Appearance-Driven Conversion of Polygon Soup Building Models with Level of Detail Control for 3D Geospatial Applications

Kan Chen[a], Henry Johan[a,b], Marius Erdt[a,b]

[a] *Fraunhofer Singapore*
[b] *Nanyang Technological University*

Abstract

In many 3D applications, building models in polygon-soup representation are commonly used for the purposes of visualization, for example, in movies and games. Their appearances are fine, however geometry-wise, they may have limited information of connectivity and may have internal intersections between their parts. Therefore, they are not well-suited to be directly used in 3D geospatial applications, which usually require geometric analysis. For an input building model in polygon-soup representation, we propose a novel appearance-driven approach to interactively convert it to a two-manifold model, which is more well-suited for 3D geospatial applications. In addition, the level of detail (LOD) can be controlled interactively during the conversion. Because a model in polygon-soup representation is not well-suited for geometric analysis, the main idea of the proposed method is extracting the visual appearance of the input building model and utilizing it to facilitate the conversion and LODs generation. The silhouettes are extracted and used to identify the features of the building. After this, according to the locations of these features, horizontal cross-sections are generated. We then connect two adjacent horizontal cross-sections to reconstruct the building. We control the LOD by processing the features on the silhouettes and horizontal cross-sections using a 2D approach. We also propose facilitating the conversion and LOD control by integrating a variety of rasterization methods. The results of our experiments demonstrate the effectiveness of our method.

Keywords: 3D geospatial; model conversion; level of detail; 3D polygon-soup building; appearance; two-manifold

1. Introduction

In various applications, for example, driving simulator, movie, 3D game, virtual reality and architecture, 3D models of buildings are largely used for the purposes of visualization. On the other hand, in various 3D geospatial simulation and analysis applications, for example, shadow/flood/wind/noise/traffic simulation and urban management/planning, 3D building models are also required. Geospatial simulations and queries often require the full vertex/polygon connectivity information of the 3D building models for the purpose of performing geometric analysis. Geometric analysis is often conducted on the exterior of the building model. For instance, in order to query all polygons of the roof of a 3D house, geometric analysis is required. Similarly, for the flood simulation, when one point (vertex) of a 3D house model is hit by the flood, in order to get the correct neighboring vertices, geometric analysis is also required. To this end, geometry-wise, geospatial applications require suitable 3D building models, which are non-self-intersecting and two-manifold, in order to perform geometric analysis.

Moreover, obtaining a suitable level of detail (LOD) representation is an essential geospatial query. The requirement of LOD for different geospatial applications varies, for example, wind or flood simulation is usually performed on lower LOD with the purpose of reducing the high cost of simulation computation without affecting much the quality of simulation output. In general, deriving LODs also requires 3D geometric analysis in order to generate different geometric complexities.

The digital entertainment industry has been growing tremendously. As shown in Figure 1, 3D models of buildings for the

(a) In Assassin's Creed Unity (game), a scene of Paris

(b) In Grand Theft Auto V (game), Los Santos vs Los Angeles

Figure 1: 3D models of buildings for the purposes of visualization are largely available.

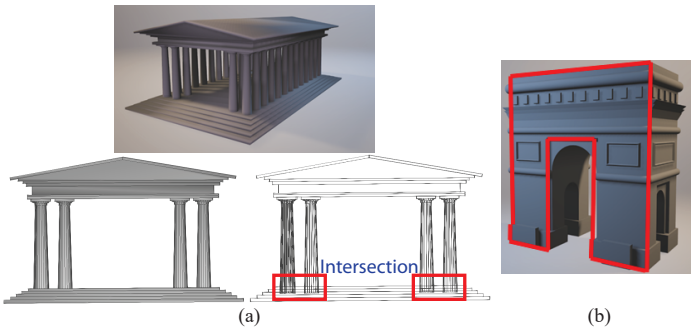

Figure 2: (a) This model of building is non-two-manifold. (b) Based on the ground print, directly performing extrusion to the roof cannot produce this building.

purposes of visualization are largely available. Utilizing these models for geospatial applications would be beneficial. 3D models of buildings for the purposes of visualization are built with right appearance, however they are mostly in the form of polygon-soup representation. Thus, they are not well-suited for geometric analysis. For instance, for the building shown in Figure 2(a), its pillar is often created only one time. After this, the pillar is simply copied and put throughout the floor. Even though the building looks fine from outside, the pillars might go into or through the roof or floor. This causes the lack of information for the connectivity between the polygons of the pillars and the polygons of the roof/floor. That is, in terms of geometry, the 3D models of buildings for the purposes of visualization can be meshes in polygon-soup representation, and they might contain internal self-intersecting components or being non-two-manifold. Hence, for geospatial applications, such representation cannot be directly adopted. Meanwhile, due to the lack of information for polygon connectivity, repairing or converting them can be challenging. Tedious manual efforts are usually required for such processes to resolve incorrect connectivity of polygons and self-intersections.

The lack of information for polygon connectivity is also a problem in LOD computation. This is because computational-wise, existing LODs generation methods usually process target geometries explicitly in 3D. Moreover, this 3D computation can also be tedious and we believe it can be handled in a simpler manner.

Moreover, existing methods for 3D geospatial building model generation might cause some of the important features of the original building to be lost. For instance, for representing the 3D digital city and building models, CityGML [15] is a widely adopted format. Nevertheless, for CityGML model generation, the most common approach is by directly extruding from the ground print of an input 3D building model to its roof [20, 57]. This extrusion-based method is not capable of producing the arc building model as shown in Figure 2(b). Preserving these representative features (like the arc), which are important for geospatial simulation and viewing, is essential.

In this research, a novel method is proposed to convert a 3D building model in polygon-soup representation to a model, which is two-manifold and well-suited for 3D geospatial applications (for example, in the format of CityGML). In addition, its LOD can also be controlled interactively during the conversion. The efforts of re-generating existing viewing-ready models of buildings or fixing the models manually for 3D geospatial usage can be saved by using our method. Moreover, their lower LODs, which may be desired by some other applications, can also be efficiently obtained. The proposed method focuses on the exterior of the 3D building models. The main features are as follows.

- Because of the limited information of polygon connectivity, it is difficult to conduct geometric processing on a model in polygon-soup representation. Nevertheless, since for a 3D model in polygon-soup representation for visualization purposes, its visual appearance is the most important attribute, artists have created it in correct appearances. This means, the rendering results (rasterized images) should be correct. As such, we propose an approach by extracting and utilizing the visual appearance with the purpose of facilitating the model conversion and LOD control by engaging tools of computer graphics, particularly, the rasterization techniques.
  We propose extracting as well as using the following visual appearances (three types): height map (top view), cross sections (outer shell) and silhouettes with side depth maps (side views).

- 3D models of buildings are often standing upright is one observation which the proposed method is based on. Furthermore, many similarities exist between levels, e.g. level one and two of a building may be similar. Therefore, in one building model, multiple similar levels could be represented using only one cross-section of the building.
  The features of the 3D building are identified based on the building silhouettes, which are computed from side views. The next step is generating horizontal cross-sections of the building, which are parallel to the ground, according to the locations of these features. After this, we connect these cross-sections to reconstruct the building model. We also propose generating side depth maps in order to assist the correspondence computation between adjacent cross-sections for the connection.

- Another observation which our method is based on is that, when we view 3D geometries, they are essentially perceived in 2D space, e.g., in screen space. Hence, instead of processing the complex geometry explicitly in 3D space, we propose focusing on its perceived 2D visual appearance, which is also an important attribute.
  To obtain the desired LOD, we propose processing the LOD of the computed silhouettes and horizontal cross-sections in 2D image space.

