An Appearance-Driven Method for Converting Polygon Soup Building Models for 3D Geospatial Applications

Kan Chen Henry Johan Marius Erdt
Fraunhofer Singapore, Nanyang Technological University
{kchen1, henryjohan, merdt}@ntu.edu.sg

***Abstract*—Polygon soup building models are fine for visualization purposes such as in games and movies. They, however, are not suitable for 3D geospatial applications which require geometrical analysis, since they lack connectivity information and may contain intersections internally between their parts. In this paper, we propose an appearance-driven method to interactively convert an input polygon soup building model to a two-manifold mesh, which is more suitable for 3D geospatial applications. Since a polygon soup model is not suitable for geometrical analysis, our key idea is to extract and utilize the visual appearance of the input building model for the conversion. We extract the silhouettes and use them to identify the features of the building. We then generate horizontal cross sections based on the locations of the features and then reconstruct the building by connecting two neighbouring cross sections. We propose to integrate various rasterization techniques to facilitate the conversion. Experimental results show the effectiveness of the proposed method.**

***Keywords*-3D building model conversion, polygon soup, two-manifold, appearance, rasterization, 3D geospatial**

I. Introduction

3D building models are widely used for visualization purposes such as in architecture, virtual reality, 3D game, movie, and driving simulator. They are also required for various 3D geospatial analysis and simulation applications such as in urban planning/management, traffic simulation, noise/wind/flood simulation, and building shadow analysis. Geospatial queries and simulations usually require complete polygon (or vertex) connectivity information of the 3D building models to conduct geometrical analysis. Such analysis is usually performed based on the exterior of the building. For example, geometrical analysis is required for users to query all the polygons of a wall of a building model. Similarly, in the case of flood simulation, such analysis is also required to retrieve the correct neighboring vertices, when the flood hits a vertex (point) of a building model. In other words, geometry-wise, geospatial applications require two-manifold, non-self-intersecting 3D building models.

With the tremendous growth in digital entertainment, 3D building models for visualization purposes are largely available as shown in Figure 1. It would be beneficial if we can use these models for geospatial applications. However, 3D building models for visualization purposes are created with correct appearance, but most of them are in polygon soup representation. As a result, they are not suitable for geometrical analysis. For example, for the model in Figure 2(a), the pillar is usually modeled only once, then it is duplicated and simply placed over the base floor. Although the model looks correct, the pillars may penetrate through the base floor or the roof. This also results in the lack of connectivity information between the polygons of the pillars and the floor/roof. In other words, geometry-wise, the 3D building models for visualization purposes, can be polygon soup meshes, and they may be non-two-manifold or containing intersecting parts internally. As such, they cannot be directly used for geospatial applications. At the same time, because of the lack of polygon connectivity information, it is challenging to convert or repair them, such processes usually require tedious manual efforts to fix the intersections as well as incorrect polygon connectivity.

(a) Paris city scene in game: Assassin's Creed Unity

(b) Los Angeles/Los Santos in game: Grand Theft Auto V

Fig. 1. 3D building models for visualization purposes are largely available.

Furthermore, existing 3D geospatial building model gener-

978-1-5386-7315-7/18/$31.00 ©2018 IEEE
DOI 10.1109/CW.2018.00075

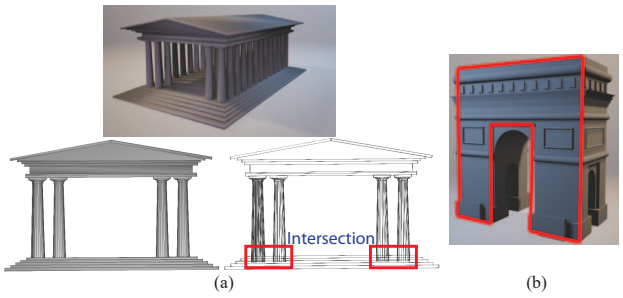

Fig. 2. (a) A non-two-manifold building model. (b) A building model which cannot be generated by simply extruding the ground print to the roof.

ation may also lose some important features of the original buildings. For example, CityGML is a popular data model format to represent digital 3D models of buildings and cities. However, besides conducting geometry processing for an input 3D building model, the most common method of CityGML model generation is based on direct extrusion from a ground print to the roof [1], [2]. This approach will not be able to generate the arc in Figure 2(b). It is essential to be able to preserve such features (e.g. arc), which can be important for viewing and geospatial simulation.

