TU Delft Department of Management in the Built Environment
BK Bouwkunde

3D modeling

Assignment 02

Enriching the 3D BAG with building metrics

Name:	**Student Number:**
Qiuxian Wei	5801737
Longxiang Xu	5722918
Bingshiuan Tsai	5511461

TU Delft

1. Volume

For volume, the idea is to calculate the sum of signed volume formed of all faces F and with a fixed point P0t. For implementation, the faces F are triangles obtained from the constrained Delaunay triangulation of CGAL. To avoid ambiguity, the fixed point P0 is set to be the first point stored in the “boundaries” attribute. The pseudocode is as follows:

```
for each building/cityobject :
        select the first point P0 stored in the boundaries
    for each face:
        create triangulation TRS:
        calculate the orientation of the first triangle TRS[0] and P0
        for each triangle:
            calculate area
            calculate distance to P0
            volume += orientation*area*distance/3
            area += area
            save area and volume to json (area will be useful for future
calculation and in the end will be deleted)
```

2. Rectangularity

For calculating the rectangularity, only the vertices of the exterior shell of each building object are extracted and pushed back to a vector named “exterior_pts”. The vector of exterior shell points is then used as an input for the CGAL::oriented_bounding_box function to compute the object oriented bounding box (OOBB). The output of the OOBB function is an eight points array representing the eight vertices of the OOBB, and its length, width and height can be derived by the subtraction of different vertices (see figure 1 for the order). Finally, the volume can be computed by the product of the length, width and height of OOBB and the rectangularity and be calculated by the fraction of the building volume obtained from part 1 and the volume of OOBB. The pseudocode is as follows, and the function is shown as the *calculate_rectangularity* function in geomtools.cpp file.

```
for each building/cityobject :
        extract the vertices of the exterior shell to create a vector of points
    for each exterior points vector:
        calculate the object oriented bounding box (OOBB)
        OOBB length = OOBB_vertex[1] - OOBB_vertex[0]
        OOBB width  = OOBB_vertex[2] - OOBB_vertex[1]
        OOBB height = OOBB_vertex[7] - OOBB_vertex[2]
        OOBB volume = length * width * height
        rectangularity = building/cityobject volume / OOBB volume
```

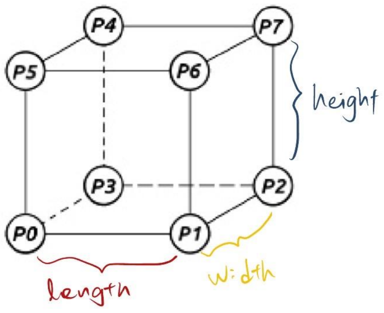

Figure 1 the order of the vertices of the derived OOBB

reference: https://doc.cgal.org/latest/BGL/group__PkgBGLHelperFct.html#gad9df350e98780f0c213046d8a257358e

3. Hemisphericality

For computing the hemisphericality, the given formula from the instruction of the assignment is used (see equation below). The $Vol(E)$ is derived from the first item of the *calculate_volume_area* function output, and the $Area(E)$ is derived from the second item of *calculate_volume_area* function. The volume and area calculated in step 1 are used as the inputs of the *hemisphericality* function in geomtools.cpp file.

$$HEM(E) = \frac{3\sqrt{2\pi}\,Vol(E)}{Area(E)^{3/2}}$$

4. Roughness index

It is a little bit tricky to calculate the average distance of points sampled on E to the center of E. To calculate the center of the building, we decided to calculate the average of the vertices in each triangle of each face. Since the averaged center of a triangle can be used as a representation of the surface of the building and the combination of these centers can fully represent the center of the building. Also, since the areas of the triangles vary, the weight (each area of the triangle/sum of the area) is used to determine the center point. The bigger the triangle is, the more influence it has on the building center.
Besides, the distance of the random points in each triangle to the center point is affected by the weight too. The triangle areas calculated in the former were returned with area and volume as a pair to reduce the calculation time.

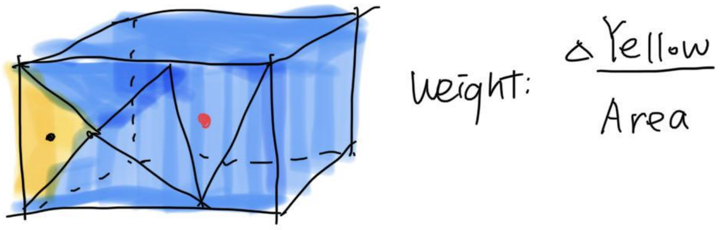

Figure 2 The weights for central points of a triangle to the center of the building

The pseudocode is as follows:

```
To calculate the weights of each triangle in each face:
```

```
for each face:
        for each triangle in each face:
        weight = triangle_area / sum_area;
        building center += each triangle_center *weight
Choose two random points in each triangle
average distance += avg_distance of two points to center * weight
```

