Fast and Robust Generation of City-Scale Seamless 3D Urban Models

Yanyan Lu, Evan Behar, Stephen Donnelly and Jyh-Ming Lien
Department of Computer Science, George Mason University

Fernando Camelli
Department of Computational Data Science, George Mason University

David Wong
Department of Earth Systems and GeoInformation Sciences, George Mason University

Abstract

Since the introduction of the concept of "Digital Earth", almost every major international city has been re-constructed in the virtual world. A large volume of geometric models describing urban objects has become freely available in the public domain via software like ArcGlobe and Google Earth. Although mostly created for visualization, these urban models can benefit many applications beyond visualization including city scale evacuation planning and earth phenomenon simulations. However, these models are mostly loosely structured and implicitly defined and require tedious manual preparation that usually takes weeks if not months before they can be used. Designing algorithms that can robustly and efficiently handle unstructured urban models at the city scale becomes a main technical challenge. In this paper, we present a framework that generates seamless 3D architectural models from 2D ground plans with elevation and height information. These overlapping ground plans are commonly used in the current GIS software such as ESRI ArcGIS and urban model synthesis methods to depict various components of buildings. Due to measurement and manual errors, these ground plans usually contain small, sharp, and various (nearly) degenerate artifacts. In this paper, we show both theoretically and empirically that our framework is efficient and numerically stable. Based on our review of the related work, we believe this is the first work that attempts to automatically create 3D architectural meshes for simulation at the city level. With the goal of providing greater benefit beyond visualization from this large volume of urban models, our initial results are encouraging.

Key words: Geometry processing, Model synthesis and repair, Urban model, Physically-based simulation

1. Introduction

Since the September 11 attacks in NYC in 2001; a toxic sludge disaster in Hungary in 2010; and the recent nuclear crisis in Fukushima, Japan, the ability to simulate large scale phenomena in urban environments (see Fig. 1) has become increasingly important to support scientific inquires and decision making. While numerical-computational models have advanced to the stage of accurately simulating various types of dynamic phenomena and replicating the reality, detailed geometric models representing the computation domain are still largely lacking [31,19].

Because of the collaborative efforts in the recent years, almost every building in the major US and international cities has been re-constructed in the virtual world. Although mostly created for visualization, these geometric models describing urban objects can potentially benefit many applications beyond visualization including video games, city scale evacuation planning, traffic simulation and natural or man-made phenomenon simulations. However, these urban models are mostly loosely structured and implicitly defined. These models require tedious manual preparation that usually takes weeks if not months before they can be used for simulation [19]. Therefore designing algorithms that can robustly and efficiently handle unstructured urban models at the city scale becomes a main technical challenge.

1.1. *Problem Statement*

Simulating large scale phenomena in urban environments usually requires techniques such as Computational Fluid Dynamics (CFD) to solve the Navier-Stokes equations numerically [3] in a tessellation of the computational domain. In this work, the computational domain is composed of a set of urban models. The tessellation process usually requires the computational domain to be represented as well-defined

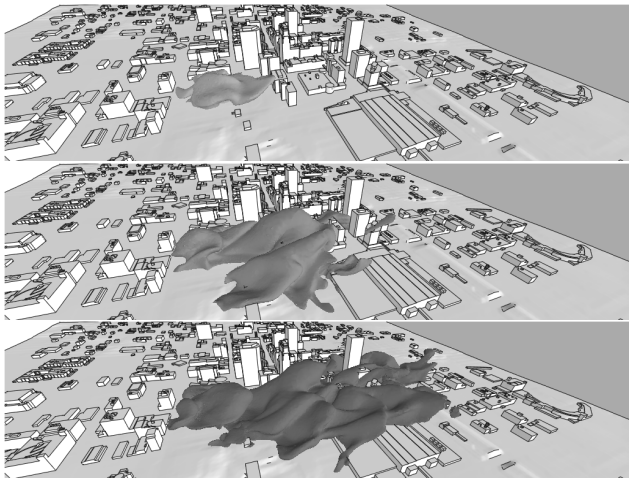
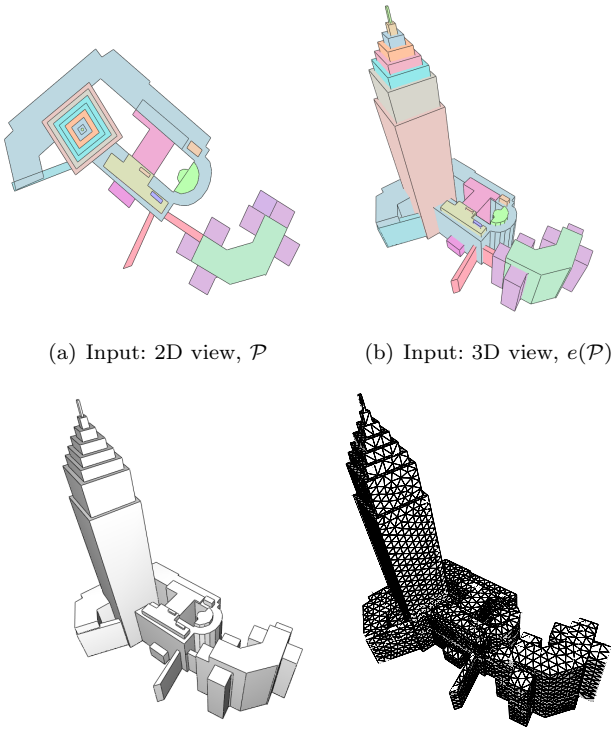


Fig. 1. A cloud is depicted at 3 different time instances in the integrated Oklahoma City model. The cloud is transported and diffused by the effects of the wind and turbulence. Clouds at 100 (top), 250 (mid) and 500 seconds (bottom) from the beginning of the release.



(a) Input: 2D view, $\mathcal{P}$ (b) Input: 3D view, $e(\mathcal{P})$

(c) Output: Seamless mesh, $\mathcal{B}$ (d) Remeshed for CFD

Fig. 2. An example input and output of our work.

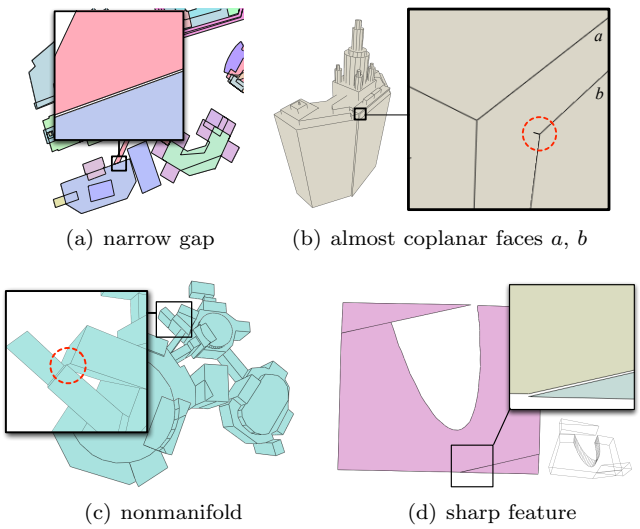
surfaces without holes [30]. There are other approaches to solve complex large urban areas without seamless surfaces, such as porosity [8], or embedded models [7]. These approaches are not as accurate as the seamless models. The main problem is the lack of well-defined surfaces to apply the right boundary conditions.

In this paper, we present a framework that generates seamless 3D architectural models (i.e., without boundary openings) from 2D ground plans with elevation and height information. These overlapping ground plans are commonly used in the current GIS software such as ESRI ArcGIS and urban model synthesis methods to depict various components of buildings. Specifically, our input is a set of 2D polygons $\mathcal{P}$ with elevation and height information. Each polygon $P \in \mathcal{P}$ can be elevated and extruded to form a 3D component $e(P)$ of a building. Here, we use the function $e(\cdot)$ to denote the transformation. Then, the set $e(\mathcal{P})$ is a collection of components that *implicitly* represent the 3D shape of buildings. For visualization applications, it is usually enough to keep the representation implicitly. However, to create simulation-ready representations our goal is to produce seamless polyhedral meshes from $\mathcal{P}$. Let us denote these building meshes as $\mathcal{B}$. An example in Figs. 2 (a-c) illustrates the problem that we will attempt to solve in this work.

