Automatic Generation of Semantic 3D City Models from Conceptual Massing Models

Kian Wee Chen [1], Patrick Janssen [2], Leslie Norford [3]

[1] CENSAM, Singapore-MIT Alliance for Research and Technology, Singapore
[2] National University of Singapore, Singapore
[3] Department of Architecture, Massachusetts Institute of Technology, USA

chenkianwee@gmail.com, patrick@janssen.name, lnorford@mit.edu

Abstract. We present a workflow to automatically generate semantic 3D city models from conceptual massing models. In the workflow, the massing design is exported as a Collada file. The auto-conversion method, implemented as a Python library, identifies city objects by analysing the relationships between the geometries in the Collada file. For example, if the analysis shows that a closed poly surface satisfies certain geometrical relationships, it is automatically converted to a building. The advantage of this workflow is that no extra modelling effort is required, provided the designers are consistent in the geometrical relationships while modelling their massing design. We will demonstrate the feasibility of the workflow using three examples of increasing complexity. With the success of the demonstrations, we envision the auto-conversion of massing models into semantic models will facilitate the sharing of city models between domain-specific experts and enhance communications in the urban design process.

Keywords: Interoperability, GIS, City Information Modelling, Conceptual Urban Design, Collaborative Urban Design Process

1 Introduction

In the early stages of urban design, designers often prefer the use of conceptual massing models for design exploration because massing models are easy to create and modify. The use of massing models enables designers to visualise and receive timely feedback on their designs. Designers will explore multiple designs, further develop a few and discard unpromising designs. The use of massing models minimises modelling efforts or “sunk cost” on the discarded designs. These massing models are usually in geometrical formats such as Collada, Wavefront and DXF. The disadvantage is that massing models do not have semantic information, which hinders the sharing of models between domain-specific simulation applications and experts. CityGML is a standard format that documents semantic 3D city data for facilitating data sharing [1]. Designers can model their designs in cityGML format to facilitate model sharing, but this requires them to specify the semantic information. As a result, modelling their design in CityGML increases the modelling effort and the "sunk cost" of discarded design models.

The usefulness of the cityGML model is it acts as the main data exchange format for sharing models with other domain experts. Domain-specific experts can visualise the model by directly importing it into a 3D Geospatial Information System (GIS) application. The 3D model will be useful for performing analyses [2] to develop the design further. The standardisation of the exchange format will streamline the process of sharing models. This paper proposes a workflow to automatically generate semantic 3D city models from the conceptual massing models. The generation process automatically identifies city objects such as buildings, terrains and land-use plots, and converts the massing model into a minimal cityGML model consisting of explicitly defined Level of Detail 1 and 0 (LOD1 and 0) city objects.

1.1 Existing Approaches

The most straightforward method is to construct the massing model within a modelling application that has CityGML modelling capability. These are usually GIS-based applications developed for managing large GIS data set, examples of which are ArcGIS [2], Autodesk Infraworks 360 [3], Autodesk Map 3D [4] and Bentley Map [5]. However, urban designers usually work on a smaller scale and thus prefer modelling in Computer-Aided Design (CAD) applications such as SketchUp and Rhinoceros3D for their more flexible and advance 3D modelling capabilities. Thus, this paper focuses on facilitating the latter; transition of geometric models authored in CAD applications into semantic models for sharing among domain-specific experts. There are two main existing methods for generating a semantic 3D city model from a conceptual massing model: 1) import of massing models into modelling tools that support cityGML export or 2) the use of visual scripting to customise the conversion from massing to cityGML.

The first method imports the conceptual model into a 3D modelling application that supports cityGML modelling. Examples of these applications include the CityEditor plugin for SketchUp [6] and RhinoCity plugin for Rhinoceros 3D [7]. In this method, designers either model the massing design in the 3D application or import the model into the 3D modelling application, explicitly declare the semantic information of each geometry and export it to cityGML format. For example, in CityEditor the declaration is based on SketchUp's geometry group, where each geometry group must be declared as a semantic object. The main disadvantage is that the semantic declaration process is inevitably workflow specific and to manually declare each geometry's semantic content can be a time-consuming and laborious task when the designer has multiple design options.

The second method is to use visual scripting to customise the conversion. Designers will create their customised procedure using a visual scripting application to convert their massing models into cityGML. One such application is FME desktop application [8], which provides readily deployable functions to facilitate setting up the conversion procedure. For example, an urban designer models his design in SketchUp and then translates the geometric data from SketchUp to CityGML by setting up a visual script in FME desktop. FME desktop provides functions to both reads and writes data from the SketchUp file into CityGML. The designer's task is to read his massing model geometries, separate the geometries into their respective semantic objects and write them into CityGML schema. The task requires him to be familiar with the FME desktop's functions and the cityGML schema. The main disadvantage is the high complexity involved in setting up the procedures. These procedures require designers, most of whom are novices in computer programming, to be familiar with modifying and adding semantics onto geometries and translating them into a specific schema using programming methods. Although visual scripting has been shown to facilitate the learning of programming methods among design students, it has also been shown that the visual scripting quickly becomes inadequate [9, 10] and confusing [11] for large and complex design tasks.

2 Method

We developed a workflow to automatically generate a cityGML model from a massing model by adapting the workflow from our previous building-level research [12]. The workflow focuses on city objects typically present in massing models: buildings, land-use plots, terrain and road networks. The automated workflow consists of the four main steps shown in Fig. 1: input model, execute analysis rules, execute template rules and retrieve model.

In the first step of the workflow – input model – the model contains the massing of a city model. The massing models can be modelled in any 3D modelling application, provided the buildings are modelled as closed poly surfaces, terrain and land-use plots as open poly surfaces, and road networks as polylines, which is how designers usually create massing models. This method does not require extra modelling effort from designers as it leverages existing modelling conventions. The polylines and poly surfaces from the model are then sorted into a topological data structure as edges and shells. An edge is defined by a line or curve bounded by the starting and ending vertexes. A surface is defined by a closed sequence of connected edges. A shell is defined by a collection of connected surfaces. A closed shell has connected surfaces that form a watertight volume without holes.

