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.
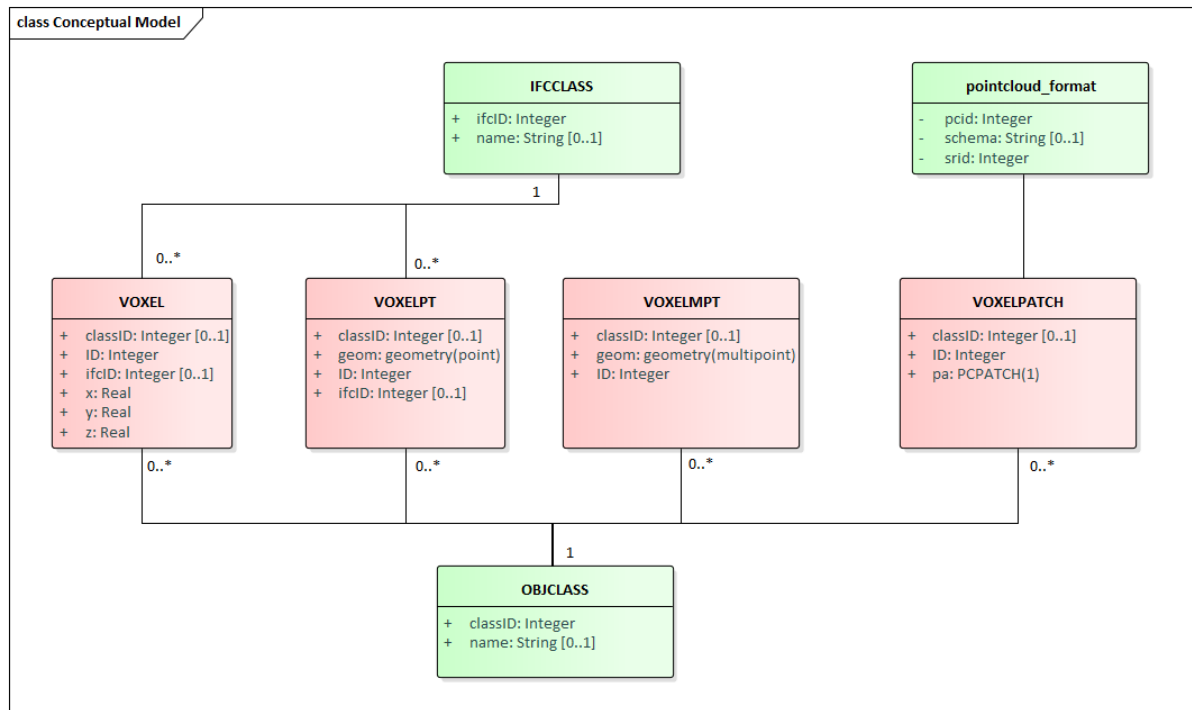
1.2 Database Schema

Figure 1-1. Conceptual Model of VoxelDB

1.3 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	1,252	1,252
Geometry	NO	POINT	MULTIPOINT	PCPATCH
Columns	id classID ifcID x y z	id classID ifcID geom	id classID ifcID geom	id classID ifcID pa
#index	1	2	2	2
General Index	YES x, y, z, classID, ifcID	YES classID, ifcID	YES classID, ifcID	YES classID, ifcID
Geometry Index	NO N/A	YES point geometry	YES multipoint geometry	YES 2D bounds of the patch
Index Size	74 GB	103 GB	264 kB	200 kB

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

```
1.  DO $$
2.  BEGIN
3.      FOR idx in 1..54
4.      LOOP
5.          IF idx = 26 OR idx = 46 THEN
6.              raise notice 'The buidling % could not be found', idx;
7.          ELSE
8.              INSERT INTO objclass(classID)
9.              VALUES (idx);
10.         END IF;
11.     END LOOP;
12. END;
13. $$
14.
```

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".

Note that, the last four IFC features (including 27-"road", 28-"greenarea", 29-"building" and 30-"tree") only serve Table – "**voxelmpt**" and "**voxelpatch**".

