Report

1. UNSW Lower Campus

1.1 Raw 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).
- For the time being, we have already transferred INTEGER coordinates into NUMERIC coordinates (i.e., real geographical coordinates in CRS EPSG 28356). Therefore, users can directly access voxels without scaling and translation.

1.2 Database Schema

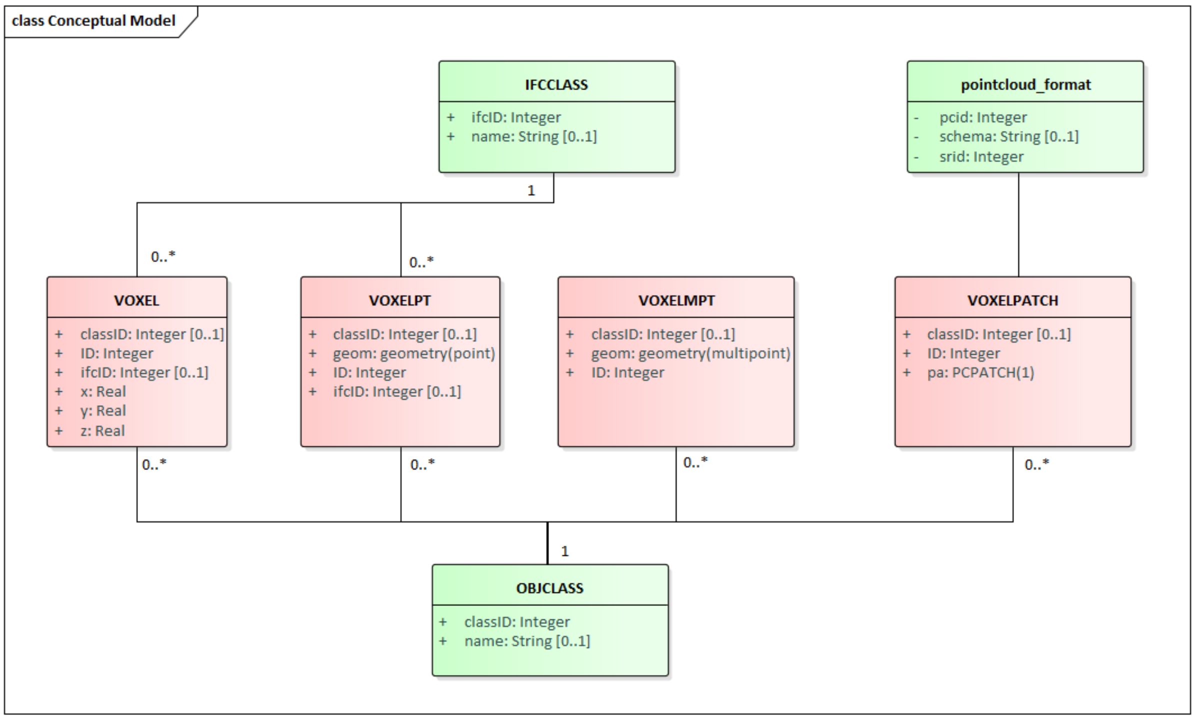


Figure 1-1. Conceptual Model of VoxelDB

1.3 Four Data Layouts in voxeldb

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

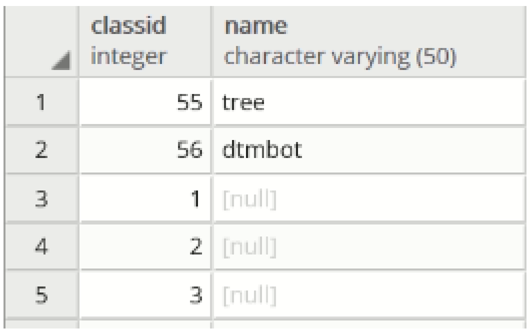
1.4 Two Semantic Tables in voxeldb

1.4.1 OBJCLASS

The adjustment is minimized by assigning a classID to the “tree” object and the “dtmbot”object, leaving the classID of the 52 “building” objects unchanged

```
DO $$
BEGIN
    FOR idx in 1..54
    LOOP
        IF idx = 26 OR idx = 46 THEN
            raise notice 'The buidling % could not be found', idx;
        ELSE
            INSERT INTO objclass(classID)
            VALUES (idx);
        END IF;
    END LOOP;
END;
$$

```

Query returned successfully in 119 msec.

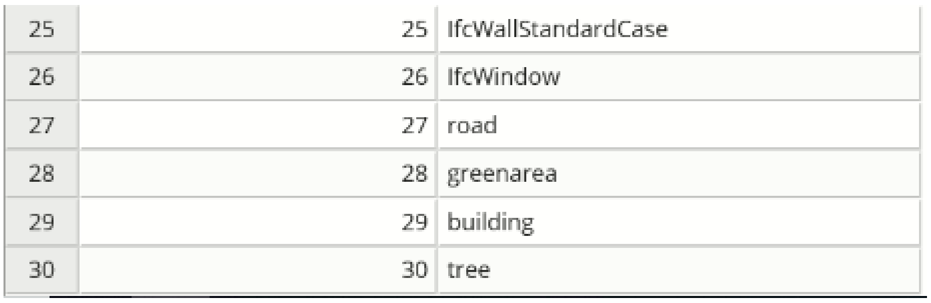


	classid integer	name character varying (50)
1	55	tree
2	56	dtmbot
3	1	[null]
4	2	[null]
5	3	[null]

Figure 1-2. Sample of table "objclass"

1.4.2 IFCCLASS

In order to adapt to the following division on "tree" and "dtmbot", we add non-IFC semantic labels for three different kinds of "dtmbot" and one non-IFC label for "tree".



25	25	IfcWallStandardCase
26	26	IfcWindow
27	27	road
28	28	greenarea
29	29	building
30	30	tree

Figure 1-3. Sample of table "ifcclass"

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

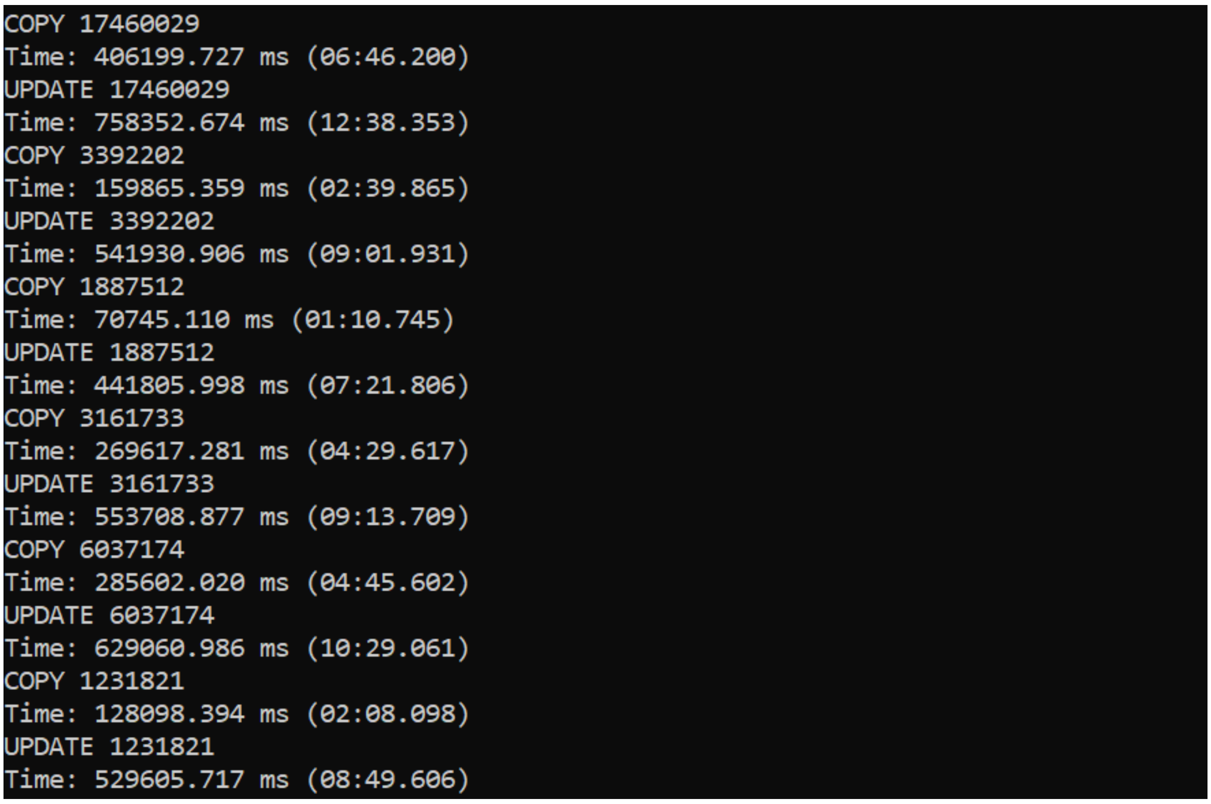
Data	#Voxels
bld1-54	241,613,693 in total and 4,646,418 per building
tree	59,640,000
dtmbot	641,624,355
be	17,460,029
blockhouse	3,392,202
dalton	1,887,512
quadrangle	3,161,733
roundhouse	6,037,174
scithe	1,231,821
Ifcid is not null	**33,170,471**
Total	**976,048,519**

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.

```sql
\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;
```