- The proposed method performs model conversion and LOD control in one unified framework. Instead of 3D geometric processing based on the information of polygon connectivity, the proposed method is realized by 2D image processing and rasterization methods. As a consequence,

an arbitrary model of 3D building in polygon-soup representation could be converted to a model, which is two-manifold and well-suited for 3D geospatial usages, with interactively controllable LOD. In addition, for an arbitrary 3D model of a building, the proposed method could be considered a technique to mend it via ensuring the property of two-manifold and fixing internal self-intersections.

The extended contributions comparing to an earlier version [12] of our work are as follows.

- In this extended version, we add and integrate a new theme: LOD control, which is an essential geospatial query. In [12], we focused mainly on the model conversion, we did not address LOD.
- We propose a novel appearance-driven 2D approach for LOD control. With the same spirit in [12] and as an unified framework, we tackle the LOD control problem.
- We propose and add a novel method to assist the correspondence computation between cross sections by generating side depth maps. In [12], we used side maps without depth, for computing the silhouettes only. In this extended version, depth information is also computed (without adding too much efforts) to help the spatial-hashing-based correspondence computation, which had difficulties sometimes in [12].
- We add new results (with LOD control) to demonstrate the effectiveness of the proposed method (new results as shown in Figure 9 and LOD control as shown in Figures 8 and 10).

2. Related work

For 3D models of buildings, the related work on model generation/representation, how to convert between different formats and LOD control, are reviewed in this section.

2.1. 3D building model representation and generation

3D digital models of buildings have a large spectrum of various applications. Visualization, computer-aided-design (CAD) and geospatial application are the main categories of their usages.

2.1.1. 3D models of buildings for visualization purposes

Movies, virtual reality simulators and games are the common applications for visualization. Polygon representation [31, 27, 29], volumetric representation [21], imagery representation [51, 33] are the major types of representations for 3D models.

In 3D visualization applications, the most commonly used representation is 3D polygon mesh (for example, FBX, Collada and OBJ formats). Usually, 3D buildings in polygon representation are modeled by 3D modelers using software for 3D modeling, such as Blender [7], Maya [4] and 3D Max [2]. In addition, to help the creation of 3D buildings in polygon representation, various automatic tools and software plugins [50, 38, 21, 45] were already available. They are mainly based on procedural heuristics. For example, a shape grammar for procedurally generating buildings was invented by Muller et al. [50]. To automatically generate buildings, shape grammar and image processing are combined in the work by Liu et al. [45]. This procedure-based method is well-suited for creating a large number of buildings, like a city. Nevertheless, the result is often less realistic but more artificial and might be limited by the pre-defined rules. 3D reconstruction based on acquisition using photogrammetry or laser scanning [43, 21] is another important category. Based on unstructured 3D point clouds, Lafarge and Mallet [43] proposed an approach for modeling 3D cities based on minimization of non-convex energy. In the work of Toshev et al. [54], structures of buildings are detected and parsed based on the point clouds. A hierarchical and compact representation can be then constructed. To acquire as well as construct a model of building from 3D point clouds, the above methods often need tedious manual and computational efforts.

Impressing viewers is the main purpose for which the 3D models of buildings for visualization are often modeled, e.g., to impress game players. The appearance of the models should be visually immersive and correct. On the other hand, the ease of performing geometric processing is not the purpose for which they are modeled. Hence, the properties of incomplete information of polygon connectivity and internal self-intersections, usually exist in the models for visualization purpose. That is, representation-wise, polygon-soup form is very common for them.

Therefore, in most real-time applications like games, the computations like AI query as well as the physics of a game, are often conducted via another form of representation, that is simplified as well as dedicated for the ease of computation in real-time [19, 25], for example, boxes and spheres. Since the movie and game industries are fast developing, models for 3D visualization are hugely available. A large number of 3D cities and buildings, which are very realistic, could be seen in many 3D movies and games. For example, in the game Assassin's Creed Unity, the 3D Paris city scene is very realistic (Figure 1(a)). A number of online portals and databases for 3D models are available [55, 47]. From there, the users can easily access 3D models of buildings created for visualization.

2.1.2. 3D models of buildings for CAD purposes

CAD models of buildings are often for purposes of construction and architecture design. Professionals like surveyors, civil engineers and architects, model and use the CAD models. Generally, professionals use professional CAD software to model 3D CAD buildings, like Rhino [5], SketchUp [24], AutoCAD [3], and so on.

For CAD models of buildings, besides 3D geometric data, professionals also create and integrate accurate and detailed architectural data, like meta semantic information, electrical/piping/structural information, measurement, and so on. In architecture, building information (BIM) data has been largely adopted [41, 1]. For BIM data, the format of Industry Foundation Classes (IFC) standard [32] is very popular.

Currently, 3D printing (additive/layered manufacturing) is a popular method for prototyping and manufacturing. The conventional 3D printing technique horizontally slices the input model with evenly separated cross-sections, and rebuilds the model by layering the cross-sections [8]. However, not every model is 3D printable, geometric processing or analysis or manual remodeling is usually required to pre-process the input model to make it printable. To make a large-scale model printable, Hao et al. [28] proposed decomposing it into small components using curvature-based clustering. Luo et al. [48] proposed a method to automatically separate a large input into printable parts. Self-intersections and the lack of polygon connectivity information are also difficult problems in 3D printing.

2.1.3. 3D models of buildings for geospatial purposes

For geospatial purposes, the integration of non-geometric data (like geographical or semantic data) and 3D geometric data are required for 3D building models. This kind of model usually can be generated by creating 3D geometric and combining with geographical data. The common approach for 3D geometric data generation is by acquiring and reconstructing from LIDAR data, photogrammetry or laser scanned 3D point clouds [17, 49, 30].

Another important step is combining with geographical data, like the semantic information of the building (wall, roof), e.g., CityGML format [40, 26]. The combination is conducted according to geographical information like IDs of buildings, ground prints of buildings and GPS. In terms of size, this kind of model is often in huge size, e.g., as large as city-scale. Geospatial queries usually require geometric processing. In order to conduct geometric processing, in general, it is necessary to have full information of polygon connectivity.

2.2. Conversion of 3D models

For two-way conversion between geospatial data models (for example, CityGML models) and CAD models (for example, IFC models), many tools were already developed. CAD models contain right 3D geometric data and semantic information. Thus, converting from CAD models to models well-suited for geospatial purposes is relatively easy [42, 18, 23, 37, 16].

From a geospatial data model such as CityGML, 3D geometric data can be directly obtained. For visualization purposes, such data can be directly applied. However, since meshes in polygon-soup representation contain only incomplete information of polygon connectivity, converting them for geospatial purposes is a difficult problem. To address this problem, less work has been done. To repair polygonal meshes for CityGML, Zhao et al. [59, 58] proposed an approach by constructing bounding surfaces based on heuristics. But, there might be some loss in visual appearance and can be limited by the heuristics. A voxel-based method was proposed in the work by Donkers et al. [16]. Nevertheless, additional storage and computation costs are required for voxel-based representation. Sampling artifacts might also be introduced in this method when sampling from the input, therefore it is not easy to preserve the shape.

It is often required to have an additional fixing step for the building model to be cleaned. Generally, existing methods for repairing are directly performed on the surfaces of the input models to detect and fix artifacts ([35, 36, 10]). In these methods, the overall shapes of the models can be preserved. On the other hand, local continuity is often required in these methods, that is, only smooth meshes can be handled. But such continuity is usually incomplete in 3D building models, e.g., walls have often right angles. Thus, for 3D building models, it is not well-suited to directly apply the methods which are surface-oriented.

