A FRAMEWORK FOR GENERALIZATION OF 3D CITY MODELS BASED ON CITYGML AND X3D

Bo Mao [a, *], Yifang Ban [a], Lars Harrie [b]

[a] Geoinformatics, KTH, Drottning Kristinas vag 30, SE-100 44 Stockholm, Sweden - (mao2, yifang)@infra.kth.se
[b] GIS-centre, Lund University, Solvegatan 12, SE-223 62 Lund, Sweden - Lars.Harrie@nateko.lu.se

Commission VI, WG II/2

KEY WORDS: CityGML, X3D, Generalization, CityTree, Visualization, Aggregation, City model, Multiple representation data structure

ABSTRACT:

In this paper, a novel framework for generalization of 3D city models based on CityGML and X3D is proposed. In the proposed framework, the CityGML files are first parsed into a Java environment to acquire the city model information using Citygml4j, an open source Java API. Generalization is then performed in the Java environment. The outcome of the generalisation is CityTree, a multiple representation data structure of the objects, based on which we can effectively realize the continual scaling and dramatically reduce the load time of 3D models. Finally, the X3D representation is generated from the CityTree Model and is displayed by Xj3D viewer. By utilising CityTree, the multiple representation data structure, it is possible to have dynamic zoom functionality in real time. The paper describes the general framework and an implementation using this framework on the aggregation of 3D building objects. The implementation confirms the applicability of the framework for generalization of 3D city models.

1. INTRODUCTION

With an increasing number of people living in or moving to cities, cities are growing and sprawling. Thus, development of effective 3D visualization tools is of critical importance for sustainable urban planning as well as effective communications to the general public. The fundamentals of these kinds of tools are the 3D city object models. In addition, it is important for the public to be able to view the 3D city scenes from the most commonly used browsers such as Microsoft IE or Mozilla Firefox, thus can tremendously increase the accessibility of the online city models.

In August 2008, OGC launched the specification CityGML (OGC, 2009) as a common information model for the representation of 3D urban objects. It defines the classes and relations for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantic and appearance properties. It is a trend to integrate the information about a city into CityGML which can be extended by users according to their application requirement (Plümer et al., 2005).

However, CityGML is made for geometric, topologic and semantic representation, but not for presentation and visualization. It is not a good idea to render the 3D scenes directly from the CityGML files. Furthermore, the CityGML models are often very detailed and should be simplified (generalized) to enable efficient and readable presentations. Our approach is hence to convert the CityGML data to a presentation format X3D (Web3D 2009) and during this converting process also simplify the model. The rest of the paper is structured as follows. Related works are given in section 2. Section 3 mainly introduces CityGML and X3D. Our framework is proposed in section 4. The generalization algorithm is specified in section 5 and experiment results are given in section 6. Section 7 summarizes whole paper.

2. RELATED WORKS

The problem of handling several level of details (LODs) in city models has been acknowledged for a long time (e.g. Köninger and Bartel, 1998) and the CityGML standard also specifies five level of details (OGC, 2009). To obtain this several level of details, which is required for efficient visualization, several methodologies have been proposed. Kolbe and Gröger (2003) proposed an MRDB-model for obtaining consistency between the different level of details where one object become a part of an object in the adjacent LoD. Other researchers have concentrated on establishing generalization methods to change the representation between different LoD (see Meng and Forberg, 2007, for an overview). Mayer (2005) and Forberg (2007) developed a scale-space technique for simplifying buildings, partly based on the morphological operators opening and closing. Kada (2006) used vertical half spaces to model the main outline of a building that were then used to simplify the building. Later, he extended the approach by also handling roof structures with using best fitting primitive roof types (Kada, 2007). Fan *et al*. (2009) proposed a methodology for efficient handling of 3D building modelled in CityGML LOD3 (corresponding to a detailed architecture model). Their research showed that good visualisation properties could be obtained by only using the exterior shell of the building model that

* Corresponding author. Bo Mao; mao2@infra.kth.se; phone 46 8 790-7676; fax 46 8 790-8580

drastically decreases the required number of polygons. However, few of these studies explicitly discuss the framework in which the generalisation process should be executed. This issue has been extensively discussed in 2D cartographic generalisation with a common research framework (Burghardt et al., *2003*). The framework proposed in this paper has similarities with earlier work by Lehto and Sarjakoski (2005), who performs generalisation by an XSLT process where they translated 2D data from GML to SVG.

3. CITYGML AND X3D

3.1 CityGML

CityGML is a common information model for the representation of 3D urban objects (OGC, 2009). It is realised as an open data model and XML-based format for the storage and exchange of virtual 3D city models.

CityGML defines five levels of detail (LOD), where objects become more detailed with increasing LOD. Meanwhile, the CityGML files can contain multiple representations for each object in different LOD simultaneously and show the generalized objects over different scales.

CityGML files are very large, often several GB for the big cities. Even though file sizes can be effectively reduced by compression method such as gzip (≈10%) the XML validation and processing can be a problem (classical DOM parsing are generally not feasible due to main memory limitations) and WFS access might have to be realized in an asynchronous way in order to avoid timeouts. Another problem with a cityGML model is the complexity. A city in itself is very complex and CityGML allow modelling of much of this complexity. This is of course nice for many applications, but it also stresses the need for efficient visualization techniques of CityGML.

3.2 X3D

X3D is an XML-based ISO standard for visualisation of 3D models in computers, the successor of Virtual Reality Modeling Language (VRML). X3D supports several pre-defined geometry objects such as box, cone, cylinder and so on, which can be used to represent CityGML models in X3D (X3D, 2009).

