3D modelling Assignment 2

Enriching the 3D BAG with new attributes

GEO1004

Yitong Xia 5445825
Yue Yang 5516862
Fengyan Zhang 5462150

03 20, 2022

GEO1004

1 Methodology

On the whole, we use cjio (develop branch) as an auxiliary tool to clean up and triangulate files. And we mainly use two inputs and outputs to calculate volume, floor, area and orientation respectively. The main process is as follows.

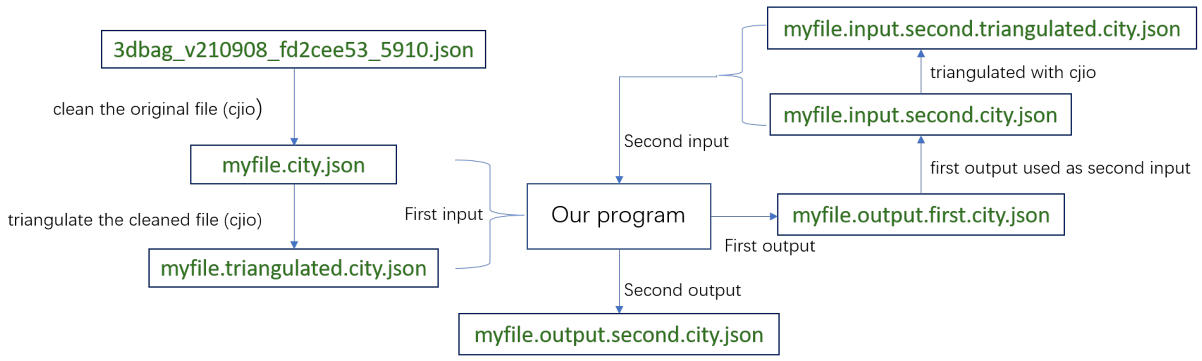

3dbag_v210908_fd2cee53_5910.json
clean the original file (cjio)
myfile.city.json
triangulate the cleaned file (cjio)
myfile.triangulated.city.json
First input
Our program
Second input
myfile.input.second.triangulated.city.json
triangulated with cjio
myfile.input.second.city.json
first output used as second input
myfile.output.first.city.json
First output
Second output
myfile.output.second.city.json

Figure 1: Overview Methodology

The reason why we have two inputs is that we use triangles to calculate the area of each roof surface and select the corresponding normal vector to estimate the orientations (which will be described in the corresponding chapters later). This requires us to identify which triangles (in the triangulated file) belong to which roof surface in the original file, therefore two inputs are needed.

In the first input, we use *myfile.city.json* file to calculate the floor and its corresponding triangulated file (*myfile.triangulated.city.json*) to calculate the volume. At the same time, we establish a unique Semantic Object for each face belonging to *RoofSurface*. We then write the calculation results and the new semantic objects into the first output file.

In the input of the second step, we use the file *myfile.input.second.city.json* and its corresponding triangulated file *myfile.input.second.triangulated.city.json* to calculate the area and orientation, and write the calculation results into the output file of the second input.

After having the result file in the second output, cjval and val3dity are used for verification. We need to process the invalid buildings according to the error report (can be downloaded from val3dity). The relevant process is as follows(see Fig. 2).

After error processing, the final outcome file (*myfile.result.city.json*) can be obtained. More descriptions are in the corresponding chapter.

