Computer Graphics International 2011

Fast and Robust Generation of City Scale Urban Ground Plan

Jyh-Ming Lien · Fernando Camelli · David Wong

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 public domain via software like Google Earth. Although mostly created for visualization, these urban models can benefit many applications beyond visualization including video games, city scale evacuation plan, traffic simulation and earth phenomenon simulations. However, these urban models are mostly loosely structured and implicitly defined and require tedious manual preparation that usually take weeks if not months before they can be used. In this paper, we present a framework that produces well-defined ground plans from these urban models, an important step in the preparation process. Designing algorithms that can robustly and efficiently handle unstructured urban models at city scale is the main technical challenge. In this work, we show both theoretically and empirically that our method is *resolution complete*, efficient and numerically stable. Based on our review of the related work, we believe this is the first work that attempts to create urban ground plans automatically from 3D architectural meshes at city level. With the goal of providing greater benefit beyond visualization from this large volume of urban models, our initial results are encouraging.

J.-M. Lien · F. Camelli · D. Wong
E-mail: {jmlien, fcamelli, dwong2}@gmu.edu
George Mason University
4400 University Dr., Fairfax, VA, USA, 22030
Tel.: +1-703-993-9546

This work is supported in part by NSF IIS-096053 and College of Science Interdisciplinary Seed Grant, "Bringing Together Computational Fluid Dynamics Models and Geospatial Modeling" at George Mason University.

Keywords Urban geometry objects · Geometry processing · Numerical robustness · Simulation

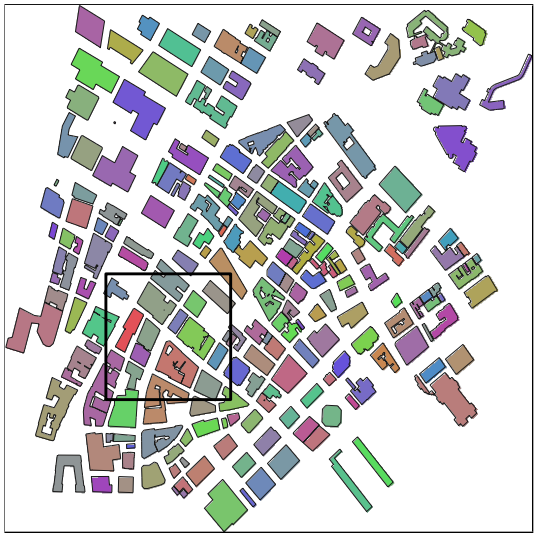
1 Introduction

Because of the recent advances in data acquisition and computer vision techniques, and collaborative efforts from hobbyists, almost every building in the major international cities has been re-constructed in the virtual world. Many of the geometric models depicting urban objects in these virtual cities are becoming broadly accessible to the general public due to tools and platforms like Google Earth and NASA World Wind, professional geographic information systems (GIS), e.g, ESRI ArcGlobe, and standards, such as CityGML [20].

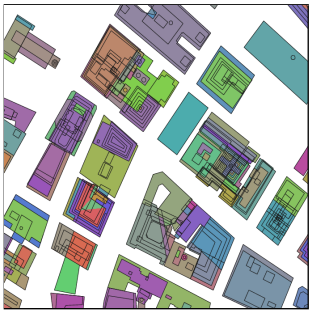
While these urban geometric models (or simply urban models) are designed mainly for visualization purposes, many applications beyond visualization should also benefit from these widely available data. Examples include video games, especially serious games, training, decision making, event design/planning, and scientific computing. For example, the urban models can provide realistic environments for *pursuit and evasion* [35] and *target chasing* games [22] for either entertaining or training purposes. These virtual cities also have potential applications in evacuation planning [32], traffic simulation or crowd control simulations [38] at city scale. These models have also been used for simulating Earth phenomena, such as the transport and dispersion of air pollutants and mudslides, in urban environments [3].