There are several toolkits available for X3D. Xj3D, for example, is an open source toolkit for X3D manipulation and visualization. Xj3D creates 3D scenes not only in Java application viewer but also in Java applet viewer embedded in web pages. It renders the 3D scenes with Java bindings for OpenGL (JOGL) and supports real time interaction with users. Xj3D supplies Scene Access Interface (SAI) to change or build X3D worlds. For details see web3D (2009).

In order to add a 3D window to an application, an X3D component should be created and added to the application. Today there are free X3D components available, e.g. the Java class X3DScene. Now we can dynamically build an X3D world by creating new X3DScene. Each X3DScene must pre-declare what Profile and X3D Components it will use. Profile is a shortcut mechanism to reduce the amount of text needed to be typed by the end user, and to also specify a set of known functionality. Once we have the Profile and Component instance, an X3DScene can be created accordingly and visualised in the application.

4. OUR FRAMEWORK

The basic framework of the proposed generalization structure is shown in Figure 1. The CityGML dataset can be stored both in files and in databases. CityGML dataset is parsed with CityGML4j (CityGML4j, 2009) and converted into Java objects representing City Objects like buildings, roads which may contain both geometry and semantic information from the input dataset. CityGML4j can directly deal with the CityGML file and if the dataset comes from a database, some extra import tools like 3DcityDBv2 (Claus, 2009) may be required.

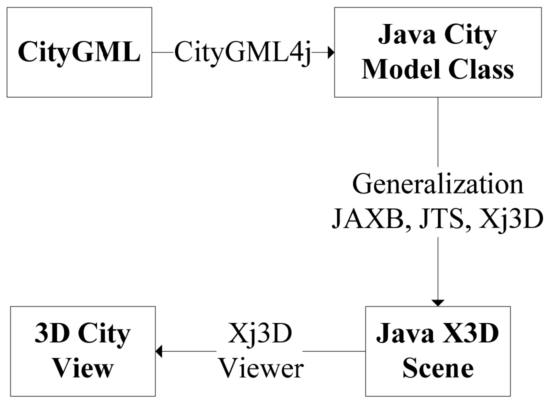


Figure 1. Framework of the 3D city model generalization.

The next step is to convert the original Java classes to one or several X3D Scenes, which are instances of the Java class X3DScene. To perform this conversion the standard Java classes, JAXB is used; for the geometrical and topological operation we utilise the open source library JTS Topology Suite (JTS, 2009). The outcome of this conversion process is the proposed multiple representation data structure, *CityTree*. An option to CityGML would be to perform the conversion using XSLT, which is a general tool for converting XML documents. However, because of difference between the geometry representations in CityGML and X3D, and the visualization of semantic information, complex calculates would be required; this implies that such a conversion would be difficult to implement using XSLT.

In the final step the 3D City view is generated from the City tree (stored as X3DScene Java objects). This is performed with a Xj3D viewer. Since all generalisations are already performed and the different levels of details are stored in the multiple representation data structure CityTree, the visualisation process can be performed dynamically in real time.

Meanwhile, the Xj3D supports Java Applet which could be used to visualize 3D City Model through Internet. Besides Java applet, Internet Brower plug-in X3D viewers such as BS Contact could also be used for online 3D City Model visualization. However, Java applet is recommended because of its standardization and smooth integration with the whole framework

5. GENERALIZATION

An implementation of the proposed framework was carried out by aggregation of buildings. The framework could be used, however, for other data types and generalisation operators.

5.1 Work flow of the generalisation

After the CityGML file is parsed by CityGML4j, all required information is stored in Java classes. Because the city models are in different LoDs, it is necessary to convert the model from higher LoDs to lower LoDs. This study focuses on the building aggregation, and therefore buildings are converted to LoD1 with only ground plan and height information in our implementation.

Figure 2 describes the work flow of the generalisation. Firstly, the ground plans of the buildings are generated. Then simplify those ground plans. Next, the buildings are divided into clusters by the road. For each building cluster, a CityTree is created for visualization. These four steps are the pre-process. With the CityTree, X3D nodes are created to represent the selected buildings. The generalization information (aggregated group buildings) is also shown in the X3D environment dynamically.

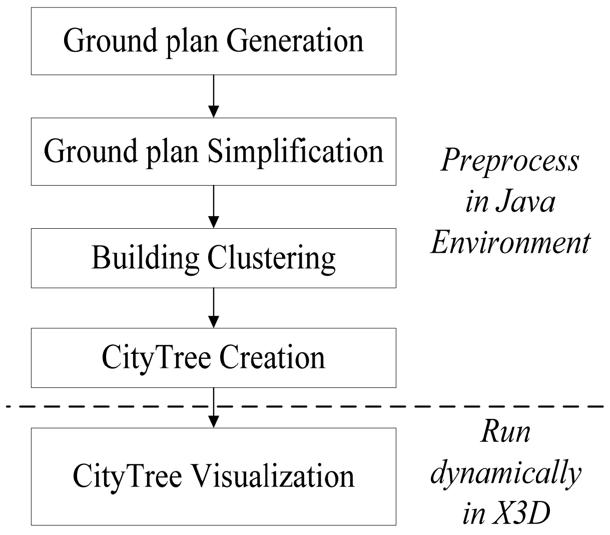


Figure 2. Work flow chat of generalization.

Next, the building clusters are generated by dividing the Delaunay triangulation of the building ground plans. Finally, for each block, the CityTree, a novel proposed structure supporting real-time generalization and continued scaling of 3D city model, is generated for visualization.

