups of 5-10 to see how well they fit with the foot prints. They are obtained from the footprints, so they have to be the same. If they are not the same I will created a MMBB of three buildings far from each other, e.g. in the corners of the area we are working on. Then I will get three points with x, y coordinates and compare with the same x,y coordinates from the 2D map. From these points I will compute affine transformation and determine shift and rotation. This shoft and rotation I will apply to the whole extruded buildings.

Then I will print each IFC building and compare with the footprint of the 2D map. This will be not straightforward because the footprint will not fit that well with the IFC building (print). So you have to visually decide which point to use for the shift and translation (no scaling should be applied).

7.1 Point cloud

Check “bld1” and “bld2” in QGIS:



“bld3” cannot be visualized in QGIS because of memory allocation problem.

Check “bld4” and “bld5” in QGIS:

Offset: **336000, 6245250, 20**

X*0.1+offset

First, we consider modifying offset only and keep each (x,y) unchanged.

Case-I: Only move y

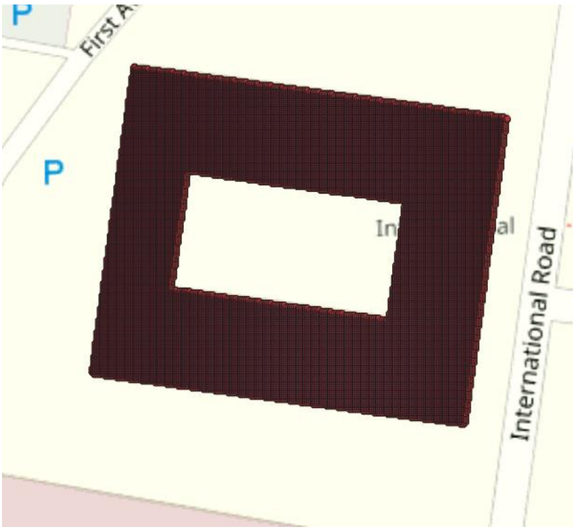
- ~~y-5~~
- y-3, 100*0.1+**6245250 = y coordinate=6245251-3=6245247**
- ~~y-2~~

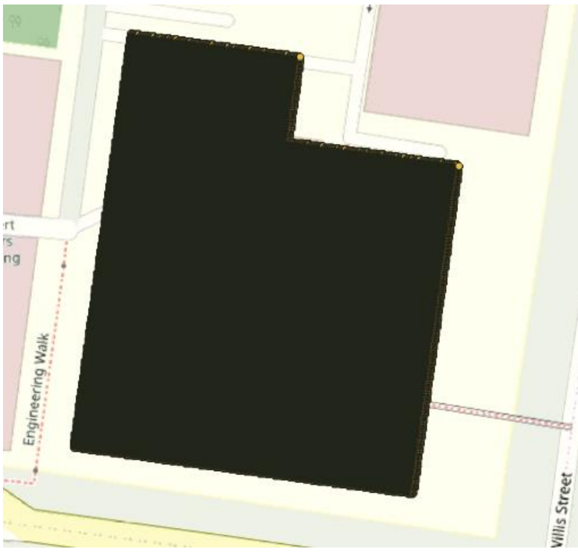
~~Case-II: Only move x~~

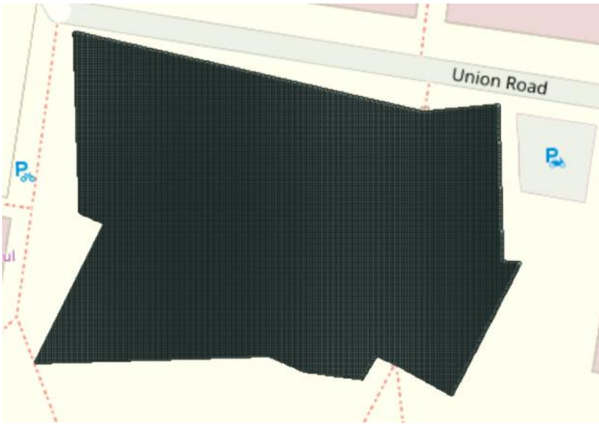
~~Case-III: Move x and y~~

==> **336000, 6245247, 20**

After “y” correction:



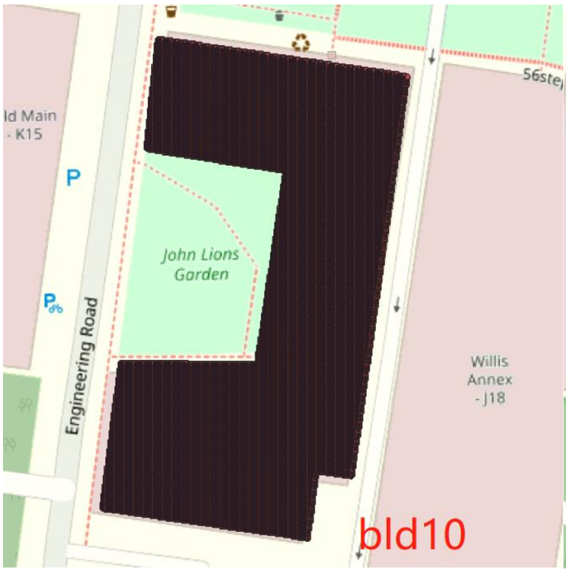


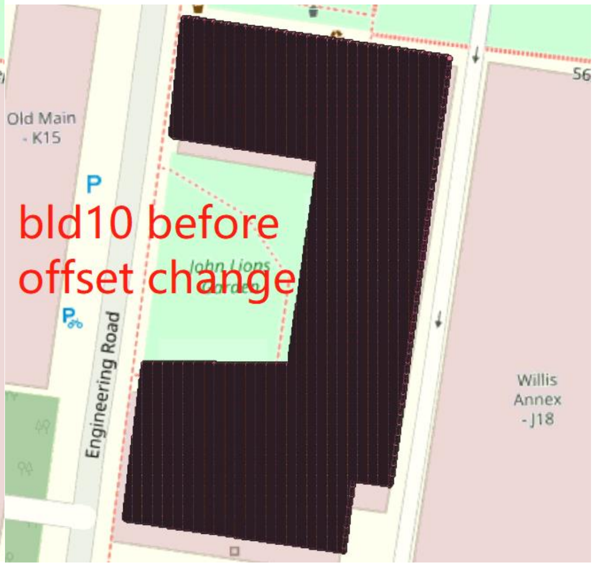


New problem comes up: Part of voxel in one object may be redundant. For instance, in “bld4”, some voxels are on the road. So the next step is to select all the buildings of this kind of problem.

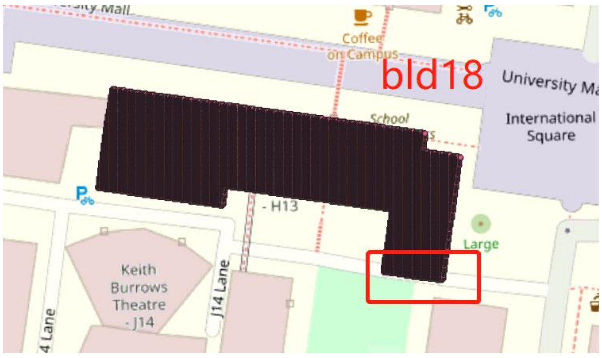
Buildings with redundant voxels:

- bld4
- bld10 contains two buildings (K17 and J17)