In this paper, we propose a new approach to convert a polygon soup 3D building model to a two-manifold model suitable for 3D geospatial applications (e.g. in CityGML format). By using our method, we can save the efforts for re-creating existing building models again or manually repairing them for 3D geospatial applications. Our proposed method focuses on the exterior of the 3D building model. It has the following main features:

- Observation 1: The visual appearance is the most important attribute in a polygon soup 3D model. Furthermore, it is difficult to perform geometrical analysis for a polygon soup model due to the incomplete polygon connectivity information. However, such model has been created with correct visual appearance. In other words, the rasterized images (rendering results) are correct. Therefore, we propose to extract and utilize the visual appearance to facilitate our model conversion by employing computer graphics tools, in particular, the rasterization methods.

  We propose to extract and use three types of visual appearance: silhouettes (side views), cross sections (outer shell), and height map (top view).

- Observation 2: Our method is based on observation that building models are usually standing upright, and there are many similarities vertically among levels. As such, one cross section of a building model can be used to represent multiple similar vertical levels in the building model.

  Based on the computed the silhouettes of the model, which are used to identify the features of the building, we propose to compute horizontal cross sections parallel to the ground and then connect them.

- Instead of geometrical processing using polygon connectivity information, our method is based on rasterization techniques. As a result, an arbitrary polygon soup 3D building mesh can be converted to a two-manifold model suitable for 3D geospatial applications (e.g. in CityGML format). Our method also can be considered as a method to repair an arbitrary 3D building mesh by resolving inner self-intersections and ensuring two-manifold property.

II. Related work

In this section, we review the prior work: 3D building model representation, generation, conversion and repair.

A. 3D building model representation and generation

3D building models are used for a wide spectrum of different purposes. They can be mainly categorized as visualization, computer aided design (CAD) and geospatial applications.

1) 3D building models for visualization: The common visualization applications are games, virtual reality simulators and movies. There are mainly three types of representation, polygonal representation [3]–[5], volumetric representation [6], imagery representation [7], [8].

3D polygonal mesh is the most commonly used representation in 3D visualization applications (e.g., OBJ, Collada and FBX)). 3D polygonal buildings are usually created by 3D artists using 3D modelling software, such as 3D Max [9], Maya [10] and Blender [11]. There are also many existing software plug-ins and automatic tools [6], [12]–[14], which are usually based on procedural heuristics, were developed to help the creation of 3D polygonal buildings. Muller et al. [12] proposed a shape grammar to procedurally generate buildings. Liu et al. [14] combined shape grammar and image processing to automatically generate buildings. This procedural approach is suitable to create a large scale of buildings, such as a city, however, it is usually more artificial and less realistic. Another common approach is based on 3D acquisition and reconstruction via laser scanning or photogrammetry [6], [15]. Lafarge and Mallet [15]'s non-convex energy minimization method models cities from unstructured point data. Toshev et al. [16] proposed to detect and parse building structures from unorganized 3D point clouds and construct a compact and hierarchical representation. These approaches usually require high computational costs and manual efforts to obtain a building model from an acquired 3D point clouds.

3D building models for visualization are usually created with the purpose of impressing viewers, for example, gamers, with visually correct and immersive appearance. They are not created for the ease of conducting geometrical analysis or computation, therefore such models may have inner self-intersections and limited polygon connectivity information. In terms of representation, it is very common that they are in the form of polygon soup. As such, in many real-time