2.3. LODs of 3D buildings

LOD is a very popular technique for many applications, such as games and simulators. Based on the requirement of a specific task or scenario, the user can choose the desired LOD. For example, if a game object is very far away from the camera, we can choose a low LOD representation for this game object. Mesh simplification based on geometric processing is commonly used in computer graphics applications to generate LODs. The most common approaches are mesh decimation [52], progressive meshes [31], quadric error metrics simplification [22]. Please refer to a survey in [14]. Moreover, for geospatial representations like CityGML, which have the information of semantics, LODs generation can be performed with the help of such information [20, 6, 57]. Since the input of our method can be a 3D polygon-soup mesh, existing geometric processing based mesh simplification is still difficult to handle such input. Furthermore, unlike most existing methods which are based on conducting 3D processing on 3D complex geometries, we think focusing on its viewer-perceived 2D appearance could be a more efficient approach.

3. Our proposed method

3.1. Basic idea and steps

Given an input of a 3D model of a building in polygon-soup representation, the goal of this paper is to convert it for geospatial applications with interactive control over its LOD. Our work focuses on the exterior (outer shell) of the building, the interior is not handled. The exterior of the building refers to the building parts that are visible to an outside viewer, such as external walls and roofs. The interior refers to the inner non-visible parts like rooms and furniture. In other words, in CityGML representation, we are mainly focusing on converting the polygon-soup model to LOD2-like CityGML model with roof/wall/ground geometry and relative semantic information.

As mentioned previously, a building model in 3D polygon-soup representation might have limited information of polygon connectivity. As such, it is challenging to perform geometric processing on them. On the other hand, the important and reliable information of the polygon-soup building model is its appearance that is perceived by viewers. Therefore, we propose extracting the appearance, and we propose achieving this by computing the outer shell of the building based on rasterization techniques. In this way, even without geometric processing, we

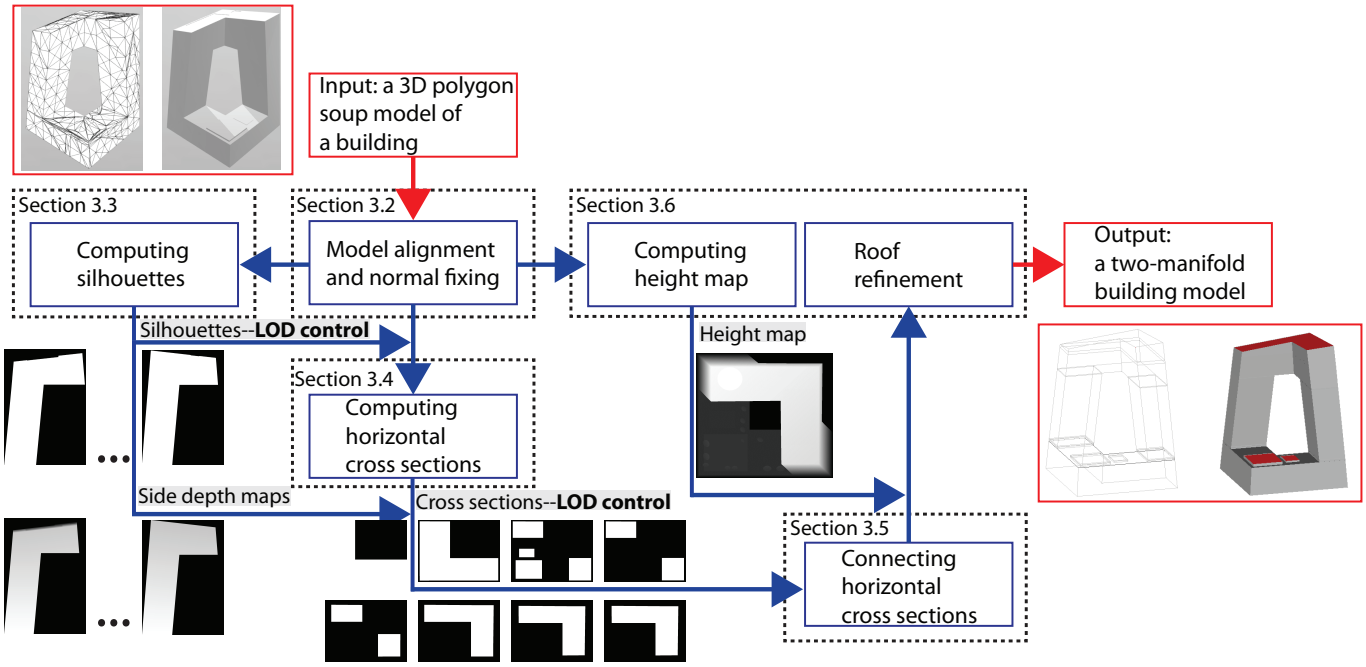

Figure 3: The steps of our method.

can still generate models which are well-suited for geospatial applications.

The silhouettes S of a 3D building model represent its overall shape: viewers can usually tell a building based on the important features F on its silhouettes. Moreover, one of our observations is that there are usually many similarities between different levels of a building. Therefore, for the given input polygon-soup building model B, we propose generating its outer shell $\Omega(B)$ by connecting the contours of its horizontal cross-sections $\Omega(C)$ that are derived from n important silhouette features $f_i \in F$, $(0 \leq i < n)$ at certain heights measured from the ground.

$$\Omega(B) \approx \bigcup \left\{ \Omega(C_{f_{0_y}}), \Omega(C_{f_{1_y}}), \ldots, \Omega(C_{f_{n-1_y}}) \right\}, \qquad (1)$$

where f_{i_y} is the height of silhouette feature f_i.

Instead of explicitly processing complex geometries in 3D, we propose controlling the LOD using a simpler 2D approach. Similar to the model conversion, the proposed LOD control is realized by focusing on the 2D appearance perceived by viewers: silhouette features and horizontal cross-sections.

As shown in Figure 3, our proposed method integrates model conversion and LOD control into one unified framework, and it has five major steps as follows.

1. We align the input 3D model and fix its face normals (Section 3.2).
2. We compute the silhouettes of the model and extract their features. Here, we can control the LOD of the silhouettes. Based on these extracted features, we determine the heights for computing horizontal cross-sections (Section 3.3). We also compute the respective side depth maps.
3. We compute horizontal cross-sections parallel to the ground (Section 3.4), and we can also control their LOD.
4. We reconstruct the building by connecting the adjacent horizontal cross-sections based on the computed side depth maps (Section 3.5).
5. We compute the height map (roof depth map) from the top view of the building model and use it to refine the roof of the building (Section 3.6).

We will present the above steps in the following subsections.

3.2. *Model alignment and normal fixing*

This pre-processing step is performed as follows.

1. The input building model is aligned, so that its up direction is aligned with the y axis and its back-front direction is aligned with the z axis. Since our horizontal cross-sections will be computed horizontally, thus the model has to be ensured as standing upright.
2. The face normals are ensured to be pointing outwards from the model. Since our cross-section computation is based on face culling, faces have to be facing away from the inside.

The above two operations could be easily performed in any popular modeling tool, such as Maya [4]. As shown in Figure 4, one pre-processed model of a building is presented.

3.3. *Computing silhouettes and side depth maps*

Silhouettes represent the vertical features of a building, e.g., from the silhouettes we can tell if a building is standing upright or leaning. We compute the silhouettes in four directions: front to back/back to front (along the z axis), and left to right/right

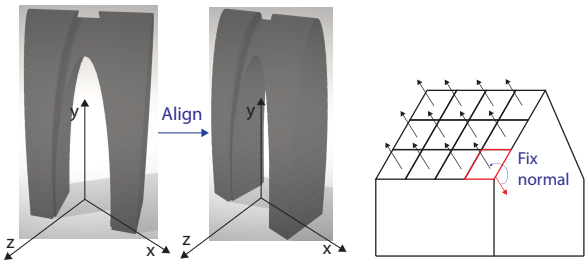

Figure 4: (a) A model which is axis-aligned. (b) Normals are facing away in this model.