5.2 Ground plan generation

Although CityGML schema use *groundsurface* type to define the ground plan of a building, there are still many models do not have such semantic information. It is necessary to find out method to create the ground plan from a simple surface set without any semantic information.

Fan et al. (2009) derived the ground plan from the exterior shell model by projecting the wall on the ground and connecting the foot-print into a closed polygon. However, this method is based on the exterior shell generated by their algorithm which is quite complex and time consuming. Actually, in lots of cases like our building group application, it is not necessary to compute the exterior shell. Meanwhile, the wall based algorithms which construct the ground plan from walls can not deal with the 3D building model with wall sounded yard. Therefore, a method that can directly derive the ground from city models and not based on wall is required.

At this paper, every surfaces of the building is used to generate the ground plan. The algorithm can be simplified described like this. First, project all surfaces into the horizontal plane (xy-plane). Then, unify the projected surfaces to the ground plan. Finally, merge the ground plan to be one polygon.

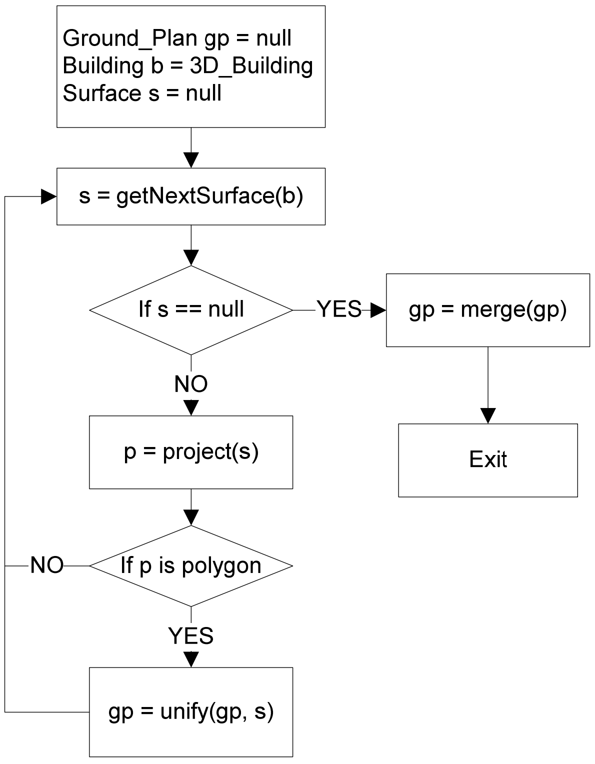
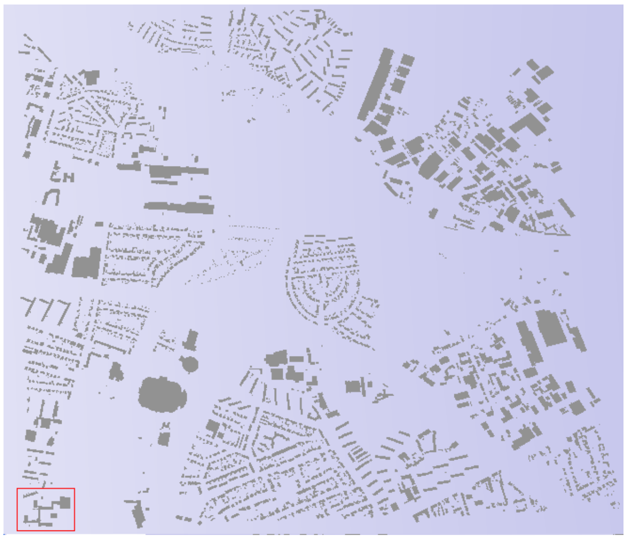


Figure 3. Flow chart of ground plan generation.

Some buildings are made up of separated parts. Therefore, the ground plans for those buildings are multi-polygons. In our implementation, Ground plan *gp* is a multi-polygon. 3D building *b* is constructed from CityGML data. Surface *s* is a common surface in 3D building model *b*. For every surface *s* in *b*, project it into the xy-plane and get *p*. If *p* is not a polygon such as a line, ignore *p*. If *p* is a polygon, unify *gp* and *p*. Finally, merge the polygons in *gp* to ground plan. JTS is used to unify the polygons.

At this step, we only merge the polygons that are close enough, otherwise, the building is considered as separated buildings. The threshold is set to 0.5 meters. For those close polygons, the same merger algorithm in section 6 which is applied for different buildings is employed.


(a)

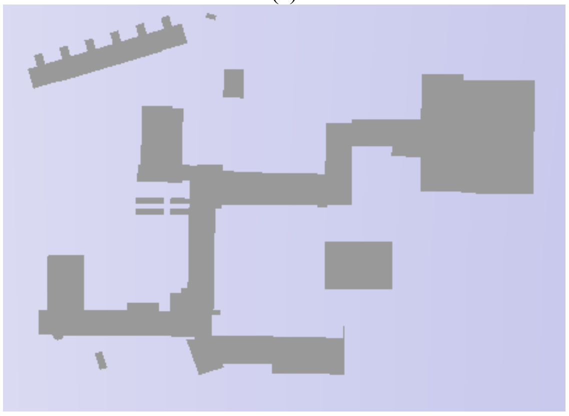
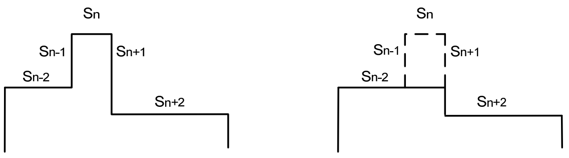

(b)

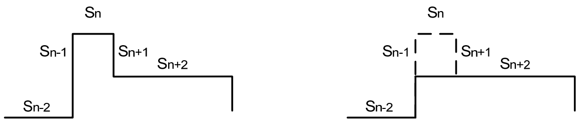
Figure 4. Generated ground plan.