Despite these potential applications, urban models are created mainly for visualization and touring purposes. As a result, these "raw" urban models are usually not suitable for computations beyond visualization. We believe that one of the main difficulties comes from the loosely structured and implicitly defined geometric de-

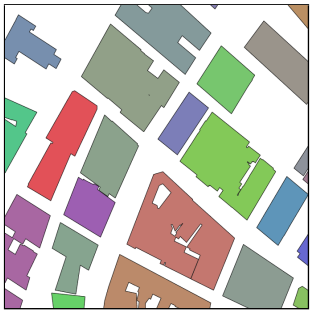
scription of the urban models. For example, in Google earth and ESRI shapefiles, each building is typically composed of a set of overlapping primitives or simple shapes. Without an explicit description of the buildings, it is difficult to understand which region of the terrain is occupied by which building, and this difficulty will later lead to problems in the aforementioned applications, such as walkable or traversable surface identification for video games, traffic, evacuation, crowd-control simulation, and building-terrain integration for 3D Earth phenomena simulation. Unfortunately, the preparation process of converting unstructured data to well-defined surfaces remains extremely labor intensive and can take several weeks to months [15].



(a) Downtown Manhattan, New York City



(b) overlapping footprints



(c) ground plans

Fig. 1 (a) There are 235 ground plans represented in this image. A region in (a) is shown in (b) and (c).

1.1 Main Contributions

With the objective to provide greater benefit beyond visualization from this large volume of urban models, we present a framework that processes implicitly defined urban models and produces well-defined ground plans. Although this is only an initial step toward this goal, our results are very encouraging. As we will demonstrate in the end of this paper, the ground plans can be readily used for the above-mentioned applications. Based on our review of the related work, we believe this is the first work that attempts to create urban ground plans automatically from 3D architectural meshes at city level.

Our main technical challenge is to design algorithms that can robustly and efficiently handle unstructured urban models at this scale. In this paper, we show both theoretically and empirically that our method is *resolution complete*, efficient and numerically stable. Here resolution complete means that the ground plan is topologically correct for the given resolution.

Moreover, because of the reconstruction and man-made errors, these urban models may not properly reside on the terrain and can contain surface degeneracies including holes and non-manifold features. We propose a footprint identification method based on the idea of surface decomposition. Details are discussed in Section 4. Then these footprints are united to generate non-overlapping ground plans. The union operation 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 perform the union operation on thousands of polygons representing the building footprints. In this paper, we developed an *online boundary evaluation* method that is designed to avoid computing the full arrangement induced by these polygons. To identify potential boundaries, our boundary evaluation method alternates the computation between CPU and GPU. To provide both robustness and efficiency, the boundary evaluation method heavily relies on an adaptive *nearest segments intersection* method that dynamically adjust the numerical precision to guarantee the correctness. These techniques are discussed in Sections 5 and 6. An example of the output generated by our method is shown in Fig. 1.

2 Related Work

Although methods have been developed to recover the 3D shape of roof surfaces, e.g., [13], building ground plans usually come from aerial data [5], the digitization and vectorization of cadastral maps or from surveying measurements. Recently, methods are proposed to extract ground plan from LiDAR data [27,14]. To the best of our knowledge, no work has focused on automatically processing urban models for simulation. Existing research often resorts to laborious manual manipulations

of the geometry data representing the topography and buildings to produce a coherent and consistent geometrical representation of the surface including landscape and buildings [15].

Although there exit 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 [23, 4, 16]. For example, ideas, such as levels of detail or texture mapping, based on the location of the viewpoint are useful for rendering but these view-dependent tricks are no longer applicable to simulation.

An important step in the proposed work is to compute the union of many footprints. The problem of geometric boolean operations has been studied more than three decades and the main focus of the research is on the *robustness* of the computation because many numerical errors and degenerate cases can creep in during the computation and 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 algorithm for computing the union of a very large number of polygons. In our example, there can have more than thousands of elements. Naïvely computing the union between pairs of elements can take very long time. A brief review of the techniques are discussed below.

2.1 Processing Urban Models

Most geometry processing methods on urban models focus on visualization. For example, Chang et al. [4] 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. Another example is Cignoni [6] who presents BlockMaps strategy that 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 [25, 34, 37, 19] and rarely focus on the city-level ground plans. For example, recently, Kada and Luo [19] simplify a building ground plan using the ideas of reducing the number line in the arrangement. See a survey in [23] for more related work.