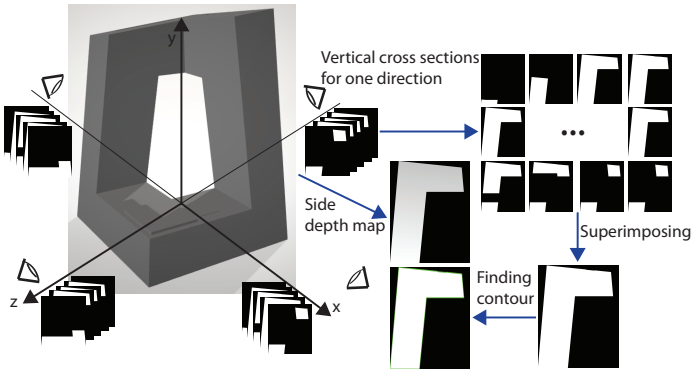

Figure 5: Computing silhouettes: superimposing vertical cross-sections and computing the contours. Computing side depth map: render depth map from the outside.

to left (along the x axis). For a regular or near-regular building (a building whose shape is in general symmetrical and box-like), silhouettes in four directions are usually sufficient to capture the important visual appearance of the building from side views. Many buildings also exhibit symmetrical property, thus silhouettes from a small number of directions is sufficient. If the building is non-regular and contains important visual features in many different directions, then we need to compute the silhouettes from more directions. Our silhouette computation algorithm for one direction consists of the steps (Figure 5) shown in Algorithm 1. Note that we introduce the main algorithm first, then the detailed respective techniques (Sections 3.3.1 to 3.3.3), which are shared in the later sections.

Input: One direction, one 3D building model.
Output: 2D silhouette with side depth map, with respect to the input direction.

1. We compute some vertical cross-sections (in our experiments, we compute 20 vertical cross-sections uniformly) from the center of the input building towards the input direction. (Section 3.3.1)

2. We superimpose them to form the silhouette image in this direction.

3. We compute the contour of the silhouette image and regard it as the silhouette in this direction. (Section 3.3.2)

4. We render from outside to get a side depth map of the input 3D building model in this direction (this will be used later in Section 3.5). We use the most exterior (nearest to the camera) point in this direction as near plane and the center as far plane. (Section 3.3.3)

Algorithm 1: Silhouette and side depth map computation

Input: Clipping direction, clipping point, one 3D building model, depth test is disabled
Output: 2D cross-section

1. To generate one cross-section, we first define the clipping plane using the clipping point and direction (the normal of the clipping plane is defined by the clipping direction) and set the stencil test to always pass (glStencilFunc(GL_ALWAYS, 1, 0xff)).

2. We render the front faces of the model using this clipping plane and increase the stencil values for the rasterized pixels in the stencil buffer (glStencilOp(GL_KEEP, GL_KEEP, GL_INCR)).

3. We render the back faces of the clipped model, and decrease the stencil values for the rasterized pixels in the stencil buffer (glStencilOp(GL_KEEP, GL_KEEP, GL_DECR)).

4. We render the front faces of the model again, and rasterize the pixels with a non-zero stencil value in the stencil buffer (glStencilFunc(GL_NOTEQUAL, 0, ¬0)). Those pixels form the cross-section image of that cutting position.

Algorithm 2: Cross-section computation

3.3.1. Cross-section computation method

We employ the stencil buffer [34] based algorithm to obtain cross-sections. This stencil buffer can control if a pixel is rasterized and its related operations are commonly available in graphical APIs like DirectX and OpenGL (we use OpenGL 4.6 in our implementation). We use orthogonal projection for all our rendering based techniques (Sections 3.3, 3.3.1 and 3.3.3), therefore we will not encounter perspective distortion. Please refer to Algorithm 2 for the algorithm. The basic idea is to mask the back faces using the front faces with the help of stencil buffer, the remaining back faces represent the cross-section, in other words, the cap of the clipped model. For performing post-processing antialiasing, we apply FXAA [46]. The pixelation artifacts because of rasterization can be reduced using antialiasing.

3.3.2. 2D contour computation method

For an input 2D cross-section image, the contours are computed using the function findContours() (based on [53]) from OpenCV [9]. With the purpose of getting less number of contour points, the function approxPolyDP() from OpenCV [9] (based on Ramer-Douglas-Peucker algorithm) are employed to approximate the contours. We specify a threshold for control-

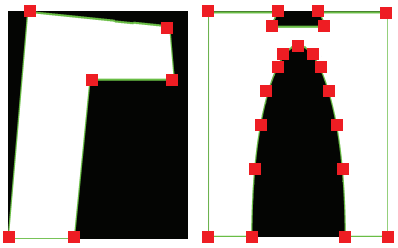

Figure 6: Examples of silhouette features.

ling the number of contour points c (we use 1% of arc length as the threshold in our experiments).

3.3.3. Depth map computation method

By rendering the model from a specified viewpoint, a depth map can be generated. The distances (depths) from the viewpoint to the surfaces of the model are recorded in the map. The per-fragment depth value we obtained in the fragment shader, is interpolated between far and near clipping planes linearly, because all rendering methods in our approach are using orthogonal projection. We use 32 bit resolution for our depth maps, thus each of them can encode 2^{32} values of distance (depth). In general, in our application, this is sufficient.

3.4. Computing horizontal cross-sections

Our method is based on utilizing and retaining the visual appearance. Conceptually, the computed silhouettes represent the vertical features, while the horizontal cross-sections represent the horizontal features. Assuming we extract m silhouettes $S_j (0 \leq j < m)$, the set F of all feature points is $F = \bigcup \{F_0, ..., F_{m-1}\}$, where $F_j (0 \leq j < m)$ is the set of feature points computed from silhouette S_j.

The silhouette feature points are determined as follows. For each computed silhouette S_j (one for each direction), we compute the features (corners) of the contour of this silhouette. Some examples are shown in Figure 6. The set F_j of feature points (corners) for silhouette S_j are computed by detecting the change in slope (second derivative) around one pixel of the contour of the 2D silhouette. If the change exceeds a threshold (t), we consider there is a feature at this silhouette pixel.

$$F_j = \left\{ \frac{d^2 S_{j_y}}{d^2 S_{j_x}} > t \right\} \tag{2}$$

Note that, S_j represents raw pixel values on the silhouette, and F_j represents the feature on the silhouette. Using the heights of the feature points computed from the silhouettes ($f_{i_y}, f_i \in F$), we compute the cross-sections (parallel to the ground) along the y axis. We use the same cross-section computation method as explained in Section 3.3.1. If two silhouette features have the same height, we separate them a bit by a small ϵ value to make sure all important silhouette features can be retained.

3.5. Connecting horizontal cross-sections

We use the same method as in Section 3.3.2 to compute the contours with a set of contour points for the horizontal cross-sections. For a building model, at one direction (with respect

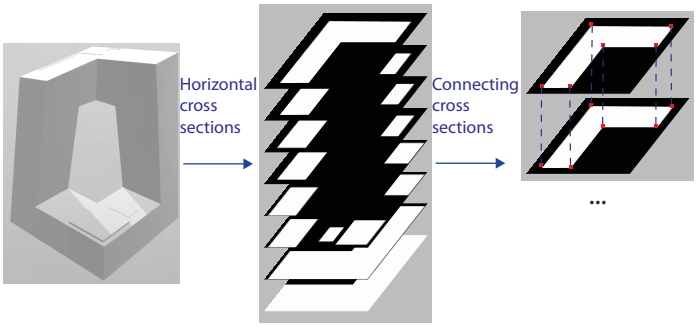

Figure 7: Horizontal cross-sections generation and connection.

to a silhouette, total four in our experiments, namely $z-$, $z+$, $x-$ and $x+$), we compute the corresponding contour points between two adjacent horizontal cross-sections (a lower one and an upper one) and connect them to form a closed mesh as the final output (Figure 7). The point correspondence computation is as follows.

