3D Validation of CityGML

Hugo Ledoux

TU Delft

GIS technology group

10 March 2009
- Information model for exchange and storage of 3D city models
- Based on GML (thus XML-based)
- Now an international standard (acc. to OGC)
- Contains 5 different levels of detail (LOD)
CityGML

- Information model for exchange and storage of 3D city models
- Based on GML (thus XML-based)
- Now an international standard (acc. to OGC)
- Contains 5 different levels of detail (LOD)
CityGML

- Information model for exchange and storage of 3D city models
- Based on GML (thus XML-based)
- Now an international standard (acc. to OGC)
- Contains 5 different levels of detail (LOD)
Validation = *geometric* validation

- Validating a spatial object means ensuring that it respects certain specifications.
- Specifications are standardised, such as by the ISO or the OGC.
- ≠ XML schema validation (*.xsd)

For Polygons:
1. closed boundary
2. no self-intersection
3. ring does not intersect boundary
4. etc
<Building gml:id="Build0815">
  <function>1000</function>
  <yearOfConstruction>1985</yearOfConstruction>
  <roofType>1030</roofType>
  <measuredHeight uom="#m">8.0</measuredHeight>
  <storeysAboveGround>2</storeysAboveGround>
  <storeyHeights uom="#m">2.5 2.5</storeyHeights>
  ...
  ...
</Building>
Validation = *geometric* validation

- Validating a spatial object means ensuring that it respects certain specifications
- Specifications are standardised, such as by the ISO or the OGC
- ≠ XML schema validation (*.xsd)

**For Polygons:**
1. closed boundary
2. no self-intersection
3. ring does not intersect boundary
4. etc

![Diagram showing validation examples]

- VALID
- NON-VALID
- NON-VALID
Why do we want to validate spatial objects?

- Non-valid spatial objects are fine if you only want to *look* at them!
- Validation is necessary to guarantee output of *processing* or *manipulation* operations:
  - calculation of area of a polygon
  - creation of a buffer
  - boolean operations such as intersection, touch, contain, etc.
  - conversion to other formats
- For 2D objects, functions implemented and used (e.g. GEOS, JTS)
- Very complex task, even in 2D
Validation of polygons = many cases
<gml:Solid>
  <gml:exterior>
    <gml:CompositeSurface>
      <gml:surfaceMember>
        <!--top surface-->
        <gml:Polygon gml:id="a">
          <gml:exterior>
            <gml:LinearRing>
              <gml:pos>0 0 1</gml:pos>
              <gml:pos>1 0 1</gml:pos>
              <gml:pos>1 1 1</gml:pos>
              <gml:pos>0 1 1</gml:pos>
              <gml:pos>0 0 1</gml:pos>
            </gml:LinearRing>
          </gml:exterior>
        </gml:Polygon>
      </gml:surfaceMember>
      ...  
      ...  
    </gml:CompositeSurface>
  </gml:exterior>
</gml:Solid>
Examples of solids

INVALID

VALID

VALID
Example of a CityGML solid
Validation with constrained Delaunay tetrahedralization

3 buildings

DT of the vertices of the buildings

Constrained DT

Constrained Delaunay triangulation (in 2D)
Validation with constrained Delaunay tetrahedralization

Constrained Delaunay tetrahedralization (in 3D)
Main ideas for geometric validation

- Using volumetric methods, since we’re dealing with volumes
- Create CDT for the objects, and then “eat away” tetrahedralization
- Permits us to easily detect connectedness problems, non-watertight situations, etc.
Main ideas for geometric validation

- Using volumetric methods, since we’re dealing with volumes
- Create CDT for the objects, and then “eat away” tetrahedralization
- Permits us to easily detect connectedness problems, non-watertight situations, etc.
3D model of the TU Delft campus

Simplest method to create 3D city models: Extrusion
Topological problems with extrusion

Touching buildings create problems

Diagram showing buildings with dimensions A=15m, B=12m, C=8m, D=9m, and E=12m.
Topological problems with extrusion

Touching buildings create problems
Our approach to extrusion

- Since no “cleaning tools” in 3D yet, we used 2D tools
- Creation of topologically consistent file for the buildings
  - Shapefile → FME + Oracle Topology
- Iteration over the edges to construct vertical planes
Tetrahedralized model of TU Delft

Validation of our approach
That's all, thanks for your attention!
Other Examples
Ambiguities in ISO/GML
What do we have to check?

1. **closedness of the rings of every surface:** easy, just check if 1st and last point given are the same. Done outside tetgen, as a pre-processing.

2. **distinct vertex:** easy to check, and must be done outside tetgen as pre-processing.

3. **orientation of points within a surface:** to make sure that the outer and the inner ring(s) have opposite orientation (CCW and CW). This must be done outside tetgen as a pre-process, and involves finding a single point that defines a hole for the inner rings.

4. **planarity of surfaces:** this is checked when tetgen makes a surface mesh, it’s a requirement of tetgen. Done pre-processing right now.

5. **non-self intersection of surfaces:** this is checked with tetgen surface mesh also.
What do we have to check?

6. **non-overlapping inner rings on a surface.** That would be checked automatically by tetgen also when meshing each surface. [again, that might not be necessary for same reason as rule #3]

7. **closedness of the solid:** Built-in tetgen.

8. **Inner shells are a volume** This is to make sure that one inner shell is not a line segment for instance. That wouldn’t be a problem for tetgen, so we have to take extra care for that one... I guess that would be done when checking the closedness in rule #7.

9. **connectedness of the volume of the solid:** rather easy to perform, once we have the CDT. Start at a point inside a IN tetra, and breath-first search, counting tetra visited on the way. Constraints obviously stops you. Then compare number of visited tetra with total number of IN tetra.
What do we have to check?

10. **inner-outer check of shells**, i.e. make sure that inner shells do not intersect outer shells. The triangle-triangle intersection series of tests will test that.

11. **orientation of surfaces**: probably the last step of the algorithm. Once we have the CDT with all tetra flagged as IN or OUT, we can then make sure that for each face, the tetra on each side are all consistent. Two cases: the point given to define the interior was based on a wrongly orientated faces; point given was based on a correctly orientated face.