Validation and automatic repair of two- and three-dimensional GIS datasets

M. Meijers  H. Ledoux  K. Arroyo Ohori  J. Zhao

OSGeo.nl dag 2013, Delft
2013–11–13
Typical error: polygon is self-intersecting
Typical error: polygon is self-intersecting
Typical error: BGT has gaps and/or overlaps
Typical error: BGT has gaps and/or overlaps
Typical error: BGT has gaps and/or overlaps
Big and complex datasets: it quickly gets out-of-control
Big and complex datasets: it quickly gets out-of-control
Big and complex datasets: it quickly gets out-of-control
Big and complex datasets: it quickly gets out-of-control
Errors in CityGML datasets
Errors in CityGML datasets

Wrong orientation of faces
Errors in CityGML datasets

Dangling face
Errors in CityGML datasets

With my colleague John Zhao, we're making an overview of the most common errors/problems, such as:

1. Automatic repairing of broken 3D city models
Errors in CityGML datasets

With my colleague John Zhao, we're making an overview of the most common errors/problems, such as:

1. Automatic repairing of broken 3D city
How do we deal with these problems?

We have solved our own problems by developing 3 prototypes:

- **prepair**: automatic repair of single polygons
- **pprepair**: automatic repair of planar partitions
- **val3dity**: validation and “simple repair” of 3D objects
repair of single polygons
Validation of a polygon = a solved problem

OGC Simple Features and ISO19107 rules:

1. no self-intersection
2. closed boundaries
3. rings can touch but not overlap
4. no duplicate points
5. no dangling edges
6. connected interior
7. etc
If it’s broken then repair it. But how?

Errors are highlighted, but not repaired. One has to manually fix them.
Our approach = constrained triangulation (CT)

Repairing = 3 simple steps:

1. CT of input polygon
2. Labelling of triangles (outside or inside)
3. Reconstruction of the rings by depth-first search on the dual graph
Our approach = constrained triangulation (CT)

Repairing = 3 simple steps:
1. CT of input polygon
2. labelling of triangles (*outside* or *inside*)
3. reconstruction of the rings by depth-first search on the dual graph
Repairing = 3 simple steps:

1. CT of input polygon
2. Labelling of triangles (*outside* or *inside*)
3. Reconstruction of the rings by depth-first search on the dual graph
Our approach = constrained triangulation (CT)

Repairing = 3 simple steps:

1. CT of input polygon
2. Labelling of triangles (*outside* or *inside*)
3. Reconstruction of the rings by depth-first search on the dual graph
Our approach = constrained triangulation (CT)

Repairing = 3 simple steps:

1. CT of input polygon
2. Labelling of triangles (*outside* or *inside*)
3. Reconstruction of the rings by depth-first search on the dual graph
Our approach = constrained triangulation (CT)

Repairing = 3 simple steps:

1. CT of input polygon
2. Labelling of triangles (*outside* or *inside*)
3. Reconstruction of the rings by depth-first search on the dual graph
Faster than PostGIS by a factor 3–4 in practice
Faster than PostGIS by a factor 3–4 in practice
repair of planar partitions
One “common” repairing solution: snapping

- Tolerance (*threshold*) is used for *snapping* vertices
- Tolerance based on scale of datasets
- Works fine for *simple* problems
One “common” repairing solution: snapping

- Tolerance (threshold) is used for snapping vertices
- Tolerance based on scale of datasets
- Works fine for simple problems
One “common” repairing solution: snapping

- Tolerance (threshold) is used for *snapping* vertices
- Tolerance based on scale of datasets
- Works fine for *simple* problems
Snapping is error-prone and “dangerous”

Spikes and punctures can create invalid polygons
Snapping is error-prone and “dangerous”

Spikes and punctures can create invalid polygons
Snapping is error-prone and “dangerous”
Snapping is error-prone and “dangerous”

Splitting of polygons into several polygons
Our solution = constrained triangulation (CT)

1. Construct CT of input polygons
2. Label each triangle with label of its polygon
3. Problems = triangles with no label or > 1 labels
4. Repair gaps/overlaps locally by changing labels
Our solution = constrained triangulation (CT)

1. Construct CT of input polygons
2. Label each triangle with label of its polygon
3. Problems = triangles with no label or > 1 labels
4. Repair gaps/overlaps *locally* by changing labels
Our solution = constrained triangulation (CT)