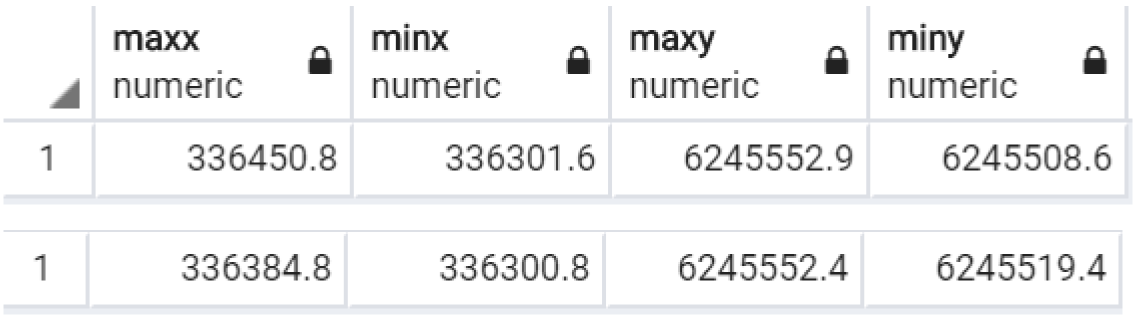
```
COPY 17460029
Time: 406199.727 ms (06:46.200)
UPDATE 17460029
Time: 758352.674 ms (12:38.353)
COPY 3392202
Time: 159865.359 ms (02:39.865)
UPDATE 3392202
Time: 541930.906 ms (09:01.931)
COPY 1887512
Time: 70745.110 ms (01:10.745)
UPDATE 1887512
Time: 441805.998 ms (07:21.806)
COPY 3161733
Time: 269617.281 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 4. 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 5. (x,y) range for BE in EPSG:28356 CRS

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

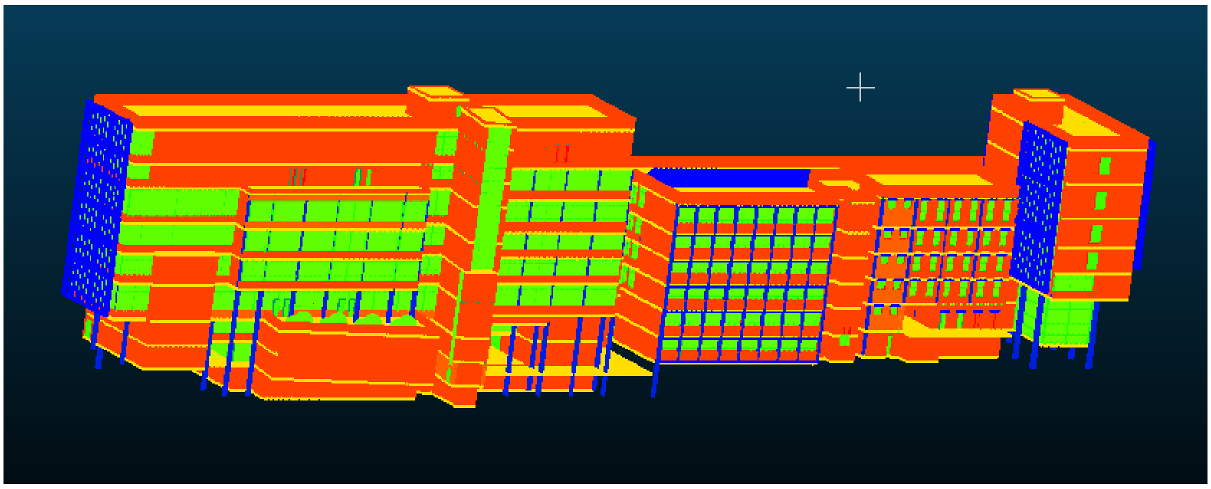


Figure 6. BE building in CloudCompare

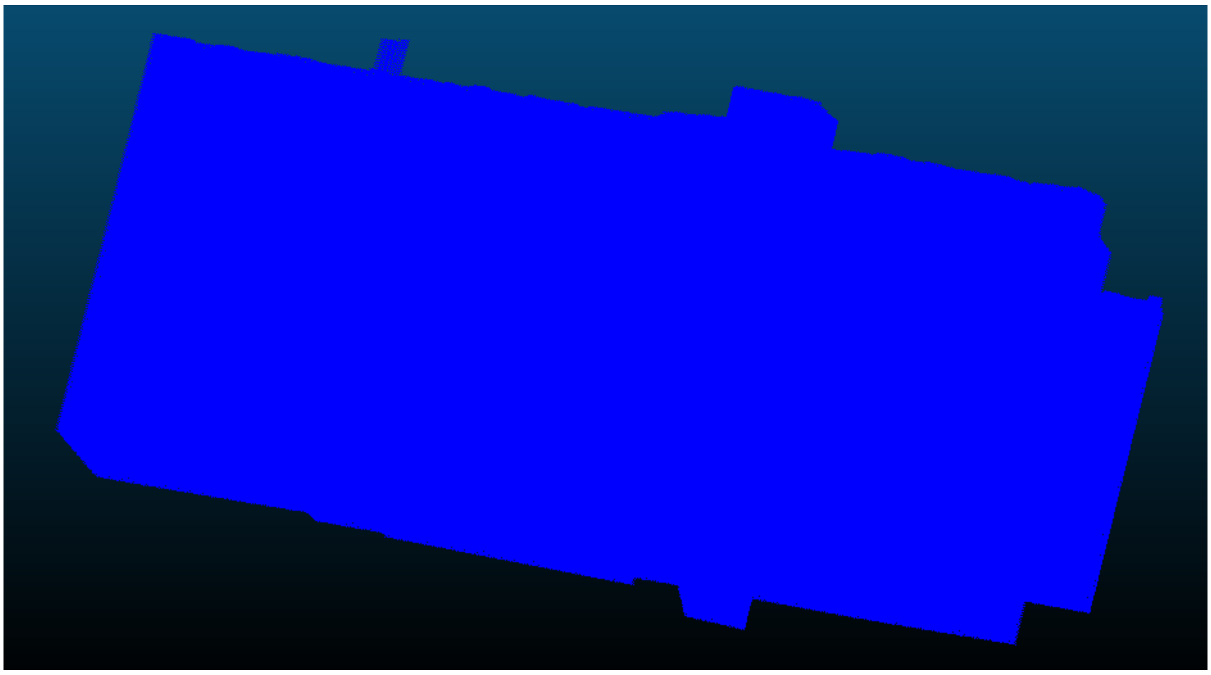
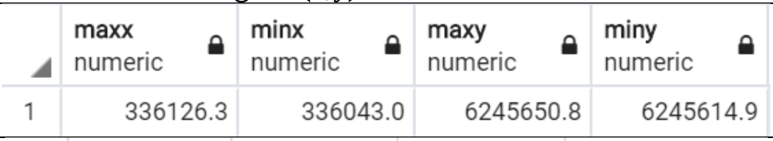
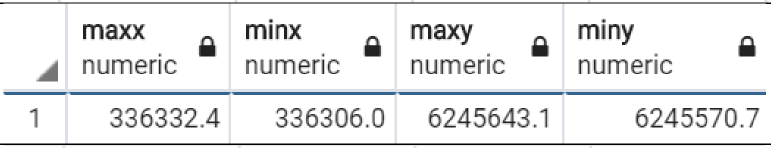
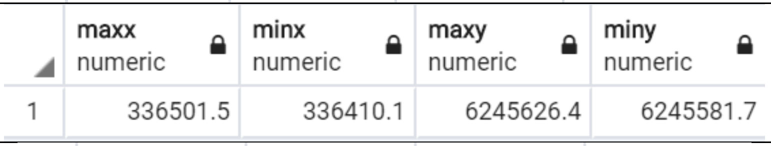
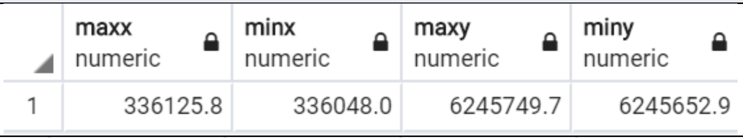
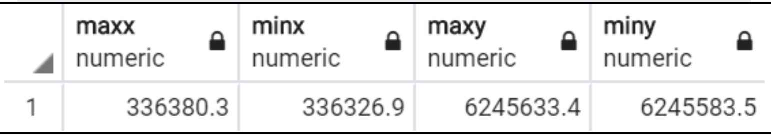


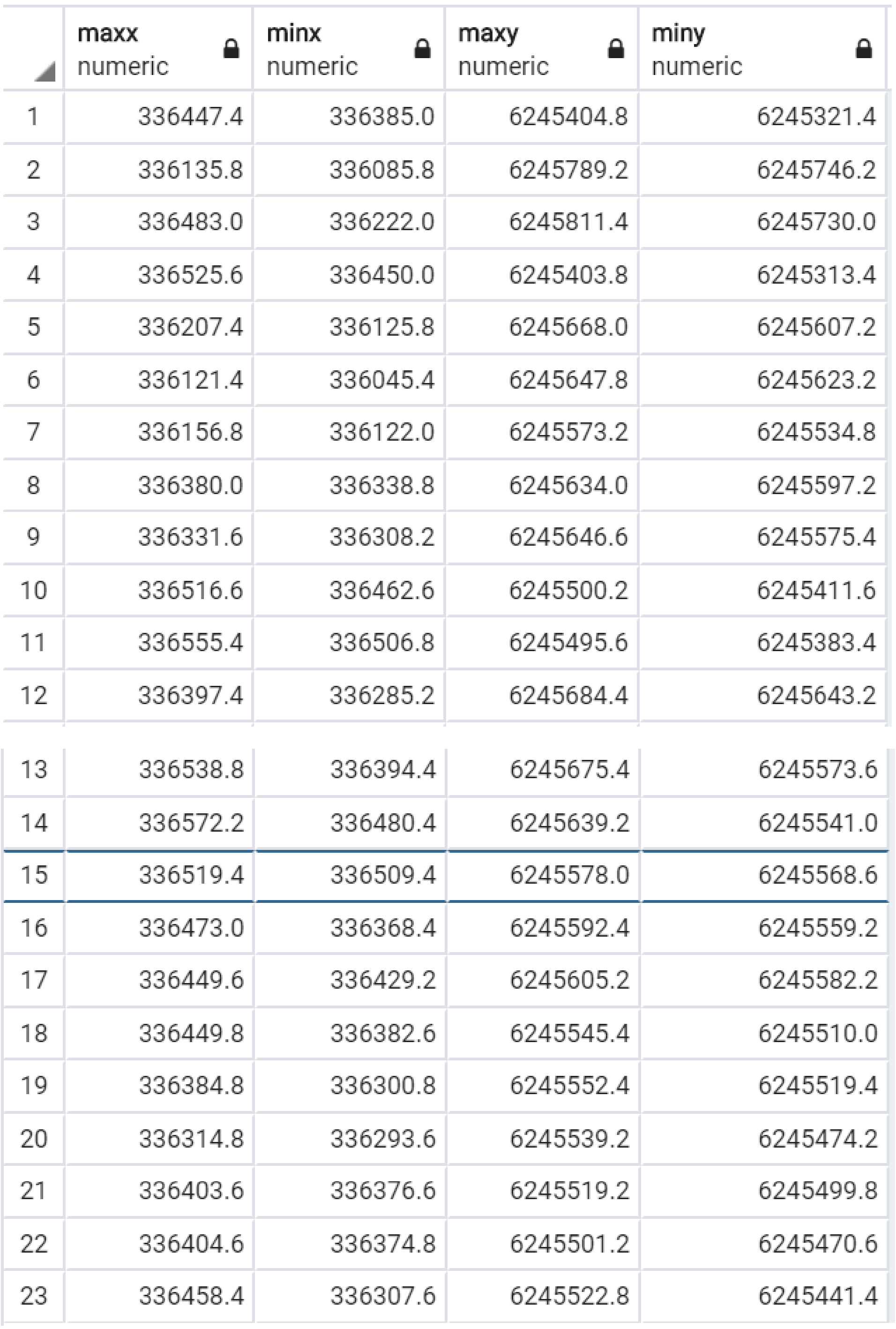
Figure 7. bld19 in CloudCompare

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.

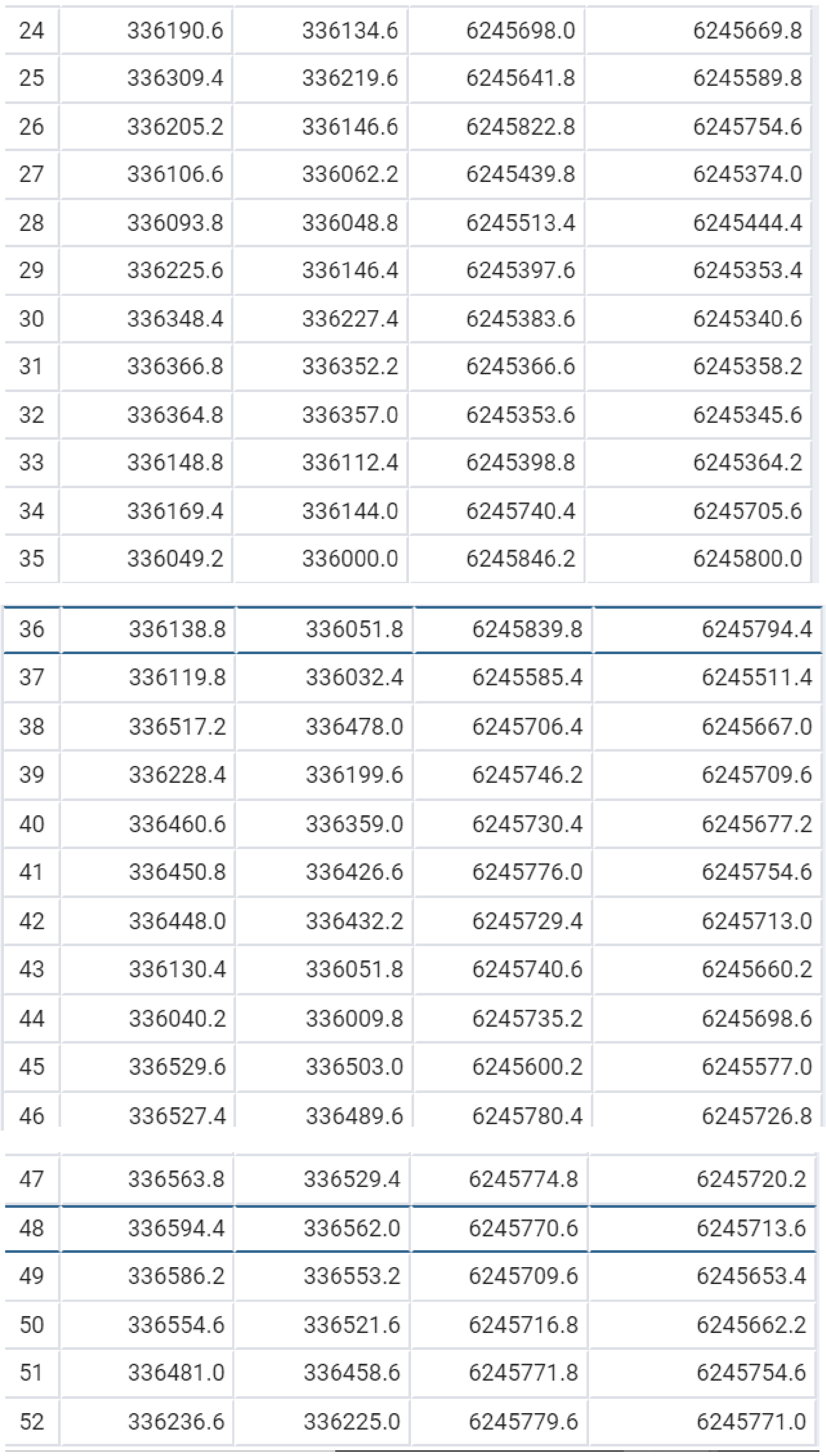
```sql
SELECT MAX(x)*0.1+336042 AS maxx, MIN(x)*0.1+336042 AS minx, MAX(y)*0.1+6245613 AS
maxy, MIN(y)*0.1+6245613 AS miny
FROM voxel
WHERE classid=58;