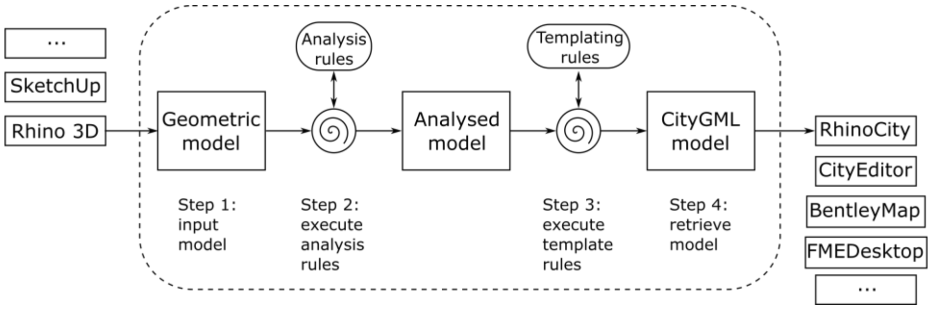


Fig. 1. Proposed workflow for automatically generating CityGML model from massing model

The second step of the workflow – executing analysis rules – starts with analysing the massing model and generating an analysed model with geometric relationship attributes. These attributes are inferred from the size, orientation, and geometrical relationship between topologies in the massing model according to the analysis rules. For example, the shells are analysed and issued a unique identification with attribute `is_shell_closed = True/False`. In order to understand the relationship between topologies in plan, the edges and shells are projected onto the XY plane and analysed. Containment relationships are determined from the analysis. For example, if the topologies are inside one or more other shells when projected to 2D, then attributes are created to capture this information, namely `is_shell_in_boundary = True/False`, `shell_boundary_contains = True/False` and `is_edge_in_boundary = True/False`.

The third step of the workflow – executing template rules – starts with the analysed model and generates the cityGML model. The template rules are matched against the attributes of the analysed model, and if a geometric topology matches the rules, it will be converted into a city object and added into the cityGML model. Designers can customise the template rules according to the type and scale of their urban design. Example rules are as follows:

- If a shell has attributes `is_shell_closed = True`, `is_shell_in_boundary = True`, and `shell_boundary_contains = False`, then a building is generated.
- If a shell has attributes `is_shell_closed = False` `is_shell_in_boundary = False`, and `shell_boundary_contains = True`, then a terrain is generated.
- If a shell has attributes `is_shell_closed = False`, `is_shell_in_boundary = True`, and `shell_boundary_contains = True`, then a land-use plot on the terrain is generated.
- If an edge has attribute `is_edge_in_boundary = True`, then roads are generated.

In the last step, the CityGML is retrieved and shared among domain-specific experts to be further developed.

3 Implementation

The method described above is implemented as four Python classes in a Python library called Pyliburo [13] (*https://github.com/chenkianwee/pyliburo*). The Python classes rely on Pyliburo's modelling kernel for analysing the geometric relationship between the topologies and the CityGML writer for reading and writing CityGML. For this implementation, the massing model is in the Collada format. Each conversion can be represented by a `Massing2Citygml` class, which reads the Collada file and stores each geometric topology as a `ShapeAttributes` class. The analysis rules and template rules are implemented as abstract classes in Python, BaseAnalysisRule and BaseTemplateRule, to facilitate reuse and extensibility.

Fig. 2 illustrates the relationships between the four classes using a Unified Modelling Language (UML) class diagram. In the diagram, the Massing2Citygml class has a one-to-many relationship (1 to N) with the ShapeAttributes and BaseTemplateRule classes. When an instance of Massing2Citygml exists, it can be associated with an unlimited number of ShapeAttributes and BaseTemplateRule classes, as it is necessary to append multiple ShapeAttributes and BaseTemplateRule classes to Massing2Citygml in defining a conversion. The same relationship applies to the BaseTemplateRule and BaseAnalysisRule classes, where multiple BaseAnalysisRule classes are required to define a BaseTemplateRule. The details of each class and their relationships are discussed below.

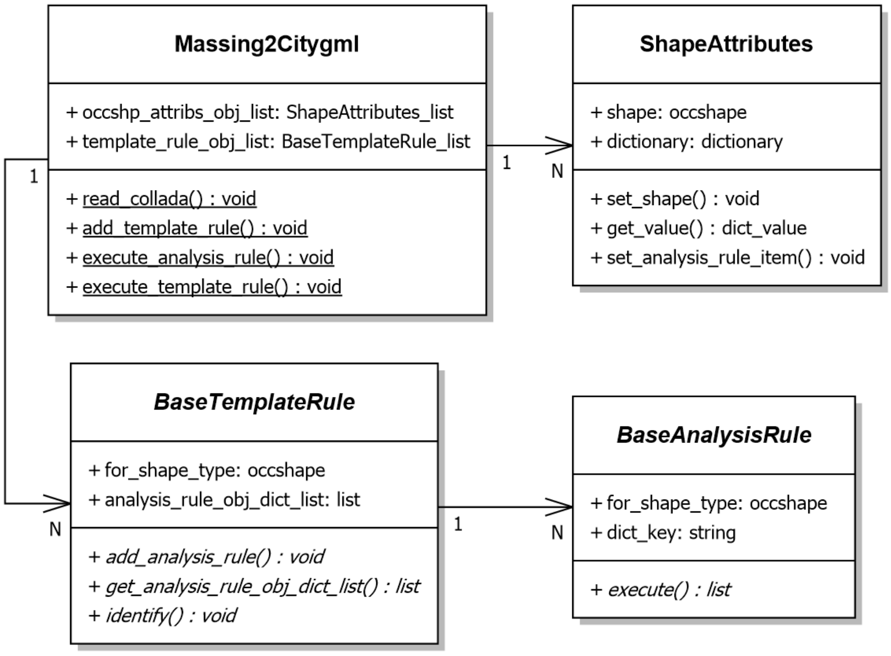

Massing2Citygml
+ occshp_attribs_obj_list: ShapeAttributes_list
+ template_rule_obj_list: BaseTemplateRule_list
+ read_collada() : void
+ add_template_rule() : void
+ execute_analysis_rule() : void
+ execute_template_rule() : void
ShapeAttributes
+ shape: occshape
+ dictionary: dictionary
+ set_shape() : void
+ get_value() : dict_value
+ set_analysis_rule_item() : void
BaseTemplateRule
+ for_shape_type: occshape
+ analysis_rule_obj_dict_list: list
+ add_analysis_rule() : void
+ get_analysis_rule_obj_dict_list() : list
+ identify() : void
BaseAnalysisRule
+ for_shape_type: occshape
+ dict_key: string
+ execute() : list
1
N
1
N
1
N

Fig. 2. UML class diagram of the relationships between the four classes

3.1 Massing2Citygml Class

The Massing2Citygml class represents any massing-to-CityGML model conversion. To set up a conversion process, users must set up a series of analysis rules (section 3.3) and then configure the analysis rules for each template rule (section 3.4). Users add the template rules into the Massing2Citygml class after it is configured through the add_template_rule method. The class will execute the analysis rules using the `execute_analysis_rule` method and the template rules using the `execute_template_rule` method to identify the city objects and write them to a CityGML file.