1.2. *Main Challenges*

Creating seamless models for simulation is an extensively studied topic in the CAD community. Although, to our knowledge, creating seamless urban models at the city scale for simulation has not been done, it is no doubt that these two problems share many similarities. One example is performing robust boundary evaluation. Because the urban models are either generated (semi-)automatically from LiDAR and satellite images or created manually by non-experts, a quick analysis of the problem and its input data will reveal many degenerate cases and numerical stability issues in the boundary evaluation. Although the polygons $\mathcal{P}$ usually have simple shapes, the arrangement of the polygons and that of their 3D transformation $e(\mathcal{P})$ usually contain non-manifold geometries, sliver polygons, coplanar or nearly coplanar faces, small features, and sharp and very narrow gaps (see Fig. 3) because of the reconstruction and man-made errors. Unlike CAD, urban models also implicitly represent plazas, roads, streets, or alleys as the void regions between buildings. In many cases, these voids are created intentionally, thus should not be removed. However, some alleys or gaps between the buildings are too narrow. This produces degenerate cases and unwanted geometries, e.g., Fig. 3(a). These (nearly) degenerate cases not only hinder the boundary evaluation process, but also the efficiency of the simulation using the extracted mesh. Therefore, it is usually not enough to just consider each building individually. The relationship between the buildings should also be taken into account. In addition, it is usually not enough to design algorithms to cope with these degeneracies. It is more desirable to remove these degeneracies to prevent creating an excessive number of small geometries that usually do not provide any significance to result of the simulation but greatly reduce the simulation efficiency.

Another main technical challenge is to design algorithms that can both robustly and efficiently handle unstructured urban models at this scale. The boundary evaluation process in the boolean operations is known to be numerically unstable, and a robust implementation is usually slow. In particular, due to the scale of the problem, we have to per-



(a) narrow gap (b) almost coplanar faces a, b

(c) nonmanifold (d) sharp feature

Fig. 3. Examples of degenerate inputs.

form the operation on thousands of meshes representing the buildings. For example, naïvely computing the union and intersection between pairs of models can be inefficient.

1.3. *Main Contributions*

In this paper, we show both theoretically and empirically that our framework for creating seamless 3D urban models is efficient and numerically stable. In a nutshell, the proposed framework achieves this goal by changing the representation from the input $\mathcal{P}$ to *layers of disjoint polygons* (see Fig. 4). The core of framework is composed of two methods: (1) k-way boolean operations with adaptively adjusted numerical precision (Section 4) and (2) the Minkowski sum operation using the reduced convolution (Section 5). The k-way boolean operations merge polygon layers into a seamless model, and the Minkowski sum operation provides the foundation for model repair. Our experimental results on two datasets, New York City (5,397 buildings, Fig. 10) and Oklahoma City (358 buildings, Figs. 1 and 12), show that the proposed framework is fast, robust, and generates high quality results. Based on our review of the related work, we believe that this is the first work that attempts to create 3D architectural meshes at the city level automatically for simulation purposes. With the goal of providing greater benefit beyond visualization from this large volume of urban models, our initial results are encouraging.

2. Related Work

To the best of our knowledge, no previous work has focused on automatically processing urban models for simulation. Existing research often resorts to laborious manual manipulations of geometric data representing topography and buildings. Without a method for automatic preprocessing, this is generally necessary to produce a coherent and consistent geometric representation of the surface, including the landscape and buildings [19]. Although there exist many methods to construct, simplify and aggregate the models depicting the urban environments, almost all of these methods focus on the issues from the rendering aspect [27,9,20]. For example, ideas, such as levels of detail or texture mapping, based on the location of the view are useful for rendering but these tricks are no longer applicable to simulation.

An important step in the proposed work is to compute the union or intersection of many models. The problem of geometric boolean operations has been studied more than three decades and the main focus of the research has been on the *robustness* of the computation, because many numerical errors and degenerate cases can result in incorrect output. In addition to the robustness issues, one of the main challenges that we face in this work is the scalability of the boolean operation for city-scale urban models. A brief review of the techniques are discussed below.

2.1. *Urban Models Synthesis and Processing*

Almost all methods for urban model synthesis and processing focus on visualization. Most of these methods come the from Computer Graphics and GIS communities. For example, several methods based on procedural modeling focus on creating large-scale urban models [36,33,32] or individual buildings [46,37,16]. Noticeably, Parish and Müller [36] design two L-systems to procedurally construct streets and the buildings. The L-system for creating the streets is subject to various global goals and local constraints specified by the user via image maps. Like many manually generated models, the models generated procedurally are usually not suitable for simulation. There are also methods developed to recover the 3D shape of roof surfaces, e.g., [16], and recover building ground plans from aerial data [10], the digitization and vectorization of cadastral maps or from surveying measurements. Recently, methods have also been proposed to extract ground plans from LiDAR data [34,18].

Because urban models tend to contain a large number of simple shapes, Chang et al. [9] propose a simplification method using the ideas from *urban legibility* in order to enhance and maintain the distinct features of a city, i.e., path, edges, district, etc. Since their focus is on visualization, the geometric errors generated due to simplification can be *hidden* from the viewers using various rendering techniques, e.g., texture mapping. Also focused on urban visualization, Cignoni [11] presents the BlockMaps strategy, which stores distant geometric and textural information in a bitmap for efficient rendering.

Other works on processing urban models can be found in GIS community. However, most of these methods focus on single buildings [28,41,43,25] and rarely focus on the city-level ground plans. For example, recently, Kada and Luo [25] simplify a building ground plan using the ideas of reducing the number of lines in the arrangement. See a survey in [27] for more related work.

Little work focused on aggregating and simplifying large scale models and this class of work tend to focus on ground plans only. Wang and Doihara [44] cluster the buildings using strategy similar to minimum spanning tree. Clustered buildings are aggregated and simplified. Rainsford and Mackaness [38] propose a template-based approach that matches the rural buildings to a set of 9 templates. The matched template is then deformed to fit the floor plan. This method is limited by the number of templates used.

2.2. *Geometric Boolean Operations*

The problem of geometric boolean operations is extensively studied because of their wide range of applications, including linear programming, robotics, solid modeling, molecular modeling, and geographic information systems. The study of the boolean operations of planar objects goes back to at least the early 1980s, when researchers were interested in the union of rectangles or disks, motivated by VLSI design, biochemistry, and other applications [1]. Starting in the mid 1990s, research on the complexity of the booleans of geometric objects has shifted to the study of instances in three and higher dimensions. The robustness issue has since been studied in great depth [22,40,39,26].

There exist several tools to compute the booleans of two polyhedra, such as CGAL [13] and Autodesk Maya. These tools are designed to handle the operation of a small number of geometries. However, the number of the building models that we will consider in this project can be much larger. CGAL's implementation is based on the Nef polyhedra [17]. Nef polyhedra is built on two critical data structures: a sphere map for each vertex v capturing v's local neighborhood and the Selective Nef Complex (SNC) storing connections between local sphere maps. Recently, Berstein and Fussell [4] construct robust and fast Boolean operations using a plane-based and binary space partitioning (BSP) representation. Experimental results show that it is 50 times faster than the robust CGAL implementation and is only twice slower than the fragile Maya method.

It is clear that the existing tools all suffer from various computational issues, such as robustness and efficiency. From our experience with the union operation of the ground plans, a large number of intermediate geometries are generated during the computation but are later deleted during or after the union process. These intermediate geometries are removed because they are inside the boundary of the final united geometry. This issue has long been ignored in the literature as most implementations consider the union operation as binary, which only takes two objects at a time. In addition, while the complexity of the union is $\Theta(n^2)$, the union of the buildings will be of much lower complexity because only a small subset of the buildings will intersect each other. From this simple observation, we can cull a lot of unnecessary computation by using some bounding volume hierarchy [21] or spatial hash table [42].