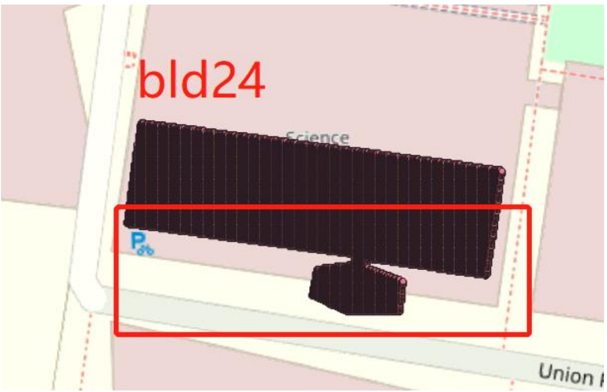




When visualizing “bld10”, new problem has arisen. That is, only changing offset is not enough, we need to change (x,y) for each voxel in bld10. Besides, some of them are redundant, some of them are missing. We can drop the redundant part, but how to fill in the missing pieces. And do we need to separate this into two buildings or keep that?

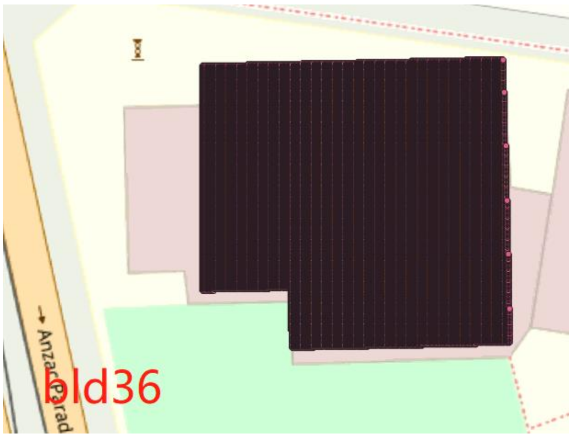

bld18
Coffee on Campus
University Ma
International Square
School
- H13
Large
Keith Burrows Theatre - J14
J14 Lane