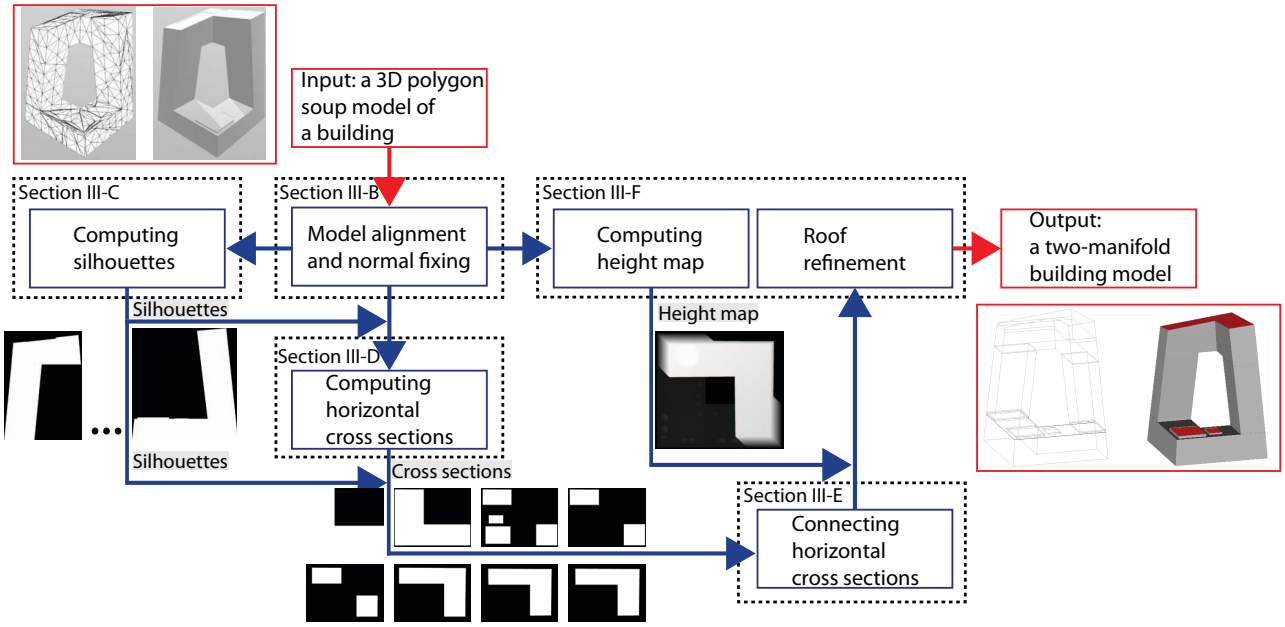

Fig. 3. The steps of the proposed method.

applications such as games, the computations, such as game physics computation and AI query, are usually performed on another type of simplified representation, which is dedicated for real-time computation [17], [18], e.g., spheres and boxes. Because game and movie industries are huge and rapidly evolving, 3D visualization models are largely available. Many realistic 3D building and city models can be found in 3D games, such as the Paris city in Assassin's Creed Unity. There are also many online 3D model databases and portals [19], [20], making 3D building models easily accessible to the users.

2) 3D building models for CAD: CAD building models are usually for architecture design and construction purposes. They are created and used by professionals such as architects, civil engineers and surveyors. 3D CAD buildings are usually created using professional CAD software, such as AutoCAD [21], SketchUp [22], Rhino [23] and etc.

For CAD building models, besides 3D geometry, detailed and accurate architectural data are created and integrated such as semantic meta data, measurement, building structural information, piping, electrical information and etc. Building information (BIM) models are widely used for architects [24], [25]. Industry Foundation Classes (IFC) standard [26] is a common format for BIM data.

3) 3D building models for geospatial applications: 3D building models for geospatial applications require the combination of 3D geometry and non-geometrical data, such as semantic or geographical information. These models are usually created through 3D geometry generation and geographical data integration. The 3D geometry is usually generated through 3D acquisition and reconstruction from laser scanned 3D point clouds, photogrammetry, or LIDAR data [[27]–[29]. The geographical data will then be integrated, such as the building semantic information (roof, wall), for example, CityGML representation [30], [31]. The integration is usually performed based on geographical references such as GPS, building footprints and building IDs. Size-wise, they are usually in large scale, such as in city-scale. 3D building models for geospatial applications are usually required to have complete polygon connectivity information to perform geometrical analysis, such as flood simulation.

B. 3D model conversion

There are many existing conversion tools developed for two-way conversion between CAD models (e.g. IFC models) and geospatial data models (e.g. CityGML). CAD models have semantic information and they usually have correct 3D geometry, therefore CAD models are easier to be converted for geospatial applications [32]–[36].