5.3 Ground plan simplification

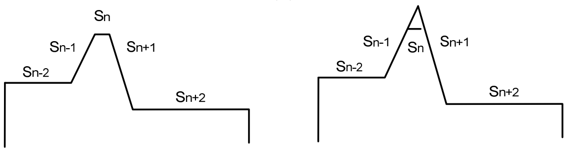
For ground plan simplification, Fan' method (Fan and Meng 2009) is used. This method is designed for the individual building ground plan simplification is used and adjusted according to the needs of building group simplification.



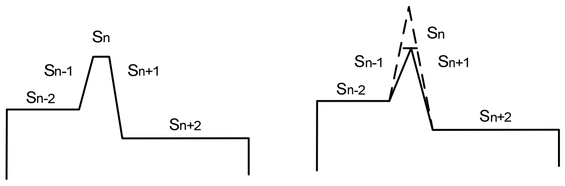
(a)



(b)



(c)



(d)

Figure 5. cases of the ground plan simplification.

The simplification is started by selecting the shortest edge of the ground plan Sn, and testing weather Sn is shorter than the threshold Smin (Sester et al., 2004). If so, delete Sn and adjust the neighbour edges. If not, the simplification process is over. There are 4 cases as discussed by Fan as shown in Figure 5. Figure 6 shows a flow chart of the ground plan simplify process.

Case 1: S_{n-1} and S_{n+1} are parallel and S_{n-1} shorter than S_{n+1}. Then delete S_{n-1} and S_n; prolong S_{n-2} and shorten S_{n+1}. As shown in Figure 5(a).

Case 2: S_{n-1} and S_{n+1} are parallel and S_{n-1} longer than S_{n+1}. Then delete S_{n+1} and S_n; prolong S_{n+2} and shorten S_{n-1}. As shown in Figure 5(b).

Case 3: S_{n-1} and S_{n+1} are not parallel and the intersection point P of S_{n-1} and S_{n+1} is near S_n. Then delete S_n; prolong S_{n-1} and S_{n+1} to P. As shown in Figure 5(c).

Case 4: S_{n-1} and S_{n+1} are not parallel and the intersection point P of S_{n-1} and S_{n+1} is far away S_n. Then delete S_n; prolong S_{n-1} and S_{n+1} to the middle point of S_n. As shown in Figure 5(d).

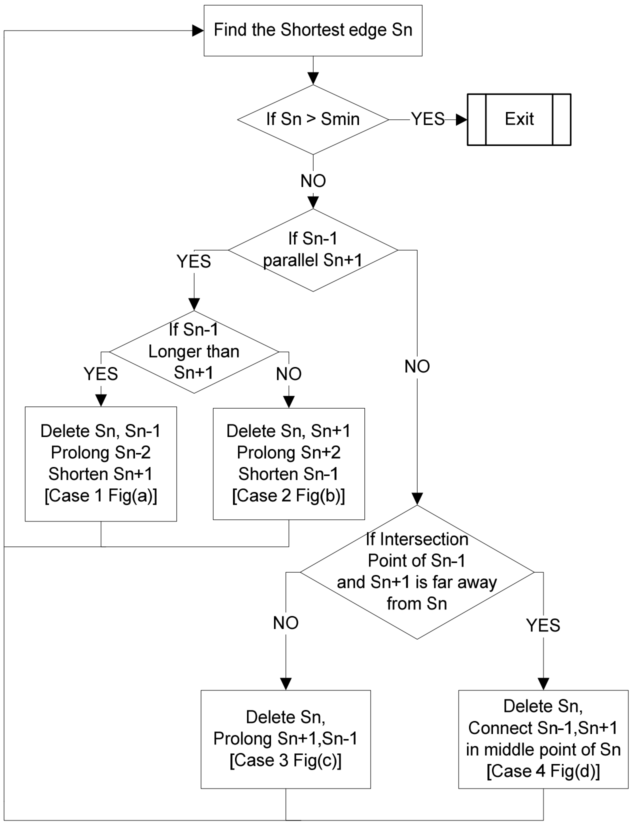

Find the Shortest edge Sn
If Sn > Smin
YES
Exit
NO
If Sn-1 parallel Sn+1
YES
If Sn-1 Longer than Sn+1
YES
NO
NO
Delete Sn, Sn-1 Prolong Sn-2 Shorten Sn+1 [Case 1 Fig(a)]
Delete Sn, Sn+1 Prolong Sn+2 Shorten Sn-1 [Case 2 Fig(b)]
If Intersection Point of Sn-1 and Sn+1 is far away from Sn
NO
YES
Delete Sn, Prolong Sn+1,Sn-1 [Case 3 Fig(c)]
Delete Sn, Connect Sn-1,Sn+1 in middle point of Sn [Case 4 Fig(d)]

Figure 6. Flowchart of the ground plan simplification. Modified from Fan and Meng (2009).

However, Fan’s method is designed for signal building ground plan simplification, but we have to deal with the aggregated building group. Therefore, we suggest use the dynamic thresholds instead of absolute thresholds. Along with the aggregation of buildings, the area of the ground plan could be changed dramatically, and the thresholds such as Smin should change accordingly. In this paper, we mainly take the area of the ground plan into consideration, and calculate the Smin and other thresholds based on the area of the ground plan. Figure 7 gives an example of ground plan simplification in detail. The original ground plan is given in figure 4(b).