3.2 ShapeAttributes Class

The `Massing2Citygml` class reads the Collada file using the `read_collada` method, converting the geometries from the file to a topology and storing it as a `ShapeAttributes` class. The `ShapeAttributes` class stores each topology from the massing model as an `OCCShape` class as defined in the modelling kernel (PythonOCC). Any additional attributes of the `OCCShape` are stored as a dictionary. The method `set_shape` adds an `OCCShape` and `get_value` access the attributes stored in the dictionary. The `ShapeAttributes` class is the data exchange format between the other three classes.

3.3 BaseAnalysisRule class

The `BaseAnalysisRule` abstract class represents any analysis rule used for analysing and generating geometric relationship attributes for a massing model. As mentioned in the example rules in section 2, we have implemented four analysis rule classes; `IsShellClosed`, `IsShellInBoundary`, `ShellBoundaryContains` and `IsEdgeInBoundary`, based on the `BaseAnalysisRule` abstract class. We will describe the `IsShellClosed` implementation to illustrate the abstract class.

The `IsShellClosed` class has attributes `for_shape_type = OCCShell` and `dict_key = "is_shell_closed"`. `OCCShell` is the topology class to be analysed by the analysis rule and is as defined in the modelling kernel, PythonOCC. The `execute` method requires one input parameter `occshp_attribs_obj_list`, which contains a list of the `ShapeAttributes` instances from the massing model. The `execute` method loops through all the `ShapeAttributes` instances that are shells and assesses if they are open or closed shells. Once determined, it will append the geometric relationship attribute `is_shell_closed = True/False` to each `ShapeAttributes` instances. The topological attribute must be either true or false; this is enforced through the `set_analysis_rule_item` method in the `ShapeAttributes` class. The method then returns the `occshp_attribs_obj_list` with the topological attribute.

3.4 BaseTemplateRule Class

The `BaseTemplateRule` abstract class represents any template rule used for identifying a city object. As mentioned in section 2 example rules, we have implemented four template rule classes: `IdentifyBuildingMassings`, `IdentifyTerrainMassings`, `IdentifyLandUseMassings` and

IdentifyRoadMassings, based on the BaseTemplateRule abstract class. We will describe the IdentifyBuildingMassings implementation to illustrate the abstract class.

The IdentifyBuildingMassings class has attribute for_shape_type = OCCShell. The identify method requires two input parameters, occshp_attribs_obj_list and the citygmlwriter object from Pyliburo. The identify method loops through all the shells and assesses if they satisfy the geometric relationship attribute conditions set in the analysis_rule_obj_dict_list, a list of dictionaries documenting the analysis rules and their corresponding attribute conditions for identifying the city object of interest. The class provides flexibility for users to define their own analysis rules and its corresponding attribute condition. To identify a building object as specified in section 3, one will add and specify the analysis rules and corresponding attribute condition IsShellClosed = True, IsShellInBoundary = True, and ShellBoundaryContains = False, using the add_analysis_rule method*.* The identify method then retrieves the dictionary that specifies the analysis rule objects and its corresponding attribute conditions using the get_analysis_rule_obj_dict_list method and writes the shell as a building city object.

4 Examples

We demonstrate the feasibility of the automated workflow on three examples. Two simpler examples illustrate how the rules operate and one complex use case illustrates the potential of the workflow. We used SketchUp for modelling the simpler cases and Rhinoceros 3D for modelling the complex case. Using the four Python classes, we wrote a Python script for the conversion, basing it on the analysis and templates rules described in section 2. The source code of the script and the example files can be obtained from GitHub (*https://github.com/chenkianwee/pyliburo_example_files/blob/master/example_scripts/collada/convert_collada2citygml.py*).

A snippet of the source code of the conversion script is shown in Fig. 3. The script requires only two inputs: the Collada file and the file path for the generated CityGML file. First, we initialise the three analysis rules classes; IsShellClosed, IsShellInBoundary and ShellBoundaryContains. Second, we specify the corresponding geometric relationship attribute conditions of each analysis rule class IsShellClosed = True, IsShellInBoundary = True, and ShellBoundaryContains = False and append it to the template class. Third, we append the configured template class to the Massing2Citygml class.

```
input1 = Collada_file
input2 = CityGML_filepath
# 1.) set up the analysis rules
is_shell_closed = IsShellClosed()
is_shell_in_boundary = IsShellInBoundary()
```

```
shell_boundary_contains = ShellBoundaryContains()
# 2.) set up template rules
id_bldgs = IdentifyBuildingMassings()
id_bldgs.add_analysis_rule(is_shell_closed, True)
id_bldgs.add_analysis_rule(is_shell_in_boundary, True)
id_bldgs.add_analysis_rule(shell_boundary_contains,
False)
# 3.) add the template rule in the massing2citygml class
massing_2_citygml = Massing2Citygml()
massing_2_citygml.read_collada(input1)
massing_2_citygml.add_template_rule(id_bldgs)
massing_2_citygml.execute_analysis_rule()
massing_2_citygml.execute_template_rule(input2)
```

Fig. 3 Snippets of the conversion script with the two inputs highlighted in bold

The generated cityGML model is validated by Val3dity [14] and the CityGML schema validator [15]. Val3dity checks and reports geometrical errors of the 3D topologies in a CityGML model. The CityGML schema validator checks and ensure a CityGML model follows its schema definition. A valid CityGML model does not contain any geometrical or schematic errors.