2.3. *Model Repair*

Mesh repair is an important step in the automatic construction of city-scale urban models, due to the frequent inaccuracies and defects of the input data. Several methods have been developed, that are capable of correcting gaps, holes, and inconsistent orientations without any user intervention. These method can be classified into surface-based [29], volumetric-based [23,6] and hybrid approaches [5]. Surface-based approaches, such as hole filling, are generally the most efficient but do not provide output-guarantees and have numeric instability. Volumetric-based methods use an intermediate volumetric representation, such as octree, and usually provide better quality. However, these methods can be very inefficient and generate large outputs, and the input's structure and details may also be lost. Hybrid algorithms attempt to maintain the quality and speed of surface reconstruction, while providing the strong guarantees of volumetric reconstruction. A more detailed summary of mesh repair techniques can be found in a recent survey [24].

3. Overview of Our Method

The main idea of our method is to exploit the fact that all components in our buildings are extruded from 2D polygons. This allows us to develop a strategy similar to *space sweep*, in which critical events representing the changes in building's shape are identified and handled either iteratively or in parallel. More specifically, if we can imagine a plane that sweeps from the bottom to the top of a given building, the intersection between the building and the plane only changes at certain critical moments. We define the critical events and the sweeping planes formally below.

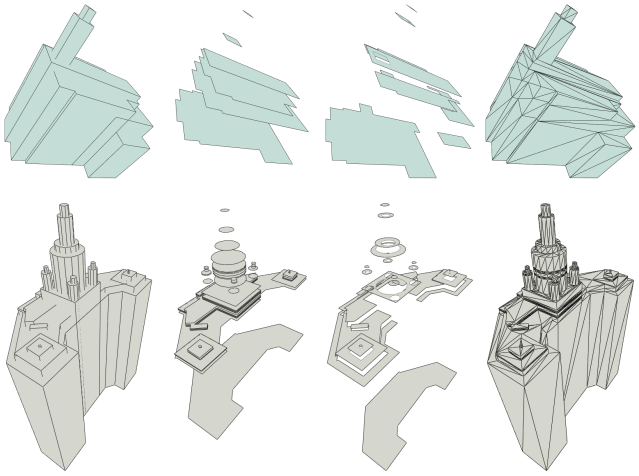
Definition 1 ***Critical events*** $\{E_i\}$ *is an ordered list of z coordinates representing the start (i.e., elevation) and the end (i.e., elevation+height) of the polygons in* $\mathcal{P}$.

Definition 2 ***Sweeping plane*** *The sweeping plane* S *is an xy-plane moving from* $-z$ *to* $+z$ *along the z-axis. We denote* $S(z)$ *as placing* S *at* z.

Our approach essentially converts the input polygons $\mathcal{P}$ whose areas, elevations and heights may overlap into another set of polygons $\mathcal{I}$ with disjoint interior. Although both $\mathcal{P}$ and $\mathcal{I}$ represent the same building B, as we will see later, using $\mathcal{I}$ allows us to construct seamless polyhedra and remove the artifacts of the building B much more easily than using $\mathcal{P}$. In fact, we can even remove some polygons from $\mathcal{I}$ while still maintaining the structure of the building. In Section 5, our model repair strategy is essentially based on repairing the polygons of $\mathcal{I}$. In the rest of this paper, we will name $\mathcal{I}$ *Invariant polygons.*

Definition 3 ***Invariant polygons*** $\mathcal{I}$. *The i-th invariant polygon* $I_i \in \mathcal{I}$ *is the intersection of* $e(\mathcal{P})$ *and the sweeping plane* S *at* z, *where* $E_i < z \leq E_{i+1}$. *More specifically,* $I_i = \bigcup_{P \in \mathcal{P}} e(P) \cap S(E_{i+1})$.

Note that I_i is usually not simple and can contain multiple connected components. Moreover, since $e(P) \cap S(E_i)$ is either $\emptyset$ or P, I_i is simply the union of some subset of



(a) $e(\mathcal{P})$ (b) $\mathcal{I}$ (c) $\mathcal{T}$ (d) $\mathcal{B}$

Fig. 4. (a) Input, $e(\mathcal{P})$. (b) Invariant polygons, $\mathcal{I}$. (c) Transition polygons $\mathcal{T}$. (d) Output, seamless building mesh $\mathcal{B}$.

Algorithm 1. Build Seamless Building Mesh

1: **procedure** SEAMLESSBUILDING($\mathcal{P}$, τ)
2: $\{E_i\} \leftarrow$ EVENTS($\mathcal{P}$, τ)
3: **for all** E_i **do** ▷ Compute invariant polygons
4: $I_{i-1} = \bigcup_{P \in \mathcal{P}} e(P) \cap S(E_i)$ ▷ see Section 4
5: $I_{i-1} \leftarrow$ REPAIR(I_{i-1}, τ) ▷ see Section 5
6: $W_i \leftarrow$ SWEEP(I_{i-1}, E_{i-1}, E_i) ▷ walls polygons
7: **for all** I_i **do**
8: $T_i = I_i \cap \overline{I_{i+1}}$ ▷ see Section 4
9: **return** $\{T_i\} \cup I_0 \cup I_{n-1} \cup \{W_i\}$

$\mathcal{P}$. In order to construct a seamless model from $\mathcal{I}$, we must also construct *transition polygons* that "connect" consecutive invariant polygons.

Definition 4 ***Transition polygons*** $\mathcal{T}$. *The i-th transition polygon $T_i \in \mathcal{T}$ is the intersection of I_i and the complement of I_{i+1} (projected on the the plane containing I_i), i.e., $T_i = I_i \cap \overline{I_{i+1}}$.*

Note that representing the complement $\overline{I_{i+1}}$ of a polygon I_{i+1} can be easily accomplished by reversing the vertex ordering of I_{i+1}. The definitions of $\mathcal{I}$ and $\mathcal{T}$ are illustrated in Fig. 4.

Our method for constructing the seamless building models is sketched in Algorithm 1. In addition to the input polygons $\mathcal{P}$, Algorithm 1 also takes a user-specified tolerance τ that will be used as the threshold to determine and remove artifacts in $\mathcal{P}$. The value τ is also used in subroutine EVENTS to ensure that two consecutive layers are at least τ units away. In most applications, τ is determined by the desired precision in the simulation, e.g., simulation step size. When there are n critical events, the seamless polyhedron B of the building simply is composed of the polygons $\mathcal{T}$, the floor I_0, the ceiling I_{n-1} and the wall polygons swept by the boundary of I_i between E_i and E_{i+1}. Note that the building polyhedron B remains seamless even when the polygons of $\mathcal{I}$ are only an approximation of those in Def. 3 or when some $\mathcal{I}$ are removed. The only step that requires more robust computation to ensure seamless output is in step 8. Other steps can tolerate lower precisions. We will discuss a new approach to provide this guarantee in Section 4.

4. 2D K-way Boolean Operations

Computing the invariant polygons requires the union of a subset of $\mathcal{P}$, and computing the transition polygons requires the intersection of the components of two consecutive invariant polygons. Traditionally, the union and intersection operations take only two input polygons, and the boundary is determined by computing the arrangement of the edges, which is a subdivision of the space into vertices, edges and faces (cells) from a set of line segments (i.e., edges). One way to extract the boundaries from such an arrangement is by finding all the faces that have positive winding numbers [14,45]. When there are k inputs, the result is computed by iteratively applying the operations. The main drawback of this approach is that many intermediate geometries are generated and then thrown away during the process [40]. In this section, we will discuss a more efficient way to compute the k-way union (Section 4.1) and intersection (Section 4.2) of k polygons in a single step from the *implicitly represented* line-segment arrangement of the input polygons. To ensure the correctness of the output, in Section 4.3, we develope a robust and efficient method to compute the nearest segment intersection by adaptively adjusting the numerical precision.

4.1. *K-way Polygon Union*

To simplify our notation, let $\{P_i\}$ be a set of polygons and Q be the boundary of the union of $\{P_i\}$, i.e., $Q = \partial(\bigcup_i P_i)$. Our goal is to compute Q. For each polygon P, we denote the vertices of P as $\{p_i\}$ and the edge that starts at vertex p_i as $e_i = \overline{p_i p_{i+1}}$. The edge e_i has two associated vectors, the vector from p_i to p_{i+1}, i.e., $\boldsymbol{v_i} = \overrightarrow{p_i\, p_{i+1}}$, and the outward normal $\boldsymbol{n_i}$. The main idea of our approach is to *incrementally* extract the *orientable loops* of the arrangement induced from the edges of the input polygons. During the extraction process, we repetitively extend the extracted loop by maintaining its desired topological properties.