3D geometry can be directly fetched from geospatial models (e.g. in CityGML format), and can be applied for visualization applications. On the other hand, since polygon soup meshes have limited polygon connectivity information, it is a challenging problem to convert them for geospatial applications. Less work has been done to address this problem. Zhao et al. [37] proposed to construct bounding surfaces to repair polygonal meshes for CityGML. However, some visual appearance may be lost. Donkers et al. [36] proposed a voxel based method. However, the volumetric representation requires additional computation and storage costs. This method may also introduce sampling artifacts to the input model, thus it is difficult to preserve the shape. A repair step is usually needed to clean the model. Existing repairing methods basically directly detect and repair artifacts on the surfaces of the input models ([38]–[40]). They can preserve the overall shape of the models. However, these methods usually require local continuity, in other words, they mainly handle smooth meshes. 3D building

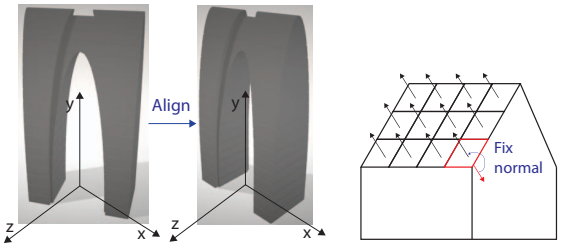

Fig. 4. (a) An axis-aligned model. (b) A model with normals facing away.

models usually lack such continuity, for example, walls are usually at 90 degrees. Therefore, the surface-based methods are not suitable for 3D building models.

III. Our proposed method

A. Basic idea and steps

Given a 3D polygon soup building model, we aim to convert it for geospatial applications. In this paper, we do not handle the interior of the building, we only focus on the exterior (outer shell) of the building model.

As explained previously, 3D polygon soup building models may have incomplete polygon connectivity information, which makes it difficult to apply geometrical analysis on them. The important and reliable information of the input polygon soup model is the appearance that is perceived by viewers. Our proposed method basically extracts the appearance, and we propose to achieve this by computing the outer shell of the building using rasterization techniques. By doing so, models suitable for geospatial applications can be generated without geometrical analysis.

The silhouettes S of a 3D building model capture its overall shape that is perceived by viewers: the viewers can usually tell the building based on the important features F on its silhouettes. Furthermore, we also observe that different levels of a building usually have many similarities. As such, for a given input polygon soup building model B, we propose to compute 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 from the ground.

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

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

Our proposed method consists of the following main steps as shown in Figure 3:

1) We align the input 3D model and fix its face normals (Section III-B).
2) We compute the silhouettes of the model and extract their features to determine the heights for computing horizontal cross sections (Section III-C).
3) We compute horizontal cross sections parallel to the ground (Section III-D) and then reconstruct the building by connecting adjacent cross sections (Section III-E).

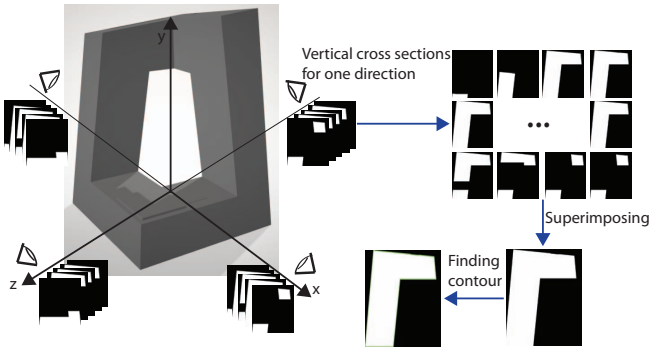

Fig. 5. Computing silhouettes: superimposing vertical cross sections and computing the contours.

4) We compute the height map from the top view of the model and use it to refine the roof of the building (Section III-F).

We introduce these steps in the following subsections.

B. Model alignment and normal fixing

We perform the following preprocessing procedures:

1) We align the model to let the up direction of the building model to align with the y axis and let the back-front direction of the building to align with the z axis. This is because we will compute the horizontal cross sections, we therefore need to ensure the model is standing upright.
2) We ensure the face normals are pointing outwards from the model. Since our cross section computation is based on face culling, faces should face away from the interior.

These two operations can easily be done in any common modelling software, such as 3Ds Max [9]. One preprocessed building model is shown in Figure 4.