4.1 Example 1

The first example is a simple case; it has a flat terrain, 44 land-use plots, 313 rectangular building extrusions and a road network of 56 edges as shown in Fig. 4a. The example contains a total of 3930 surfaces. We modelled the example using geometry groups as suggested in the SketchUp manual [16]. Extruded buildings, land-use plots, terrain and road networks are modelled as separate geometry groups. Each group is translated into a mesh when exported into Collada. Meshes in Collada contain both surfaces and lines, and meshes that contain surfaces are essentially shells. As a result, building extrusions, land-use plots and terrain geometry groups in SketchUp are automatically exported as closed shells and open shells respectively. The network lines are also automatically exported as edges in Collada. The exported Collada file is triangulated to ensure the geometries are properly translated, as we have experienced inaccurate export of complex geometries, such as those in example 2 and 3, with the non-triangulated option.

Lastly, satisfying all the requirements as mentioned in section 2, the Collada file is converted into a CityGML model as shown in Fig. 4b. Fig. 5 shows the difference between a building extrusion documented in Collada (Fig. 5a) and CityGML (Fig. 5b) after the conversion. The main difference is the building extrusion is explicitly declared as a building object in CityGML, while it is only documented as a mesh in Collada. This is also the case for all the other identified city objects; land-use plots, terrain and roads, in which their semantic information is explicitly declared in the CityGML file.

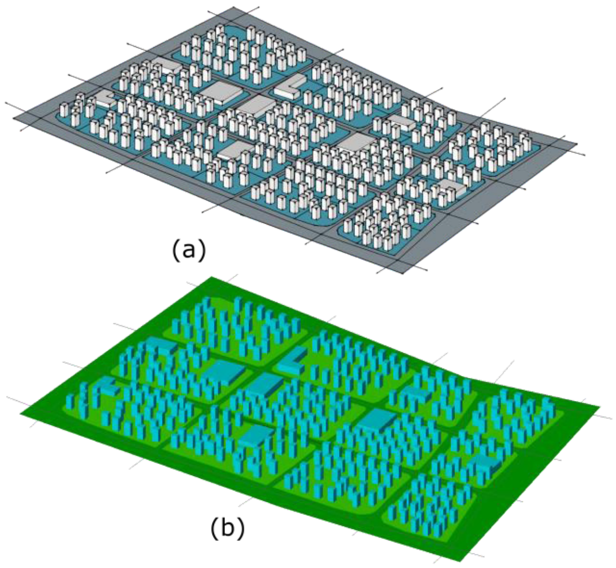
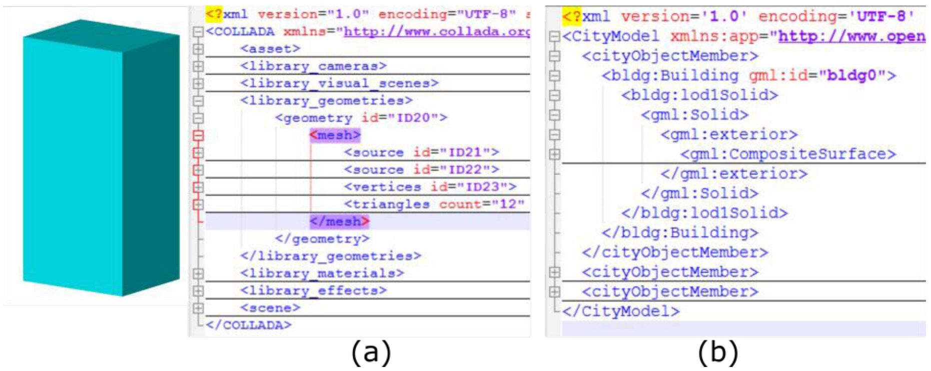


Fig. 4. Example 1 (a) SketchUp massing model (b) Converted CityGML model from the massing model



```
<?xml version="1.0" encoding="UTF-8" s
<COLLADA xmlns="http://www.collada.org
    <asset>
    <library_cameras>
    <library_visual_scenes>
    <library_geometries>
        <geometry id="ID20">
            <mesh>
                <source id="ID21">
                <source id="ID22">
                <vertices id="ID23">
                <triangles count="12"
            </mesh>
        </geometry>
    </library_geometries>
    <library_materials>
    <library_effects>
    <scene>
</COLLADA>
```

```
<?xml version='1.0' encoding='UTF-8'
<CityModel xmlns:app="http://www.open
  <cityObjectMember>
    <bldg:Building gml:id="bldg0">
      <bldg:lod1Solid>
        <gml:Solid>
          <gml:exterior>
            <gml:CompositeSurface>
          </gml:exterior>
        </gml:Solid>
      </bldg:lod1Solid>
    </bldg:Building>
  </cityObjectMember>
  <cityObjectMember>
  <cityObjectMember>
</CityModel>
```

Fig. 5. (a) Building extrusion exported as Collada from SketchUp (b) Converted CityGML building extrusion with explicit building semantic information.

4.2 Example 2

The second example is a more complex case; it has an elevated terrain, 59 land-use plots, 453 building extrusions and a road network of 1125 edges. The added complexities are the TIN (Triangulate Irregular Network) mesh of the elevated terrain consisting of 4961 triangulated surfaces (Fig. 6a) and the non-rectangular building extrusions (Fig. 6b).

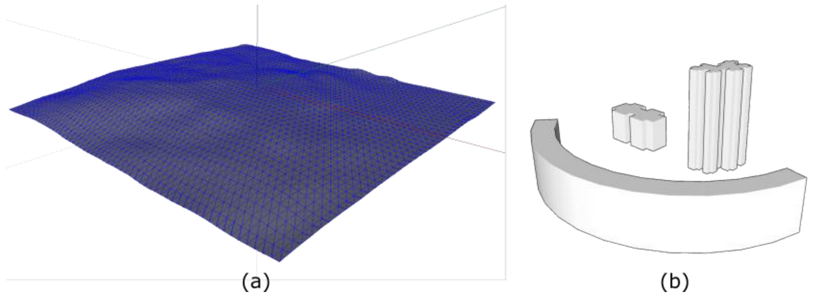


Fig. 6. Added complexities of example 2 (a) TIN mesh of the elevated terrain (b) Examples of non-rectangular building extrusions

All the geometries are modelled according to the recommended SketchUp modelling workflow (Fig. 7a). The example as shown in Fig. 7 contains a total of 37,794 surfaces. The conversion script converted the exported Collada into CityGML (Fig. 7b). The script was able to successfully identify the open shell terrain of 4961 surfaces and non-rectangular extrusions of 76 surfaces, and convert them into the CityGML object as shown in Fig. 8 and Fig. 9.