Without loss of generality, we assume that cross-section C_b (with respect to feature f_b on the silhouette) has a greater number of contour points compared to cross-section C_a (with respect to feature f_a on the silhouette). For each contour point ($c_{b_i} \in \Omega(C_b)$) of cross-section C_b, we propose applying a depth-map-based mapping, to find a corresponding point on $\Omega(C_a)$. However, for feature points that are occluded or not visible, this means the side depth map does not capture their information, we then apply a spatial hashing instead to find the nearest corresponding contour point on cross-section C_a. If the nearest distance is greater than a threshold γ, we project c_{b_i} onto $\Omega(C_a)$. Similarly, for each unmapped point ($c_{a_j} \in \Omega(C_a)$, we map it onto $\Omega(C_b)$.

Depth-map-based mapping: For a contour point ($c_{b_i} \in \Omega(C_b)$) of cross-section C_b, we first find its corresponding side depth map. Among the four directions: $z-$, $z+$, $x-$ and $x+$, we choose the side depth map associated with the direction that has the minimum angle to vector $c_{b_{i_{xz}}} - O_{xz}$, where O is the center of the model. Suppose the corresponding side depth map of c_{b_i} is associated with $z-$ direction (similar computation for other directions), we sample the side depth map using $(c_{b_{i_x}}, f_{a_y})$ to get the z value of c_{b_i}'s corresponding point $c_{a_{i'}}$ in C_a, note as z_d. As such, the full 3D coordinates of $c_{a_{i'}}$ are $(c_{b_{i_x}}, f_{a_y}, z_d)$.

Spatial-hashing-based mapping: We first normalize these two cross-sections C_a and C_b, by non-uniformly scaling the upper cross-section to align with the lower one in four directions: $z-$, $z+$, $x-$ and $x+$. If there are multiple subsections in one cross-section, the normalization is conducted by considering them as one whole section. Note that this normalization is only performed for computing correspondences, not for connecting points to form the output mesh. The scaling factor to scale cross-section C_a (with respect to feature f_a on the silhouette) to align with cross-section C_b (with respect to feature f_b on the silhouette), in the direction $u \in \{z-, z+, x-, x+\}$, is computed based on:

$$\frac{(u_{f_b,max} - u_{f_b,min})}{(u_{f_a,max} - u_{f_a,min})}, \tag{3}$$

where $u_{f_b, max\,or\,min}$ is the u value (right or left) of the respective boundary (x or z value in the model coordinates) on the silhouette associated with the height of the feature f_b. In other words, we compute the widths of the silhouette at the heights of the features f_b and f_a, and scale the cross-section a accordingly.

We explain our projection method using the example of projecting a point c_{b_i} onto contour $\Omega(C_a)$. We use spatial hashing to find c_{b_i}'s nearest corresponding contour point $c_{a_j} \in \Omega(C_a)$. From c_{a_j}'s two neighboring contour points $c_{a_{j-1}}$, $c_{a_{j+1}}$, we compute and insert the corresponding point $c_{a_{i'}}$ in $\Omega(C_a)$:

$$\begin{aligned} c_{a_{i'}} &= \arg\min_{i'} \min_{k} \left\| c_{b_{i_{xz}}} - p_{k_{xz}} \right\|, \; 0 \leqslant k \leqslant 1, \\ p_k &\in \{proj_{c_{a_{j-1}} c_{a_j}}(c_{b_i}),\; proj_{c_{a_j} c_{a_{j+1}}}(c_{b_i})\}, \end{aligned} \tag{4}$$

where $proj_{c_{a_{j-1}} c_{a_j}}(c_{b_i})$ finds the closest point on line segment $c_{a_{j-1}} c_{a_j}$ to point c_{b_i}.

In the case if two cross-sections are only separated by ϵ in height, which means they are very close to each other, and exhibit a large change in the slope on the respective silhouette, we simply project (simply using the height of another cross-section) the contour points on the cross-section with a smaller area to the other cross-section with a larger area. This can be considered extruding the smaller cross-section onto the bigger cross-section. To ensure the two manifold property, for the bigger cross-section, a constrained Delaunay triangulation can be performed.

3.6. Roof refinement and conversion completion

Using the same method as in Section 3.3.3, from the top, we render the input 3D building model to obtain a depth (height) map. This roof depth map is used to refine the roof (top) of the connected cross-sections, because roof plays an important role in many geospatial applications. Instead of letting the roof vertices placed among various cross-sections, this step is helpful to ensure that a roof with vertices grouped together can be obtained.

From all computed silhouettes, we choose all feature points at the top. Based on the feature point with the lowest height (y) value, $f_{top, min}$, we apply a cut and ignore the rest of the features. We duplicate the cross-section corresponding to $f_{top, min}$, as the roof. For each contour point on the roof cross-section, based on its (x, z) value, we sample the corresponding roof height value from the roof depth map and use this to lift the roof contour point accordingly.

In our experiments, we adopt CityGML as the output format for our conversion. The output is LOD2-like: it has roof, wall and ground floor geometry as well as relative semantic information. Other than the roof, to complete the conversion to CityGML, the bottom (lowest) cross-section of the building model is triangulated and labeled as the ground floor, and the outer shell, which is constructed by connecting horizontal cross-sections (as mentioned in the previous section), is considered and labeled as the walls.

3.7. LOD control

The proposed LOD control is integrated into the model conversion framework. It aims to reduce the model complexity (the input model is regarded as the highest LOD) according to the need of a specific geospatial application. In other words, we generate lower LOD from the highest LOD. The proposed method focuses on 2D appearance as follows.

1. Side view: silhouette LOD (vertical). This LOD controls the silhouette features (F) for computing cross-sections.
2. Outer shell: cross-section LOD (horizontal). This LOD controls the contour points of the cross-section to reconstruct the outer shell.

Lower LOD leads to less number of features/points while retaining the important ones as many as possible. These features/points are basically features (corners) in the contours. The essential visual properties of a feature (corner) are its size and sharpness. A bigger and sharper feature (corner) usually attracts more attention, thus we consider it as more important. Both two LOD controls mentioned above are using the same 2D technique, which consists of two image processing steps as follows.

1. Image blurring: we first blur the silhouette or cross-section contour in order to reduce features (corners) that are relatively small in size even though they can be sharp. We implement this image blurring based on image dilation then erosion (dilate() and erode() in OpenCV [9]). By setting the parameters to control the kernel size and number of iterations, we can control the LOD in terms of the feature (corner) size.
2. Corner detection: we then perform corner detection and ignore features (corners) that are relatively less sharp. We implement this in Equation 2, by setting the threshold t, we can control the LOD in terms of the feature (corner) sharpness.

Note that the contour approximation method (approxPolyDP()) mentioned in Section 3.3.2 acts as an initialization for the contour points. By setting its threshold parameter for the approximation, we can control the LOD in terms of the overall number of feature points.

4. Results and discussions

In our experiments, the inputs are polygon-soup 3D buildings in OBJ format. The conversion outputs are two-manifold models in CityGML format, which is a popular geospatial data format. The conversion results of various representative and challenging models of buildings such as CCTV building (China, Beijing, Figure 3), Marina Bay Sands (MBS) (Singapore, with LODs, Figure 8), China Pavilion Expo (China, Shanghai), a Tower, a Bridge (Figure 9), Temple of ancient Greek, East Gate (China, Suzhou), Arc de Triomphe (France,Paris, with LODs, Figure 10), show the effectiveness of our proposed method. These models were originally created by 3D modelers for visualization purposes. We adopt the FZKViewer [39] from KIT to visualize our generated CityGML buildings.