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.

```
1.  \COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\BE\
    classmodel.xyz' DELIMITER ' ';
2.  UPDATE voxel SET classID=57 WHERE classid IS NULL;
3.  \COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\Blo
    ckHouse\classmodel.xyz' DELIMITER ' ';
4.  UPDATE voxel SET classID=58 WHERE classid IS NULL;
5.  \COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\Dal
    ton\classmodel.xyz' DELIMITER ' ';
6.  UPDATE voxel SET classID=59 WHERE classid IS NULL;
7.  \COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\Qua
    drangle\classmodel.xyz' DELIMITER ' ';
8.  UPDATE voxel SET classID=60 WHERE classid IS NULL;
9.  \COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\Rou
    ndhouse\classmodel.xyz' DELIMITER ' ';
10. UPDATE voxel SET classID=61 WHERE classid IS NULL;
11. \COPY voxel(x, y, z, ifcID) FROM 'C:\Users\z5039792\Documents\Vox3DMod\data\bim\Sci
    The\classmodel.xyz' DELIMITER ' ';
12. 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.

```
1.  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
2.  FROM voxel
3.  WHERE classid=57;
4.
5.  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
6.  FROM voxel
7.  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

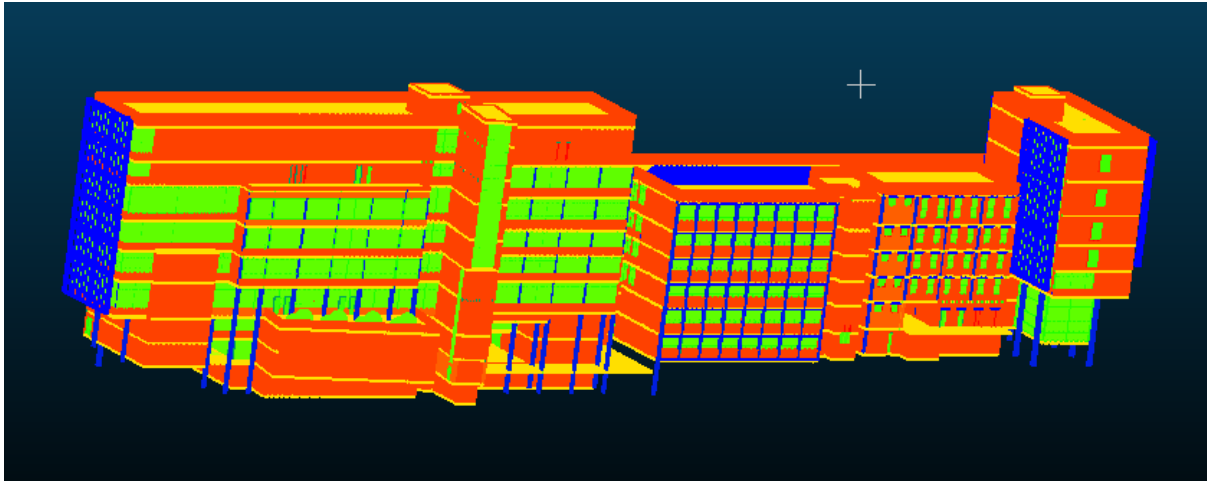Then, visualizing above two buildings in CloudCompare, it looks similar, at least in shape.