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 8. (x,y) range for all 52 building

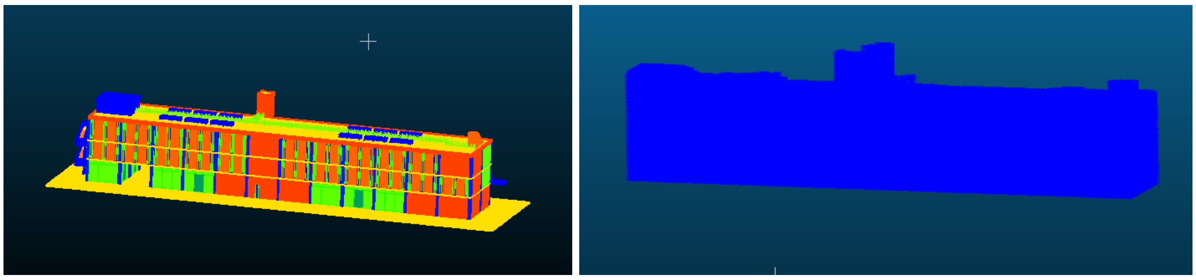


Figure 9. Blockhouse

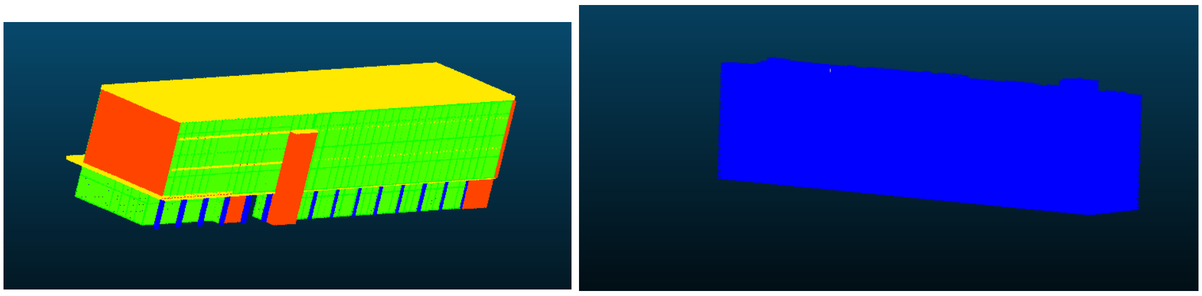


Figure 10. Dalton

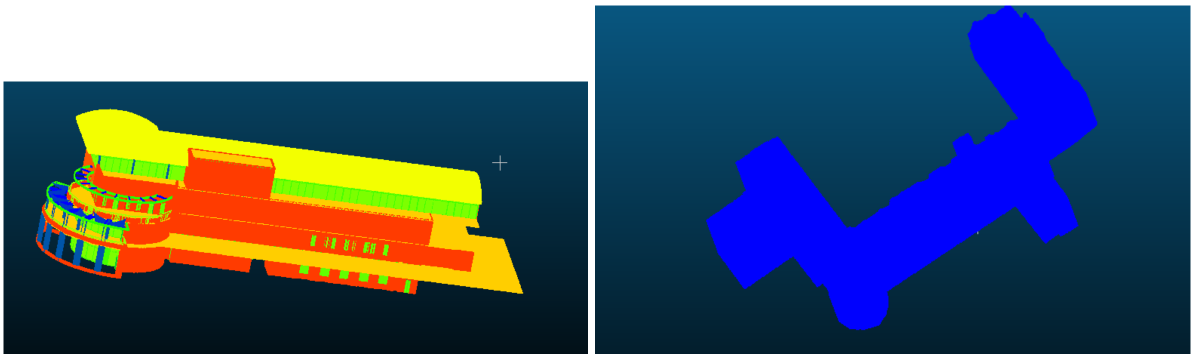


Figure 11. Quadrangle

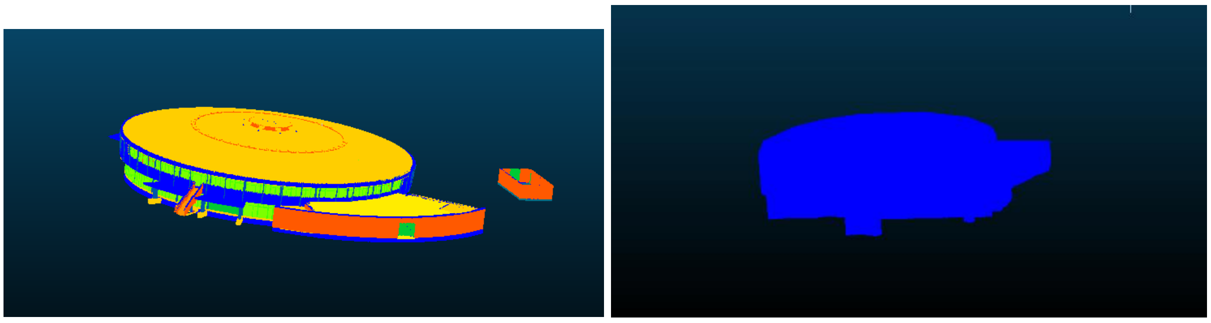


Figure 12. Roundhouse

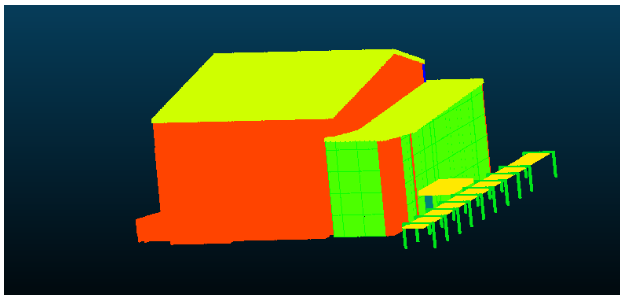
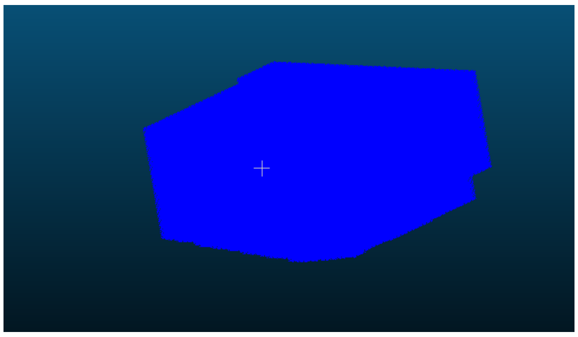



Figure 13. Science Theatre

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

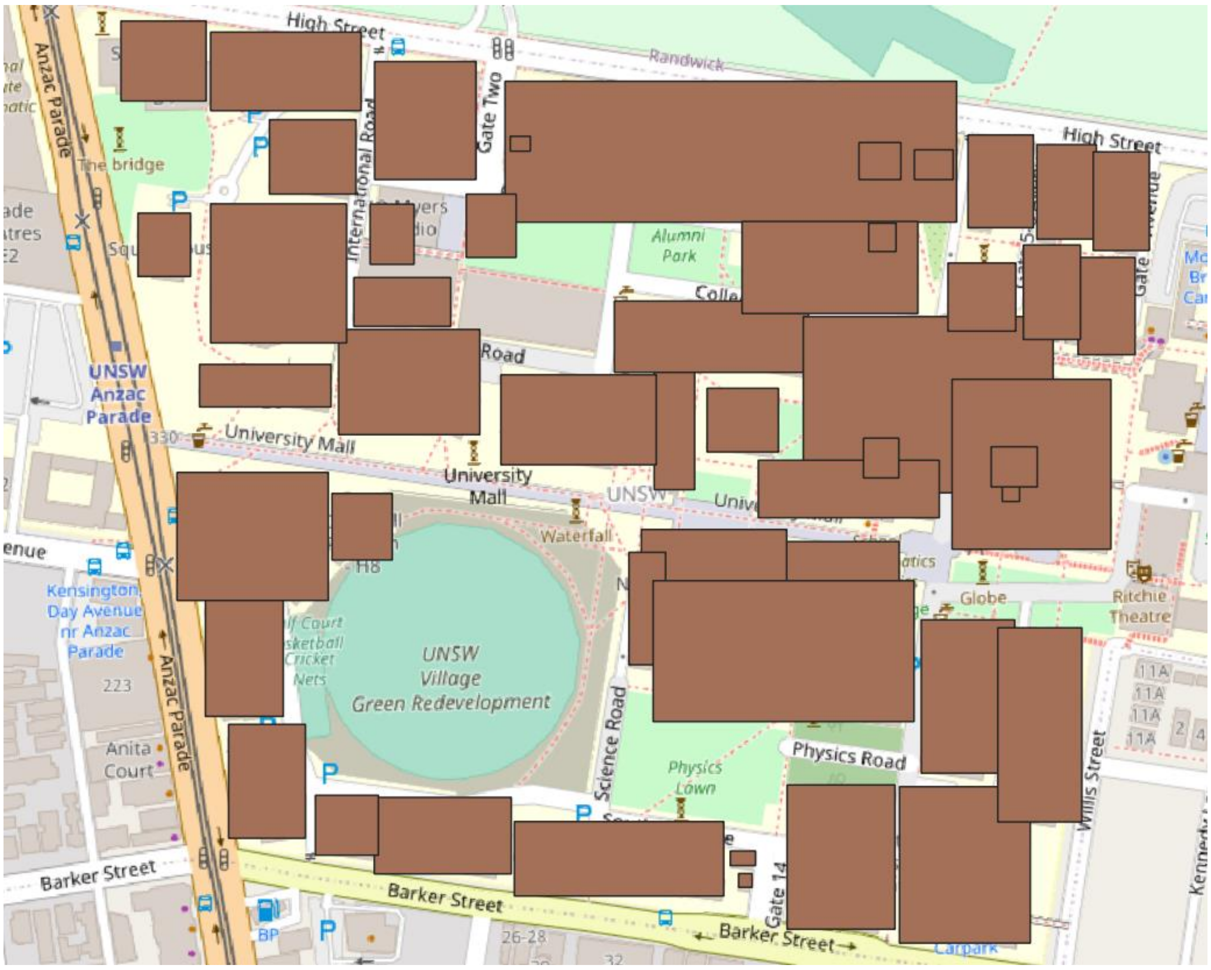
Update the 6 buildings:

```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;
```