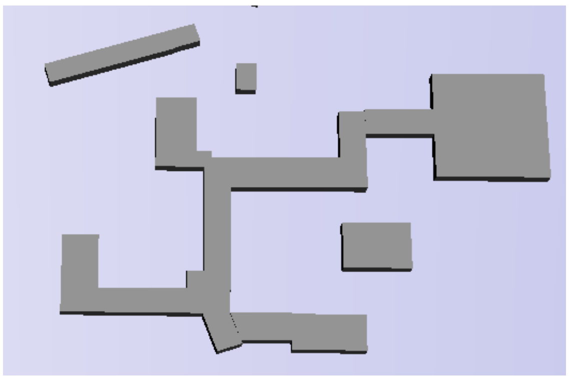


Figure 7. Ground plan simplification in detail.

Figure 8 shows the simplified buildings in whole study area. There are 5330 buildings which contain 35215 points in their ground plans. After the simplification, 14150 points are left. More than 60% points are deleted, while the grounds plans are still quite similar.

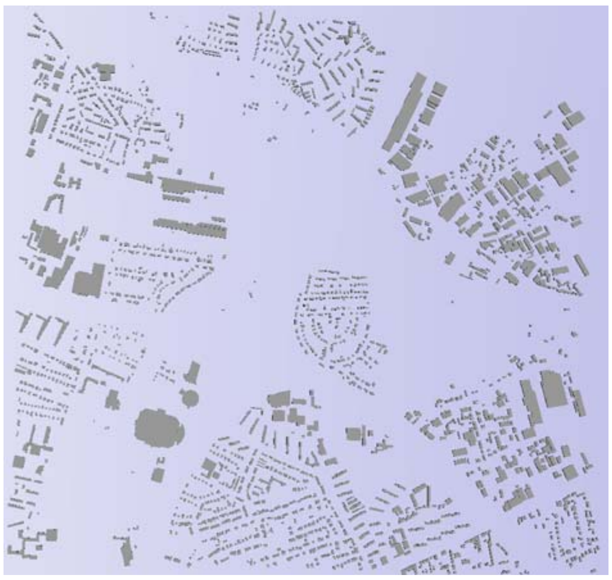
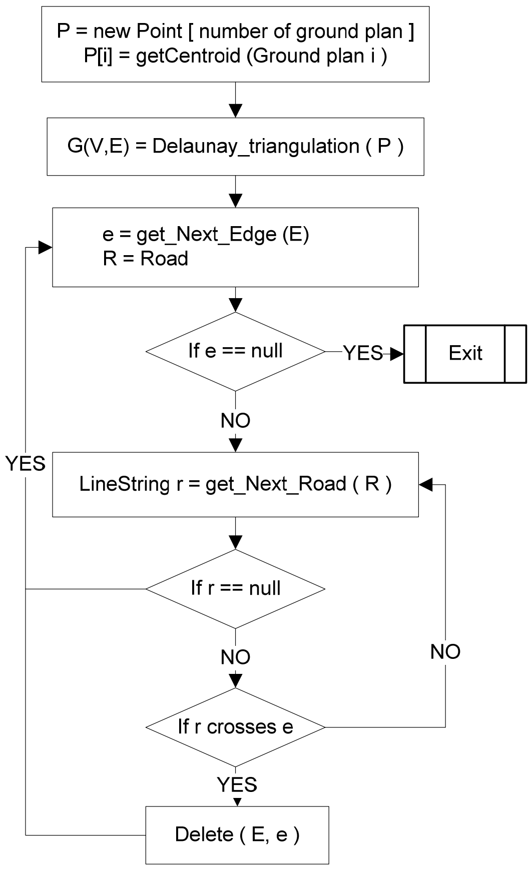


Figure 8. Ground plan simplification whole area.

5.4 Building Clustering

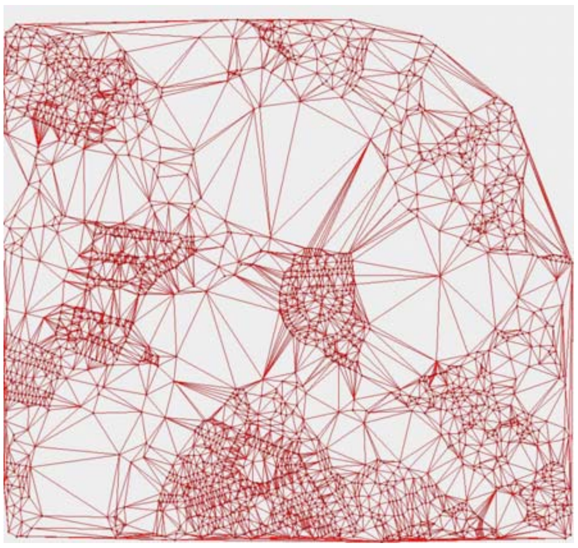
It is reasonable to divide the buildings into groups according to the road in the area. First, buildings in the two sides of the road should not be aggregated as long as the road exists. Second, it will improve the process performance by clustering buildings, since the building number is smaller in a group than whole area, which is essential to certain algorithm such as search and sorting.

It seems that divided certain area by road is not a difficult test. However, in realistic, the data about road is quite mess. A road may be composed by several line strings which might be connected, parallel or not related. In order to deal with this kind of data, Delaunay triangulation of all buildings which is represented by its centroid is generated. Then, for each edge in the triangulation network, test if it crosses some road, if so, delete the edge. By deleting the edges, sub graphs is generated. Each sub graph will represent a building group.