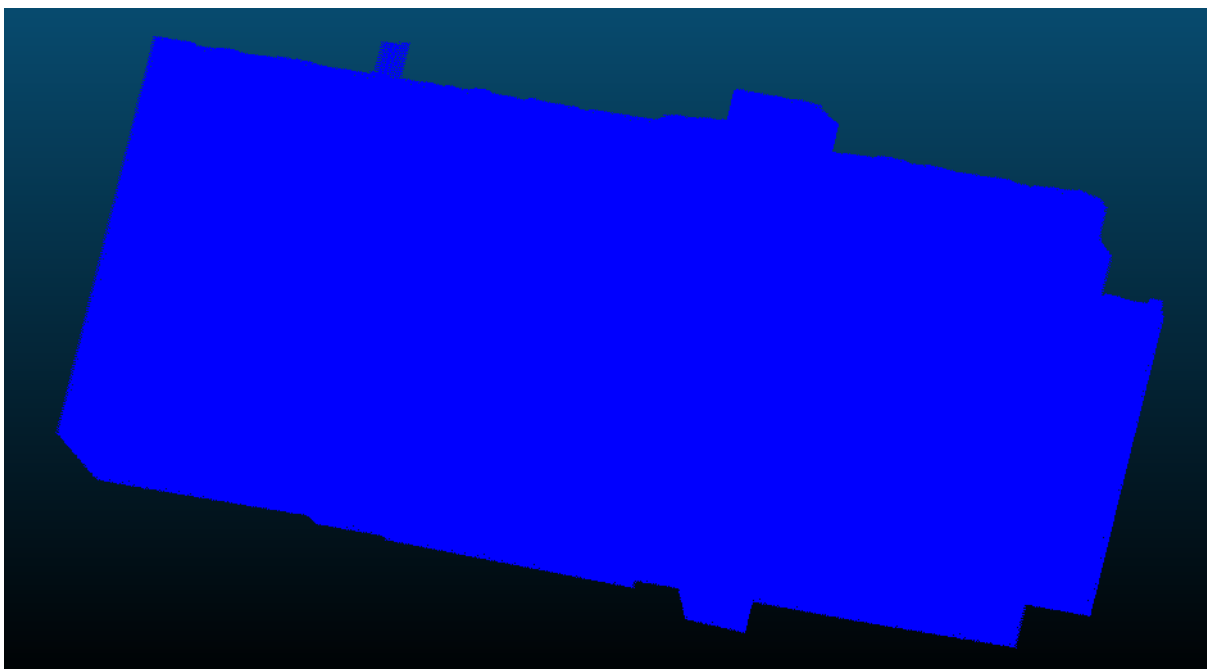Figure 6. BE building in CloudCompare

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.

```
1.  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
2.  FROM voxel
3.  WHERE classid=58;
4.
5.  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
6.  FROM voxel
7.  WHERE classid=59;
8.
9.  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
10. FROM voxel
11. WHERE classid=60;
12.
```

```
13. 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
14. FROM voxel
15. WHERE classid=61;