Our method was implemented on a PC with Intel E5 2.6GHz Xeon CPU, 16.0G RAM and Nvidia K5000 GPU. The rasterization components of the proposed method were programmed

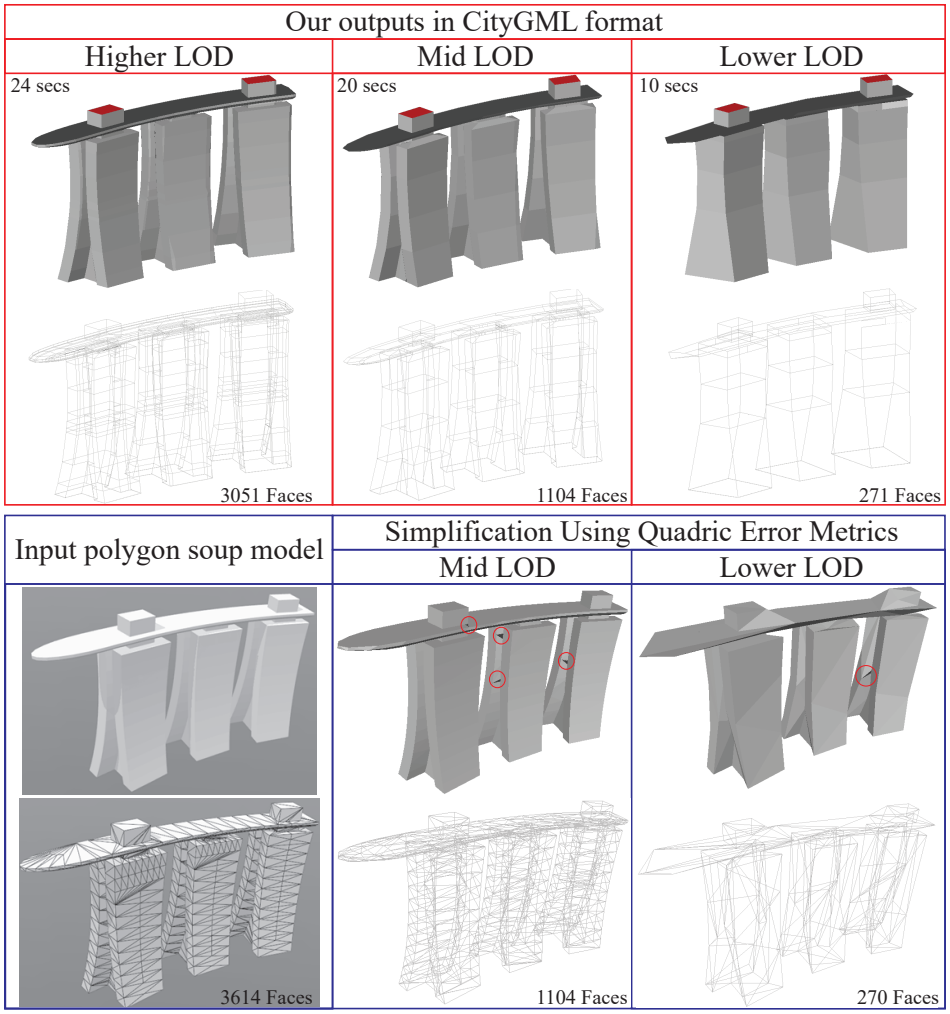

Figure 8: Comparison of LODs generation. Upper: our generated different LODs in CityGML format, the numbers of faces are counted in OBJ format which is converted from the generated CityGML file. Lower: input model and the generated LODs in OBJ format by simplification using quadric error metrics [22], artifacts are circled in red.

in OpenGL and C++. The other components, like outputting CityGML, 2D image processing, were programmed in Python. In our experiments (as shown in Figure 10), each model conversion of building or LODs generation took only 10 to 40 seconds.

Evaluations: (1) Our model conversion approach is evaluated by comparing our outputs (e.g. in Figures 3, 8, 9 and 10 (Higher LOD column)) with the input buildings. In these results, we can observe that the important visual appearances of the input buildings can be preserved, for example, the gate shape, the arc shape and the special shape of the MBS building. However, using the conventional method based on extrusion from ground print to generate CityGML models, it is often a challenging task to preserve such shapes. Moreover, our proposed method does not need the information of polygon connectivity to perform geometric processing, therefore it is capable of dealing with polygon-soup buildings which can be problematic. For instance, the temple of ancient Greek model, which contains self-intersections, can be handled using our method. Our method is also evaluated by checking the two manifold property in our outputs by loading them to MeshLab [13]. This property can be ensured in our outputs.

Besides geometry-wise evaluation, it is also important to evaluate the generated CityGML models from geospatial perspective [56]. Besides loading and viewing them using FZKViewer [39], they are validated using val3dity [44] and the validation results show: all 3D primitives are valid. They are also validated using CityDoctor [11] and no errors are observed.

(2) Our LODs generation method is evaluated by comparing our generated LODs (Figure 10) with the input building models. We also compare our method with a conventional method, which is LODs generation by simplification using quadric error metrics [22] (Figure 8). As shown in Figures 8 and 10, when reducing LOD, our proposed method is helpful to preserve the important visual appearance. As shown in Figure 8, our appearance-based method can achieve a comparable simplification result as [22]. Moreover, our method can handle polygon-soup buildings, which are in general difficult to be processed and may produce unwanted artifacts (circled in red in

Figure 8) using existing methods, due to the lack of connectivity information. Furthermore, different from the conventional methods, our computation is only performed in 2D instead of 3D.

The effectiveness and simplicity of the proposed method allow it to be used as a practically useful technique to convert an arbitrary 3D models of buildings for visualization to 3D models for geospatial applications while controlling their LODs.

Limitations: Because the proposed method is appearance-driven based on viewing results and we only compute silhouettes from a certain number of views, some features which are not visible in these views, may be missed. The silhouettes are generated in four directions in our experiments. For regular or near-regular buildings, which are box-like, this is in general sufficient. In order to capture more features, users can choose to compute silhouettes in more directions. However, in this case, more computational costs are needed. Similarly, the users can also choose to increase the number of horizontal cross-sections.

Furthermore, within one horizontal cross-section, the features could be complex, this may cause the difficulty of computing the correspondences between adjacent cross-sections. In the proposed method, the computation of correspondence for features which are not visible from side views, is still realized using the nearest matching. Therefore, for buildings containing twisted walls, the proposed method might not be able to obtain the right correspondences between adjacent cross-sections. A UI to further refine the correspondence computation can be provided to the users to resolve this issue.

5. Conclusion and future work

We have proposed an appearance-driven approach to convert models of buildings in polygon-soup representation to models, which are two-manifold and suitable for 3D geospatial applications, as well as to interactively control its LOD. The inputs were initially modeled with the purposes of visualization. The difficult task of handling models in polygon-soup representation with incomplete information of connectivity or internal self-intersections is tackled by extracting and using the visual appearances of the models.

We proposed first identifying the feature points of an input 3D building model from its silhouettes generated from a number of views. According to these feature points, a number of horizontal cross-sections are then computed. After this, adjacent horizontal cross-sections are connected based on the computed side depth map to construct a mesh, which has no internal self-intersections and is two-manifold. The resulting mesh is more well-suited for geometric processing hence 3D geospatial usages. In one unified appearance-driven framework, we also proposed and integrated a LOD control method using a 2D approach by controlling the details in the silhouettes and horizontal cross-sections.

As a future work, we plan to improve our method to handle one whole city with a large scale of buildings as the input, currently, we still need to process the buildings one by one. We also plan to generalize the proposed approach to deal with different forms of 3D models like vegetation, terrain and furniture.