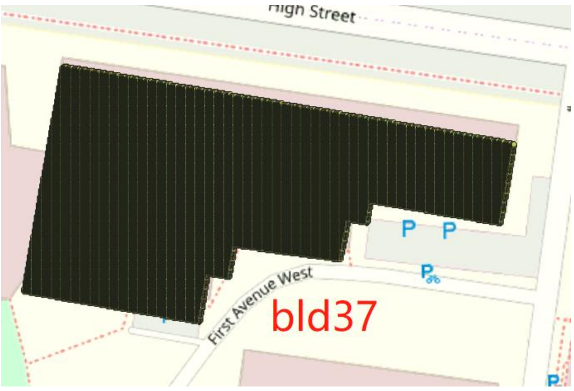

University Mall
bld19
Newton - J12
Keith


bld24
Science
Union

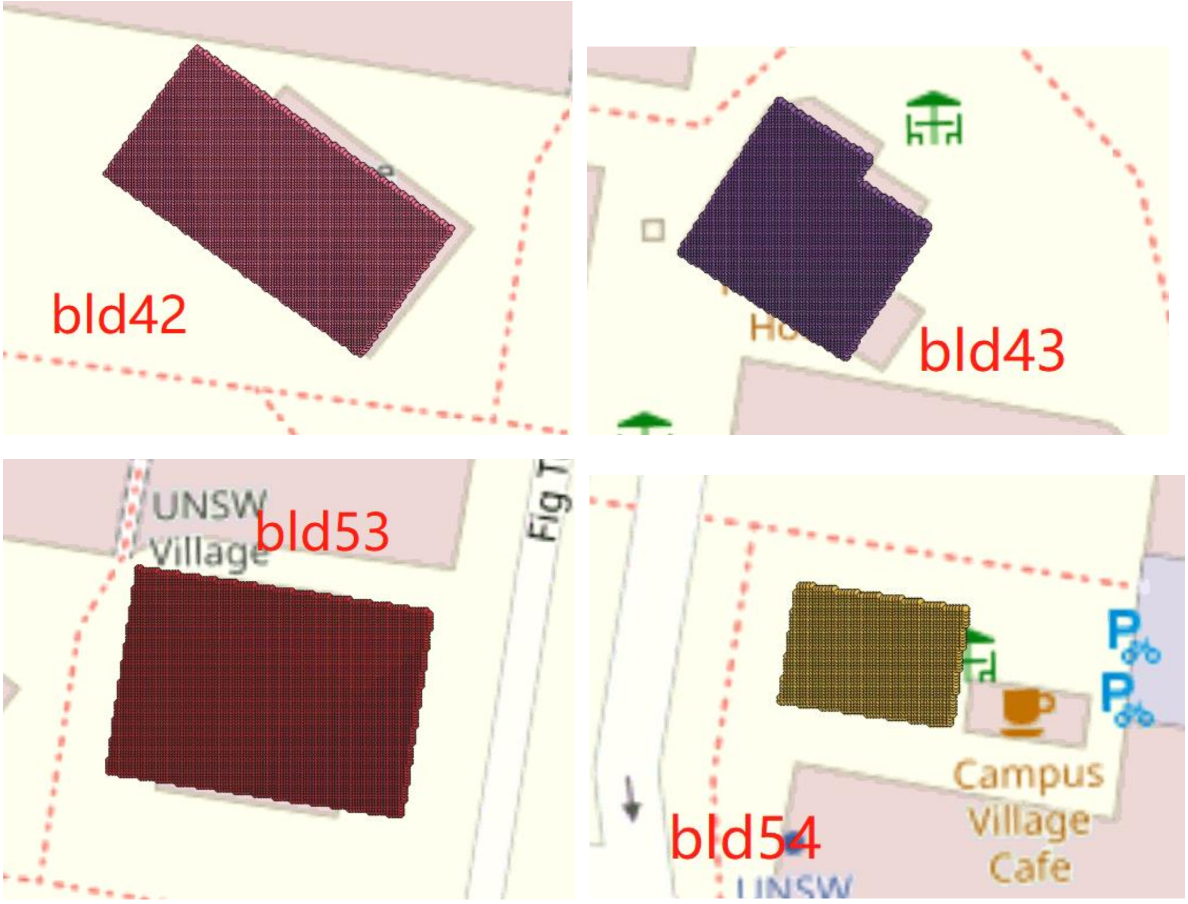

Randwick, UNSW Gate 2,
bld27
Gate Two
Gate 2
International House
First Avenue East
UNSW Campus Security


bld35
Science and Engineering Building (E8)