Little work focused on aggregating and simplifying large scale ground plans. Wang and Doihara [39] cluster the buildings using strategy similar to MST extraction. The clustering process is performed on a graph whose nodes are the buildings and whose edge weight is the distance between the building centers. Clustered buildings are aggregated and simplified. Rainsford and Mackaness [29] propose a template-based approach that matches the rural buildings to a set of 9 templates. The floor plans are extracted and simplified before matching the a template. The matched template is then deformed and transformed to fit the floor plan. This method is limited to the number of templates.

2.2 Geometric Union

The problem of geometric union is extensively studied. Several combinatorial and algorithmic problems in a wide range of applications, including linear programming, robotics, solid modeling, molecular modeling, and geographic information systems, can be formulated as problems of the union of a set of objects. A recent survey on the geometric union operation can be found in [1]. Briefly, the study of the union 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 [18, 31, 21]. Starting in the mid 1990s, research on the complexity of the union of geometric objects has shifted to the study of instances in three and higher dimensions.

There exist several tools to compute the union of two polyhedra, such as CGAL [10] and Autodesk Maya. These tools are designed to handle the union operation of a small number of geometries. However, the scale of the number of the building models that we will consider in this project can be in a much larger. It is clear that the existing tools will suffer from various computational issues, such as robustness and efficiency. In this paper, we develop a method to increase the efficiency of the union operation by avoiding generating a full arrangement. From our experience with the union operation of the shape files, 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 Boolean, 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 [17] or spatial hash table [36].

The robustness issue has been studied in great depth since it is likely that the output geometry of the union operation is non-manifold while the input models are manifold and well behaved [33,30]. In order to provide both efficiency and robustness, we present a segment intersection method that can dynamically adjust the needed numerical precision.

3 Overview of Our Method

In this section, we provide an overview of our method using Algorithm 3.1. The input of the algorithm is a set of building model $\mathcal{P}$ describing urban objects, such as buildings, bridges, and landmarks. Output is a set of non-overlapping polygons $\mathcal{G}$ representing the ground plans of $\mathcal{P}$.

Algorithm 3.1: UrbanGroundPlan($\mathcal{P}$)

footprints $\mathcal{F}$ = Projection($\mathcal{P}$)
building blocks $\mathcal{B}$ = BlockExtraction($\mathcal{F}$)
for each block $b \in \mathcal{B}$
do
- $G_b = \emptyset$ **comment:** b's ground plan
- **while** ($S_b \neq \emptyset$)
  - **do**
    - $s = S_b.pop()$
    - g_s = ExtractBoundary(s)
    - $G_b = G_b \cup g_s$
    - Update(b, G_b, S_b)
- $\mathcal{G} = \mathcal{G} \cup G_b$

return ($\mathcal{G}$)

In Algorithm 3.1, first, a set of footprints $\mathcal{F}$ are created from the input building polyhedra $\mathcal{P}$. Note that we abuse the term "footprint" to distinguish the ground plan of each mesh in $\mathcal{P}$ from the ground plan of the urban models, which is the union of the footprints. In our implementation, the sub-routine Projection, $\mathcal{F}$ is a set of 2d polygons projected from the *lower wall boundaries* of the polyhedra in $\mathcal{P}$. In Section 4, we will discuss a surface decomposition method to identify $\mathcal{F}$.

Next, a set of blocks $\mathcal{B}$ are computed by rasterizing $\mathcal{F}$. Each building block $b \in \mathcal{B}$ consists of (1) a set of *potentially* overlapping footprints $F_b \subset \mathcal{F}$, (2) a set of connected pixels R_b rasterized by F_b, and (3) a list of pixels called *seed pixels* S_b that will be used as the starting points for boundary extraction. This step is done on GPU kernel functions and will be discussed in Section 5.

Finally, an iterative process is applied to each building block $b \in \mathcal{B}$ to extract the united ground plans G_b from the footprints F_b and the seed pixels S_b. The output ground plan G_b is a set of polygons that may have holes. A boundary g_b of G_b is extracted using a seed and the function ExtractBoundary that implements the online boundary evaluation method discussed in Section 6. The iteration process terminates when there all seeds are processed. Note that the for-loop can be trivially parallelized since each $b \in \mathcal{B}$ is independent.