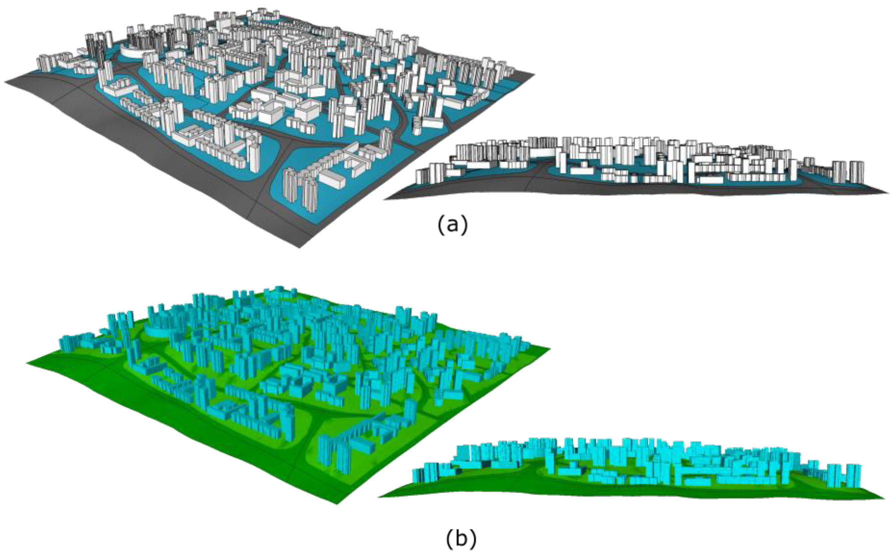
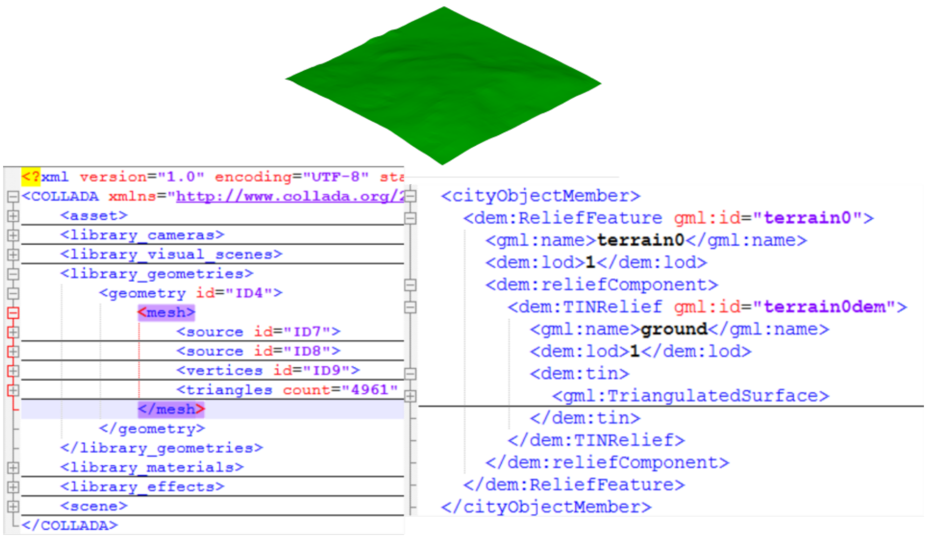


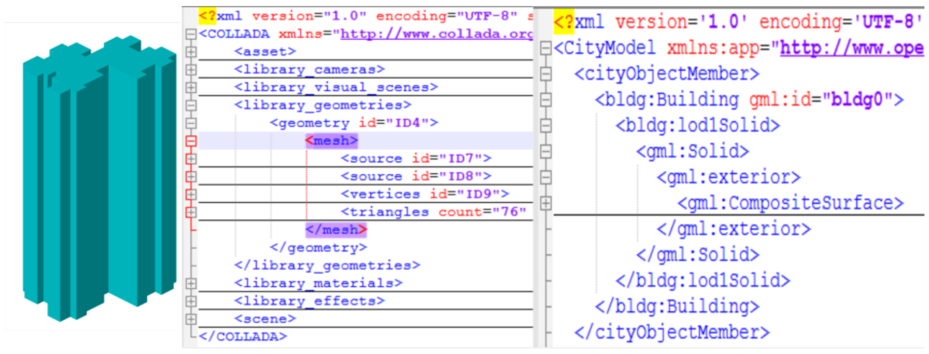
Fig. 7. Example 2 (a) SketchUp massing model with elevated terrain and non-rectangular extrusions (b) Converted CityGML model from the massing model



```
<?xml version="1.0" encoding="UTF-8" sta
<COLLADA xmlns="http://www.collada.org/2
    <asset>
    <library_cameras>
    <library_visual_scenes>
    <library_geometries>
        <geometry id="ID4">
            <mesh>
                <source id="ID7">
                <source id="ID8">
                <vertices id="ID9">
                <triangles count="4961"
            </mesh>
        </geometry>
    </library_geometries>
    <library_materials>
    <library_effects>
    <scene>
</COLLADA>
```

```
<cityObjectMember>
  <dem:ReliefFeature gml:id="terrain0">
    <gml:name>terrain0</gml:name>
    <dem:lod>1</dem:lod>
    <dem:reliefComponent>
      <dem:TINRelief gml:id="terrain0dem">
        <gml:name>ground</gml:name>
        <dem:lod>1</dem:lod>
        <dem:tin>
          <gml:TriangulatedSurface>
        </dem:tin>
      </dem:TINRelief>
    </dem:reliefComponent>
  </dem:ReliefFeature>
</cityObjectMember>
```

Fig. 8. (a) Terrain shell of 4961 surfaces exported as Collada from SketchUp (b) Converted CityGML terrain with explicit terrain semantic information



```
<?xml version="1.0" encoding="UTF-8" s
<COLLADA xmlns="http://www.collada.org
    <asset>
    <library_cameras>
    <library_visual_scenes>
    <library_geometries>
        <geometry id="ID4">
            <mesh>
                <source id="ID7">
                <source id="ID8">
                <vertices id="ID9">
                <triangles count="76"
            </mesh>
        </geometry>
    </library_geometries>
    <library_materials>
    <library_effects>
    <scene>
</COLLADA>
```

```
<?xml version='1.0' encoding='UTF-8'
<CityModel xmlns:app="http://www.ope
  <cityObjectMember>
    <bldg:Building gml:id="bldg0">
      <bldg:lod1Solid>
        <gml:Solid>
          <gml:exterior>
            <gml:CompositeSurface>
          </gml:exterior>
        </gml:Solid>
      </bldg:lod1Solid>
    </bldg:Building>
  </cityObjectMember>
```