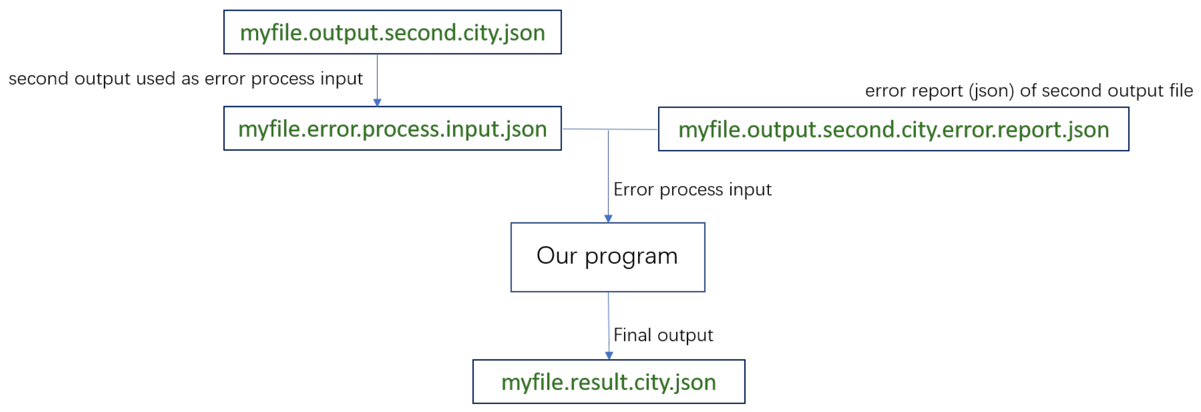

myfile.output.second.city.json
second output used as error process input
error report (json) of second output file
myfile.error.process.input.json
myfile.output.second.city.error.report.json
Error process input
Our program
Final output
myfile.result.city.json

Figure 2: Error Process

2 Volume

Because the shape of buildings is mostly irregular, the initial idea is to decompose each building into tetrahedrons, and then calculate the volume of each tetrahedron to obtain the volume of the whole building. However, in practice, we found that for buildings with arbitrary shape and irregularity, it is difficult to select a random point outside and then form tetrahedrons with the vertices of the building. The artificially specified external point forms a tetrahedron with every three points in the vertex sequence of each building, but this is not universal, different building structures will affect the order of vertices when constructing tetrahedrons(see Fig. 3). Later, we found that in the actual calculation, manually dividing

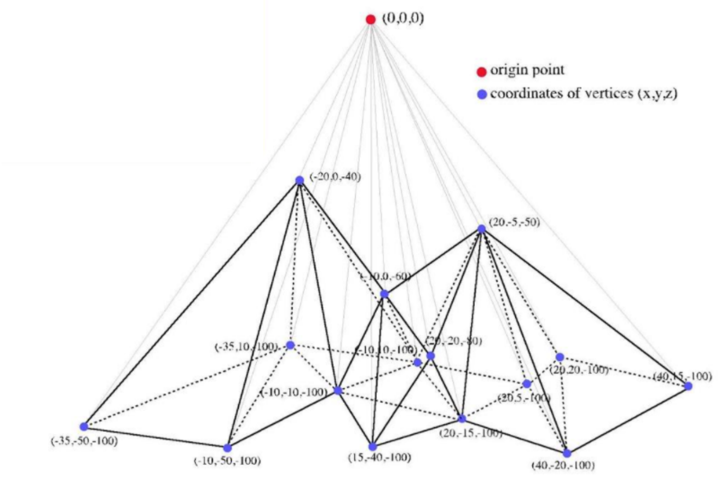


Figure 3: Construct Tetrahedrons(From Tetrahedral Shoelace Method)

the building into tetrahedrons is not necessary. When the vertex order of all faces is the same when observed from a reference point (all clockwise or counterclockwise), then the volume can be calculated by using the following formula when traversing each triangulated face:

$$V = \frac{1}{6} \times \left| det \begin{bmatrix} v1 \\ v2 \\ v3 \end{bmatrix} \right| \tag{1}$$

The algorithm goes as follows:

Algorithm 1 algorithm for volume

```
for each Building Part do
    $V \leftarrow 0$
    for each triangle IN each Surface do
        $v1 \leftarrow triangle[0]$        ▷ $v1$, $v2$, $v3$ indicates the three vertices of the triangle
        $v2 \leftarrow triangle[1]$
        $v3 \leftarrow triangle[2]$
        $V \mathrel{+}= (1/6) \times \left| det \begin{bmatrix} v1 & v2 & v3 \end{bmatrix}^{\mathrm{T}} \right|$
    end for
end for
```

For invalid *BuildingPart*, volume is set to *null*.

3 Floor

When calculating the floor, we only calculated those CityObjects whose type is *Building*. To calculate the height of the roof, we read "h_dak_max" and "h_dak_min" and "h_maaiveld" of each building from the attributes and apply the formula below:

$$Floor = \frac{((\text{“}h\_dak\_max\text{”} - \text{“}h\_dak\_min\text{”}) \times 0.7) + (\text{“}h\_dak\_min\text{”} - \text{“}h\_maaiveld\text{”})}{3.0} \tag{2}$$

where 3.0 is the typical height of ceilings, we need to round the value to the nearest integer after the calculation.