4.1.1. *Orientable Loop*

We say that a loop is orientable if all the normal directions of the edges in the loop are all either pointing inward or outward. Note that the segments we considered are edges from $\{P_i\}$, therefore, they are directional (as vertices in $\{P_i\}$ are ordered) and include normal directions pointing outward.

Observation 5 *We observe that the boundary of the union must be an orientable loop (if it encloses an area, either positive or negative).*

Therefore, given two adjacent segments $s = \{u, v\}$ and $s' = \{v, u'\}$ sharing an end point v, we can check whether s and s' belong to an orientable loop if

$$(\overrightarrow{u\,v} \times \boldsymbol{n_s}) \cdot (\overrightarrow{v\,u'} \times \boldsymbol{n_{s'}}) > 0\ , \qquad (1)$$

where $\boldsymbol{n_x}$ is the normal vector of segment x, and $\times$ is the cross product. If s and s' satisfy Eq. 1, we say they are compatible segments.

4.1.2. *Boundary Orientable Loop Extraction*

There can be many orientable loops in the arrangement. We are only interested in finding those that can be a member of Q, i.e., a boundary. We call these loops *boundary orientable loop.*

To extract all boundary orientable loops, we start from an arbitrary edge e that has not been considered. Our method then proceeds by incrementally discovering the compatible edges of a loop L from e. To abuse the notation a little bit, we let e be the latest edge of L discovered, and let v be starting end point of e. In every incremental step, our method will need to identify (1) the portion of e and (2) the next compatible edge so that L remains on the boundary of the union until all edges of L are discovered.

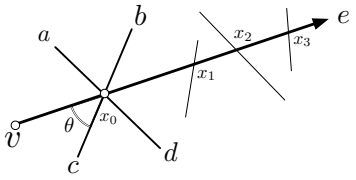


Fig. 5. Given the last vertex v and a potential edge e discovered in the extraction process, the segment $\overline{vx_0}$ must be an edge of L and the next v is x_0 and the next e is the edge containing $\overline{x_0c}$.

To identify the portion of e that contributes to L, let x_j be a sorted list of intersections between e and other line segments $e_j \neq e$ in $\{P_i\}$. The intersections x_j are sorted in non-decreasing order using the distance to v. Therefore x_0 is the intersection closest to v. See Fig. 5. Now we claim that if we expand the loop by appending $\overline{vx_0}$ to L and replacing e by the edge that makes the largest right turn from the current edge e at x_0, then L will remain on the boundary of the union Q if there exist an edge of L is in Q. This observation is proved by Lemma 6.

Lemma 6 *The orientable loop L extracted must be a boundary orientable loop if an edge of L is on Q.*

PROOF. If L ever reaches Q, then we assume v is the first vertex of L on Q. See Fig. 5. Assume that x_0 is not on Q. Then x_0 must be interior to Q. Since we know that v is a vertex of Q, when we move from v to x_0, there must be a point $x' \in Q$ before we reach the interior of Q. If we wish to remain on the boundary of Q, we must move another edge of Q at x'. Therefore, x' must be an intersection of e and another segment from $\{P_i\}$. However, we know that x_0 is the intersection closest v. This means x_0 cannot be interior to Q, and in fact x_0 and x' must be the same point and the segment $\overline{vx_0}$ must be on Q. □

Now, with e updated, we repeat the process until a closed loop is found. Note that since we start from an arbitrary edge, the loop may not close at the first edge of L. When this happens we simply prune away edges of L that are not part of the final loop since these edges cannot be on the boundary.

4.1.3. *Boundary Filtering*

The orientable loops identified in the previous section can only represent potential boundaries of the union if an edge of the loop is on the boundary. Filters are needed to determine the final boundaries. Our first filter is based on the nesting relationships of the loops.

Observation 7 *By definition, the simple polychains in a polygon must obey the nesting property, i.e., the loops that are* directly *enclosed by the external loop must be holes and will have negative areas, and the loops that are directly enclosed by the holes must have positive areas.*

Because the loop-extraction step discussed above does not allow loops to share an edge, all orientable loops we generated are non-overlapping (i.e., they don't intersect or touch, except at vertices). Therefore, the nesting property can be determined efficiently using a plane sweep algorithm, e.g., [2], in $O(n \log n)$ time for n segments. Loops that do not have proper nesting property are filtered.

Finally, in the second filter, we verify each nested loop against the input polygons $\{P_i\}$ to remove the rest of false boundaries (if any). More specifically, for each hole boundary L, we check if a point inside L is outside all polygons $\{P_i\}$. If the check failed, L is a false loop. Since all orientable loops form a superset of the final boundaries (Lemma 6) and each loop is checked for its validity, the remaining loops must correctly represent the boundaries of the k-way union of $\{P_i\}$.

4.2. *K-way Polygon Intersection*

The k-way intersection can be computed in the same way. Instead of making the largest right turns, we will make the largest left turns at the nearest segment intersection, and we filter false loops by determine if a point inside each external loop is inside all polygons in $\{P_i\}$.

The proposed k-way union and intersection methods have many advantages over the existing approaches. First, in contrast to the traditional boolean operation approach, the proposed method can handle arbitrary number of elements in $\{P_i\}$ at once without produce intermediate geometries. Second, we do not have to compute the arrangement of the input segments, i.e., we avoid computing all the intersections for all the line segments in $\{P_i\}$. Instead, we compute only the intersections of all e_r discovered during the construction of Q. This is extremely helpful when the size of $\{P_i\}$ is large and the boundary of Q has only a few features (edges and vertices). This observation is usually true when the size of $\{P_i\}$ is large and for the architectural models in which many parts only contribute a small portion to the external boundary. Because of this feature, our method is more sensitive to the output complexity than the existing methods. Third, the proposed method can handle degenerate cases easily, i.e., two polygons that touch at a single vertex or a line. The

proposed approach can even handle non-simple polygon, whose edges may self intersect, and polychains which do not form a loop or enclose an area.

4.3. *Robust and Efficient Nearest Segment Intersection*

As we have seen earlier, the main step that we used to determine the boundaries of the union and intersection is to find the *closest* intersection to the boundary (e.g., an end point or an edge) of a segment. A straightforward approach is to compute all the intersections for a given segment, and then the intersection that are closest to the given boundary can be determined. However, computing the intersections is known to be prone to numerical errors [35] and can be inefficient if exact arithmetic is used to overcome the numerical problems.

In this section, we will discuss our approach to handle this problem. We will show that our approach can easily identify *precision inefficiency* and dynamically increase the numerical precision when needed. The proposed method is similar to algorithms designed for the k-th order statistic [12]. The main idea of our approach is to determine the segments that will create the closest intersection *without* computing the intersection.

Given a 2-d segment s, one end point p of s and a set of 2-d segments $\mathcal{S}$ intersecting with s, our goal here is to determine a segment t in S such that the intersection of t is closer to p than other segments in $\mathcal{S}$. That is, given s, p and $\mathcal{S}$, we try to solve the following:

$$\arg\min_{t\in\mathcal{S}} \mathbf{d}(p, \mathbf{int}(s,t)) \ , \tag{2}$$

where $\mathbf{d}(x,y)$ is the distance between two points x and y, and $\mathbf{int}(s,t)$ is the intersection between two segments s and t. Note that since $\mathcal{S}$ are polygon edges, each segment is directional and has a normal direction.

Instead solving Eq. 2 numerically, we approach the problem algorithmically. We use the *visibility* between a point $q \in s$ and $t \in \mathcal{S}$ to recursively determine the closest intersection. More specifically, we classify the visibility between q and s as the value v of $\overrightarrow{qr} \cdot \boldsymbol{t_n}$, where $r \in t$ is a point on t and $\boldsymbol{t_n}$ is the normal direction of t. If v is greater than zero, we say q is visible to t. If v is zero, then q is invisible to t. Otherwise, we say q is on t.