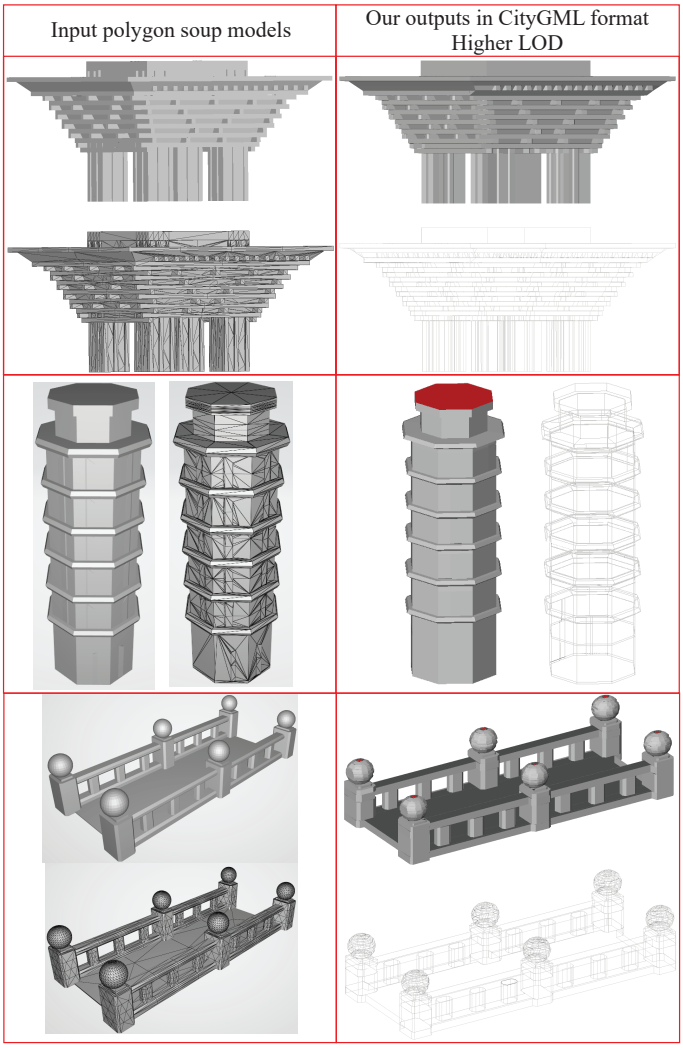

Figure 9: Some results. Left: input OBJ models in polygon-soup representation. Right: our results (visualized in FZKViewer [39] from KIT) in the format of CityGML. Top-down: China Pavilion Expo; A tower; A bridge.

Another possible future work is to utilize the proposed method in the area of additive manufacturing.

Acknowledgments: We gratefully thank the reviewers for their constructive comments. This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under the Virtual Singapore Programme.

References

[1] Atazadeh, B., Kalantari, M., Rajabifard, A., Ho, S., Champion, T., 2017. Extending a bim-based data model to support 3d digital management of complex ownership spaces. International Journal of Geographical Information Science 31, 499–522.

[2] Autodesk, a. 3D Max. www.autodesk.com/products/3ds-max (visited on 20/11/2019).

[3] Autodesk, b. AutoCAD. www.autodesk.com/products/autocad (visited on 20/11/2019).

[4] Autodesk, c. Maya. www.autodesk.com/products/maya (visited on 20/11/2019).

[5] Autodesk, d. Rhinoceros 3D. www.rhino3d.com (visited on 20/11/2019).

[6] Biljecki, F., Ledoux, H., Stoter, J., 2016. An improved lod specification for 3d building models. Computers, Environment and Urban Systems 59, 25–37.

[7] Blender Foundation, . Blender. www.blender.org (visited on 20/11/2019).

[8] Bogue, R., 2013. 3d printing: the dawn of a new era in manufacturing? Assembly Automation 33, 307–311.

[9] Bradski, G., Kaehler, A., 2000. Opencv. Dr. Dobb's journal of software tools 3.

[10] Campen, M., Attene, M., Kobbelt, L., 2012. A practical guide to polygon mesh repairing., in: Eurographics (Tutorials).

[11] Casper, E., . CityDoctor. www.citydoctor.eu (visited on 20/11/2019).

[12] Chen, K., Johan, H., Erdt, M., 2018. An appearance-driven method for converting polygon soup building models for 3d geospatial applications, in: 2018 International Conference on Cyberworlds (CW), pp. 383–390.

[13] Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G., 2008. MeshLab: an Open-Source Mesh Processing Tool, in: Sixth Eurographics Italian Chapter Conference, The Eurographics Association. pp. 129–136.

[14] Cignoni, P., Montani, C., Scopigno, R., 1998. A comparison of mesh simplification algorithms. Computers & Graphics 22, 37–54.

[15] CityGML Standard, . www.opengeospatial.org/standards/citygml (visited on 20/11/2019).

[16] Donkers, S., Ledoux, H., Zhao, J., Stoter, J., 2016. Automatic conversion of ifc datasets to geometrically and semantically correct citygml lod3 buildings. Transactions in GIS 20, 547–569.

[17] Dorninger, P., Pfeifer, N., 2008. A comprehensive automated 3d approach for building extraction, reconstruction, and regularization from airborne laser scanning point clouds. Sensors 8, 7323–7343.

[18] El-Mekawy, M., Östman, A., Hijazi, I., 2012. A unified building model for 3d urban gis. ISPRS International Journal of Geo-Information 1, 120–145.

[19] Ericson, C., 2004. Real-Time Collision Detection. CRC Press, Inc., Boca Raton, FL, USA.

[20] Fan, H., Meng, L., 2009. Automatic derivation of different levels of detail for 3d buildings modeled by citygml, in: ICC'09: In Proceedings of 24th International Cartography Conference, pp. 15–21.

[21] Garcia-Dorado, I., Demir, I., Aliaga, D., 2013. Technical section: Automatic urban modeling using volumetric reconstruction with surface graph cuts 37, 896–910.

[22] Garland, M., Heckbert, P.S., 1997. Surface simplification using quadric error metrics, in: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co.. pp. 209–216.

[23] Geiger, A., Benner, J., Haefele, K.H., 2015. Generalization of 3d ifc building models, in: 3D Geoinformation Science. Springer, pp. 19–35.

[24] Google, . SketchUp. www.sketchup.com (visited on 20/11/2019).

[25] Gregory, J., 2014. Game Engine Architecture, 2 edition. A K Peters/CRC Press.

[26] Gröger, G., Plümer, L., 2012. Citygml–interoperable semantic 3d city models. ISPRS Journal of Photogrammetry and Remote Sensing 71, 12–33.

[27] Haala, N., Kada, M., 2010. An update on automatic 3d building reconstruction. ISPRS Journal of Photogrammetry and Remote Sensing 65, 570 – 580. ISPRS Centenary Celebration Issue.

[28] Hao, J., Fang, L., Williams, R.E., 2011. An efficient curvature-based partitioning of large-scale stl models. Rapid Prototyping Journal 17, 116–127.

[29] Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A., 2013. Procedural content generation for games: A survey. ACM Trans. Multimedia Comput. Commun. Appl. 9, 1:1–1:22.

[30] Henn, A., Gröger, G., Stroh, V., Plümer, L., 2013. Model driven reconstruction of roofs from sparse lidar point clouds. ISPRS Journal of photogrammetry and remote sensing 76, 17–29.

[31] Hoppe, H., 1996. Progressive meshes, in: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, ACM, New York, NY, USA. pp. 99–108.

[32] Howard, R., Bjork, B.C., 2007. Building information models–experts' views on bim/ifc developments, in: Proceedings of the 24th CIB-W78 Conference, pp. 47–54.

[33] Jiang, N., Tan, P., Cheong, L.F., 2009. Symmetric architecture modeling with a single image. ACM Trans. Graph. 28, 113:1–113:8.

[34] John Kessenich, Graham Sellers, and Dave Shreiner, 2016. The OpenGL Programming Guide, in: The OpenGL Programming Guide, Addison Wesley Professional.

[35] Ju, T., 2004. Robust repair of polygonal models 23, 888–895.