We think the step of converting the result of the calculation to an integer is worth noting; if we convert the data type with int(), we will not get the correct result because it simply removes the decimal part. If the decimal part is greater than 0.5, we round the value to the next integer; if the decimal part is less than 0.5, we round the value to the current integer.

In the result file, the value of floor is set to *null* in three buildings without geometry.

4 Area

For calculating the area of the roof surfaces, surface triangulation method is selected to complete the area calculation, as it is easier and more accurate to calculate the area of a triangle (using Heron's formula) than that of a polygon. At the same time, triangulation can well solve the problem of holes inside the surfaces. We tested the cjio triangulation using a roof surface with a hole and proved that triangulation can avoid the holes well in the triangulation process(see Fig. 4).

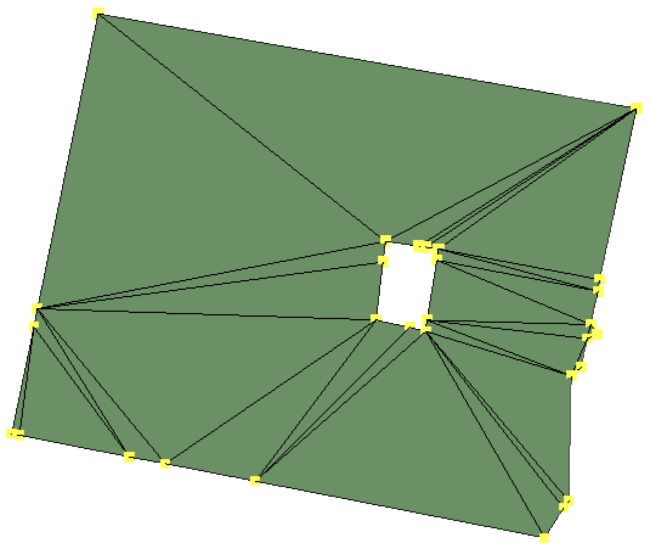


Figure 4: The Triangulation of the Roof Surface Can Prevent the "Hole" Well

Before calculating the area, creating new and unique semantic objects for surfaces which belong to *RoofSurface* is needed. In this way, each roof surface has a corresponding semantic object. At the same time, updating the index value of the corresponding new semantic surface in the *value* array. After all the new semantic surfaces are created, export the current json file and triangulate it using cjio. The following command for triangulation is used:

```
cjio myfile.city.json triangulate save myfile.triangulated.city.json
```

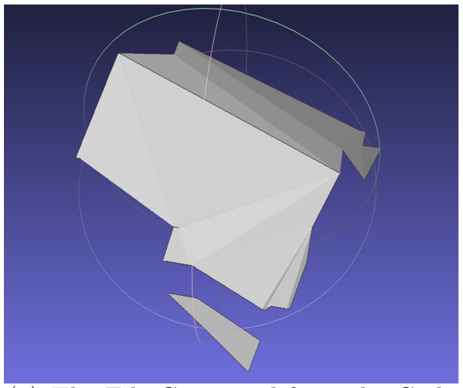
Then two files can be built, one is *myfile.output.first.city.json* , another one is the corresponding triangulated json file. Now we start iterating each *geometry* whose type is *Solid* in the triangulated json file, calculate the area of each triangle whose type is *RoofSurface*, and accumulate the areas to the corresponding roof surface. After finishing calculating the area of all triangles in one geometry, we assign the summed values to the corresponding *RoofSurface* in the *myfile.output.first.city.json* file.

Overall, we have completed the area calculation with the aid of the triangulation.

Another method has been tested as well, a brief explanation is to first calculate the projected area of each roofsurface in the XY plane (pay attention to subtracting the inner ring area), and then convert the plane area into the area of this face according to the angle between this face and the XY plane.