1. Construct CT of input polygons
2. Label each triangle with label of its polygon
3. Problems = triangles with no label or > 1 labels
4. Repair gaps/overlaps *locally* by changing labels
Our solution = constrained triangulation (CT)

1. Construct CT of input polygons
2. Label each triangle with label of its polygon
3. Problems = triangles with no label or > 1 labels
4. Repair gaps/overlaps \textit{locally} by changing labels
Our solution = constrained triangulation (CT)

1. Construct CT of input polygons
2. Label each triangle with label of its polygon
3. Problems = triangles with no label or > 1 labels
4. Repair gaps/overlaps *locally* by changing labels
Our solution = constrained triangulation (CT)

1. Construct CT of input polygons
2. Label each triangle with label of its polygon
3. Problems = triangles with no label or > 1 labels
4. Repair gaps/overlaps *locally* by changing labels
Our solution = constrained triangulation (CT)

1. Construct CT of input polygons
2. Label each triangle with label of its polygon
3. Problems = triangles with no label or > 1 labels
4. Repair gaps/overlaps *locally* by changing labels
Our solution = constrained triangulation (CT)

1. Construct CT of input polygons
2. Label each triangle with label of its polygon
3. Problems = triangles with no label or > 1 labels
4. Repair gaps/overlaps locally by changing labels
Local control with simple rules
Local control with simple rules
Local control with simple rules
Local control with simple rules
Local control with simple rules
Experiments with large real-world datasets

CORINE dataset: land-use of Europe
tiles of 100km X 100km
Experiments with large real-world dataset

(a) E41N27

(b) 4tiles

(c) 16tiles

(d) Mexico
Experiments with large real-world datasets

<table>
<thead>
<tr>
<th></th>
<th># polygons</th>
<th># pts</th>
<th>largest polygon</th>
<th># pts</th>
<th>avg # pts per polygon</th>
</tr>
</thead>
<tbody>
<tr>
<td>E41N27</td>
<td>14 969</td>
<td>496 303</td>
<td>26 740</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>4tiles</td>
<td>4 984</td>
<td>365 702</td>
<td>16 961</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>16tiles</td>
<td>63 868</td>
<td>6 622 133</td>
<td>95 112</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>26 866</td>
<td>4 181 354</td>
<td>117 736</td>
<td>156</td>
<td></td>
</tr>
</tbody>
</table>
## Comparison with other GIS packages

<table>
<thead>
<tr>
<th></th>
<th># pts</th>
<th>memory</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>E41N27</td>
<td>496</td>
<td>145 MB</td>
<td>19s</td>
</tr>
<tr>
<td>4tiles</td>
<td>365</td>
<td>116 MB</td>
<td>17s</td>
</tr>
<tr>
<td>16tiles</td>
<td>6 622</td>
<td>1.45 GB</td>
<td>4m47s</td>
</tr>
<tr>
<td>Mexico</td>
<td>4 181</td>
<td>1.01 GB</td>
<td>3m31s</td>
</tr>
</tbody>
</table>
val3dity

validation of 3D solids
ISO 19107 rules also in 3D

s1 invalid
s2 valid
s3 valid
s4 invalid

s5 invalid
s6 invalid
s7 valid
s8 invalid

s9 invalid
s10 invalid
s11 valid
s12 invalid
ISO 19107 rules also in 3D

1 distinct vertex
2 closedness of the rings of every surface
3 orientation of points within a surface (with inner rings)
4 planarity of surfaces
5 non-self intersection of surfaces
6 non-overlapping inner rings on a surface
7 orientation of normal vectors
8 “watertightness” of every shell
9 “connectedness” of the interior
10 how inner/outer shells interact with each others
11 ...
None of the (commercial) tools are ISO compliant
The implementation

- As compliant to ISO 19107 as possible
- Use of CGAL: robust and fast
- C++
- Be kind to the user
- Try to automatically repair invalid solids (work in progress)
Validation is performed hierarchically.

1. Planarity of surface
2. Combinatorial consistency
3. Geometric consistency
4. Orientation of normals

Interactions between the shells with Nef polyhedra and Boolean operations.

2D validation rules applied to primitives embedded in 3D space.
Automatic repair = our current work
Thanks for your attention

www.github.com/tudelft-gist/prepair
pprepair
val3dity