C. Computing silhouettes

We compute the silhouettes in four directions: front to back/back to front (along the z axis), and left to right/right to left (along the x axis). For a regular or near-regular building (a building whose shape is in general symmetrical and box-like), four direction silhouettes are usually sufficient to capture the important visual appearance of the building from side views. In other words, the silhouettes represent the vertical features of a building, e.g., from the silhouettes we can tell if a building is standing up right or leaning. 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.

1) Cross section computation method: To obtain the cross sections, we employ the stencil buffer (to control if a pixel is rasterized [41]) based algorithm that is commonly available in graphics APIs such as OpenGL and DirectX (our

Input: One 3D building model, one direction
Output: 2D silhouette with respect to the direction

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

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.

Algorithm 1: Silhouette computation

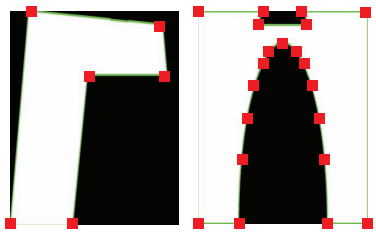

Fig. 6. Examples of silhouette features.

implementation is based on OpenGL 4.6). Note that all our rendering based methods (Sections III-C, III-C1 and III-F) are using orthogonal projection, thus no perspective distortion is generated. The algorithm is shown in Algorithm 2. We use FXAA to perform the postprocessing anti-aliasing [42]. This operation is helpful to reduce the pixelation artifacts due to rasterization. By doing so, there will be less noises in the silhouettes.

2) 2D contour computation method: We compute the contours (findContours() [43] in OpenCV library [44]) of a 2D cross section image. They are then approximated using the Ramer-Douglas-Peucker algorithm (approxPolyDP() in OpenCV library [44]) to get a smaller number of contour points. A threshold in this algorithm can be specified to control the number of contour points c (in our experiments, we use 1% of arc length as the threshold).

D. 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 computed 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

Input: One 3D building model, clipping point, clipping direction
Output: 2D cross section

1. To generate one cross section, we first define the clipping plane using the clipping point and direction (clipping plane's normal 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 again 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 clipped 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

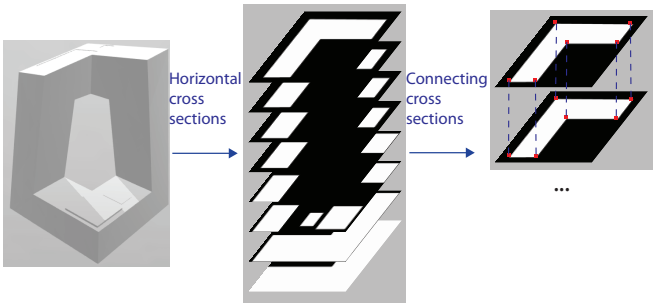

Fig. 7. Generating and connecting horizontal cross sections.

(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 III-C1. 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.

E. Connecting horizontal cross sections

We use the same method as in Section III-C2 to compute the contours with a set of contour points for the horizontal cross sections. For a building model, at one direction (with

respect to a silhouette, total four in our experiments, namely $x+$, $x-$, $z+$ and $z-$), 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.

We first normalize these two cross sections, by non-uniformly scaling the upper cross section to align with the lower one in four directions: $x+$, $x-$, $z+$ and $z-$. If a cross section contains multiple subsections, the normalization is performed 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 $u \in \{x+, x-, z+, z-\}$, 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 respective boundary's (right or left) u value (x or z value in the model coordinates) on the silhouette associated with feature f_b's height. In other words, we compute the widths of the silhouette at features f_b's and f_a's heights, and scale cross section a accordingly.

Without loss of generality, we assume that cross section C_b has a smaller number of contour points compared to cross section C_a. For each contour point ($c_{b_i} \in \Omega(C_b)$) of cross section C_b, we apply spatial hashing 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 project it onto $\Omega(C_b)$.

We introduce 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 $\Omega(C_a)$:

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

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 that 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 another cross sections' height) the contour points on the cross section with a smaller area to the other cross section with a bigger area. This also can be considered as extruding the smaller cross section onto the bigger cross section. We also perform constrained Delaunay triangulation for the bigger cross section to ensure the two manifold property.