16.
17. 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
18. FROM voxel
19. WHERE classid=62;
20.
21. 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
22. FROM voxel
23. WHERE classID<=54
24. 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

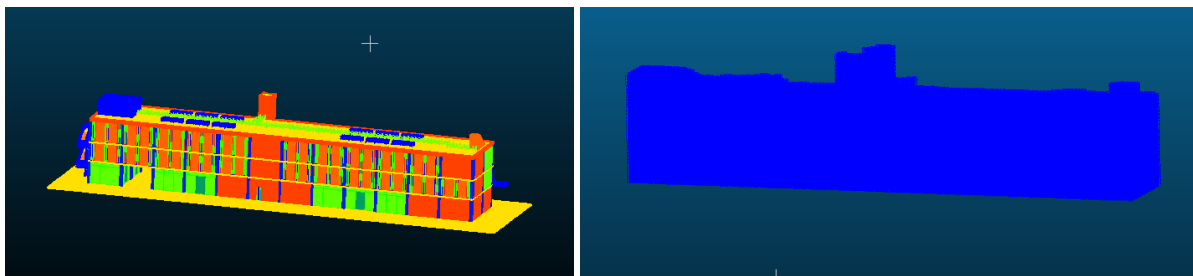Figure 8. (x,y) range for all 52 building
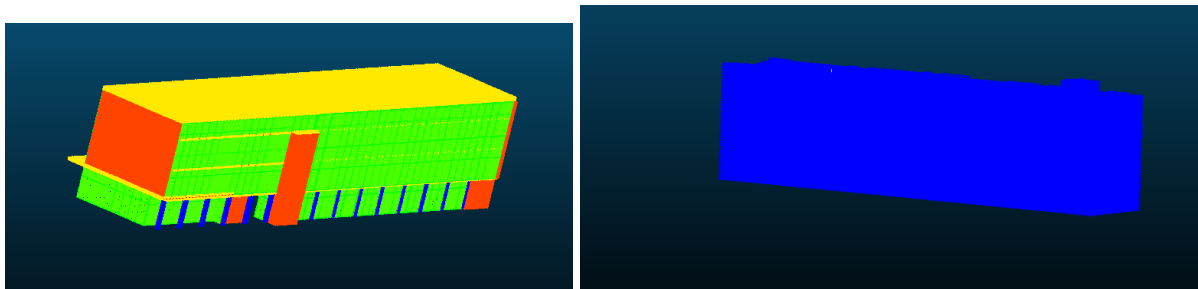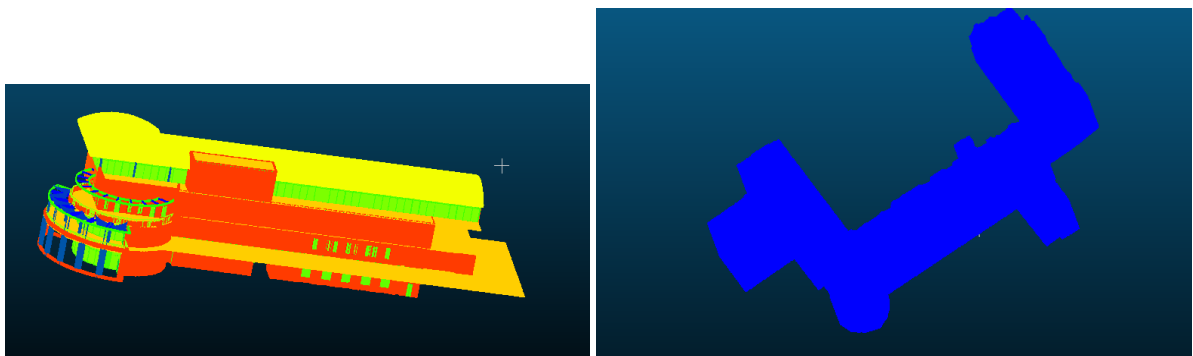
Figure 9. Blockhouse
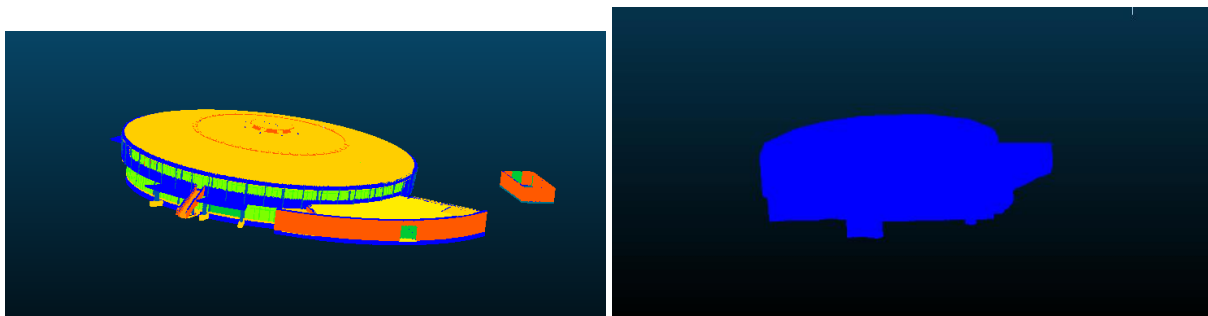

Figure 10. Dalton