Results Analysis

To validate our results, the area of surfaces in the file *myfile.city.json* is checked through MeshLab. It is found that our results are close to area computations in the software, but there are still some differences. Taking NL.IMBAG.Pand.0503100000004247 that contains three roof surfaces as an example, for each surface, our results are 44.53, 2.33, 42.63, while the result given by MeshLab are around 44.48, 2.38, 42.70 respectively. It may be because the operation of geometric calculation, depending on the algorithm, inevitably leads to different degrees of loss. In addition, we reconstruct the roof surfaces using an obj file generated with the coordinates output from the code and check it in the software, in comparison with the obj file exported from *myfile.city.json* using cjio. We find the latter surfaces are smoother, which may also prove the loss of numerical accuracy in the calculation.


(a) The File Generated from the Code

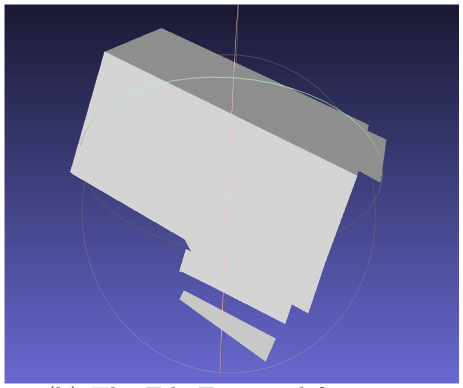

(b) The File Exported from cjio

Figure 5: Comparison of Surfaces

5 Orientation

Each *RoofSurface* will have its own normal vector. By projecting this normal vector onto the XY plane, a 2D vector or a point (when the *RoofSurface* is horizontal) can be obtained. The positive direction of the y-axis is set to the north direction, and the direction of the 2D vector can be determined by the sign of the XY coordinate and the included angle between the 2D vector and the y-axis (see eq. (3) and Algorithm. 2).

$$\alpha = \arctan(x/y) \tag{3}$$

Before applying this formula, it is necessary to judge whether the *RoofSurface* is horizontal or not by comparing the length of the normal vector projected onto the XY plane with a manually set threshold. Please note that the value of this threshold can be changed, after randomly selecting buildings for testing, we set the threshold to 1. The process of estimating orientations is as follows. The direction of the *RoofSurface* to which the normal

Algorithm 2 algorithm for orientation

```
if $length < horizontal\_epsilon$ then
    $orientation \leftarrow$ "horizontal"
else
    if $|y| = 0$ then
        if $x > 0$ then
            $orientation \leftarrow$ "EN"
        else
            if $|x| = 0$ then
                $orientation \leftarrow$ "horizontal"
            else
                $orientation \leftarrow$ "WN"
            end if
        end if
    else                                                        ▷ $y \neq 0$
        $quadrant \leftarrow 1, 2, 3, 4$    ▷ four quadrants assigned by the sign of xy coordinates
        $\alpha = \arctan(x/y)$
        $orientation \leftarrow result$    ▷ use $\alpha$ and quadrant to decide the orientation value
    end if
end if
```

vector belongs can be determined by the quadrant where the normal vector is projected and the included angle with the y-axis (which can be positive or negative, see Fig. 6).

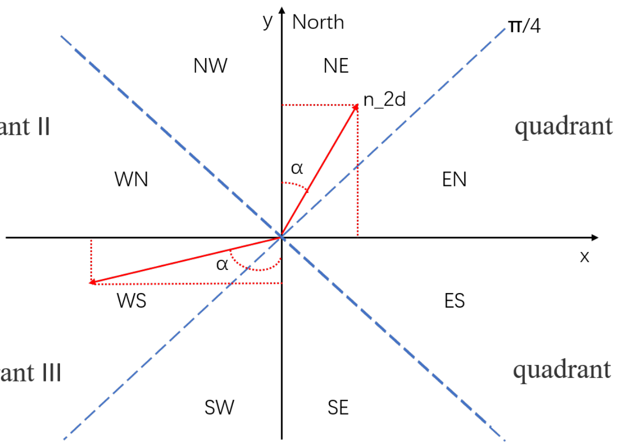
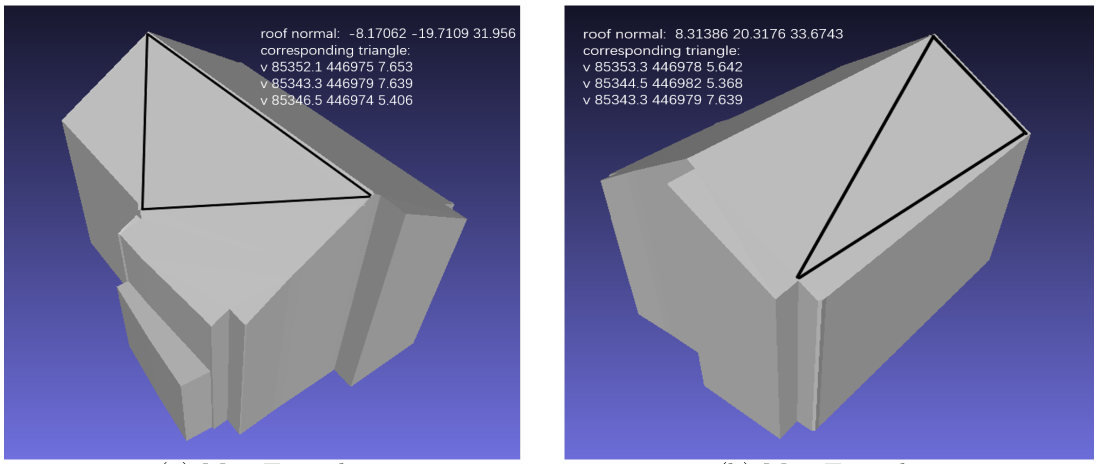