bld36

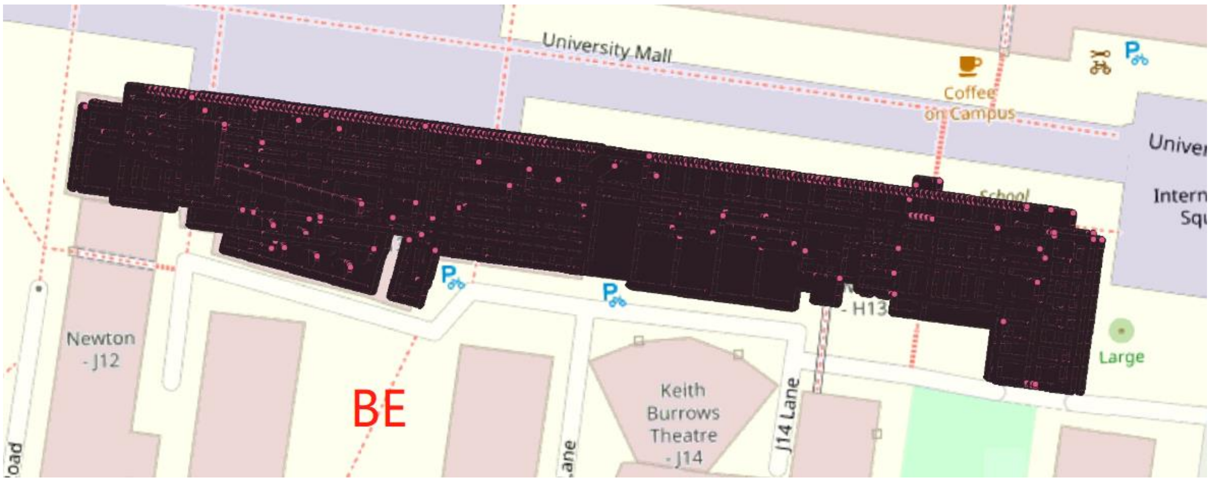

High Street
First Avenue West
bld37


Avenue East
Campus Security
bld40

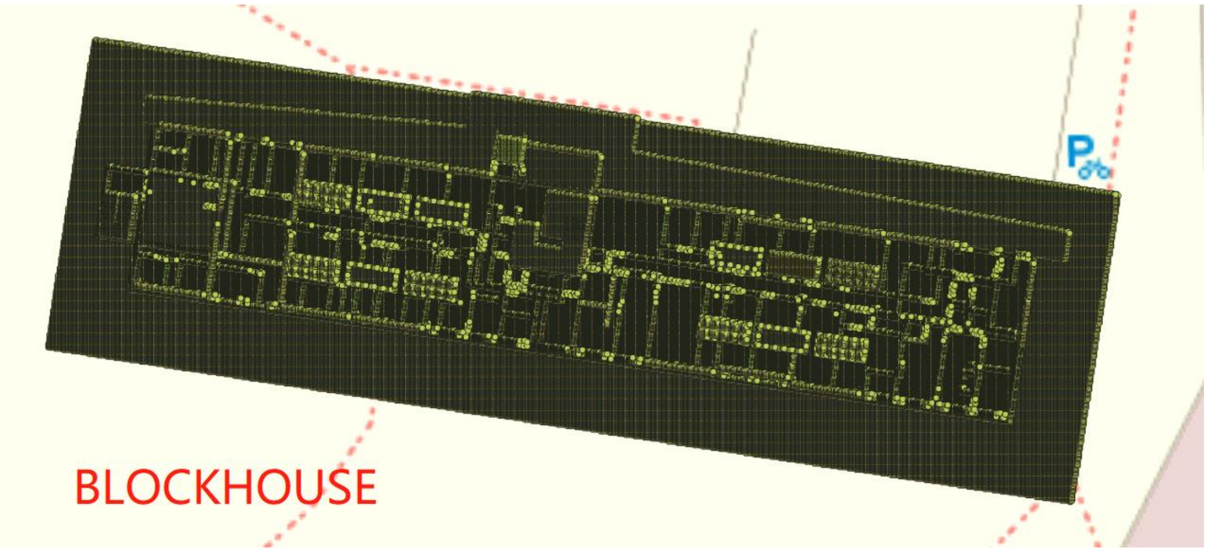


7.2 IFC