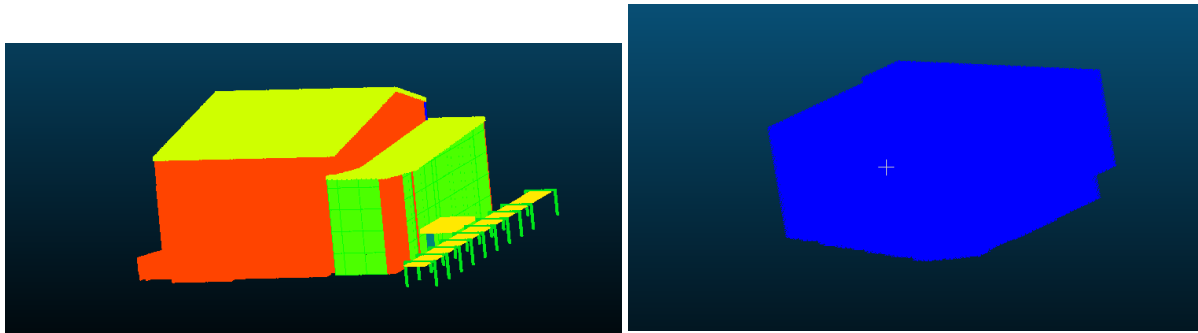

Figure 11. Quadrangle

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
1.  UPDATE voxel SET classID=19 WHERE classid=57;
2.  UPDATE voxel SET classID=6 WHERE classid=58;
3.  UPDATE voxel SET classID=9 WHERE classid=59;
4.  UPDATE voxel SET classID=13 WHERE classid=60;
5.  UPDATE voxel SET classID=44 WHERE classid=61;
6.  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 …
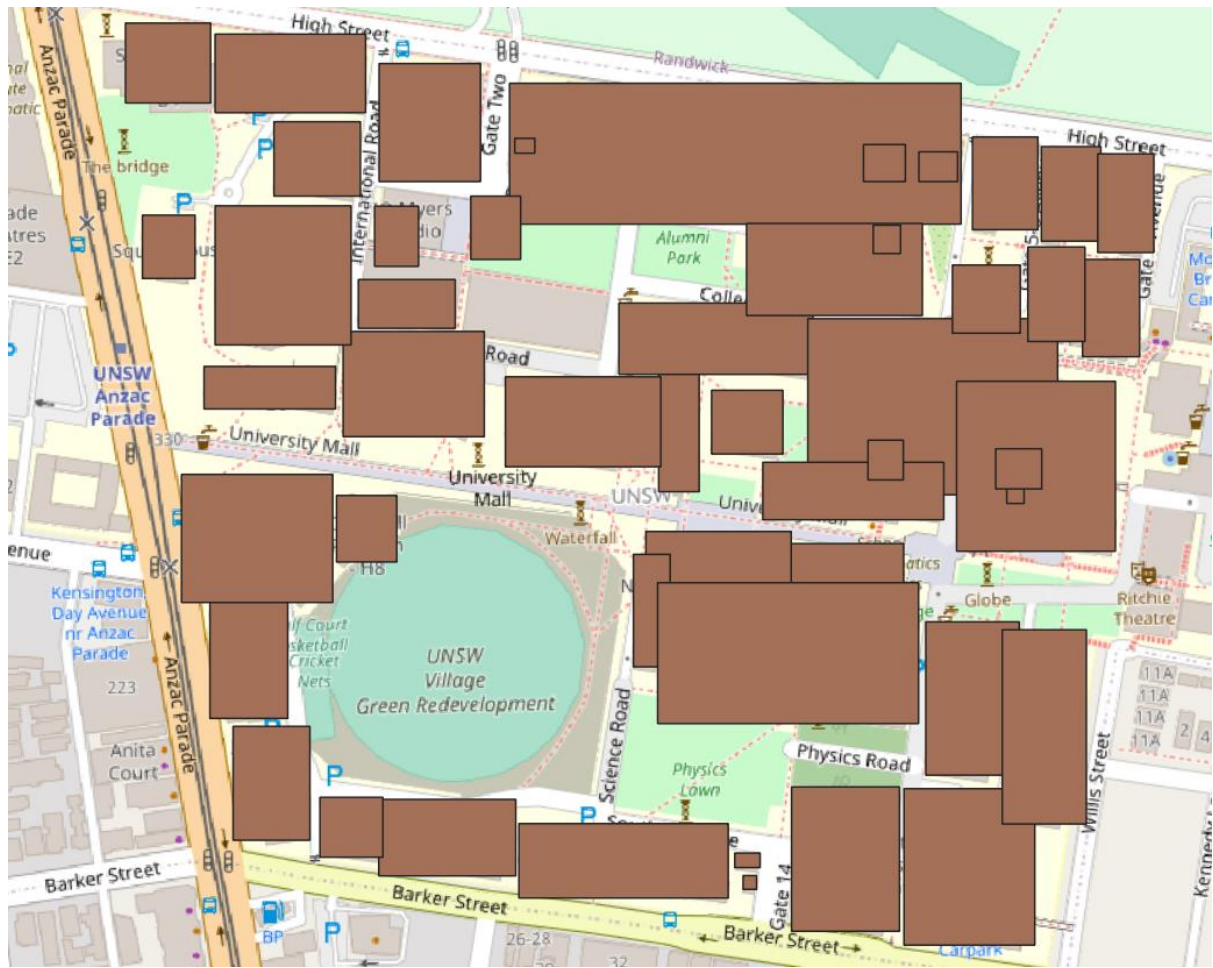
3. Create Flat ARRAY Table

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