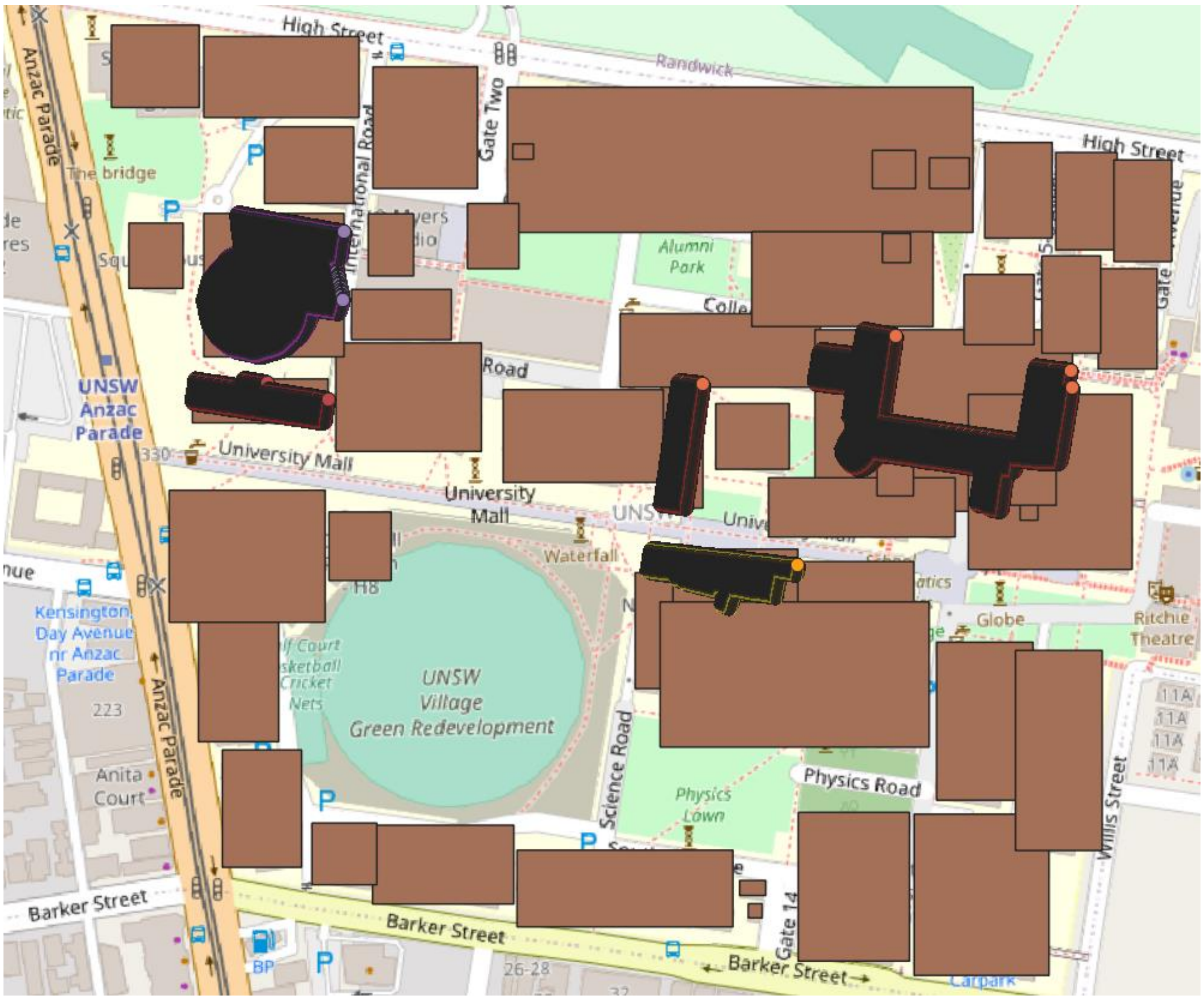
Time: 1h 12m 19s

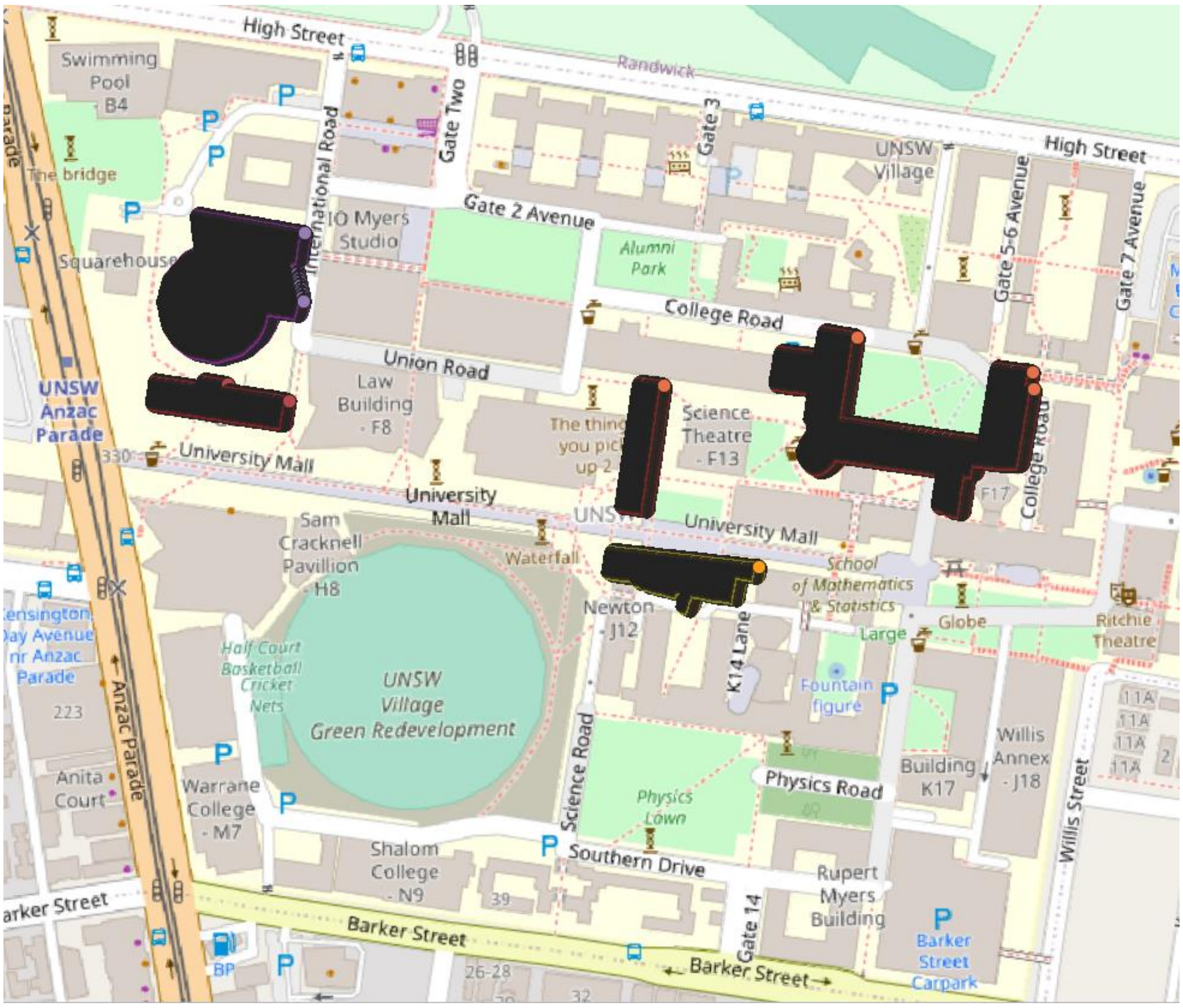
2.4 Assign Name for Each Building in Lower Campus

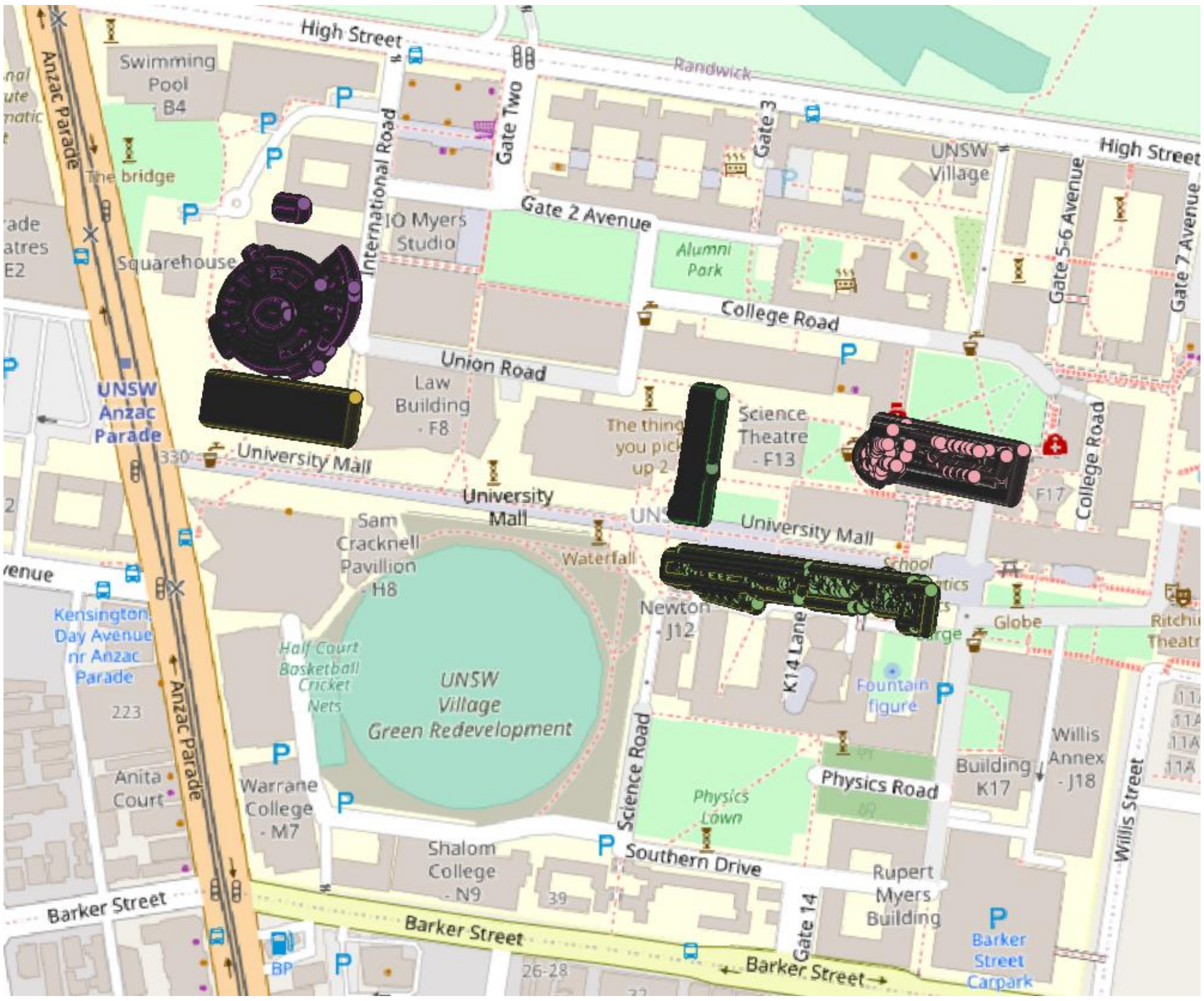
Ignore … maybe later …


High Street
Randwick
High Street
Anzac Parade
The bridge
International Road
Gate Two
Alumni Park
Road
UNSW Anzac Parade
University Mall
University Mall
UNSW
Waterfall
H8
UNSW Village Green Redevelopment
Kensington Day Avenue nr Anzac Parade
223
Anita Court
Science Road
Physics Lawn
Physics Road
Globe
Ritchie Theatre
Willis Street
Barker Street
Barker Street
Barker Street
Gate 14
BP
26-28
Carpark