F. Roof refinement

We render from the top to get a depth (height) map of the input 3D building model. We use this depth map to refine the roof (top) of the connected cross sections since in many geospatial applications, the roof plays an important role. This step ensures that we can obtain a roof with vertices grouped together instead of distributed among many cross sections.

Depth map is generated by rendering the model based on a given viewpoint. It records the distance of the surfaces of the model from a viewpoint. Since our rendering is based on orthogonal projection, the depth value per fragment we get in the fragment shader, is linearly interpolated between near and far clipping planes. The resolution of our depth map is 32 bit, so it can encode 2^{32} distance (depth) values which are sufficient in our application.

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 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 from the depth map to retrieve the corresponding roof height value and use this to lift the roof contour point accordingly. In addition, the bottom cross section is triangulated and labeled as the ground floor, and the outer shells are considered as the walls.

IV. Results and discussions

We applied our method to convert 3D polygon soup buildings in OBJ format to a geospatial data format, CityGML. We demonstrated the effectiveness of our method using some challenging and representative building models: CCTV building in Beijing, China (Figure 3), Marina Bay Sands (MBS) in Singapore, Arc de Triomphe in Paris, France, Ancient Greek temple and East Gate building in Suzhou, China (Figure 8). These models were created by 3D artists for games applications. Our results are visualized using the FZKViewer [45] from KIT.

We tested our method on a workstation with Intel Xeon E5-2650 2.6 GHz CPU, 16 G Memory and Nvidia Quadro K5000 GPU. The rendering parts of our method were implemented using C++ and OpenGL, the other parts, such as image processing and saving to CityGML format, were implemented using Python. In our experiments, each building model conversion took only 10 to 30 seconds. We evaluate our method by comparing our results with the input building models. The important visual appearance can be preserved in our results, such as the shape of the gate and the unique shape of the MBS building. It is usually difficult to preserve such shape using the conventional geospatial 3D model (e.g. in CityGML format) generation method, which is based on extrusion from ground prints. Furthermore, our method does not rely on polygon connectivity information to conduct geometrical analysis, thus our method can handle problematic polygon soup buildings. For example, we can handle the Ancient Greek temple model with self-intersections. We also evaluate our method by checking the two manifold property of our results using MeshLab [46].

Our method can ensure this property. The effectiveness of our method allows it to be used as a practical tool for converting arbitrary 3D building models for visualization to 3D models (e.g. in CityGML format) for geospatial applications.

Limitations: Since our method is appearance-driven, and based on computing silhouettes from a number of views, some features which are not visible in these views may be missed. In our experiments, we generated silhouettes from four directions. This is usually sufficient for regular or near-regular buildings. Increasing the number of views and more silhouettes will be helpful to capture more features. Similarly, users can also increase the number of horizontal cross sections to capture more features. Moreover, the features in a horizontal cross section can be complex, in this case, the correspondences between adjacent cross sections are not easy to compute. Our correspondence computation is based on nearest matching, thus if the outer walls are twisted, our method may fail to compute correct correspondences. A user interface can be provided to the users to further refine the correspondences.

V. Conclusion and future work

In this paper, we have presented an appearance-driven method for converting polygon soup building models, which are originally created for visualization purposes, to two-manifold models for 3D geospatial applications. We tackle the challenging problem of handling polygon soup models with limited connectivity information by utilizing their visual appearance. Based on the silhouettes of a 3D building model, we identify the feature points and compute a set of horizontal cross sections accordingly. We then connect adjacent horizontal cross sections to form a two-manifold mesh without inner part intersections, which is suitable for geometrical analysis thus 3D geospatial applications.

In the future, we plan to interactively generate and control the levels of details for the geometry of 3D building models. Another possible future work is to generalize our method to handle more types of 3D objects such as furniture, terrain and vegetation. Since they are also important components in a 3D city, this can be beneficial to 3D city geospatial applications. We also plan to apply our approach for 3D printing.

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

References

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

[2] J. Xie and C.-C. Feng, "An integrated simplification approach for 3d buildings with sloped and flat roofs." *ISPRS International Journal of Geo-Information.*, vol. 5, no. 8, p. article number:128, 2016.

[3] H. Hoppe, "Progressive meshes," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 99–108.