[36] Ju, T., 2009. Fixing geometric errors on polygonal models: a survey. Journal of Computer Science and Technology 24, 19–29.

[37] Kang, T., Hong, C.H., 2015. Ifc-citygml lod mapping automation based on multi-processing, in: Proceedings of the 32nd International Symposium on Automation and Robotics in Construction (ISARC), pp. 1–8.

[38] Kelly, G., Mccabe, H., 2006. A survey of procedural techniques for city generation 14.

[39] KIT IAI, . FZKViewer.

[40] Kolbe, T.H., 2009. Representing and exchanging 3d city models with citygml, in: 3D geo-information sciences. Springer, pp. 15–31.

[41] Kolbe, T.H., 2012. Bim, citygml, and related standardization, in: Proceedings of the 2012 Digital Landscape Architecture Conference, Bernburg/Dessau, Germany.

[42] de Laat, R., Van Berlo, L., 2011. Integration of bim and gis: The development of the citygml geobim extension, in: Advances in 3D geo-information sciences. Springer, pp. 211–225.

[43] Lafarge, F., Mallet, C., 2011. Building large urban environments from unstructured point data, in: 2011 International Conference on Computer Vision, pp. 1068–1075.

[44] Ledoux, H., 2018. val3dity: validation of 3d gis primitives according to the international standards. Open Geospatial Data, Software and Standards 3, 1.

[45] Liu, K., Chen, J., Wang, S., Zhu, X., 2014. Procedural modeling of buildings based on facade image segmentation, in: 2014 International Conference on Audio, Language and Image Processing, pp. 797–801.

[46] Lottes, T., 2009. FXAA, in: NVDIA White Paper.

[47] Louisiana Entertainment, . www.turbosquid.com (visited on 20/11/2019).

[48] Luo, L., Baran, I., Rusinkiewicz, S., Matusik, W., 2012. Chopper: Partitioning models into 3D-printable parts. ACM Trans. Graph. 31.

[49] Malamboa, L., Hahnb, M., 2010. Lidar assisted citygml creation. AGSE 2010 13.

[50] Müller, P., Wonka, P., Haegler, S., Ulmer, A., Van Gool, L., 2006. Procedural modeling of buildings. ACM Trans. Graph. 25, 614–623.

[51] Poullis, C., You, S., 2009. Photorealistic large-scale urban city model reconstruction. IEEE Transactions on Visualization and Computer Graphics 15, 654–669.

[52] Schroeder, W.J., Zarge, J.A., Lorensen, W.E., 1992. Decimation of triangle meshes, in: ACM siggraph computer graphics, ACM. pp. 65–70.

[53] Suzuki, S., Abe, K., 1985. Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing 30, 32–46.

[54] Toshev, A., Mordohai, P., Taskar, B., 2010. Detecting and parsing architecture at city scale from range data, in: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 398–405.

[55] Trimble Inc, . www.3dwarehouse.sketchup.com (visited on 20/11/2019).

[56] Wagner, D., Alam, N., Wewetzer, M., Pries, M., Coors, V., 2015. Methods for geometric data validation of 3d city models. International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences 40.

[57] Xie, J., Feng, C.C., 2016. An integrated simplification approach for 3d buildings with sloped and flat roofs. ISPRS International Journal of Geo-Information. 5, article number:128.

[58] Zhao, J., Ledoux, H., Stoter, J., Feng, T., 2018. Hsw: Heuristic shrink-wrapping for automatically repairing solid-based citygml lod2 building models. ISPRS Journal of Photogrammetry and Remote Sensing 146, 289–304.

[59] Zhao, Z., Ledoux, H., Stoter, J.E., 2013. Automatic repair of citygml lod2 buildings using shrink-wrapping, in: ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, pp. II–2/W1, 309–317.

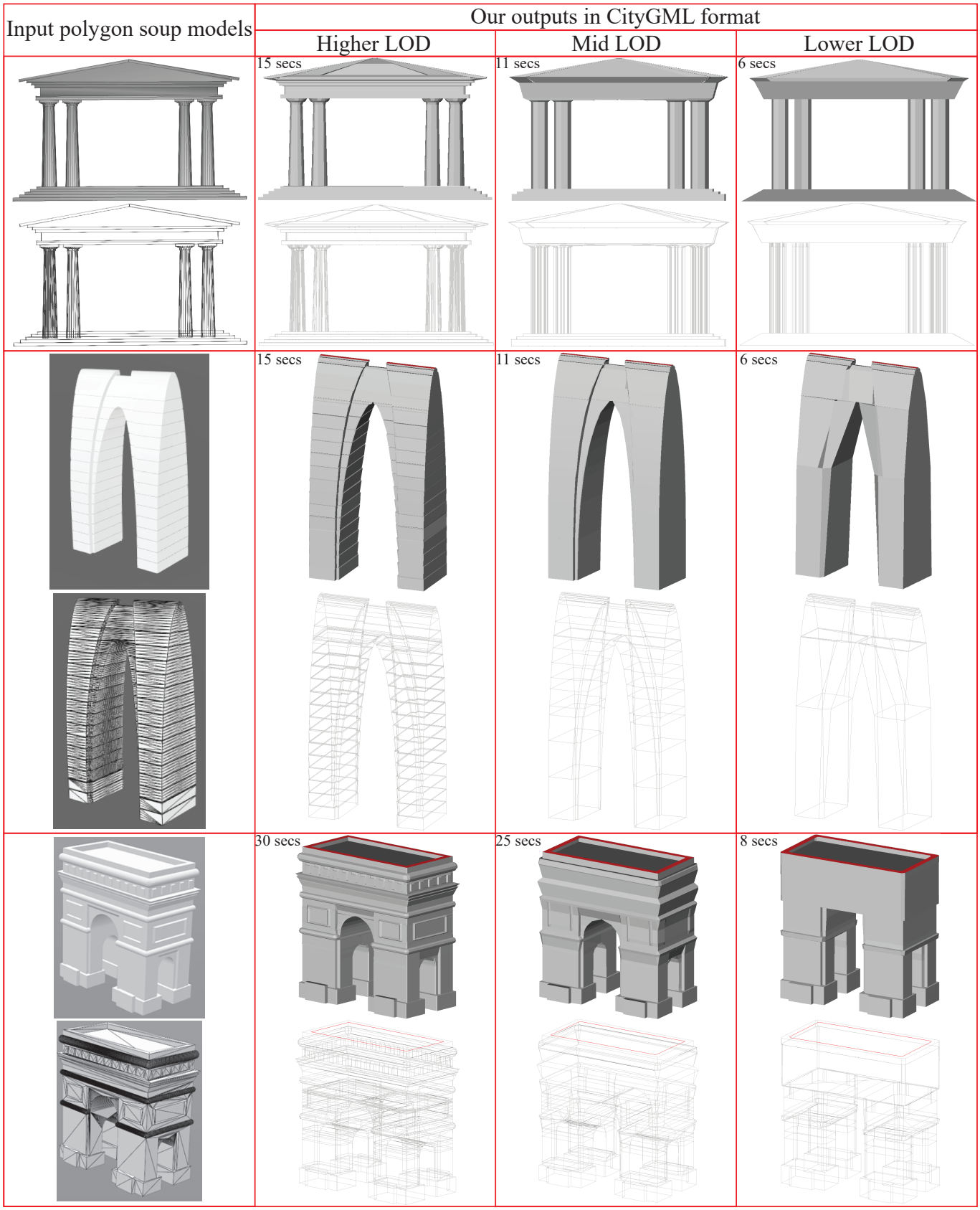

Figure 10: Some results. Left: input OBJ models in polygon-soup representation. Right: our results (visualized in FZKViewer [39] from KIT) in different LODs in the format of CityGML. Top-down: Temple of ancient Greek; East Gate; Arc de Triomphe. The conversion time is also shown.