Figure 6: Orientation

When calculating the normal vector, we need to ensure that the direction of the normal vector is correct, so the triangulated file of the second input is used. At the same time, due to the existence of conservative points, for each triangle constituting each face, three vertices may be collinear (for instance, two of the three vertices are repeated). In order to ensure that the normal vector can be calculated correctly, the three vertices of the triangle with the largest area are selected(see Fig. 7).



```
roof normal:  -8.17062 -19.7109 31.956
corresponding triangle:
v 85352.1 446975 7.653
v 85343.3 446979 7.639
v 85346.5 446974 5.406
```

(a) Max Triangle-1

```
roof normal:  8.31386 20.3176 33.6743
corresponding triangle:
v 85353.3 446978 5.642
v 85344.5 446982 5.368
v 85343.3 446979 7.639
```

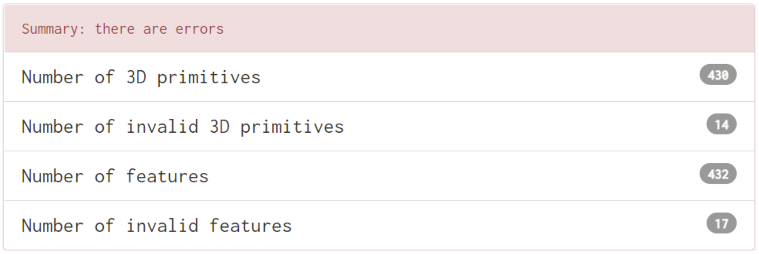
(b) Max Triangle-2

Figure 7: Vertices of Max Triangles

6 Invalid Cases

Invalid Geometry

In the input file *myfile.city.json*, we use val3dity to check its geometry and find that there are some invalid buildings, as shown below(see Fig. 8). We try to export the specific invalid building as an OBJ file and then open it in MeshLab to observe the invalid part. The main categories and examples are as follows.