P = new Point [number of ground plan]
P[i] = getCentroid (Ground plan i)
G(V,E) = Delaunay_triangulation (P)
e = get_Next_Edge (E)
R = Road
If e == null
YES
Exit
NO
YES
LineString r = get_Next_Road (R)
If r == null
NO
NO
If r crosses e
YES
Delete (E, e)
Figure 9. Flow chart for the building clustering.

A flow chart for the building clustering is given in Figure 9. First, create an array of Point with the centroid of ground plan. Then, generated the Delaunay triangulation graph G(V, E) of P, in which V is set of the vertexes and E is the set of edges. Then, get an edge e from E, and reset the R to contain all roads. If all edges in E have been processed (e == null) then the algorithm finished. If not, for every road r in R, test whether r crosses e. If so delete e from edge set E. Figure 10 shows the result of segmentation. It is clear to see that our algorithm can perform the building clustering effectively.

(a) Building and road data.


(b) Triangulation of the buildings (one node per building).

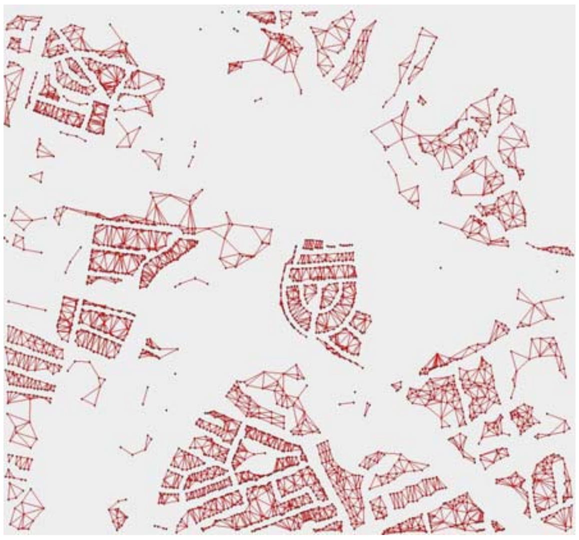

(c) Edges that intersects road and/or are longer than100m are removed

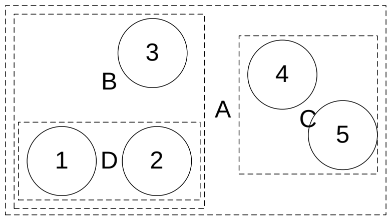
Figure10. Building Clustering of whole study area.

5.5 CityTree Generation

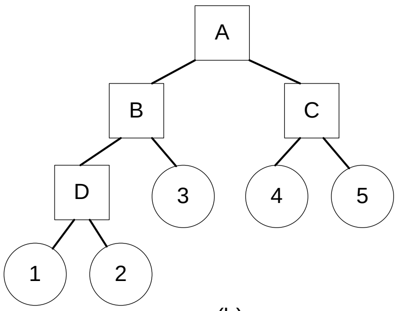
So far we have only made a first example of implementing a first algorithm into our framework. A novel structure CityTree is proposed to support real-time building aggregation and continued scaling of 3D city. This structure is based on a binary tree in which leaf nodes represent the original 3D city objects (mainly the buildings) and the other nodes represent the generalization models of their children. In visualization, selected nodes of the CityTree are shown to the user according to his/her view point. When the view point is changed, the new selected nodes in CityTree will replace the previous ones. With CityTree, the real-time generalization of 3D city can be efficiently realized.

For each building clustering, the CityTree is generated based on "closeness value" between city objects. First, city objects like building are represented by their centre of gravity and the Delaunay triangulation is used to compute the neighbourhood relations between buildings which can be directly got from building clustering. Second, get the "closeness value" between neighbour buildings based on not only geometry information such as the distance, height, area, but also semantics ones like usage, owner and so on. In this paper, we mainly consider the distance between ground plans and the difference of height. Third, merge the closest two nodes to create a new node and repeat this step to create a tree—CityTree. The structure is implemented by CityGML and X3D and the experiment shows that the load time and space in visualization 3D city model can dramatically reduced by using CityTree.

Figure 11 gives a demo example of the CityTree. Figure 11(a) shows the distribution of the original city object (1~5). The rectangle areas (A~D) are created by selecting nearby objects. Then, the CityTree is generated as shown in Figure 11(b). The leaf nodes (1~5) are original objects in city model. The other nodes (A~D) are new generated middle nodes to represent their child nodes.



(a)



(b)

Figure11. CityTree Model.

Each building cluster Bi is represented by triangulation sub-graph Gi = (V, E) in which V is the set of vertexes, and E is the set of edges. The pseudocode of creating CityTree is shown in following steps.

Step1. For every edge e composed by vertex node A and B in E, calculate the "distance" between node A and B, set the root of A and B to themselves, and children to null, in which A and B represent a building respectively. A node contains 4 parts: ground plan polygon, height, root, and children. The distance may contain the ground plan distance, the height difference, and other semantic difference. In this paper, we only consider the distance between ground plans. The root is the root node in CityTree. Children is its child-nodes.

Step2. Get the edge e_{min} with the smallest "distance" value from E.

Step3. Let Node A and B be the two nodes of e_{min}. If RA the root of A is not equal with RB the root of B. Then create a new node C, set root of RA and root of RB to C, children of C to RA and RB, ground plan of C to be the aggregation of RA and RB, and the height of C to be the area weighted average of the height RA and RB, and the root of C to C. Else set C to RA and go to Step 4.

Step4. Delete e_{min} from E. If E is empty set, return CityTree C and Exit. Else, go to Step2.