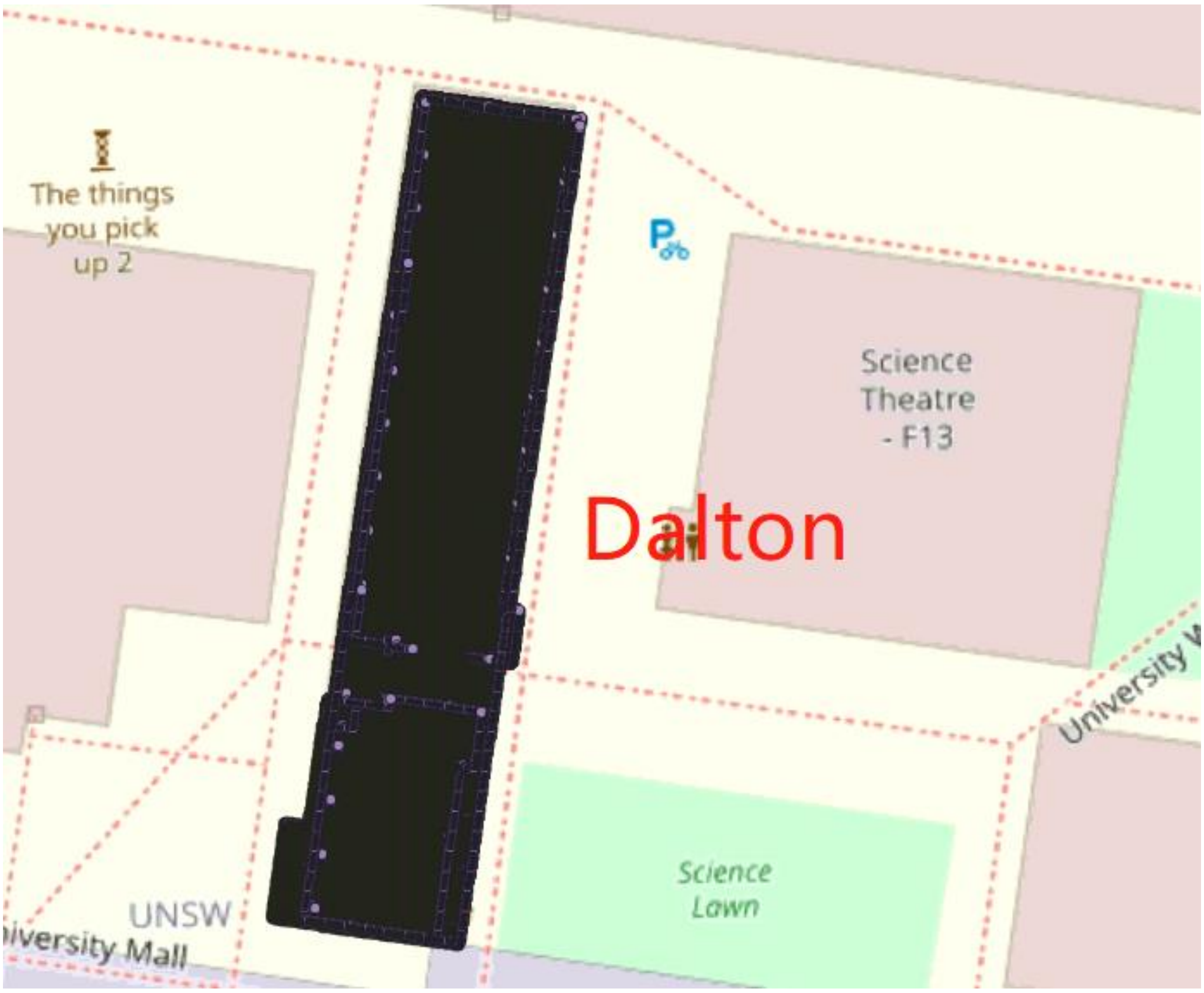
select ST_Translate(ST_Scale(geom, 0.1,0.1,0.1), 336300, 6245507, 25) as geom from voxelmpt where classid=19 and ifcid is not null;