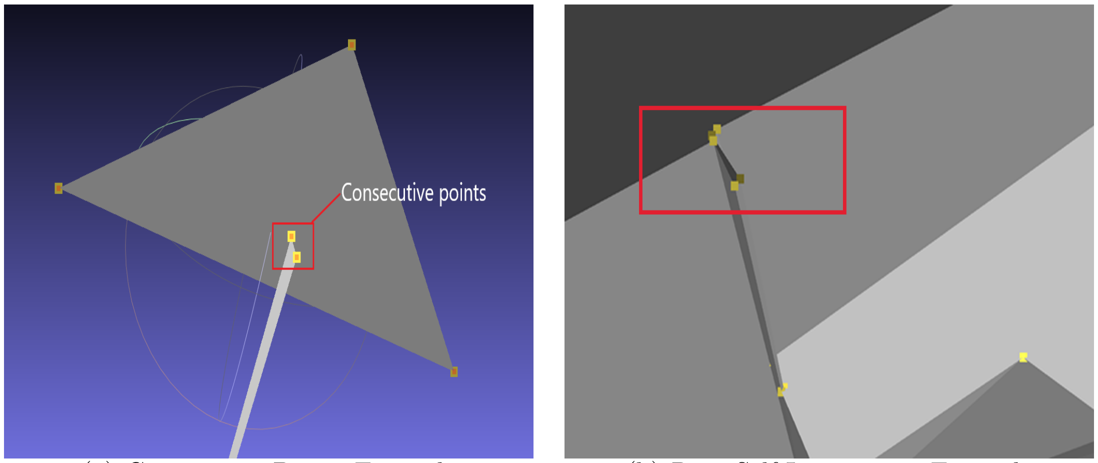
Summary: there are errors

Number of 3D primitives	430
Number of invalid 3D primitives	14
Number of features	432
Number of invalid features	17

Figure 8: Error Summary in myfile.city.json

ConsecutivePointsSame(details)

In short, there are consecutive vertices with very similar positions, or two repeated vertices with the same coordinates in the file(see Fig. 9(a)).



(a) Consecutive Points Example (b) Ring Self Intersection Example

Figure 9: Examples

RingSelfIntersection(details)

There are self intersecting rings or, for instance, rings that are (partially) collapsed to a line in the file (see Fig. 9(b)).

NonPlanarPolygonNormalsDeviation(details)

The deviation normals of the following two files exceeds the limit tolerance (see Fig. 10)

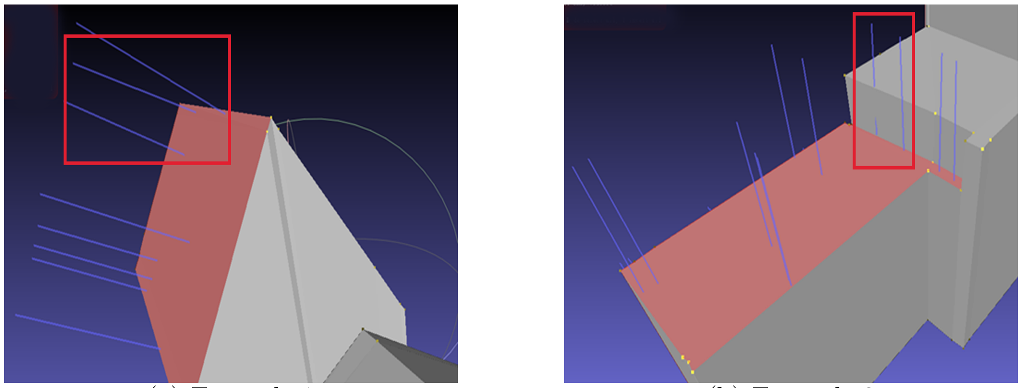
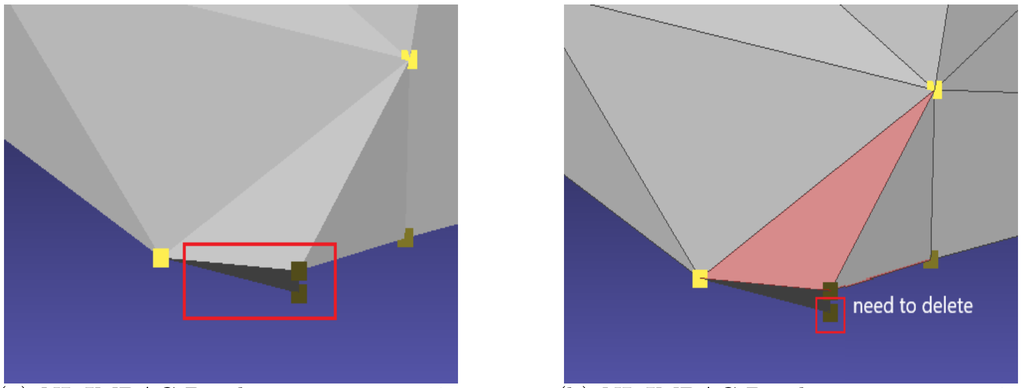

(a) Example-1 (b) Example-2

Figure 10: Non Planar Polygon Normals Deviation

ShellNotClosed(details)

The shell in the building part contains "holes"(see Fig. 11).