High Street
Randwick
High Street
National Institute of Dramatic Art
Anzac Parade
The bridge
Parade Theatres - E2
International Road
Gate Two
Alumni Park
UNSW Anzac Parade
University Mall
University Mall
Waterfall
Day Avenue
Kensington Day Avenue nr Anzac Parade
UNSW Village Green Redevelopment
Science Road
Physics Road
Physics Lawn
Globe
Ritchie Theatre
Houston Road
Anita Court
Barker Street
Barker Street
Barker Street
Gate 14
Willis Street
Kennedy
BP
Carpark


High Street
Anzac Parade
International Road
Gate Two
Randwick
Alumni Park
University Mall
UNSW Anzac Parade
Waterfall
UNSW Village Green Redevelopment
Science Road
Physics Lawn
Physics Road
Globe
Ritchie Theatre
Willis Street
Kensington Day Avenue nr Anzac Parade
Anita Court
Barker Street
Gate 14
BP


High Street
Swimming Pool - B4
Randwick
Gate Two
Gate 3
International Road
The bridge
UNSW Village
High Street
IO Myers Studio
Gate 2 Avenue
Alumni Park
Gate 5-6 Avenue
Gate 7 Avenue
Squarehouse
College Road
Union Road
UNSW Anzac Parade
Law Building - F8
The thing you pick up 2
Science Theatre - F13
College Road
F17
University Mall
University Mall
University Mall
Sam Cracknell Pavillion - H8
Waterfall
School of Mathematics & Statistics
Newton - J12
Globe
Large
Ritchie Theatre
Kensington Day Avenue nr Anzac Parade
Half Court Basketball Cricket Nets
UNSW Village Green Redevelopment
K14 Lane
Fountain figure
223
Anzac Parade
Science Road
Willis Annex - J18
Building K17
Physics Road
Physics Lawn
Anita Court
Warrane College - M7
Willis Street
Shalom College - N9
Southern Drive
Rupert Myers Building
Barker Street
Barker Street
Barker Street
Gate 14
Barker Street Carpark
BP
39
26-28
32
330


High Street
Swimming Pool - B4
The bridge
Squarehouse
UNSW Anzac Parade
Anzac Parade
International Road
Gate Two
IO Myers Studio
Gate 2 Avenue
Alumni Park
Gate 3
Randwick
UNSW Village
Gate 5-6 Avenue
Gate 7 Avenue
College Road
Union Road
Law Building - F8
University Mall
The thing you pick up 2
Science Theatre - F13
Sam Cracknell Pavillion - H8
Waterfall
Newton - J12
K14 Lane
Globe
Ritchie Theatre
Kensington Day Avenue nr Anzac Parade
Half Court Basketball Cricket Nets
UNSW Village Green Redevelopment
Science Road
Fountain figure
Physics Road
Building K17
Willis Annex - J18
Willis Street
Anita Court
Warrane College - M7
Physics Lawn
Shalom College - N9
Southern Drive
Rupert Myers Building
Gate 14
Barker Street
Barker Street Carpark
BP

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”.

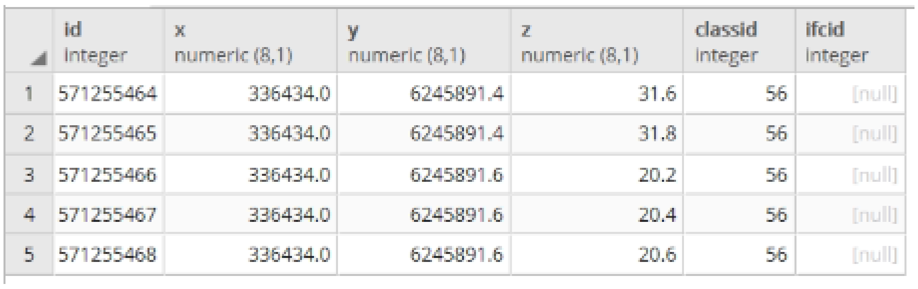
```sql
DROP TABLE voxel;
ALTER TABLE voxelflat RENAME TO voxel;
```

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

```sql
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

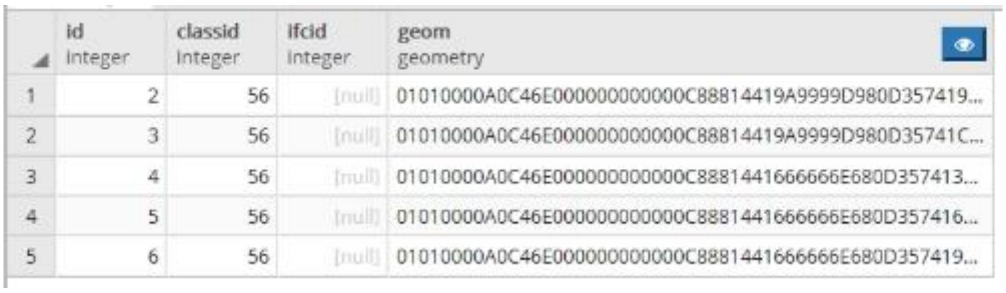
```sql
INSERT INTO voxelpt(classID, ifcID, geom) SELECT classID, ifcID, ST_SetSRID (ST_MakePoin
t(x,y,z), 28356) FROM voxel AS VALUES;
```

```
Query returned successfully in 2 h 21 min 52 secs.
```

4.3 Build Index

```sql
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

```sql
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

```sql
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 119 min 40 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
```

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'.

```
INSERT INTO ifcclass(ifcid, name) VALUES (30, '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 14 min 17 secs.

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

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 $$
BEGIN
    FOR idx in 1..118
    LOOP
        INSERT INTO voxelmpt(classID, ifcID, geom)
        VALUES (55, 27, ST_Collect(ARRAY(
            SELECT V.geom
            FROM voxelpt V
            JOIN road R ON ST_WITHIN(V.geom, R.geom)
            WHERE V.classid=55 AND R.id=idx)));
    END LOOP;
END;
$$
```

Query returned successfully in 145 min 16 secs.

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 AND V.geom IS NOT NULL)));
    END LOOP;
END;
$$
```

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 $$
BEGIN
    FOR idx in 1..120
    LOOP
        INSERT INTO voxelmpt(classID, ifcID, geom)
        VALUES (55, 28, ST_Collect(ARRAY(
            SELECT V.geom
            FROM voxelpt V
            JOIN greenareas G ON ST_WITHIN(V.geom, G.geom)
            WHERE V.classid=55 AND G.gid=idx)));
    END LOOP;
END;
$$

```

```
Query returned successfully in 58 min 23 secs.
```

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 (55, 29, ST_Collect(ARRAY(
            SELECT V.geom
            FROM voxelpt V
            JOIN building B ON ST_WITHIN(V.geom, B.geom)
            WHERE V.classid=55 AND B.id=idx)));
    END LOOP;
END;
```

```
Query returned successfully in 119 min 18 secs.
```

Don’t forget to update the “ifcID” for building voxels without IFC semantic.

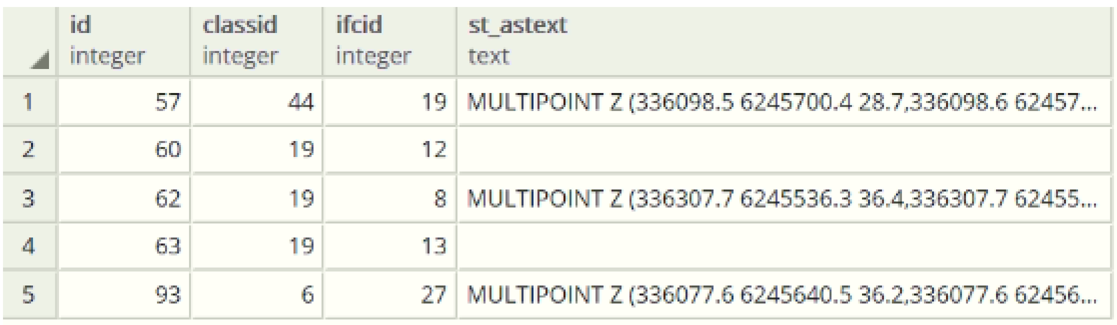
```
UPDATE voxelmpt SET ifcID=29 WHERE classid=ifcID IS NULL;
```

```
Query returned successfully in 107 msec.
```

5.3.4.4 Others

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

5.4 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.
```

~~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)~~
~~);~~
```

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.

~~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;~~

~~$$~~

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

```
DO $$
BEGIN
    FOR idx in 1..1345
    LOOP
        INSERT INTO voxelpatch(classID, ifcID, pa)
        VALUES (54, 30, PC_Patch(ARRAY(
            SELECT V.pt
            FROM (SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM voxel WHERE classID=54) AS V
            JOIN tree T ON ST_WITHIN(V.pt::geometry, ST_Buffer(T.geom, 4, 'quad_segs=8'))
            WHERE T.id=idx)));
    END LOOP;
END;
$$
```

~~DO $$~~

~~BEGIN~~

~~FOR idx in 1..1345~~

~~LOOP~~

~~INSERT INTO voxelpatch(classID, ifcID, pa)~~

~~VALUES (54, 30, PC_Patch(ARRAY(~~

~~SELECT V.pt~~

~~FROM (SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM voxel WHERE classID=54) AS V~~

~~JOIN tree T ON ST_WITHIN(V.pt::geometry, ST_Buffer(T.geom, 4, 'quad_segs=8'))~~

~~WHERE T.id=idx)));~~

~~END LOOP;~~

~~END;~~

~~$$~~

6.3.4 Dtmbot

7.3.4.1 Road Surface

```
DO $$
BEGIN
```

```sql
    FOR idx in 1..118
    LOOP
        INSERT INTO voxelpatch(classID, ifcID, pa)
        VALUES (55, 27, PC_Patch(ARRAY(
            SELECT V.pt
            FROM (SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM voxel WHERE cla
ssID=55) AS V
            JOIN road R ON ST_WITHIN(V.pt::geometry, R.geom)
            WHERE R.id=idx)));
    END LOOP;
END;
$$
```

```sql
DO $$
BEGIN
   FOR idx in 1..118
   LOOP
      INSERT INTO voxelpatch(classID, ifcID, pa)
      VALUES (55, 27, PC_Patch(ARRAY(
         SELECT V.pt
         FROM (SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM voxel WHERE classID=55)
AS V
         JOIN road R ON ST_WITHIN(V.pt::geometry, R.geom)
         WHERE R.id=idx)));
   END LOOP;
END;
$$
```

6.3.4.2 Greenarea Surface

```sql
DO $$
BEGIN
   FOR idx in 1..120
   LOOP
      INSERT INTO voxelpatch(classID, ifcID, pa)
      VALUES (55, 28, PC_Patch(ARRAY(
         SELECT V.pt
         FROM (SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM voxel WHERE classID=55)
AS V
         JOIN greenareas G ON ST_WITHIN(V.pt::geometry, G.geom)
```

```
            WHERE G.gid=idx)));
    END LOOP;
END;
$$
```

6.3.4.3 Building Surface

```sql
DO $$
BEGIN
    FOR idx in 1..121
    LOOP
        INSERT INTO voxelpatch(classID, ifcID, pa)
        VALUES (55, 29, PC_Patch(ARRAY(
            SELECT V.pt
            FROM (SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM voxel WHERE classID=55) AS V
            JOIN building B ON ST_WITHIN(V.pt::geometry, B.geom)
            WHERE B.id=idx)));
    END LOOP;
END;
$$
```

```
DO $$
BEGIN
    FOR idx in 1..121
    LOOP
        INSERT INTO voxelpatch(classID, ifcID, pa)
        VALUES (55, 29, PC_Patch(ARRAY(
            SELECT V.pt
            FROM (SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM voxel WHERE classID=55)
AS V
            JOIN building B ON ST_WITHIN(V.pt::geometry, B.geom)
            WHERE B.id=idx)));
    END LOOP;
END;
$$
```

7. Align the IFC with the Extrusion Buildings

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.