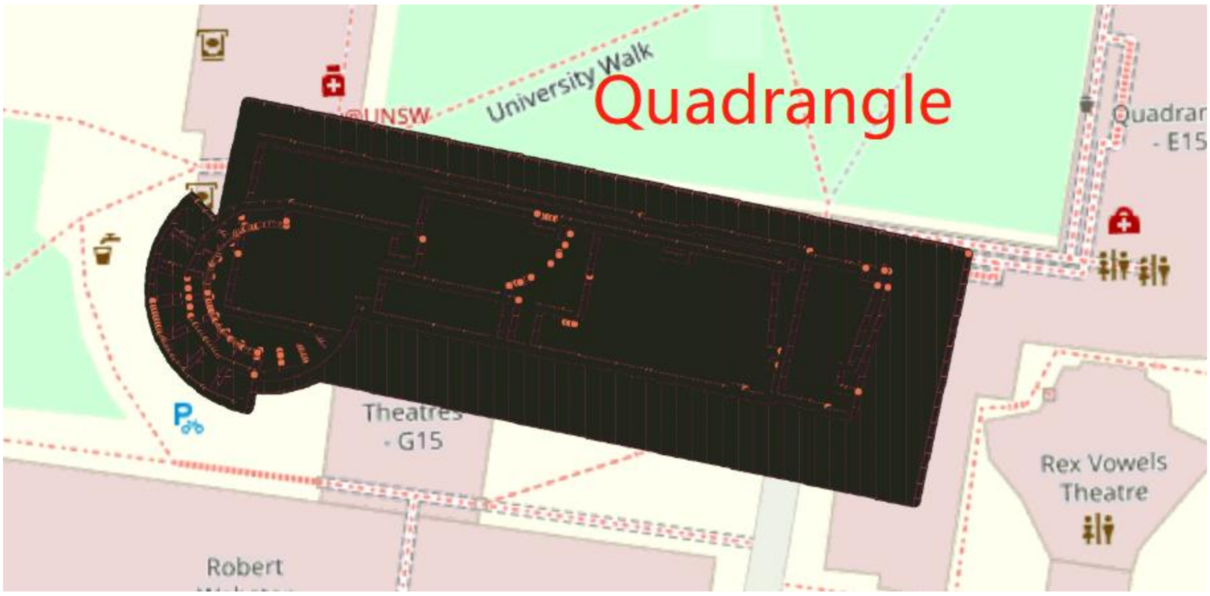
select ST_Translate(ST_Scale(geom, 0.1,0.1,0.1), 336042, 6245613, 27) as geom from voxelmpt where classid=6 and ifcid is not null;



select ST_Translate(ST_Scale(geom, 0.1,0.1,0.1), 336305, 6245569, 29) as geom from voxelmpt where classid=9 and ifcid is not null;