We aggregate the ground plans by selecting and union the convex hole generated from the nearest edges of the ground plans. Figure 12 shows an example of ground plan aggregation. G1 and G2 are ground plans Figure 12(a). First, find out the nearest edges between them: A1B1 with A2D2, B1C1 with A2A2 are the nearest pairs. Then create the convex holes with the nodes of closest edges shown in Figure 12(b). Next select the convex hole with smallest area but not zero, and union the ground plans with the convex hole shown in Figure 12(c).

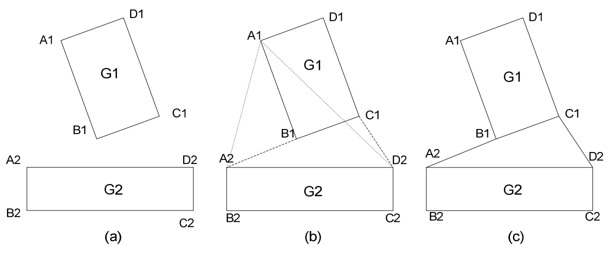


Figure 12. Aggregation of ground plans.

After the aggregation, the new generated ground plan is simplified with the simplification method described in 5.3 with adjusting the threshold accordingly. In this paper, we adjust the threshold according to the area of the ground plan. Figure 13 gives a example of ground plan aggregation. From (a) to (c), the scale is decrease and more buildings are aggregated.

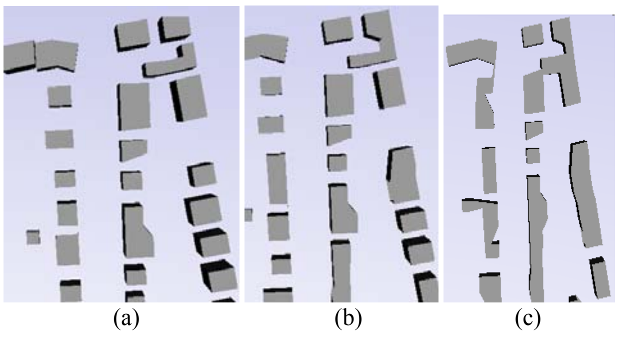


Figure13. Ground plan aggregation result.

5.6 CityTree Visualization

In the visualization step, the CityTree nodes are selected based on the user's view point and the features of the node. In this paper, the distance between the view point and the visible area of the node is considered.

Every node in CityTree has a function to create the X3D scene. First, the root of the CityTree tests relationship of the ground plan area and distance to the view point. We calculate the rate of distance square and the area, if the rate is bigger than a threshold, then this node is shown in the X3D scene, else we test the children of the node to see if they should be visualize in the X3D scene. For the leaf nodes with no children, we just show them with out test the rate. The pseudocode of CityTree visualization is given in Figure 13

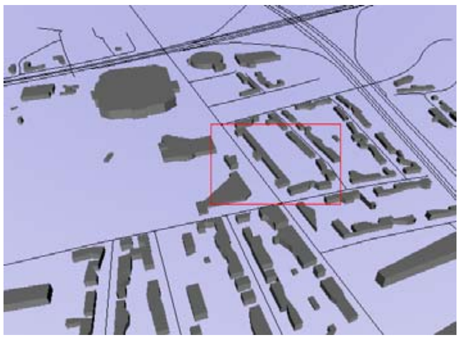
```
void createX3DScene(X3DScene mainscene, Point vp)
{
    Point cent = getCenter();
    double distance = getDistance(vp, cent);
    double area = getGroundPlanArea();
    double rate = area/(distance*distance);

    if(rate<SET_VALUE)
    {
        show(mainscene);
    }
    else
    {
        CityTreeNode lch = getLeftChirld();
        CityTreeNode rch = getRightChirld();
        if(lch!=null)
            lch.createX3DScene(mainscene, vp);
        if(rch!=null)
            rch.createX3DScene(mainscene, vp);
        if(rch==null&&lch==null)
            show(mainscene);
    }
}
```

Figure 13. Pseudocode of CityTree visualization

By invoking the createX3DScene function of a CityTree root, the "most suitable" nodes are selected for visualization. At this paper, we only consider the relationship between view point (vp) and the building ground plan area.

In the Xj3D visualization environment, we can get the view point in real time, based on that, the 3D city models are created dynamically. Figure 14 gives some results of scaling. More 3D buildings in detail can be seem when zoom in.


(a) CityTree visualization in higher view point

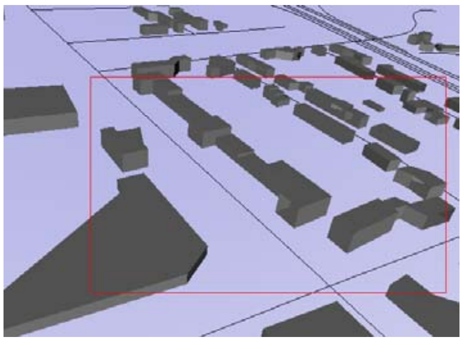
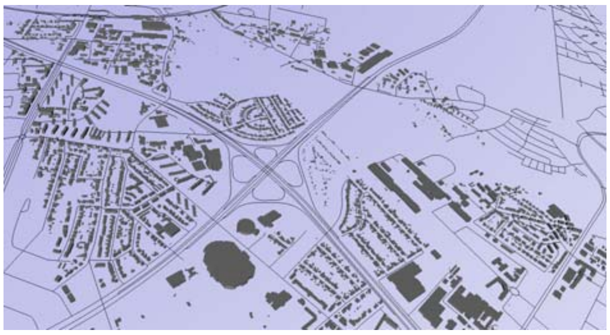

(b) CityTree visualization in closer view point

Figure 14. CityTree Visualization

6. EXPERIMENT RESULT

The framework was implemented Java application. The platform is Eclipse 3.4.1 running on a PC with Inter 2.4GHz Core2 Duo CPU, 2.39GHz 3.25GB RAM, and Microsoft Window XP SP3. The CityGML data is parsed by citygml4j 0.2.0. The 3D city model is visualized with Xj3D 2.0.0. The test datasets come from the CityGML.org (CityGML, 2009).