Now, our goal is to find a point $q \in s$ that is invisible to a segment but is visible to all the rest of the segments in $\mathcal{S}$. As described above, given any $q \in s$ we can classify the segments in S into three sets of segments: $\mathcal{V}$, $\mathcal{I}$, $\mathcal{O}$, which are visible, invisible, and on segments. If the set $\mathcal{I}$ contains exactly one segment, then we have found our solution. If I has more than one segment, then we let $s = \overline{pq}$ and $\mathcal{S} = \mathcal{I}$ and perform the classification recursively. If I is empty, then we analyze $\mathcal{O}$ and then $\mathcal{V}$ in a similar way in this order. The only difference is that if $\mathcal{O}$ has more than one segment then that means that all segments in $\mathcal{O}$ intersect s at q and are equidistant to p. In this case, we will be looking for the segment that makes the smallest angle to s. Again we can use the idea of visibility to find this segment.

The point $q \in s$ is determined in a way similar to binary search. First the mid point of s is used as q. If the next searching range is in $\mathcal{I}$, then q is the mid point of p and q. If the next searching range is in $\mathcal{V}$, then q is the mid point of q and p' (the other end point of s). Now, for fixed-precision floating-point computation, it is possible that there is not enough precision to distinguish between the segments in $\mathcal{S}$. This happens when the size of $\mathcal{S}$ is greater than one while the length of the search range collapses to zero. When this happens, we dynamically increase the precision.

Analysis. The main step in our approach is the visibility test, which involves a dot product (two multiplications and a summation). The asymptotic time complexity of the proposed approach is $O(n)$ for n segment in $\mathcal{S}$ which the same as that of k-th order selection of n values. On the contrary, the traditional approach that computes the (parameterized) intersection between two line segments will require two divisions, 14 multiplications and 10 summations to compute the parameterizations of the intersection. As a result much higher precision is needed for the traditional approach. More importantly, the traditional approach has no way to tell if the fixed-precision is enough to handle the given input. Therefore, in order to provide error-free computation, high-precision floating points are used regardless the input configuration. Consequently, the traditional approach can be very slow. On the other hand, the proposed method provides the same accuracy and robustness but is more efficient.

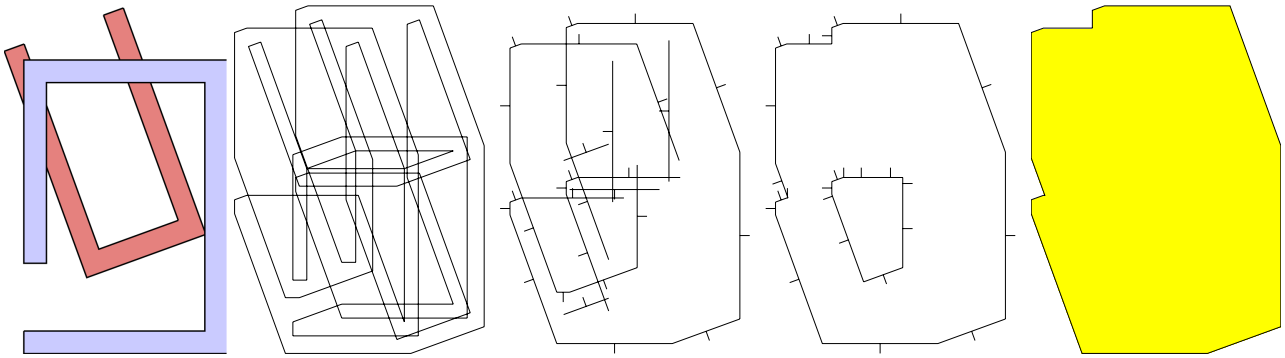
5. Model Repair

Due to errors in the input polygons and their arrangements in the space, many small features, and narrow gaps can be generated. Removing these geometries in 3D can be difficult [24]. Several volumetric-based methods have been proposed to use mathematical morphology the repair the models. In this section, we take the same approach but perform the operations in the continuous domain to avoid the drawbacks of volumetric-based methods.

Since mathematical morphology is closely related to Minkowski sum (M-sum), we will first discuss an efficient method to compute the Minkowski sum in Section 5.1. Then we will extend the Minkowski sum to compute the "closing" operation in Section 5.2.

5.1. *Minkowski Sum using Reduced Convolution*

The M-sum of two shapes P and Q is defined as: $P \oplus Q = \{p + q \mid p \in P, q \in Q\}$. We propose a convolution-based method to compute the M-sum of non-convex polygons. Unlike the classic approach, e.g., [45], the proposed method avoids computing (1) the complete convolution, (2) the arrangement of the segments of the convolution, and (3) the winding number for each arrangement cell. Our method first computes a subset of the segments that is from the



(a) inputs (b) full convolution (c) reduced convolution (d) orientable loops (e) M-sum

Fig. 6. Steps for computing the M-sum of two simple polygons.

convolution of the inputs. We call this subset a "reduced convolution" as illustrated in Fig. 6(d).

Definition 8 *The convolution of two shapes P and Q, denoted as $P \otimes Q$, is the M-sum of the boundary, i.e., $P \otimes Q = \partial P \oplus \partial Q$.* If both P and Q are convex, $\partial(P \oplus Q) = P \otimes Q$. Otherwise, it is necessary to trim the convolution to obtain the M-sum boundary.

Definition 9 *A* **reduced convolution** *is a set of convolution segments $\overline{p_i p_{i+1}} \oplus q_j$ and $p_k \oplus \overline{q_l q_{l+1}}$ and q_j and p_k must be convex.*

Similar to the proposed k-way boolean operations, we identify boundary orientable loops. These loops form potential boundaries of the M-sum and are further filtered by analyzing their nesting relationship. Finally, the remaining boundaries are filtered by checking the intersections between the input polygons placed at the configurations along these loops. Each of these steps is discussed in detail below.

5.1.1. *Reduced Convolution*

In the first step of the algorithm, we compute a subset of the segments of the convolution based on the following observation.

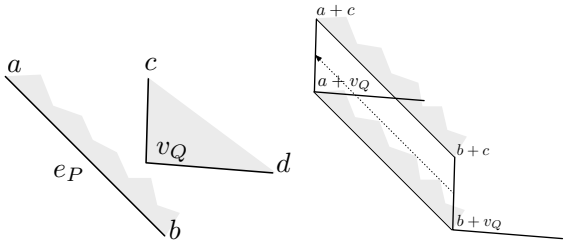


Fig. 7. Figures used in the proof of Observation 10. The shaded areas indicate the internal of the polygons.

Observation 10 *Given a convolution segment $s = e_i \oplus q$ of an edge $e_i \in P$ and a vertex $q \in Q$, if q is a reflex vertex, s must not be a boundary of the M-sum of P and Q. This observation remains true if $s = p \oplus e_j$, where $p \in P$ is a reflex vertex and $e_j \in Q$ is an edge.*

PROOF. Assume that v_Q is convex, and e_P and v_Q are compatible. An example of $e_P = \overline{ab}$ and v_Q is shown in Fig. 7(a). In the figure, v_Q is adjacent to two vertices c and d. Now, let us consider the Minkowski sum locally involving only $e_P = \overline{ab}$, $\overline{cv_Q}$ and $\overline{v_Q d}$, which are shown in Fig. 7(b). First, one should see that the segment s has end points $a + v_Q$ and $b + v_Q$. In this case, the segment s is locally on the boundary of the Minkowski sum because all other segments (i.e., $\overline{(a+v_Q)(a+c)}$, $\overline{(b+v_Q)(b+c)}$, $\overline{(a+c)(b+c)}$, etc.) are interior to s. Therefore, the question becomes: if we increase the internal angle of v_Q while maintaining the compatibility of e_P and v_Q, when will s become an interior segment?

First, observe that when the internal angle of v_Q increases, the segment $\overline{(a+c)(b+c)}$ and s become closer. Moreover, when c, v_Q, d are collinear, they must be also parallel to e_P (otherwise e_P and v_Q are not compatible), thus, $\overline{(a+c)(b+c)}$ and s overlap. When the internal angle of v_Q increases more, v_Q becomes reflex and the segment s becomes interior to $\overline{(a+c)(b+c)}$. Therefore, if v_Q is reflex, s must not be on the boundary. □

Because of the definition of a reflex angle, the number of edges that are compatible with any convex vertex in Q form a lower bound on the number of edges compatible with any reflex vertex in Q. Due to this, the number of segments filtered by Observation 10 is significant. An example of this is demonstrated in Fig. 6. Figs. 5 and 6(b) show the input polygons and the full convolution of the inputs, respectively. Fig. 6(c) demonstrates the reduced convolution of the same imput polygons, in which there are are significantly fewer convolution edges.