[4] N. Haala and M. Kada, "An update on automatic 3d building reconstruction," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 65, no. 6, pp. 570 – 580, 2010, iSPRS Centenary Celebration Issue.

[5] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1, pp. 1:1–1:22, Feb. 2013.

[6] I. Garcia-Dorado, I. Demir, and D. Aliaga, "Technical section: Automatic urban modeling using volumetric reconstruction with surface graph cuts," vol. 37, pp. 896–910, 11 2013.

[7] C. Poullis and S. You, "Photorealistic large-scale urban city model reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 4, pp. 654–669, Jul. 2009.

[8] N. Jiang, P. Tan, and L.-F. Cheong, "Symmetric architecture modeling with a single image," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 113:1–113:8, Dec. 2009.

[9] Autodesk. 3D Max.

[10] ——. Maya.

[11] Blender Foundation. Blender.

[12] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural modeling of buildings," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 614–623, Jul. 2006.

[13] G. Kelly and H. Mccabe, "A survey of procedural techniques for city generation," vol. 14, 01 2006.

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

[15] F. Lafarge and C. Mallet, "Building large urban environments from unstructured point data," in *2011 International Conference on Computer Vision*, Nov 2011, pp. 1068–1075.

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

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

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

[19] Trimble Inc. 3dwarehouse.sketchup.com.

[20] Louisiana Entertainment. turbosquid.com.

[21] Autodesk. AutoCAD.

[22] Google. SketchUp.

[23] Autodesk. Rhinoceros 3D.

[24] T. H. Kolbe, "Bim, citygml, and related standardization," in *Proceedings of the 2012 Digital Landscape Architecture Conference, Bernburg/Dessau, Germany*, vol. 31, 2012.

[25] B. Atazadeh, M. Kalantari, A. Rajabifard, S. Ho, and T. Champion, "Extending a bim-based data model to support 3d digital management of complex ownership spaces," *International Journal of Geographical Information Science*, vol. 31, no. 3, pp. 499–522, 2017.

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

[27] P. Dorninger and N. Pfeifer, "A comprehensive automated 3d approach for building extraction, reconstruction, and regularization from airborne laser scanning point clouds," *Sensors*, vol. 8, no. 11, pp. 7323–7343, 2008.

[28] L. Malamboa and M. Hahnb, "Lidar assisted citygml creation," *AGSE 2010*, vol. 13, 2010.

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

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

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

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

[33] M. El-Mekawy, A. Östman, and I. Hijazi, "A unified building model for 3d urban gis," *ISPRS International Journal of Geo-Information*, vol. 1, no. 2, pp. 120–145, 2012.