Fig. 9. (a) Non-rectangular building extrusion of 76 surfaces exported as Collada from SketchUp (b) Converted CityGML non-rectangular building extrusion with explicit building semantic information

4.3 Example 3

The last example is the most complex case of all; it has an elevated terrain, 60 land-use plots, 174 buildings and a road network of 1512 edges. The complexity of this example is that each building is a complex solid consisting of hundreds of thousands of polygon surfaces (Fig. 10). SketchUp's push/pull modelling technique [17] is not suitable for modelling such complex solids. We used a NURBS modelling application, Rhinoceros 3D, and modelled the geometries according to this application's recommended modelling workflow. The loft command was used extensively for modelling the twisting and slanting towers (Fig. 10a). The join command was then used to join all the lofted surfaces together to form a closed shell. For more complex geometries that are made up of multiple complex solids (Fig. 10b), the boolean union command was used to fuse multiple solids into a single solid.

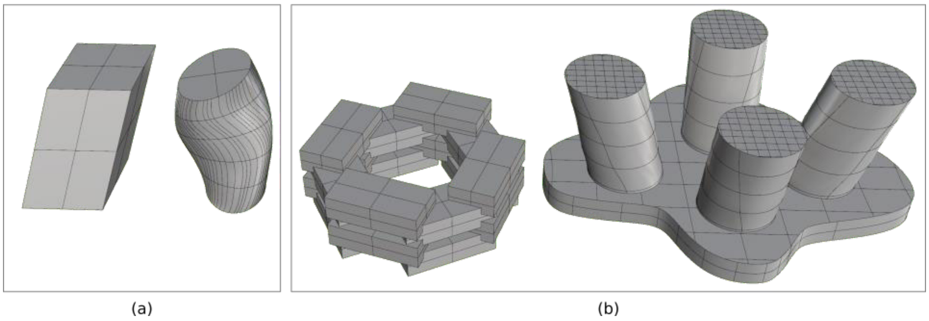


Fig. 10. Complex building solids from example 3 (a) twisting and slanting tower constructed with loft and join command (b) building consisting of multiple solids fuse into a single solid with the boolean union command.

Unfortunately, Rhinoceros 3D is only able to export the building, land-use plot and terrain surfaces and is not able to export the network edges into the Collada format. The workaround is to first export the surfaces from Rhinoceros 3D into SketchUp through the .3ds format and then continue to model the network edges in SketchUp as shown in Fig. 11a. The geometries are eventually exported as Collada and converted into CityGML using the conversion script as shown in Fig. 11b. The example contains a total of 255,953 surfaces. The script was able to successfully convert all the complex building solids into CityGML building objects. Fig. 12 shows an example of a twisting tower of 2960 surfaces and Fig. 13 an example of a building that is made up of multiple complex solids of 3270 surfaces that were converted into CityGML building objects.

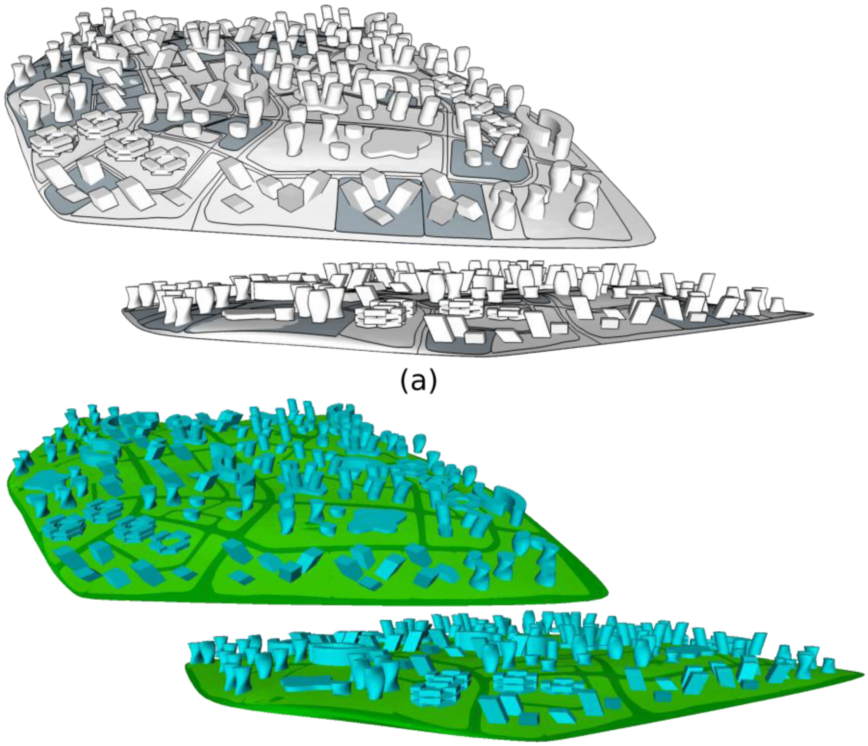
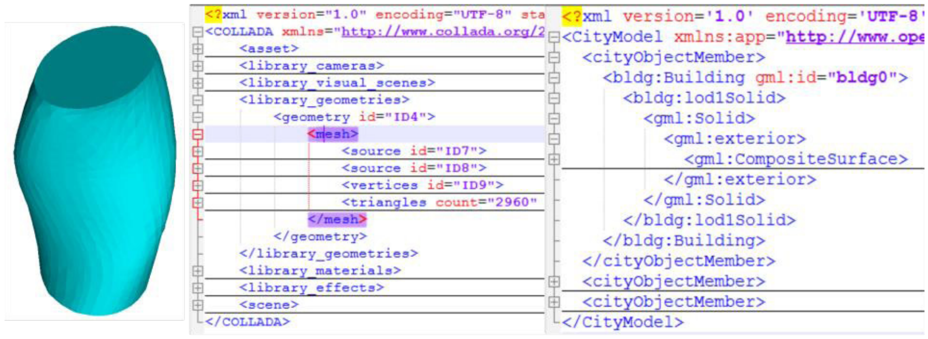