5.1.2. *Boundary Orientable Loop Extraction*

Now, since the segments that we will be working with are no longer a complete convolution, we cannot apply the idea of computing the winding number for each arrangement cell to extract the M-sum boundary as done in [45]. Note that the segments we considered are edges from P and Q, therefore, they are directional (as vertices in P and Q are ordered) and include normal directions pointing outward (to P or Q). Therefore, we proceed as the k-way boolean

operations by extracting the boundary orientable loops discussed in Section 4.1.2.

5.1.3. *Boundary Filtering*

As discussed in Section 4.1.3, the loops extracted are potential boundary loops. For example, the polygon in Fig. 6(d) has an improperly nested loop (positive area inside positive area) that must be removed; once the loop is removed, we get the correct M-sum as seen in Fig. 6(e).

So far, we have introduced three quite efficient filters. Unfortunately, some of the remaining loops may not be boundaries of the M-sum. These false loops are the direct result from the filters that exploit only local topological properties on the convolution. Therefore, we will have to resort to collision detection, a global operation, to remove all the false loops. Given a translational robot P and obstacles Q, the contact space of P and Q can be represented as $\partial((-P) \oplus Q)$, where $-P = \{-p \mid p \in P\}$. If a point x is on the boundary of the M-sum of two polygons P and Q, then the following condition must be true:

$$(-P^\circ + x) \cap Q^\circ = \emptyset \ ,$$

where Q° is the open set of Q and $(P + x)$ denotes translating P to x.

Although there are many methods to optimize the computation time for collision detection, collision detection is more time consuming than the previous filters. Fortunately, only a single collision detection is needed to reject or accept a loop based on the following lemma.

Lemma 11 *All the points on a false hole loop must make P collide with Q.*

PROOF. Let A be the arrangement of the segments in the complete convolution. Let ℓ be a hole loop extracted using our method. It is guaranteed that ℓ must be empty since we always make the largest right turns when we trace the hole (note that this may not be true if ℓ is not a hole, i.e., when ℓ encloses a positive area, e.g., Fig. 6(d)). Since ℓ is empty, we know that $\ell \subset A$. Furthermore, since we know that all vertices in each cell of A must to have the same winding number [45]. Therefore, we know that all points on ℓ will have the same winding number. If ℓ is a false loop, then all points on ℓ will have positive winding numbers, thus, are all interior to the Minkowski sum boundary. This means that all the points on a false hole loop must make P collide with Q. □

5.2. *Mathematical Morphology*

It is a common practice to remove small gaps and sharp features by performing mathematical morphology operations, such as erosion and dilation. These operations are usually done in a discretized domain by converting the polygons or polyhedra into pixels or voxels. This approach is usually robust, but well-known issues include the difficulty of choosing a good discretization resolution, space and time inefficiency and the loss of data fidelity.

We extend the proposed Minkowski sum method using reduced convolution to compute the "closing" operation. The closing operation is the application of dilation and then erosion to the model. Given two polygons P and Q, the dilation operation is simply the Minkowski sum of P and Q. The erosion operation denoted as $P \ominus Q$ can also be computed using Minkowski sum: $P \ominus Q = \overline{\overline{P} \oplus Q}$, where $\overline{X}$ is the complement of a set X. The complement of a polygon P is simply a polygon with reverse ordering of P's vertices. Finally, the closing operation of P and Q is defined as

$$P \bullet Q = (P \oplus Q) \ominus Q \ .$$

The proposed closing method operates directly on the polygons thus avoiding the aforementioned issues. A straightforward approach is to compute the Minkowski sum twice using definition above. Fig. 9 shows three examples that are commonly seen in our dataset.

We observe that the second Minkowski sum can be avoided if Q is a small circle. If Q is a small circle, the convolution for computing $P \oplus Q$ can be easily obtained by translating each edge e of P by the amount of Q's radius along e's outward normal, and replacing each convex vertex v of P with an arc of Q connecting v's incident edges. Once we have the convolution, the boundary of $P \oplus Q$ can be obtained using the method discussed in Section 5.1. In order to compute $P \bullet Q$, we will have to apply another Minkowski sum operation on the complement of $P \oplus Q$. However, we observe that the erosion operation can simply be done reversing the dilation process. That is, for a given loop L of $P \oplus Q$, we identify the *origin* of each edge of L. If the edge is an edge of P, we translate the edge back. The rest of edges that form (partial) arcs must be from the vertices of P. We replace each arc with its original point.

Next the vertices of $P \bullet Q$ can also be determined. Let's analyze the vertices of the loop. These vertices can only come from two sources. The vertices can either be created by the sum of a vertex of P and a vertex of Q, or from the intersection of edges (of the convolution). The first type of vertex (e.g., a in Fig. 8) is automatically generated when the edges of $P \oplus Q$ are translated back. The second type of vertex requires more attention. We observe that the intersection can come from three sources: (1) the intersection of two edges of P (e.g., d in Fig. 8), (2) the intersection of an edge of P and a vertex arc (e.g., c in Fig. 8), and (3) the intersection of two arcs (e.g., b in Fig. 8). In erosion, when edges are moved back and arcs are shrunk to points, these intersections may disappear. In the first case, if the intersection disappears, the edges of P are then connected via their boundary end points. In the second case, the vertex is connected to the closest point on the edge (e.g., r to e in Fig. 8). In the third case, two vertices of P are simply connected (e.g., p to q in Fig. 8).

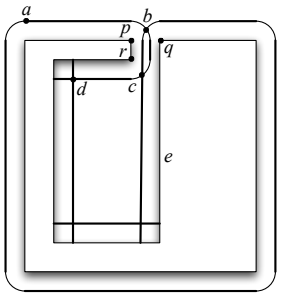
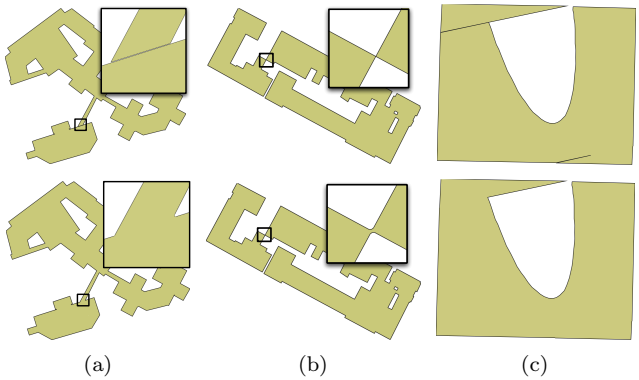


Fig. 8. closing operation



(a) (b) (c)

Fig. 9. Before (top row) and after (bottom row) the application of the closing operation. (a) Narrow gap between two buildings. (b) Non-manifold vertex. (c) Narrow gap within the building.

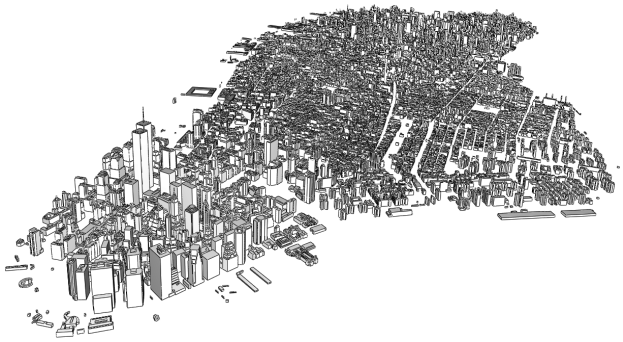


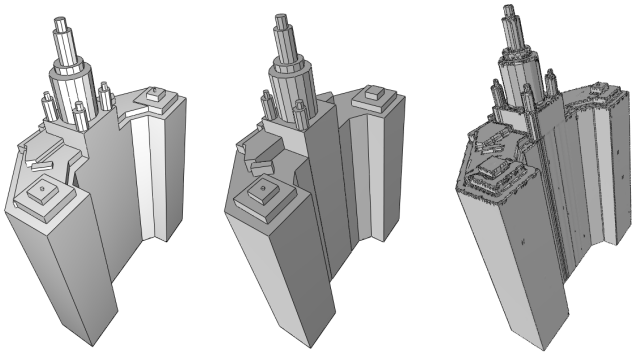
Fig. 10. Seamless NYC (south of Central Park). There are 1,161,850 vertices, 2,310,610 triangles, and 5,397 buildings in this example. The input composed of 45,496 polygons.