```
1.  DROP TABLE IF EXISTS voxelflat CASCADE;
2.  CREATE TABLE voxelflat
3.  (
4.      id serial PRIMARY KEY,
5.      x NUMERIC(8,1) NOT NULL,
6.      y NUMERIC(8,1) NOT NULL,
7.      z NUMERIC(8,1) NOT NULL,
8.      classID INTEGER,
9.      ifcID INTEGER
10. );
```

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.

```
1.  INSERT INTO voxelflat(x,y,z,classID,ifcID)
2.  SELECT x*0.2+336000, y*0.2+6245250, z*0.2+20, classID, ifcID FROM voxel
3.  WHERE ifcID IS NULL;
4.  INSERT INTO voxelflat(x,y,z,classID,ifcID)
5.  SELECT x*0.1+336300, y*0.1+6245507, z*0.1+25, classID, ifcID FROM voxel
6.  WHERE classID=19 AND ifcID IS NOT NULL;
7.  INSERT INTO voxelflat(x,y,z,classID,ifcID)
8.  SELECT x*0.1+336042, y*0.1+6245613, z*0.1+27, classID, ifcID FROM voxel
9.  WHERE classID=6 AND ifcID IS NOT NULL;
10. INSERT INTO voxelflat(x,y,z,classID,ifcID)
11. SELECT x*0.1+336305, y*0.1+6245569, z*0.1+29, classID, ifcID FROM voxel
12. WHERE classID=9 AND ifcID IS NOT NULL;
13. INSERT INTO voxelflat(x,y,z,classID,ifcID)
14. SELECT x*0.1+336409, y*0.1+6245580, z*0.1+31, classID, ifcID FROM voxel
15. WHERE classID=13 AND ifcID IS NOT NULL;
16. INSERT INTO voxelflat(x,y,z,classID,ifcID)
17. SELECT x*0.1+336047, y*0.1+6245651, z*0.1+25, classID, ifcID FROM voxel
18. WHERE classID=44 AND ifcID IS NOT NULL;
19. INSERT INTO voxelflat(x,y,z,classID,ifcID)
20. SELECT x*0.1+336325, y*0.1+6245582, z*0.1+28, classID, ifcID FROM voxel
21. 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".

```
1.  DROP TABLE voxel;
2.  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".

```
1.  DROP INDEX IF EXISTS idx_voxel CASCADE;
2.  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

```
1.  CREATE EXTENSION IF NOT EXISTS POSTGIS;
2.  DROP TABLE IF EXISTS voxelpt CASCADE;
3.  CREATE TABLE voxelpt
4.  (
5.      id serial PRIMARY KEY,
6.      classID INTEGER,
7.      ifcID INTEGER,
8.      geom geometry(POINTZ,28356)
9.  );
```

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

```
1.  DROP INDEX IF EXISTS idx_voxelpt CASCADE;
2.  DROP INDEX IF EXISTS geom_ voxelpt CASCADE;
3.  CREATE INDEX idx_voxelpt ON voxelpt(classID, ifcID);
4.  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]	01010000A0C46E000000000000000C88814419A9999D980D357419...
2	3	56	[null]	01010000A0C46E000000000000000C88814419A9999D980D35741C...
3	4	56	[null]	01010000A0C46E000000000000000C8881441666666E680D357413...
4	5	56	[null]	01010000A0C46E000000000000000C8881441666666E680D357416...
5	6	56	[null]	01010000A0C46E000000000000000C8881441666666E680D357419...

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
1.  DROP TABLE IF EXISTS voxelmpt CASCADE;
2.  CREATE TABLE voxelmpt
3.  (
4.      id serial PRIMARY KEY,
5.      classID INTEGER,
6.      ifcID INTEGER,
7.      geom geometry(MULTIPOINTZ,28356)
8.  );
```
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
1.  DO $$
2.  DECLARE
3.      f record;
4.  BEGIN
5.      FOR idx in 1..54
6.      LOOP
7.          IF idx = 26 OR idx = 46 THEN
8.              raise notice 'The buidling % could not be found', idx;
9.          ELSE
10.             INSERT INTO voxelmpt(classID, geom)
11.             VALUES (idx, ST_Collect(ARRAY(SELECT geom FROM voxelpt WHERE ifcID IS
     NULL AND classID=idx)));
12.         END IF;
13.     END LOOP;
14. END;
15. $$
16.
```
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.

```
1.  DO $$
2.  DECLARE
3.      f record;
4.  BEGIN
5.      FOR f in SELECT DISTINCT classID, ifcID
6.          FROM voxelpt
7.      LOOP
8.          INSERT INTO voxelmpt(classID, ifcID, geom)
9.          VALUES (f.classID, f.ifcID, ST_Collect(ARRAY(SELECT geom FROM voxelpt WHE
    RE ifcID IS NOT NULL AND classID=f.classID AND ifcID=f.ifcID)));
10.     END LOOP;
11. END;
12. $$
13.
```

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.

```
1.  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
2.  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'.

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

10.             WHERE V.classid=55 AND T.id=idx)));
11.         DELETE FROM voxelmpt WHERE classID=55 AND ifcID=30 AND geom IS NULL;
12.     END LOOP;
13. END;
14. $$
15.
```

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.

```
1.  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
2.  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
3.  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:

```
1.  \i C:/Users/z5039792/Documents/Vox3DMod/data/lawn.sql
2.  \i C:/Users/z5039792/Documents/Vox3DMod/data/road.sql
3.  \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.