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

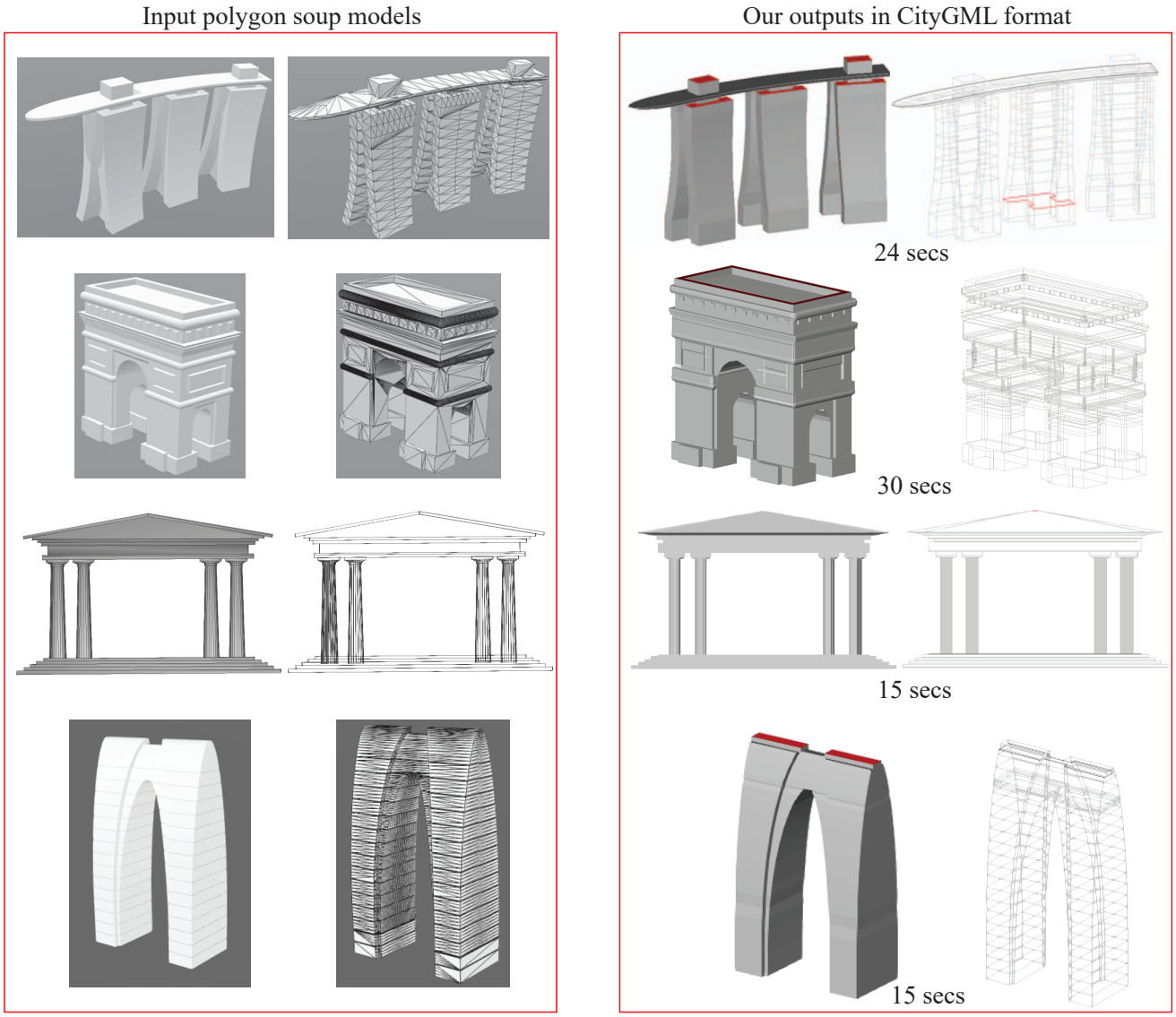

Fig. 8. Some results of the proposed method. Left: input polygon soup models in OBJ format. Right: our results in CityGML format using FZKViewer [45] from KIT. From top to bottom: Marina Bay Sands in Singapore; Arc de Triomphe in Paris, France; Ancient Greek temple; East Gate building in Suzhou, China. The conversion time is also shown.

[35] T. Kang and C. H. Hong, "Ifc-citygml lod mapping automation based on multi-processing," in *ISARC. Proceedings of the international symposium on automation and robotics in construction*, vol. 32. Vilnius Gediminas Technical University, Department of Construction Economics & Property, 2015, p. 1.

[36] S. Donkers, H. Ledoux, J. Zhao, and J. Stoter, "Automatic conversion of ifc datasets to geometrically and semantically correct citygml lod3 buildings," *Transactions in GIS*, vol. 20, no. 4, pp. 547–569, 2016.

[37] Z. Zhao, H. Ledoux, and J. Stoter, "Automatic repair of citygml lod2 buildings using shrink-wrapping," in *8th 3DGeoInfo Conference & WG II/2 Workshop, Istanbul, Turkey, 27–29 November 2013, ISPRS Archives Volume II-2/W1*. ISPRS, 2013.

[38] T. Ju, "Robust repair of polygonal models," in *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3. ACM, 2004, pp. 888–895.

[39] T. Ju, "Fixing geometric errors on polygonal models: a survey," *Journal of Computer Science and Technology*, vol. 24, no. 1, pp. 19–29, 2009.

[40] M. Campen, M. Attene, and L. Kobbelt, "A practical guide to polygon mesh repairing." in *Eurographics (Tutorials)*, 2012.

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

[42] T. Lottes, "Fxaa," in *NVDIA White Paper*, 2009.

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

[44] opencv.org. OpenCV Library.

[45] KIT, IAI. FZKViewer.

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

[47] Ubisoft. Assassin's Creed Unity.

[48] Rockstar Games. Grand Theft Auto V.