6. Results

Our framework is implemented in C++. The exact number type uses MPFR [15]. We use two examples, New York City (NYC, 5,397 buildings) and Downtown Oklahoma City (OKC, 357 buildings), shown in Figs. 10 and 12, to demonstrate the results of the proposed method.

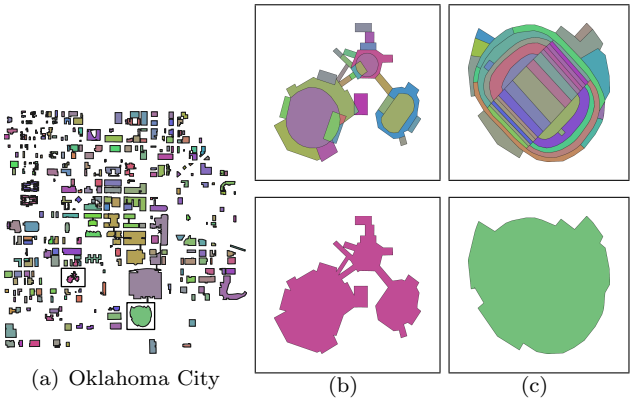
The total running time for generating the seamless models for NYC in Fig. 10 is 119 seconds (about 2 minutes). The total running time for generating the seamless models for Downtown OKC in Fig. 10 is 7.2 seconds. All these experiments are performed on a dual core 2.54 GHz Intel CPU. The reported running times are based on our initial implementation, whose efficiency can still be significantly improved, e.g., using parallelization and ideas from broad-phase collision detection.

To show the significance of our results, we compare our results to Maya 2010 and a volumetric approach using Marching Cube. We use a building from NYC composed of 37 components. The results are shown in Fig. 11. Our method produces the result shown in Fig. 11(a) in 105 milliseconds. Fig. 11(b) shows the best result using Maya as Maya produces completely incorrect output when more components are added to Fig. 11(b). Finally, we compare to the result generated using distance field and a Marching Cube method (from VCG library) shown in Fig. 11(c).



(a) Our result (b) Maya (c) Marching Cube

Fig. 11. Comparing to results from Maya and Marching Cube.



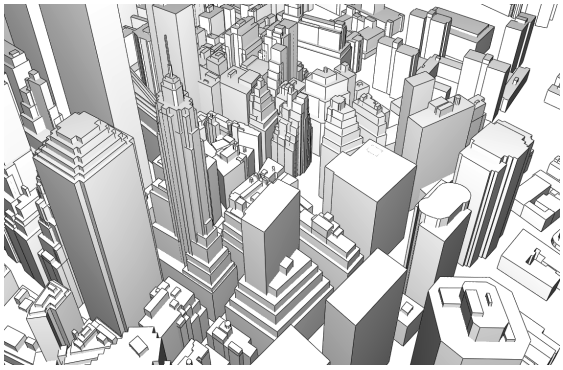
(a) Oklahoma City (b) (c)

Fig. 12. In total 358 ground plans are represented in this image. The ground plans are created from 1454 footprints. Two interesting regions are highlighted.

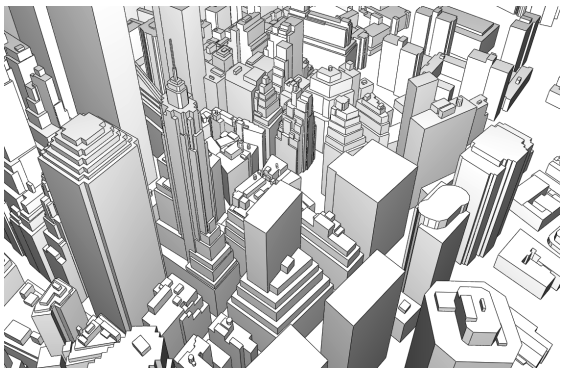
Although the quality of the mesh can be improved using better Marching Cube variants, this volumetric approach takes several minutes and produces an excessive number of polygons (138,288 triangles for a 150×150×150 grid). Thus, the volumetric approach is clearly not scalable to the city-scale problem considered in this work. No results are shown for CGAL because CGAL Nef polyhedra cannot handle our extruded meshes.

We also compare the proposed method to CGAL's robust 2D boolean operations package. We setup the experiment by generating the lowest invariant polygons (i.e., ground plans) of the entire OKC by computing the union of 1,454 footprints. The results are shown in Fig. 12. Since CGAL's union operation takes only two polygons, we compute the ground plans by incrementally adding each of the footprints to its current union. Both approaches generate the same result. Our approach takes 124 milliseconds to generate all ground plans, and the incremental approach using CGAL takes 55,516 milliseconds.

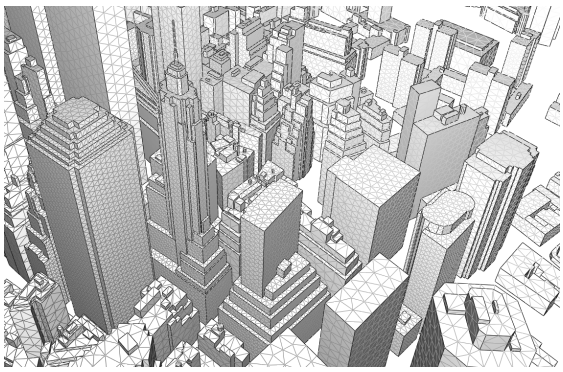
To further verify our method, we show that the seamless meshes can be readily remeshed for CFD simulations. Fig. 13 shows our results zoomed into the area near the financial district in NYC dataset, and Fig. 14 shows the entire OKC dataset. We used the identified ground plans (Fig. 12) to merge the seamless surface of the buildings and



(a) Input



(b) Seamless meshes



(c) Remeshed for CFD

Fig. 13. An area of Fig. 10 near the financial district in NYC.

the surface terrain from DEM and obtain a computational domain suitable as input for CFD models. We simulate a hypothetical transport and dispersion event using FEFLO-Urban [31]. A simulation of the flow, and the transport and dispersion of a gas was performed using the volume mesh produced with the proposed data processing methodology. Figs. 1 and 14 show snapshots of air pollutant dispersion simulation in the integrated OKC model.

7. Conclusion

We developed the first known framework to construct seamless 3D building models from 2.5D ground plans. Our main idea to provide both efficiency and robustness is to generate polygon layers, repair the layers and then stitch

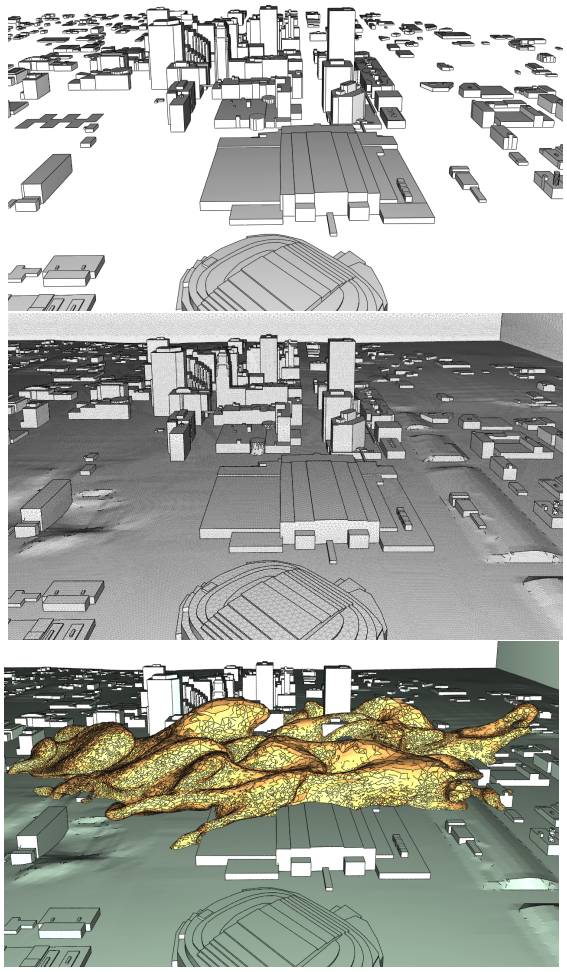


Fig. 14. Using the OKC dataset for pollutant dispersion simulation. (top) Input meshes. (mid) CFD meshes integrated with DEM terrain data. (bottom) A simulation snapshot.

the layers into seamless models. The core methods for stitching and repairing are a k-way boolean operation and a Minkowski sum operation, respectively. Finally, we show both theoretically and empirically that our framework is efficient and numerically stable.