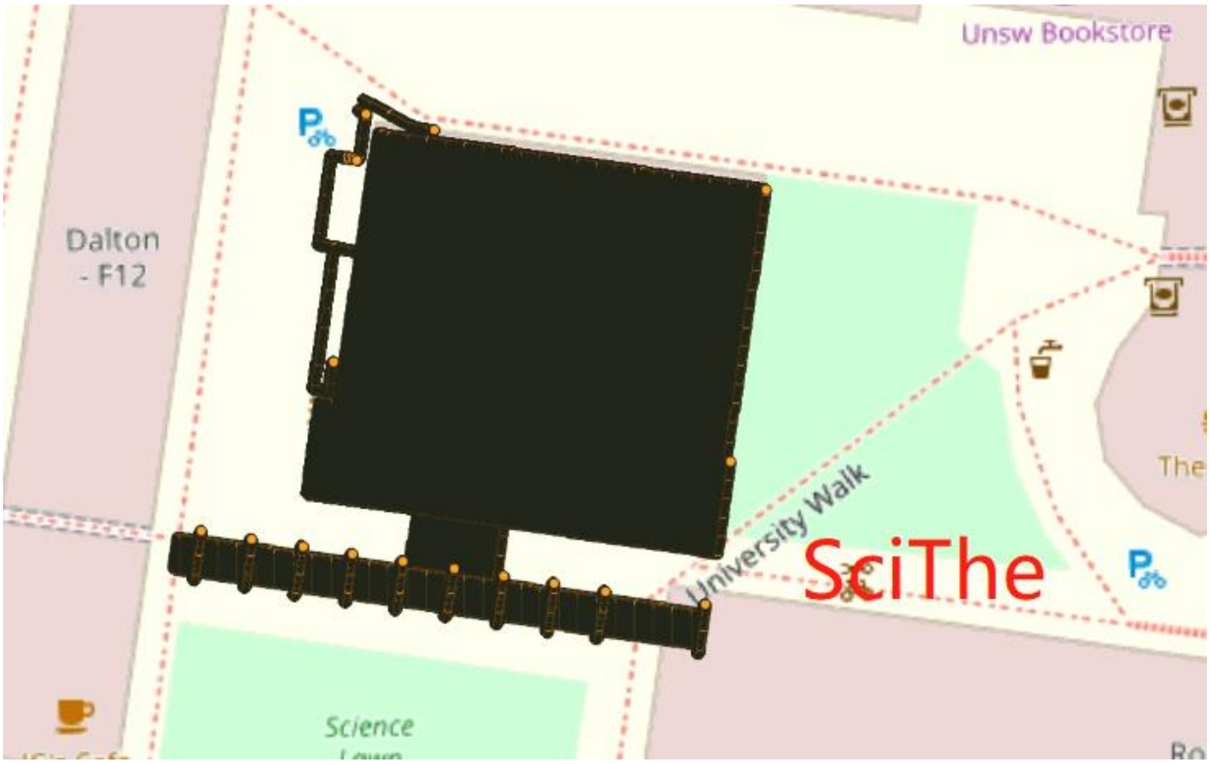


select ST_Translate(ST_Scale(geom, 0.1,0.1,0.1), 336409, 6245580, 31) as geom from voxelmpt where classid=13 and ifcid is not null;



select ST_Translate(ST_Scale(geom, 0.1,0.1,0.1), 336047, 6245651, 25) as geom from voxelmpt where classid=44 and ifcid is not null;

select ST_Translate(ST_Scale(geom, 0.1,0.1,0.1), 336325, 6245582, 28) as geom from voxelmpt where classid=8 and ifcid is not null;


Unsw Bookstore
Dalton
- F12
University Walk
SciThe
Science

8. Sample Queries and QGIS Visualization (ON-GOING)

A high performance HP Z4 G4 Workstation with 64GB RAM and a 2.90GHZ Intel(R) Xeon(R) W-2102 CPU, running on a64bit Windows 10 and equipped with an internal 512GB SSD and 1TB HDD for disk storage was used for all the tests. As a RDBMS, we used PostgreSQL 11.2, together with pgAdmin (version 4.3) for running the queries. To evaluate the performance of a PostgreSQL query, the "Query Tool" in pgAdmin was used to run each query. However, so as to obtain an accurate time, for each query we issued the command "*EXPALIN ANALYZE stmt*", where *stmt* is the query that was run. The query tie is the sum obtained from the planning and execution times and it is these values which are reported in the result tables.

8.1 Sample Query and Visualization

As for the queries:

A. Queries that return subset of voxels

1. Select all voxels in a given MINMAX space (radius). Several options for MINMAX: rectangular space, very long but narrow space, different size space and at different locations (at the beginning, middle and end of the entire voxel space of the campus)
2. Select all voxels in a given MINMAX/radius from a voxel (x,y,z)
3. Select all voxels that have a specific attribute
4. Combination of 3 and 4: e.g. all trees in a given MINMAX/radius around a given location (x,y,z)
5. Select all the information for a specific voxel (x,y,x)
6. Select MAX or MIN on the basis of value (for example ' the highest point of DTM of a building X)
7. Select a specific object (e.g. Building X, DTM, Vegetation)
8. Select all objects (e.g. buildings) with a given attribute (e.g. have indoor)

B. Some more complicated selections

1. Give all neighbours of a voxel (x,y,z)
2. Give the outer shell of an object
3. Give the buffer of an object

C. Queries that return a result of computation

1. Give the volume of an object (objects)
2. Give the area of the footprint of an object
3. Give the distance between voxel 1 and voxel 2
4. …

Q1: Load all voxels that are in a given rectangle $100<x<200$ and $100<y<200$ and $100<z<200$.

References

[1] Introduction to UML, https://github.com/ISO-TC211/UML-Best-Practices/wiki/Introduction-to-UML

[2] Gröger, G., Kolbe, T.H., Nagel, C. and Häfele, K.H., 2012. OGC city geography markup language (CityGML) encoding standard.

[3] Pointcloud, https://github.com/pgpointcloud/pointcloud

[4] Li, W., Zlatanova, S., Diakite, A.A., Aleksandrov, M. and Yan, J., 2020. Towards Integrating Heterogeneous Data: A Spatial DBMS Solution from a CRC-LCL Project in Australia. ISPRS International Journal of Geo-Information, 9(2), p.63.

[5]