(a) Before Aggregation

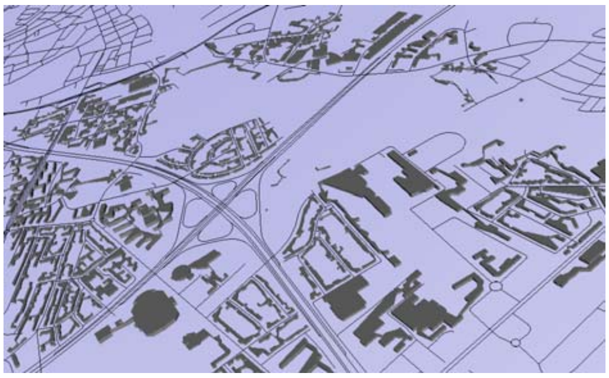

(b) After Simplification and Aggregation

Figure 15. Experiment result

Figure 15 shows an X3D city model of an area around Leverkusen in Germany. Figure 15 (a) is the visualized 3D city model with out generalization, while Figure 15 (b) shows the result of generalization with our proposed method.

7. CONCLUSION

In this paper, CityTree, a framework to support the generalization of 3D city model, is proposed. CityTree is implemented based on CityGML and X3D which are used to represent and visualize the 3D city model respectively. According to the experiment results, CityTree can effectively realize the continual scaling and dramatically reduce the load time of 3D models. However, it is still need to be improved in the generalization of a group of 3D city models.

ACKNOWLEDGE

This paper is supported by the VisuCity Project funded by the Knowledge Foundation, Swedish Foundation for Strategic Research, Vinnova, The Swedish Foundation for Health Care Sciences and Allergy Research, and Invest in Sweden. Bo Mao is thankful to the Royal Institute of Technology - KTH and the China Scholarship Council for the PhD fellowship.

REFERENCE

Burghardt, D., Edwardes, A., Weibel, R., Harrie, L., and Sester, M., 2003. Map generalisation technology: addressing the need for a common research platform, *Proceedings of ICC 2003*, Durban, South Africa

CityGML, 2009 dataset, http://www.citygml.org/1539/

CityGML4j, 2009 http://opportunity.bv.tu-berlin.de/software/projects/show/citygml4j

Claus Nagel , Alexandra Stadler, Gerhard Koenig, Thomas H. Kolbe, CityDBV2, http://opportunity.bv.tu-berlin.de/software/projects/show/3dcitydb (accessed 28 June. 2009)

Forberg, A. (2007). Generalization of 3D building data based on scale-space approach. In: *ISPRS Journal of Photogrammetry and Remote Sensing 62* (2007), pp. 104-111.

Fan, H.; Meng, L.; Jahnke, M. (2009): Generalization of 3D buildings modeled by CityGML. In: Lecture Notes in Geoinformation and Cartography, Advances in GIScience. Springer Heidelberg Berlin, ISSN 1863-2246. pp. 387-405. 12th AGILE International Conference on Geographic Information Science. 02-05 June, 2009, Hannover, Germany.

JTS (2009). JTS Topology Suite, http://www.vividsolutions.com/jts/jtshome.htm.

Köninger, A., and S. Bartel, (1998). 3D-GIS for Urban Purposes. *GeoInformatica*, Vol. 2, No. 1, pp. 79-103.

Kada, M. (2006). 3D Building Generalization based on Half-Space Modeling. In:Proceedings of the ISPRS Workshop on Multiple Representation and Interoperability of Spatial Data, Hannover.

Kada, M. (2007). Generalisation of 3D Building Models by Cell Decomposition and Primitive Instancing. In: Proceedings of the Joint ISPRS Workshop on "Visualization and Exploration of Geospatial Data", Stuttgart, Germany.

Kolbe, T. H. and G. Gröger (2003). Towards unified 3D city models. Proceedings of the ISPRSCommision IV Joint Workshop on Challenges in Geospatial Analysis, Integration and Visualization II, Stuttgart.

Lehto, L. and Sarjakoski, L. T. (2005). Real-time generalization of XML-encoded spatial data for the Web and mobile devices. *International Journal of Geographical Information Science*, 19:8, 957 — 973.

Lutz Plümer, Thomas H. Kolbe, Gerhard Gröger., CityGML-Interoperable Access to 3D City Models, in International Symposium on Geoinformation for Disaster Management. 2005: Delft, Netherlands

Mayer, H. (2005). Scale-spaces for generalization of 3D buildings. In: *International Journal of Geographical Information Science*. Vol. 19, No. 8-9, September-October 2005, pp. 975-997.

Meng, L. and A. Forberg (2007). 3D Building generalisation, In: W. Mackaness, A. Ruas, and L. T. Sarjakoski (eds.), *Generalisation of Geographic Information: Cartographic Modelling and Applications*, Elsevier, pp. 211-232.

OGC, 2009. *CityGML specification*. http://www.opengeospatial.org/standards/citygml.

Web3D, 2009. X3D http://www.web3d.org/x3d/

X3D, 2009. http://www.web3d.org/x3d/specifications/