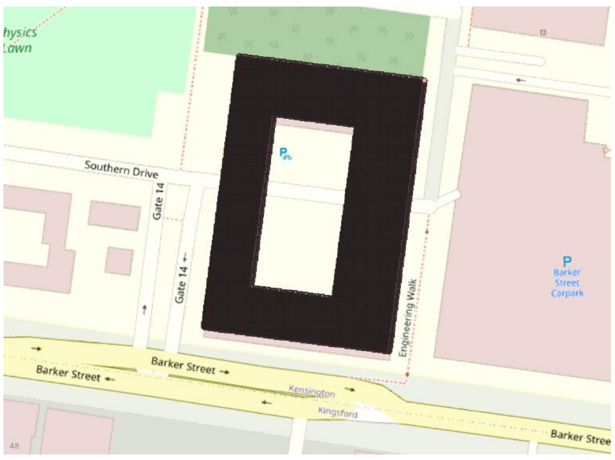
What has been given as one footprint on the 2D map might have been two buildings and vice versa… two footprints have been drawn for one building. This might happen with buildings like Red Centre (check OSM map: https://www.openstreetmap.org/#map=19/-33.91822/151.22950). What is on the maps in not wrong: 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.

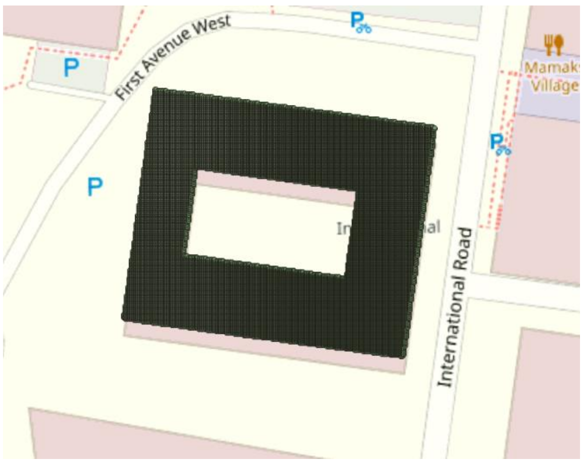
I suggest you measure the offset in QGIS with respect to the footprint of the map. What I will do I would print all extruded buildings in groups of 5-10 to see how well they fit with the foot prints. They are obtained from the footprints, so they have to be the same. If they are not the same I will created 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:

Physics Road
Rupert Myers Building
Engineering Walk
Willis Street
Barker Street
Kensington
Kingsford

Union Road
Science and Engineering Building (E10)
Thoughtful Foods
University Mall
Seeing the wood for the trees

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~~

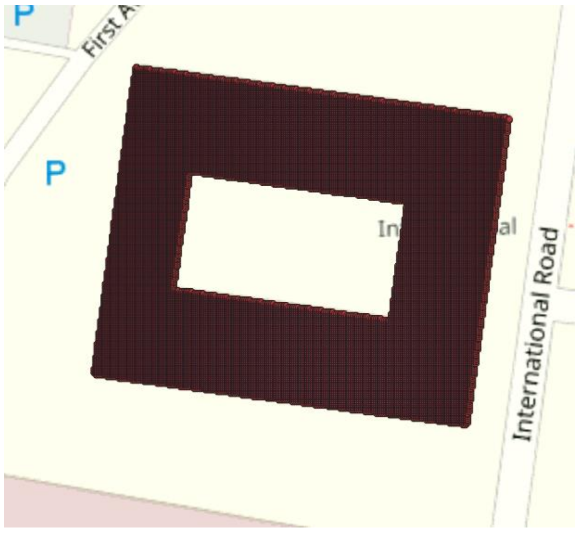
~~Case-II: Only move x~~

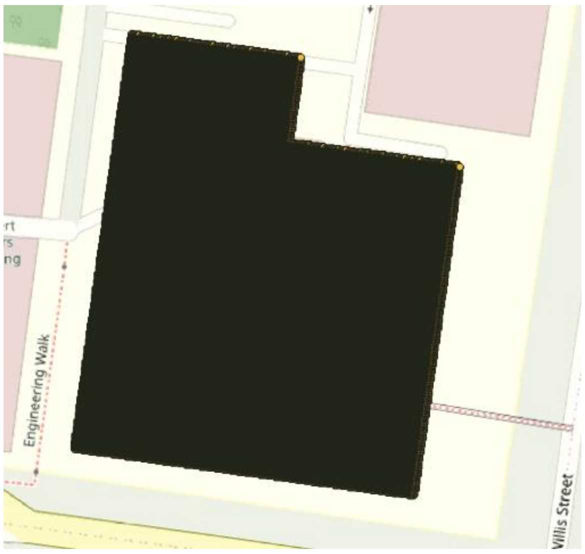
~~Case-III: Move x and y~~

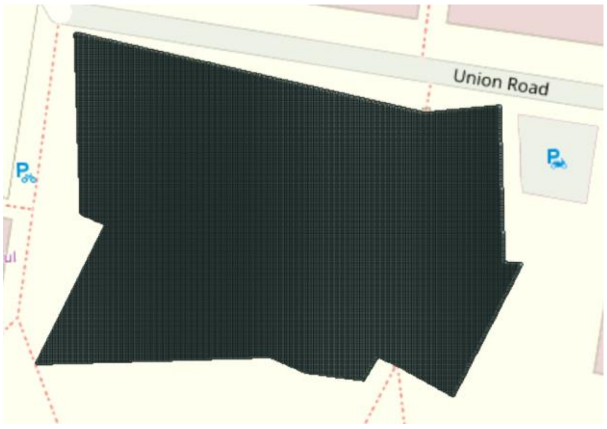
==> **336000, 6245247, 20**

After "y" correction:

Southern Drive
Gate 14
Gate 14
Engineering Walk
Barker Street
Kensington



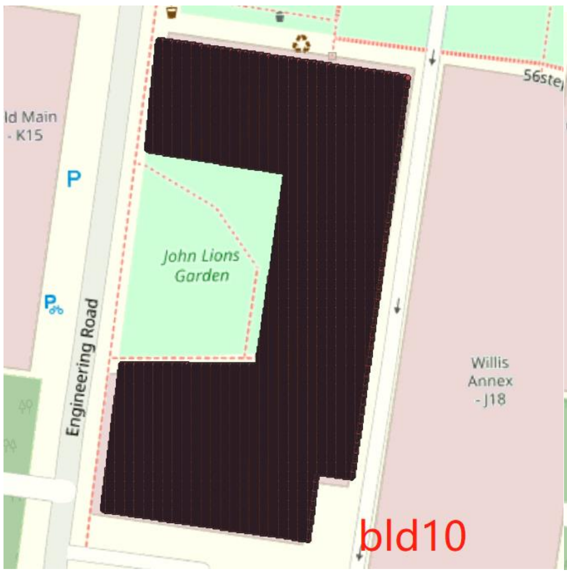


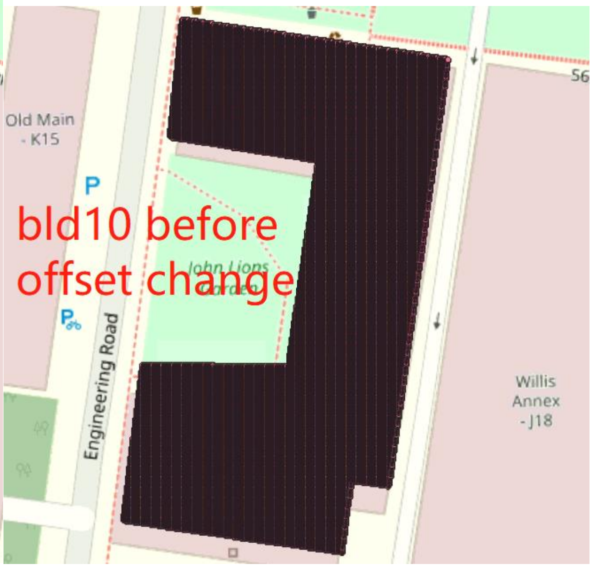


New problem comes up: Part of voxel in one object may be redundant. For instance, in "bld4", some voxels are on the road. So the next step is to select all the buildings of this kind of problem.

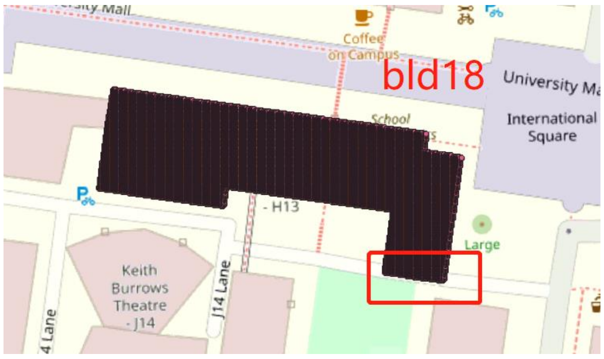
Buildings with redundant voxels:

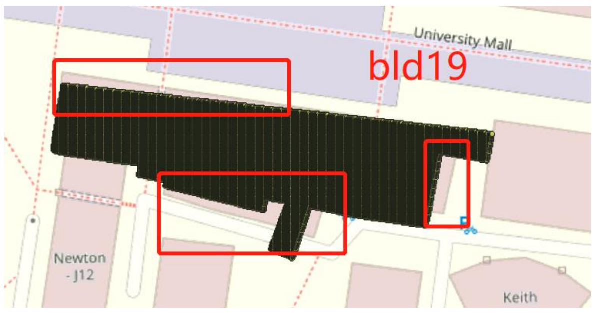
- bld4
- bld10 contains two buildings (K17 and J17)

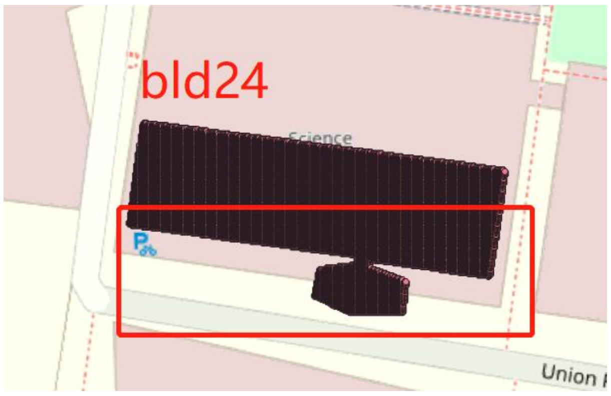


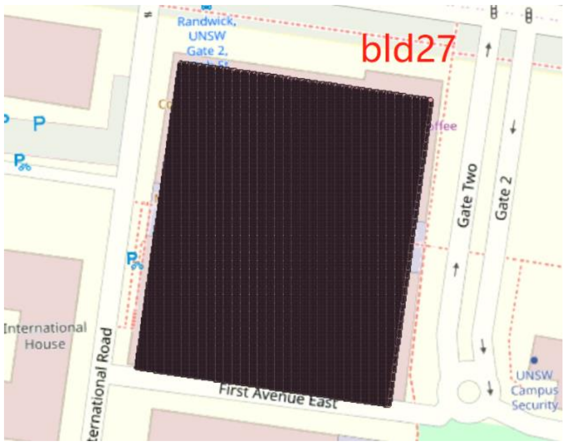


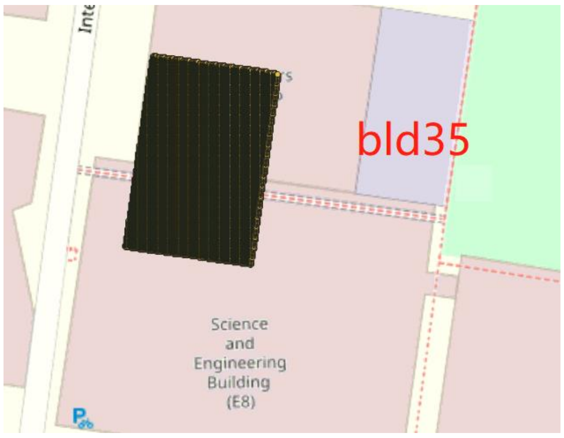
When visualizing "bld10", new problem has arisen. That is, only changing offset is not enough, we need to change (x,y) for each voxel in bld10. Besides, some of them are redundant, some of them are missing. We can drop the redundant part, but how to fill in the missing pieces. And do we need to separate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