Limitations and Future Work. A main limitation of the proposed method is that, despite the effort to remove small and sharp features, sliver triangles can still be found in our output mesh. Another limitation is that our method can only handle extruded polygons. We are currently extending the proposed work to handle more general 3D urban models, such as those from Google Earth. Finally, we are interested in developing statistical methods to better visualize the simulation data and urban models.

References

[1] P. Agarwal, J. Pach, and M. Sharir. State of the union, of geometric objects: A review. *Surveys on Discrete and Computational Geometry,(J. Goodman, J. Pach and R. Pollack, eds.), AMS, Providence, RI*, pages 9–48, 2008.

[2] C. Bajaj and T. K. Dey. Polygon nesting and robustness. *Inform. Process. Lett.*, 35:23–32, 1990.

[3] G. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1983.

[4] G. Bernstein and D. Fussell. Fast, exact, linear booleans. In *Computer Graphics Forum*, volume 28, pages 1269–1278. Wiley Online Library, 2009.

[5] S. Bischoff and L. Kobbelt. Structure preserving cad model repair. In *Computer Graphics Forum*, volume 24, pages 527–536. Wiley Online Library, 2005.

[6] S. Bischoff, D. Pavic, and L. Kobbelt. Automatic restoration of polygon models. *ACM Trans. Graph.*, 24:1332–1352, October 2005.

[7] F. Camelli. FEFLO CFD model study of flow and dispersion as influenced by tall buildings in New York city. In *Sixth Symposium on the Urban Environment*, 2006.

[8] S. Chan and M. Leach. A validation of FEM3MP with Joint Urban 2003 data. *Journal of Applied Meteorology and Climatology*, 46(12):2127–2146, 2007.

[9] R. Chang, T. Butkiewicz, C. Ziemkiewicz, Z. Wartell, W. Ribarsky, and N. Pollard. Legible simplification of textured urban models. *IEEE Computer Graphics and Applications*, 28(3):27, 2008.

[10] L. Cheng, J. Gong, X. Chen, and P. Han. Building boundary extraction from high resolution imagery and lidar data. *ISPRS08, p. B3b*, 693, 2008.

[11] P. Cignoni, M. Di Benedetto, F. Ganovelli, E. Gobbetti, F. Marton, and R. Scopigno. Ray-Casted BlockMaps for Large Urban Models Visualization. In *Computer Graphics Forum*, volume 26, pages 405–413. Wiley Online Library, 2007.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[13] A. Fabri, G. Giezeman, L. Kettner, S. Schirra, and S. Sch "onherr. On the design of CGAL a computational geometry algorithms library. *Software Practice and Experience*, 30(11):1167–1202, 2000.

[14] E. Flato. *Robust and E cient Construction of Planar Minkowski Sums*. PhD thesis, Tel-Aviv University, 2000.

[15] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software (TOMS)*, 33(2):13, 2007.

[16] C. Frueh and A. Zakhor. Constructing 3d city models by merging ground-based and airborne views. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2. IEEE, 2003.

[17] P. Hachenberger and L. Kettner. Boolean operations on 3d selective nef complexes: optimized implementation and experiments. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 163–174, New York, NY, USA, 2005. ACM Press.

[18] K. Hammoudi, F. Dornaika, and N. Paparoditis. Extracting building footprints from 3D point clouds using terrestrial laser scanning at street level. *ISPRS/CMRT09*, 38:65–70, 2009.

[19] S. Hanna, M. Brown, F. Camelli, S. Chan, W. Coirier, O. Hansen, A. Huber, S. Kim, and R. Reynolds. Detailed simulations of atmospheric flow and dispersion in downtown Manhattan. *Bulletin of the American Meteorological Society*, 87(12):1713–1726, 2006.

[20] J. Haunert and A. Wolff. Optimal simplification of building ground plans. In *Proceedings of XXIst ISPRS Congress Beijing 2008, IAPRS Vol. XXXVII (Part B2)*, pages 372–378, 2008.

[21] H. Haverkort. Introduction to bounding-volume hierarchies, 2004. Part of the PhD thesis, Utrecht University.

[22] C. Hoffmann. *Geometric and solid modeling: an introduction*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1989.

[23] T. Ju. Robust repair of polygonal models. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 888–895. ACM, 2004.

[24] T. Ju. Fixing geometric errors on polygonal models: a survey. *Journal of Computer Science and Technology*, 24(1):19–29, 2009.

[25] M. Kada and F. Luo. Generalisation of building ground plans using half-spaces. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36, 2006.

[26] J. Keyser, T. Culver, M. Foskey, S. Krishnan, and D. Manocha. ESOLID–a system for exact boundary evaluation. *Computer-Aided Design*, 36(2):175–193, 2004.

[27] D. Lee. Geographic and cartographic contexts in generalization. In *ICA Workshop on Generalisation and Multiple Representation, Leicester, UK, August*. Citeseer, 2004.

[28] W. Lichtner. Computer-assisted processes of cartographic generalization in topographic maps. *Geo-Processing*, 1(2):183–199, 1979.

[29] P. Liepa. Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 200–205. Eurographics Association, 2003.

[30] R. Lohner. *Applied computational fluid dynamics techniques: an introduction based on finite element methods*. Wiley, 2008.

[31] R. Löhner, J. Cebral, F. Camelli, S. Appanaboyina, J. Baum, E. Mestreau, and O. Soto. Adaptive embedded and immersed unstructured grid techniques. *Computer Methods in Applied Mechanics and Engineering*, 197(25-28):2173–2197, 2008.

[32] P. Merrell and D. Manocha. Continuous model synthesis. In *ACM Transactions on Graphics (TOG)*, volume 27, page 158. ACM, 2008.

[33] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. *ACM Transactions on Graphics (TOG)*, 25(3):614–623, 2006.

[34] H. Neidhart and M. Sester. Extraction of building ground plans from Lidar data. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37:405–410.

[35] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2nd edition, 1998.

[36] Y. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM, 2001.

[37] C. Poullis and S. You. Photorealistic large-scale urban city model reconstruction. *IEEE transactions on visualization and computer graphics*, pages 654–669, 2008.

[38] D. Rainsford and W. Mackaness. Template matching in support of generalisation of rural buildings. In *Advances in Spatial Data Handling, 10th International Symposium on Spatial Data Handling*, pages 137–152, 2001.

[39] A. Ricci. A constructive geometry for computer graphics. *The Computer Journal*, 16(2):157–160, 1973.

[40] J. Rossignac and A. Requicha. Constructive non-regularized geometry. *Computer-Aided Design*, 23(1):21–32, 1991.

[41] M. Sester. Generalization based on least squares adjustment. *International Archives of Photogrammetry and Remote Sensing*, 33(B4/3; PART 4):931–938, 2000.

[42] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of Vision, Modeling, Visualization VMV'03*, pages 47–54. Citeseer, 2003.

[43] F. Thiemann and M. Sester. 3D-symbolization using adaptive templates. *Proceedings of the GICON*, 2006.

[44] P. Wang and T. Doihara. Automatic Generalization of Roads and Buildings. In *ISPRS Congress*, 2004.

[45] R. Wein. Exact and efficient construction of planar Minkowski sums using the convolution method. In *Proc. 14th Annual European Symposium on Algorithms*, pages 829–840, 2006.

[46] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Transactions on Graphics*, 22(4):669–677, july 2003. Proceeding.