```
1.  INSERT INTO ifcclass(ifcid, name) VALUES (27, 'road');
2.  DO $$
3.  BEGIN
4.      FOR idx in 1..118
5.      LOOP
6.          INSERT INTO voxelmpt(classID, ifcID, geom)
7.          VALUES (56, 27, ST_Collect(ARRAY(
8.              SELECT V.geom
9.              FROM voxelpt V
10.             JOIN road R ON ST_WITHIN(V.geom, R.geom)
11.             WHERE V.classid=56 AND R.id=idx)));
12.         DELETE FROM voxelmpt WHERE classID=56 AND ifcID=27 AND geom IS NULL;
13.     END LOOP;
14. END;
15. $$
16.
```

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.

```
1.  INSERT INTO ifcclass(ifcid, name) VALUES (28, 'greenarea');
2.  DO $$
3.  BEGIN
4.      FOR idx in 1..120
5.      LOOP
6.          INSERT INTO voxelmpt(classID, ifcID, geom)
```

```
 7.            VALUES (56, 28, ST_Collect(ARRAY(
 8.                SELECT V.geom
 9.                FROM voxelpt V
10.                JOIN greenareas G ON ST_WITHIN(V.geom, G.geom)
11.                WHERE V.classid=56 AND G.gid=idx)));
12.            DELETE FROM voxelmpt WHERE classID=56 AND ifcID=28 AND geom IS NULL;
13.        END LOOP;
14. END;
15. $$
16.
```

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").

```
 1. INSERT INTO ifcclass(ifcid, name) VALUES (29, 'building');
 2. DO $$
 3. BEGIN
 4.    FOR idx in 1..121
 5.    LOOP
 6.        INSERT INTO voxelmpt(classID, ifcID, geom)
 7.        VALUES (56, 29, ST_Collect(ARRAY(
 8.            SELECT V.geom
 9.            FROM voxelpt V
10.            JOIN building B ON ST_WITHIN(V.geom, B.geom)
11.            WHERE V.classid=56 AND B.id=idx)));
12.        DELETE FROM voxelmpt WHERE classID=56 AND ifcID=29 AND geom IS NULL;
13.    END LOOP;
14. END;
15. $$
16.
```

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

```
 1. DROP INDEX IF EXISTS idx_voxelmpt CASCADE;
```

```
2.  DROP INDEX IF EXISTS geom_voxelmpt CASCADE;
3.  CREATE INDEX idx_voxelmpt ON voxelmpt(classID, ifcID);
4.  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

```
1.  CREATE EXTENSION IF NOT EXISTS pointcloud;
2.  CREATE EXTENSION IF NOT EXISTS pointcloud_postgis;
3.  DELETE FROM pointcloud_formats;
4.  INSERT INTO pointcloud_formats (pcid, srid, schema) VALUES (1, 28356,
5.  '<?xml version="1.0" encoding="UTF-8"?>
6.  <pc:PointCloudSchema xmlns:pc="http://pointcloud.org/schemas/PC/1.1"
7.      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8.    <pc:dimension>
9.      <pc:position>1</pc:position>
10.     <pc:size>4</pc:size>
11.     <pc:description>X coordinate as a float.</pc:description>
12.     <pc:name>X</pc:name>
13.     <pc:interpretation>double</pc:interpretation>
14.   </pc:dimension>
15.   <pc:dimension>
16.     <pc:position>2</pc:position>
17.     <pc:size>4</pc:size>
18.     <pc:description>Y coordinate as a float.</pc:description>
19.     <pc:name>Y</pc:name>
20.     <pc:interpretation>double</pc:interpretation>
21.   </pc:dimension>
22.   <pc:dimension>
23.     <pc:position>3</pc:position>
24.     <pc:size>4</pc:size>
25.     <pc:description>Z coordinate as a float.</pc:description>
26.     <pc:name>Z</pc:name>
27.     <pc:interpretation>double</pc:interpretation>
28.   </pc:dimension>
29.   <pc:metadata>
30.     <Metadata name="compression">dimensional</Metadata>
31.   </pc:metadata>
32. </pc:PointCloudSchema>');
33. DROP TABLE IF EXISTS voxelpatch CASCADE;
34. CREATE TABLE voxelpatch (
35.   id SERIAL PRIMARY KEY,
36.   classID INTEGER,
37.   ifcID INTEGER,
38.   pa PCPATCH(1)
39. );
40.
```

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

```
1.  DO $$
```

```
2.  BEGIN
3.      FOR idx in 1..54
4.      LOOP
5.          IF idx = 26 OR idx = 46 THEN
6.              raise notice 'The buidling % could not be found', idx;
7.          ELSE
8.              INSERT INTO voxelpatch(classID, pa)
9.              VALUES (idx, PC_Patch(ARRAY(SELECT PC_MakePoint(1, ARRAY[x,y,z]) as p
    t FROM voxel WHERE classID=idx AND ifcID IS NULL)));
10.         END IF;
11.     END LOOP;
12. END;
13. $$
```

Query returned successfully in 353 min 50 secs.