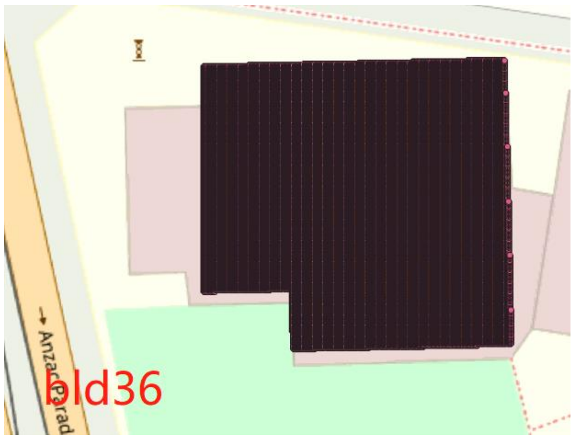

bld18
University Mall
Coffee on Campus
University Ma
International Square
School
- H13
Large
Keith Burrows Theatre - J14
J14 Lane
4 Lane

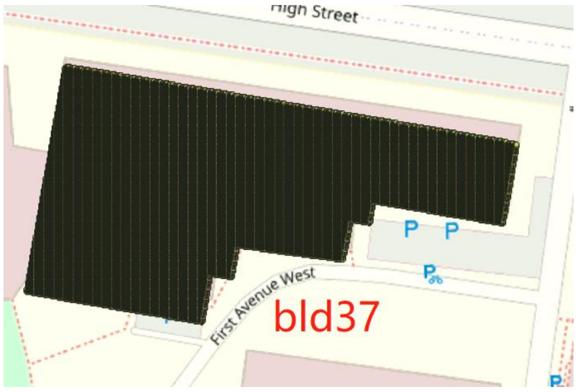

University Mall
bld19
Newton - J12
Keith

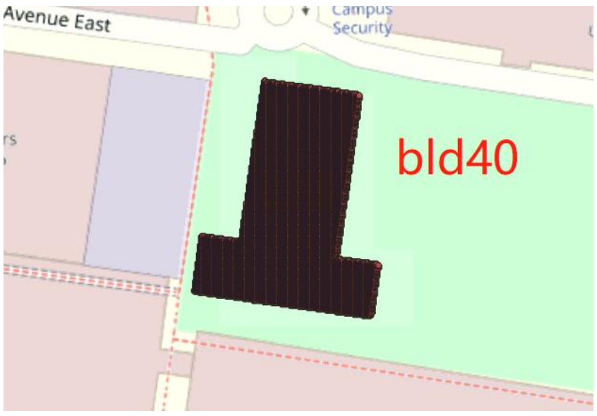

bld24
Science
Union

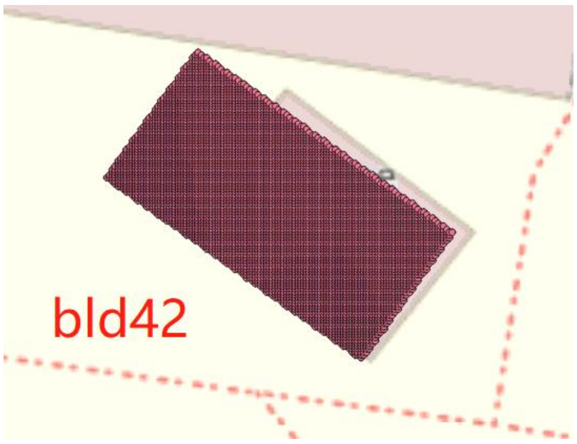

Randwick, UNSW Gate 2,
bld27
Gate Two
Gate 2
International House
International Road
First Avenue East
UNSW Campus Security

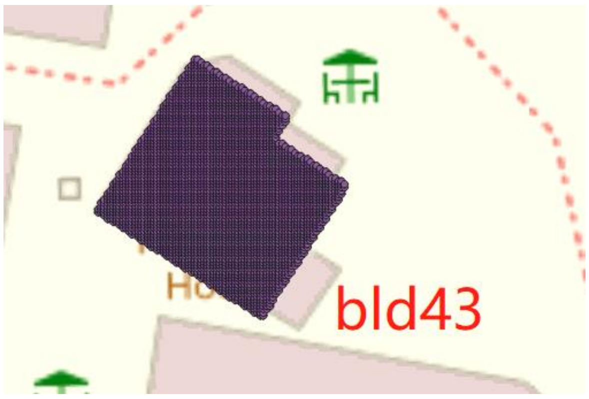

bld35
Science and Engineering Building (E8)


bld36
Anzac Parad


High Street
First Avenue West
bld37


Avenue East
Campus Security
bld40



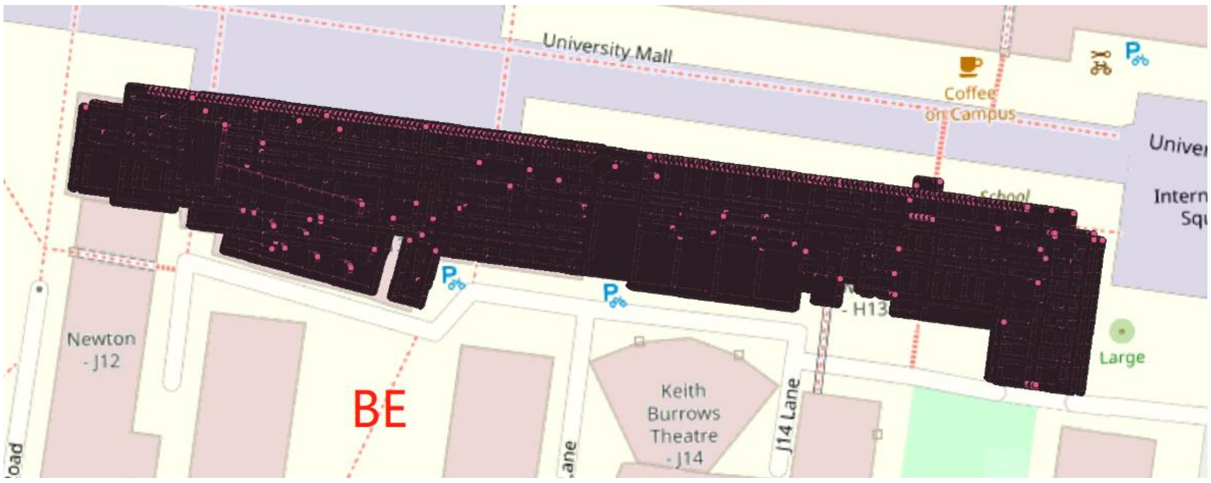


UNSW
Village
bld53
Fig T

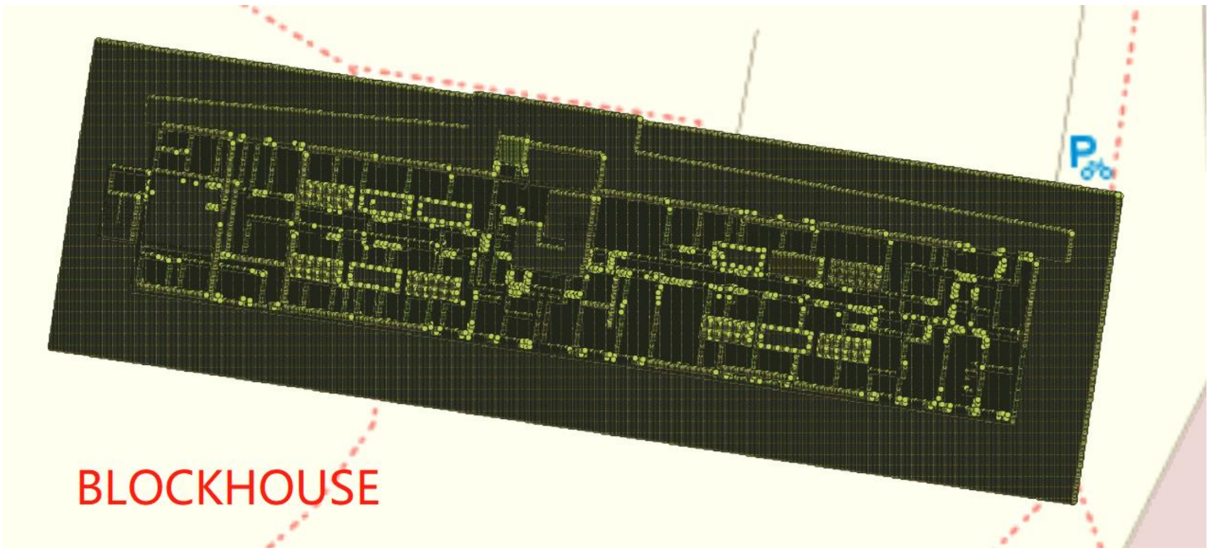
bld54
Campus
Village
Cafe
UNSW

7.2 IFC

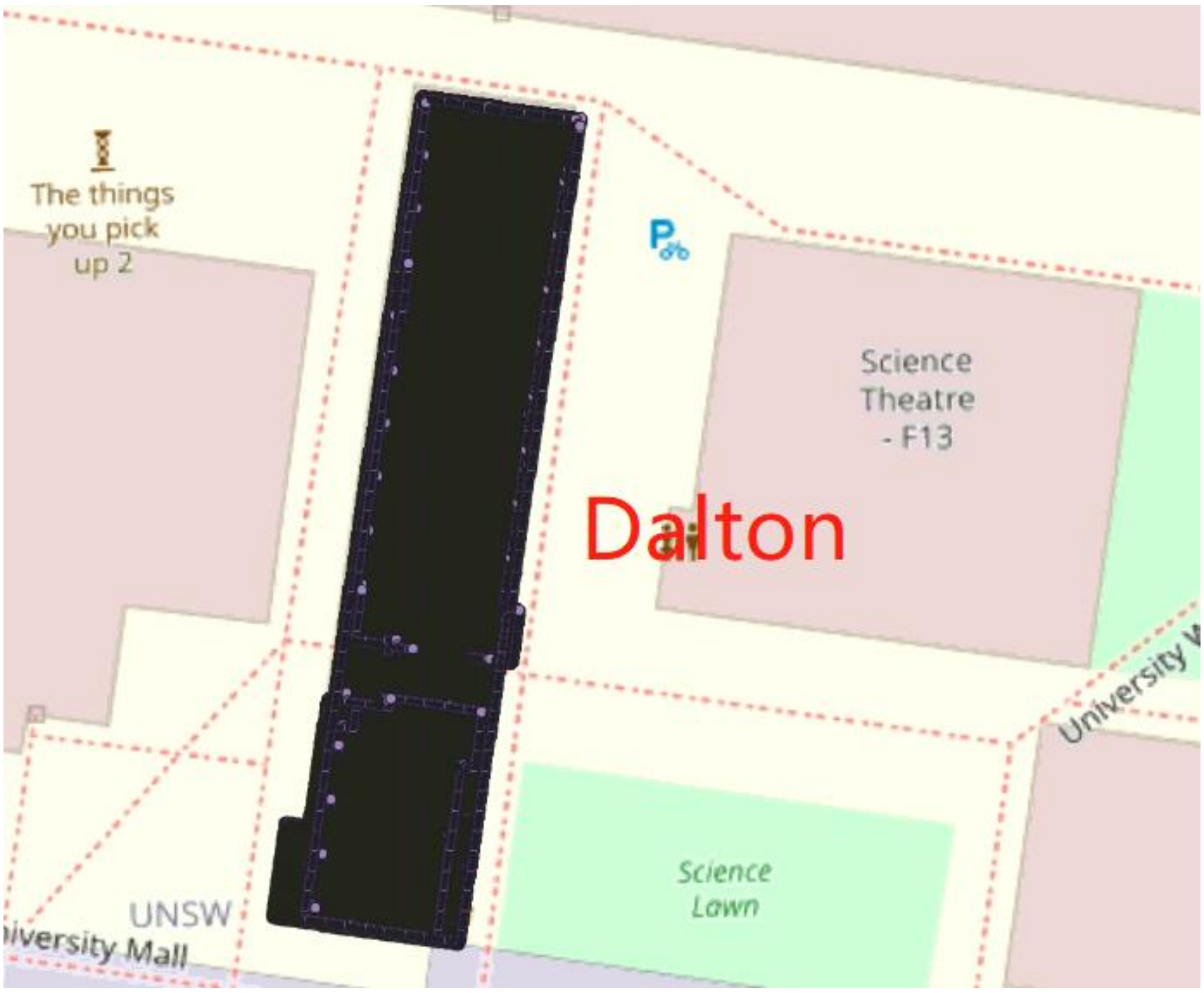
select ST_Translate(ST_Scale(geom, 0.1,0.1,0.1), 336300, 6245507, 25) as geom from voxelmpt where classid=19 and ifcid is not null;