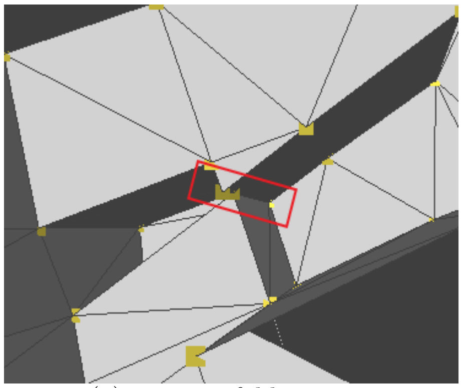


(a) NL.IMBAG.Pand.0503100000020272 (b) NL.IMBAG.Pand.0503100000020272

Figure 11: Shell Not Closed

NonManifoldCase(details)

This error might be returned for shells having an edge shared by more than two faces, also in cases where surfaces of shells are inconsistently oriented (see Fig. 12).


(a) nonmanifold.31387-1

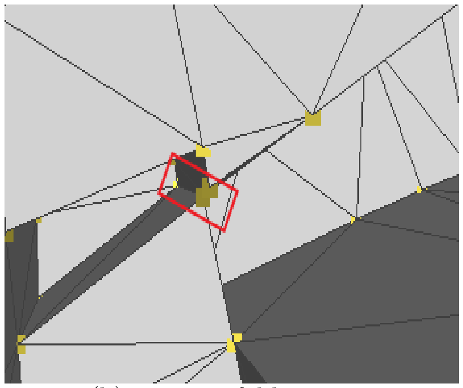
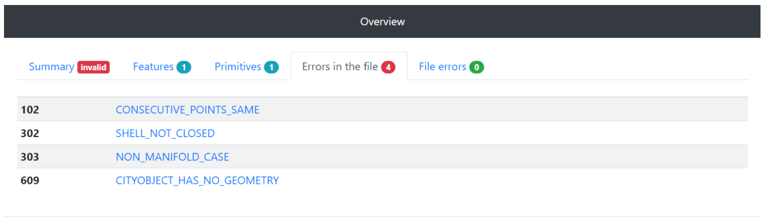

(b) nonmanifold.31387-1

Figure 12: Non Manifold Case

Invalid Building Impact

We also check the geometry in the file *myfile.triangulated.city.json* using val3dity. It is found that there are still some invalid buildings, as shown below(see Fig. 13). After triangulation, the two buildings which were reported as the error "NON PLANAR POLYGON NORMALS DEVIATION" are considered valid, the building that introduced "RING SELF INTERSECTION" is still considered invalid and returns the error "NON MANIFOLD CASE", which might also account for the reason why the area of the roof surface ("roof id": "NL.IMBAG.Pand.0503100000033695-0.roof.2") is null.


Overview

Summary invalid | Features 1 | Primitives 1 | Errors in the file 4 | File errors 0

102	CONSECUTIVE_POINTS_SAME
302	SHELL_NOT_CLOSED
303	NON_MANIFOLD_CASE
609	CITYOBJECT_HAS_NO_GEOMETRY

Figure 13: Overview of Validation in myfile.triangulated.city.json

The impact of the six errors not eliminated in the original document on the four attributes is evaluated as follows

error type	Volume	Floor	Area	Orientation
Consecutive Points Same	-	-	-	+
Ring Self Intersection	+	-	+	+
Non Planar Polygon Normals Deviation	-	-	-	+
Shell Not Closed	+	-	-	-
Non Manifold Case	-	-	-	-
CityObject Has No Geometry	+	+	+	+

The + sign represents a great influence (which may lead to failure of calculation or great difference in calculation accuracy), and the $-$ sign represents a small influence.

The specific impact needs further analysis in the future.

7 Workload

Yitong did the floor calculation, Yue Yang did the area calculation, Fengyan did the volume and orientation calculation.

For area calculation, we tried multiple methods and worked on the area part together.

We write the report together.

8 Appendix

The relevant code is in this git repository.

In our result file, $floor$ and $volume$ are written in Building $->$ attributes. In semantics $->$ surfaces, each $RoofSurface$ contains two area attributes:

$area:$ area calculated using projection method.

$area\_tri:$ area calculated using triangulation method.

Both area values are kept because we want to compare and analyze the accuracy of the two different methods.