6.3.2 Building with IFC

```
1.  DO $$
2.  DECLARE
3.      f record;
4.  BEGIN
5.      FOR f in SELECT DISTINCT classID, ifcID
6.          FROM voxel WHERE ifcID IS NOT NULL
7.      LOOP
8.          INSERT INTO voxelpatch(classID, ifcID, pa)
9.          VALUES (f.classID, f.ifcID, PC_Patch(ARRAY(SELECT PC_MakePoint(1, ARRAY[x
    ,y,z]) as pt FROM voxel WHERE ifcID IS NOT NULL AND classID=f.classID  AND ifcID=
    f.ifcID)));
10.     END LOOP;
11. END;
12. $$
```

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" .

```
1.  DO $$
2.  DECLARE
3.      f record;
4.  BEGIN
5.      FOR f in SELECT geom
6.          FROM voxelmpt WHERE classID=55 AND ifcID=30 AND geom IS NOT NULL
7.      LOOP
8.          INSERT INTO voxelpatch(classID, ifcID, pa)
9.          VALUES (55, 30, PC_Patch(ARRAY(
10.             SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM (
11.                 SELECT ST_X((ST_DumpPoints(f.geom)).geom) AS x,
12.                 ST_Y((ST_DumpPoints(f.geom)).geom) AS y,
13.                 ST_Z((ST_DumpPoints(f.geom)).geom) AS z
14.                 ) AS g
15.             )));
16.     END LOOP;
17. END;
18. $$
19.
```

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

6.3.4 Dtmbot

7.3.4.1 Road Surface

```
1.  DO $$
2.  DECLARE
3.      f record;
4.  BEGIN
5.      FOR f in SELECT geom
6.          FROM voxelmpt WHERE classID=56 AND ifcID=27 AND geom IS NOT NULL
7.      LOOP
8.          INSERT INTO voxelpatch(classID, ifcID, pa)
9.          VALUES (56, 27, PC_Patch(ARRAY(
10.            SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM (
11.                SELECT ST_X((ST_DumpPoints(f.geom)).geom) AS x,
12.                ST_Y((ST_DumpPoints(f.geom)).geom) AS y,
13.                ST_Z((ST_DumpPoints(f.geom)).geom) AS z
14.                ) AS g
15.             )));
16.      END LOOP;
17. END;
18. $$
19.
```

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

6.3.4.2 Greenarea Surface

```
1.  DO $$
2.  DECLARE
3.      f record;
4.  BEGIN
5.      FOR f in SELECT geom
6.          FROM voxelmpt WHERE classID=56 AND ifcID=28 AND geom IS NOT NULL
7.      LOOP
8.          INSERT INTO voxelpatch(classID, ifcID, pa)
9.          VALUES (56, 28, PC_Patch(ARRAY(
10.            SELECT PC_MakePoint(1, ARRAY[x,y,z]) as pt FROM (
11.                SELECT ST_X((ST_DumpPoints(f.geom)).geom) AS x,
12.                ST_Y((ST_DumpPoints(f.geom)).geom) AS y,
13.                ST_Z((ST_DumpPoints(f.geom)).geom) AS z
14.                ) AS g
15.             )));
16.      END LOOP;
17. END;
18. $$
```

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

6.3.4.3 Building Surface

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

```
11.                SELECT ST_X((ST_DumpPoints(f.geom)).geom) AS x,
12.                ST_Y((ST_DumpPoints(f.geom)).geom) AS y,
13.                ST_Z((ST_DumpPoints(f.geom)).geom) AS z
14.                ) AS g
15.            )));
16.     END LOOP;
17. END;
18. $$
19.
```

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

6.4 Build Index

```
1.  DROP INDEX IF EXISTS idx_voxelpatch CASCADE;
2.  DROP INDEX IF EXISTS geom_voxelpatch CASCADE;
3.  CREATE INDEX idx_voxelpatch ON voxelpatch(classID, ifcID);
4.  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]	0101000000000100000056AD6900031941010078DAECC5CB55D5501400D...
2	2	2	[null]	0101000000001000000193C2200038D6A000078DAECC5C791D45014004...
3	3	3	[null]	
4	4	4	[null]	
5	5	5	[null]	

Figure 6-1. Sample of "voxelpatch" table.