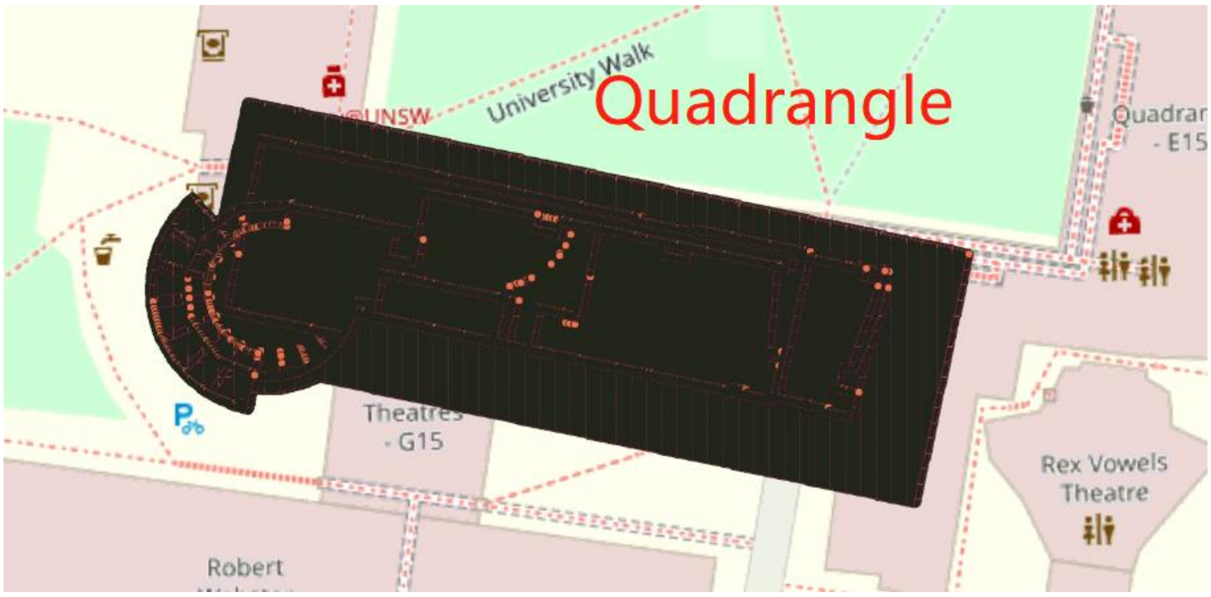
select ST_Translate(ST_Scale(geom, 0.1,0.1,0.1), 336042, 6245613, 27) as geom from voxelmpt where classid=6 and ifcid is not null;



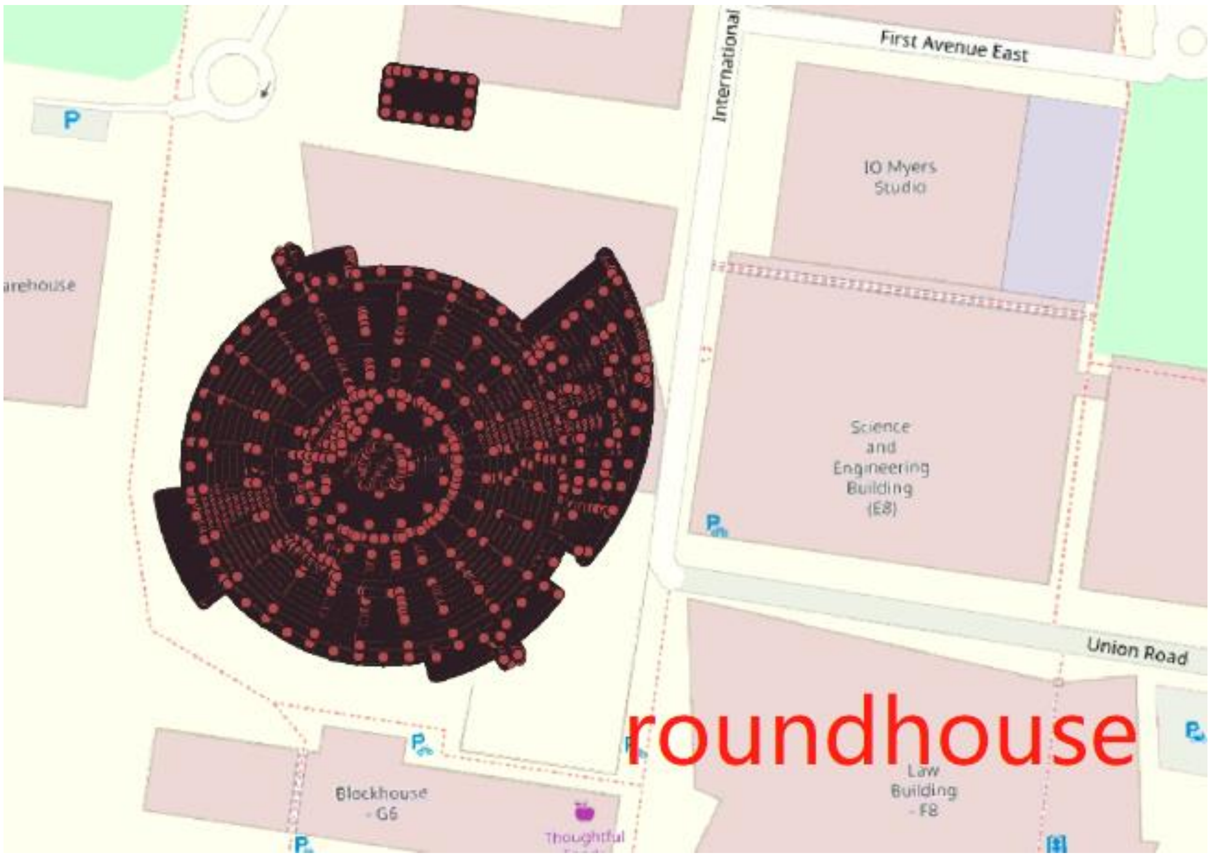
select ST_Translate(ST_Scale(geom, 0.1,0.1,0.1), 336305, 6245569, 29) as geom from voxelmpt where classid=9 and ifcid is not null;



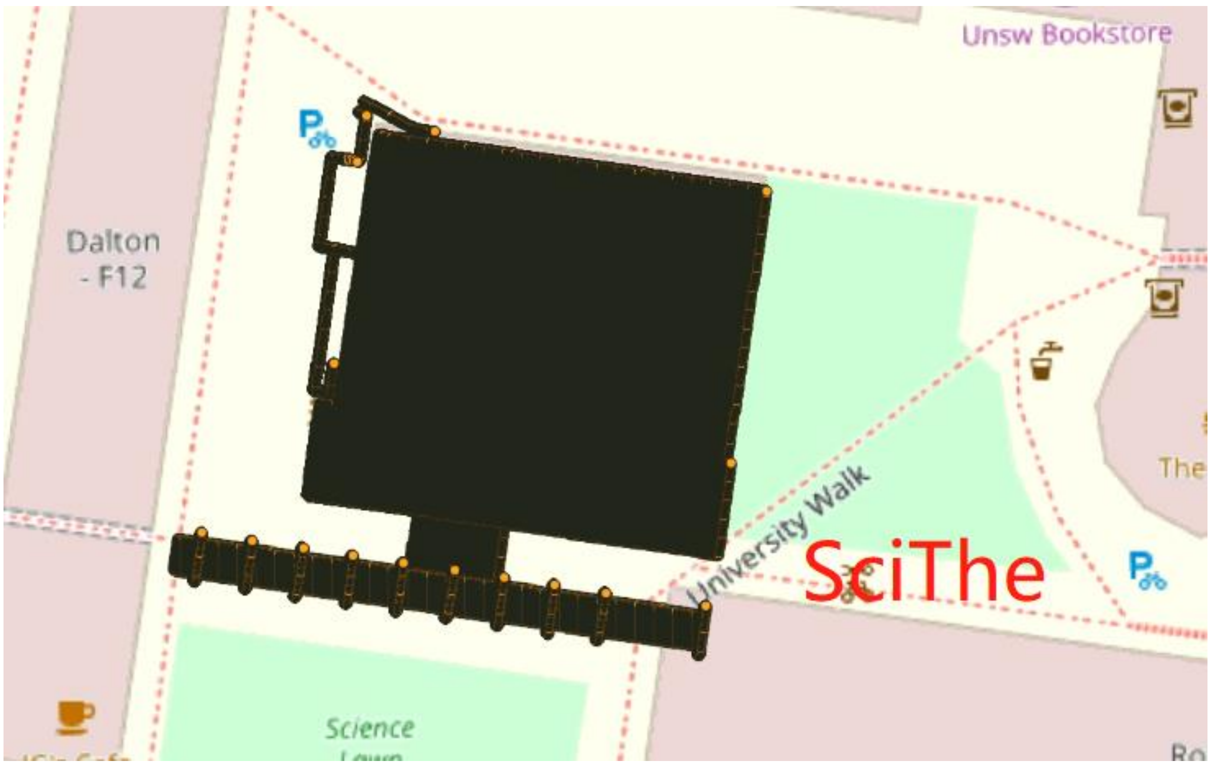
select ST_Translate(ST_Scale(geom, 0.1,0.1,0.1), 336409, 6245580, 31) as geom from voxelmpt where classid=13 and ifcid is not null;



```
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 Lawn

8. Sample Queries and QGIS Visualization

The data query is processed using a HP laptop. Its processor is Intel(R) Core (TM) i7-7600 CPU @ 2.80GHz and its installed memory is 16.0 GB. Its operating system is 64-bit Windows 10. And the test is performed on PostgreSQL (11.2), PostGIS (2.5.2), and QGIS (3.6.1).

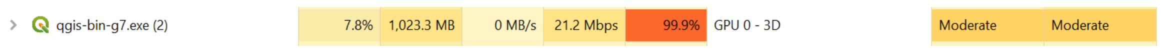
8.1 Sample Visualization

Considering bld52 (we don't have building name at this moment), 29.6MB and 1,881,847 voxels, we extract "geom" first, and then convert its coordinate into EPSG:28356.

Figure 3 shows how to execute query in QGIS.

Figure 3. Query execution in QGIS

Note that, loading the above query result as a new layer in QGIS may take several minutes and 3D view is as well. Once choosing 3D view, please keep an eye out for your GPU and memory changes, if you crash, kill the task or stop doing the work at hand and continue to wait patiently. If not necessary or you are not confident in your PC, don't try 3D view.



qgis-bin-g7.exe (2)	7.8%	1,023.3 MB	0 MB/s	21.2 Mbps	99.9%	GPU 0 - 3D	Moderate	Moderate

Figure 3. 2D and 3D visualization of "bld52" in QGIS