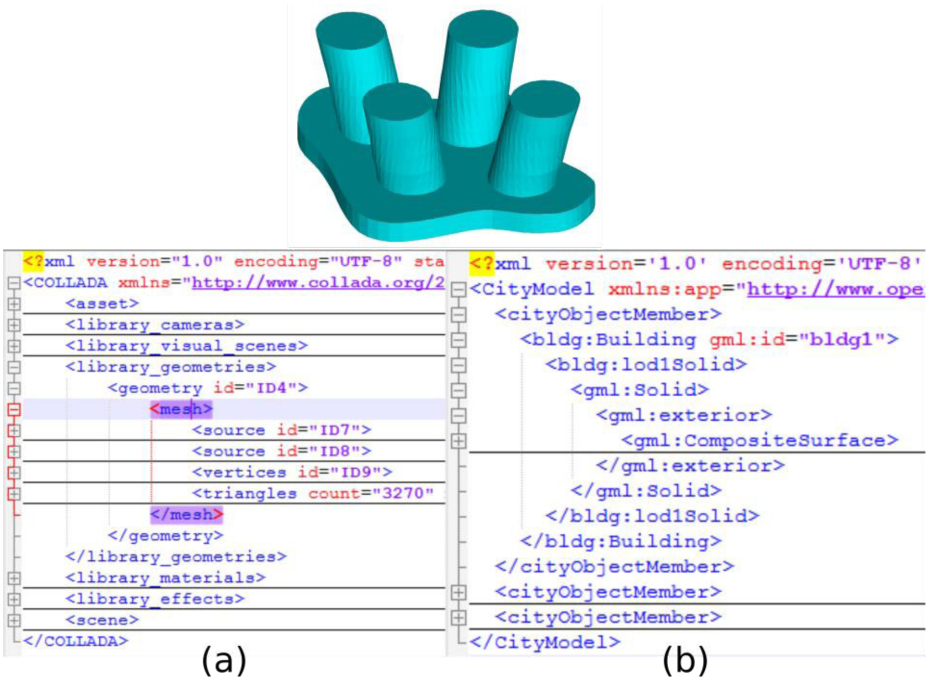
Fig. 11. Example 3 (a) SketchUp massing model with elevated terrain and complex building solids (b) Converted CityGML model from the massing model



```
<?xml version="1.0" encoding="UTF-8" sta
<COLLADA xmlns="http://www.collada.org/2
    <asset>
    <library_cameras>
    <library_visual_scenes>
    <library_geometries>
        <geometry id="ID4">
            <mesh>
                <source id="ID7">
                <source id="ID8">
                <vertices id="ID9">
                <triangles count="2960"
            </mesh>
        </geometry>
    </library_geometries>
    <library_materials>
    <library_effects>
    <scene>
</COLLADA>
```

```
<?xml version='1.0' encoding='UTF-8'
<CityModel xmlns:app="http://www.ope
  <cityObjectMember>
    <bldg:Building gml:id="bldg0">
      <bldg:lod1Solid>
        <gml:Solid>
          <gml:exterior>
            <gml:CompositeSurface>
          </gml:exterior>
        </gml:Solid>
      </bldg:lod1Solid>
    </bldg:Building>
  </cityObjectMember>
  <cityObjectMember>
  <cityObjectMember>
</CityModel>
```

Fig. 12. (a) Complex solid geometry of 2960 surfaces constructed with Rhinoceros loft command and exported to SketchUp then to Collada (b) Converted CityGML complex solid with explicit building semantic information.



```
<?xml version="1.0" encoding="UTF-8" sta
<COLLADA xmlns="http://www.collada.org/2
    <asset>
    <library_cameras>
    <library_visual_scenes>
    <library_geometries>
        <geometry id="ID4">
            <mesh>
                <source id="ID7">
                <source id="ID8">
                <vertices id="ID9">
                <triangles count="3270"
            </mesh>
        </geometry>
    </library_geometries>
    <library_materials>
    <library_effects>
    <scene>
</COLLADA>
```

```
<?xml version='1.0' encoding='UTF-8'
<CityModel xmlns:app="http://www.ope
  <cityObjectMember>
    <bldg:Building gml:id="bldg1">
      <bldg:lod1Solid>
        <gml:Solid>
          <gml:exterior>
            <gml:CompositeSurface>
          </gml:exterior>
        </gml:Solid>
      </bldg:lod1Solid>
    </bldg:Building>
  </cityObjectMember>
  <cityObjectMember>
  <cityObjectMember>
</CityModel>
```

Fig. 13. (a) Complex solid geometry of 3270 surfaces constructed with Rhinoceros loft and boolean union commands and exported to SketchUp then to Collada (b) Converted CityGML complex solid with explicit building semantic information.

4.4 Discussion

The auto-conversion of the massing models to cityGML in examples 1-3 takes 1, 14 and 145 minutes respectively, on a workstation laptop with an i7 processor and 16GB RAM. The complexity in example 3 demands substantially more time for the conversion as compared to examples 1 and 2. However, 145 minutes of computational time is a considerable improvement compared to the required time for manually remodelling the cityGML model. Moreover, it is not common to model in such complexity in the early design stages; we foresee most applications will have the complexity of examples 1 or 2. As the Python library is still an early prototype, further improvement will be made to speed up the conversion process for complex examples. The working prototype will be open and free for usage and feedback (*https://github.com/chenkianwee/pyliburo/blob/master/massing2citygml.py*).

For designers with no programming background who follow the recommended modelling workflow of the respective 3D modelling applications, we have introduced a configuration that only requires two inputs for the conversion and demonstrated its feasibility with the three examples. We envision that the auto-conversion workflow would be used in conjunction with our previous research that generates a 3D semantic city model from open data online [18]. Based on the workflow introduced in [18], designers can acquire all the available data online for reconstructing the existing project site into a CityGML model and use it for the massing design stage. Unlike the conversion from a geometric to a semantic model where there is a need for inferring semantic information, all the geometric data is present in the semantic model. Thus, the CityGML model of the existing site can be easily converted into the geometric model and imported into any 3D modelling application for use in the massing design stage. The massing model will be constructed based on the site model. Essentially, with a streamlined workflow based on an open-standard city model, we hope to enhance communication between experts and facilitate the urban design process.