Then just follow the formula below. The $Vol(E)$ is derived from the first item of the *calculate_volume_area* function output, and the $Area(E)$ is derived from the second item of *calculate_volume_area* function and is represented as a vector of area of each triangle of each face. The volume and area calculated in step 1 are used as the inputs of the *Roughness_index* function in geomtools.cpp file.

$$RI(E) = \frac{\mu_E^3}{Vol(E) + Area(E)^{3/2}} \times 48.735$$

5. Orientation of the "RoofSurface" surfaces

The 9 possible orientations are first assigned a number and added to the “semantic” attribute; the order of the numbers in the value list is a reference for the surface attribute, and the number of the order will be replaced by the newly added orientation attribute number if it refers to the “RoofSurface”. For addressing the orientation of each"RoofSurface"of each Building, the triangulation of each building face is used as the first triangle of the face is extracted to compute its normal vector. Since the vertices of the triangle are in Counterclockwise (CCW) order from the external view of an building, the normal vector generated by the cross product of vector 1(P2 - P1) and vector 2 (P3 - P1) is guaranteed to be pointing outward, which its x, y coordinates and tangent ($\frac{y}{x}$) can then be used to determine the orientation of the whole face (see figure 3).
The challenging part is that since some of the vertices of a face might contain duplicate points or other erroneous geometry that results in the failure of triangulation, e.g. the buildingPart with the key value “NL.IMBAG.Pand.0503100000033695-0”, which there is no valid triangle accessible for computing the normal vector and it introduces a memory leak issue while compiling. In this case, the *get_best_fitted_plane* function is used to derive the best fitted plane representing the problematic face, and the coefficients of the plane equation obtained from the function are the normal vector of the plane (see figure 4), which then can be used for determining orientation. The result is shown as the *roof_orientation* function in geomtools.cpp file. The pseudocode is as follows:

```
for each building/cityobject :
    add all 9 possible orientation back to the “surface”
    for each face:
        create triangulation TRS:
    if face is RoofSurface and the triangulation exits:
        extract the first triangle, and compute its normal vector
        V1 = triangle_pt[1] -triangle_pt[0]
        V1 = triangle_pt[2] -triangle_pt[0]
        normal_vector = cross_product(V1, V2)
        x = normal_vector.x()
        y = normal_vector.y()
        tan = y / x
        use x, y & tan to determine the orientation
        replace the number at the corresponding order in the “value” list
    else
        extract the vertices of the face, and compute its best_fitted plane
        x = best_fitted_plane.a()
        y = best_fitted_plane.b()
        tan = y / x
        use x, y & tan to determine the orientation
        replace the number at the corresponding order in the “value” list
```

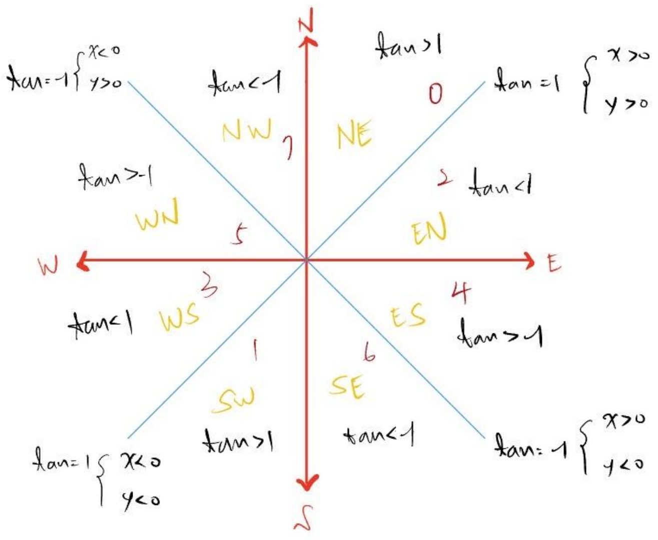

Figure 3 illustration for the orientation determination

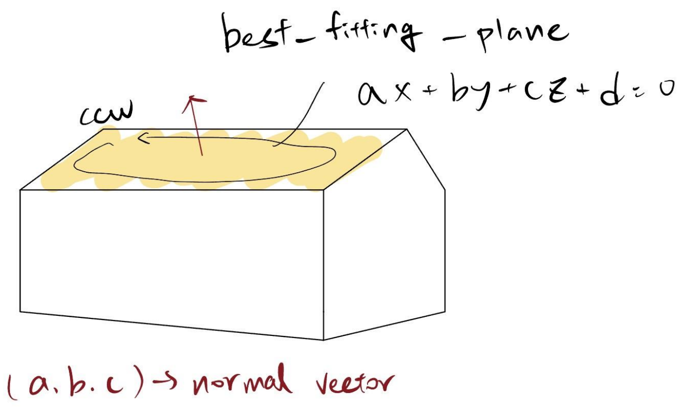

Figure 4 Normal vector computation of the faces without valid triangulation

6. Workload:

- Volume and area & code integration: Longxiang Xu
- Hemisphericality & Roughness index: Qiuxian Wei
- Rectangularity & Orientation: Bingshiuan Tsai