Currently, the auto-conversion method can only identify four types of city objects; buildings, terrain, land-use plots and roads. However, the library's flexibility and extensibility, described in section 3, allows designers with a programming background to easily configure or create new analysis and template rules catering to their own modelling workflow. One can easily reconfigure the current analysis and template rules for identifying exceptions. One such exception is that land-use plots demarcated for recreational use do not have to contain buildings; to identify such land-use plots one can easily add another land-use template rule with existing analysis rules, `IsShellClosed = False, IsShellInBoundary = True, and ShellBoundaryContains = False.`

When the existing palette of rules is not sufficient for identifying city objects of interest, Pyliburo provides the building blocks, a modelling kernel and a CityGML writer, for creating new rules. For example, one can identify LOD1 road networks by modelling roads as shells instead of edges. In this scenario, one can model the roads as open shells that are contained within a terrain shell and not containing other objects. This only requires the designers to implement a new template rule to identify LOD1 roads, while reusing the `IsShellClosed = False, IsShellInBoundary = True, and ShellBoundaryContains = False` analysis rules to identify the shells as roads.

Similarly, for identifying other transportation infrastructures such as tunnels and bridges, designers will have to decide the modelling procedure for such infrastructures. Tunnels can be modelled as closed shells that are below the terrain and bridges as closed shells that are floating above the terrain. This will require the designer to implement a new set of analysis rules and template rules. First of all, designers have to implement two analysis rules; `IsShellUnder` and `IsShellFloating`. To implement `IsShellUnder`, project the shell upwards; if it hits another shell, it means the projected shell is placed under another shell. Similarly for `IsShellFloating,` project the shell downwards; if it hits another shell and has a distance from the shell, it means the projected shell is floating above another shell. Then append these analysis rules to the template rules: `IsShellClosed = True` and `IsShellUnder = True` for identifying tunnels and `IsShellClosed = True` and `IsShellFloating = True` for identifying bridges.

5 Conclusion

This paper shows the feasibility of the workflow for automatically generating a semantic 3D city model, cityGML, from a conceptual massing model. The workflow does not require extra modelling effort from designers while modelling their massing design, as it leverages existing modelling workflows. The auto-conversion requires only two inputs, the massing model for the conversion and the file path to store the generated cityGML model. It eliminates the time-consuming and laborious task of remodelling massing models into cityGML models so that urban designers can focus on design rather than modelling technicalities. The cityGML model documents partial data from the early design stages in a standard format that can be readily viewed and modified by other 3D GIS applications, thus streamlining the process of sharing models between domain-specific experts. We envision this would facilitate communications between experts in an urban design process.

Further improvements of the auto-conversion include the development of an auto-correct feature for the massing geometries and a Graphical User Interface (GUI) for the library. First, for the conversion of complex geometries consisting of thousands of surfaces, it is demanding for the designers to ensure that each surface is error-free. We would like to implement an auto-correct feature to address this issue. Initially, we need to integrate the Val3dity library for identifying invalid geometries. According to the error identified, we will then develop algorithms using the modelling kernel from Pyliburo to fix the geometries. By doing so, the workflow will be more designer-friendly.

Second, we propose the development of a GUI for the library so that designers who are non-programmers are also able to change the analysis and template rule configurations. We propose developing a parameter tree GUI similar to feature-based modellers such as CATIA. In the parameter tree GUI, the analysis rules are nested within a template rule and the template rules within a `Massing2Citygml` conversion; one can readily remove or append rules to configure the conversion.

Acknowledgements. This research was supported by the National Research Foundation Singapore through the Singapore MIT Alliance for Research and Technology's Center for Environmental Sensing and Modeling interdisciplinary research program.

References

1. Gröger, G. and Plümer, L.: CityGML – Interoperable semantic 3D city models, ISPRS Journal of Photogrammetry and Remote Sensing, 71, 12–33 (2012)
2. <https://www.arcgis.com/features/index.html>, accessed on 1 June 2017
3. <http://www.autodesk.com/products/infraworks-360/overview>, accessed on 1 June 2017
4. <http://www.autodesk.com/products/autocad-map-3d/overview>, accessed on 1 June 2017
5. <https://www.bentley.com/en/products/product-line/infrastructure-asset-performance-software/bentley-map>, accessed on 1 June 2017
6. 3DISGmbH, CityEditor User Manual, Bocholt, Germany (2016)

7. <http://www.rhinoterrain.com/en/rhinocity-3.html>, accessed on 1 June 2017
8. <http://www.safe.com/fme/fme-desktop/>, accessed on 1 June 2017
9. Leitão, A. L. Santos and Lopes, J.: Programming Languages For Generative Design: A Comparative Study, International Journal of Architectural Computing, 10, 139–162 (2012)
10. Celani, G. and Vaz, C.: CAD Scripting And Visual Programming Languages For Implementing Computational Design Concepts: A Comparison From A Pedagogical Point Of View, International Journal of Architectural Computing, 10, 121–138 (2012)
11. Janssen, P.: Visual Dataflow Modelling: Some thoughts on complexity, in Proceedings of the 32nd eCAADe Conference, Newcastle, UK (2014)
12. Janssen, P., Chen, K. W., Mohanty, A.: Automated Generation of BIM Models, in Proceedings of the 34th eCAADe Conference, Oulu, Finland (2016)
13. Chen, K. W. and Norford, L.: Developing an Open Python Library for Urban Design Optimisation - Pyliburo, in Building Simulation 2017, San Francisco, USA (2017)
14. Ledoux, H.: On the Validation of Solids Represented with the International Standards for Geographic Information, Computer-Aided Civil and Infrastructure Engineering, 28, 693–706 (2013)
15. <http://geovalidation.bk.tudelft.nl/schemacitygml/>, accessed on 1 June 2017
16. <https://help.sketchup.com/en/article/3000120>, accessed on 1 June 2017
17. Schell, B., Esch, J. L., Ulmer, J. E.: System and method for three-dimensional modeling, Google Patents, <https://www.google.com/patents/US6628279>, accessed on 1 June 2017
18. Chen, K. W. and Norford, L. K.: Workflow for Generating 3D Urban Models from Open City Data for Performance-Based Urban Design, in Asim 2016 IBPSA Asia Conference, Jeju, Korea (2016)