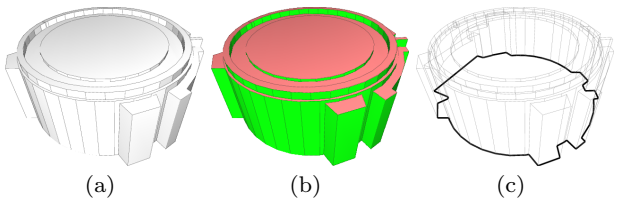
Seed pixels and Update function. The seed pixel is a pixel that must overlap with an external or hole boundary of the ground plan G_b and is the key for boundary extraction. Initially, S_b contains only a single seed pixel, i.e., the rightmost pixel in R_b. The new seed pixels are identified in each iteration through the Update function. In addition to hole boundaries, due to discretization resolution in R_b, F_b may also have more than one external boundaries (e.g., when two or more buildings are very close but do not overlap). The Update function essentially determines the pixel-wise difference between the rasterization of the ground plan G_b discovered so far and the block pixels R_b. When there are non-empty pixels outside G_b, we know that there must be more external boundary. When there are empty pixels inside G_b, we know that there must be hole boundary inside G_b. The Update function identifies these unbounded components and then determines their extreme pixels (e.g., the rightmost) as the new seed pixels. Update is also implemented as a GPU kernel function.

4 Footprint Identification

Urban objects come in many different formats. For example, architectural structures in ArcGIS are collectively represented by extruding the footprints defined in a given ESRI shapefile [9]. However, in most 3D city models, meshes are defined without footprints. These include both manually created models (e.g., in Google Earth, the are defined by 3D COLLADA meshes.) or automatically generated architectural structures [13]. In addition, several open source GIS tools have also adopted CityGML standard[20], a markup language for describing urban objects, where buildings can be explicitly defined as 3D meshes (i.e., CompositeSurface).

When the inputs are a set of 3d polyhedra $\mathcal{P}$, we need to identify their footprints in order to generate the ground plans. One approach is to compute the intersection between the terrain and a polyhedra P using a collision detection library. However, these urban objects are created with various modeling, measuring and reconstruction errors. The geospatial information of these urban objects (e.g., location and orientation from the KML files in Google Earth) may also be inaccurate. Because of these errors, a building component $P \in \mathcal{P}$ may reside partially on the terrain, and the intersection between P and the terrain can be *non-simple*

or even empty. Therefore, it is unreliable to extract the footprint based on the intersection.



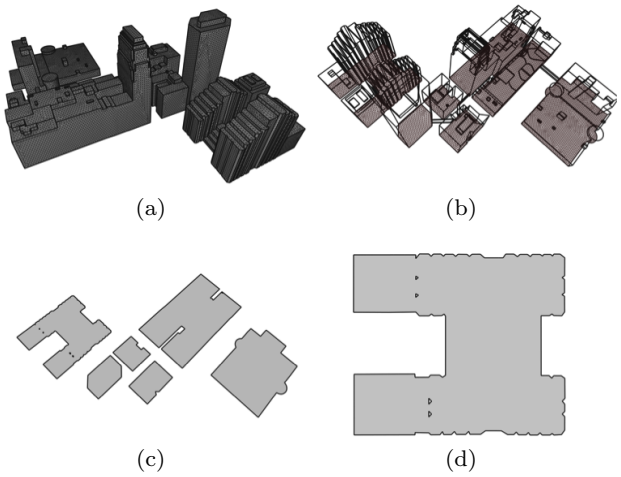
(a) (b) (c)

Fig. 2 A footprint (c) extracted from a building model (a) based on surface decomposition (b), in which green facets are the walls and red facets are the ceilings.

We propose to identify the footprint by decomposing the surface of polyhedra in $\mathcal{P}$ into wall, floor and ceiling patches. A surface patch is a set of connected facets. Floors and ceilings are surface patches that comprises nearly horizontal facets with the normal direction pointing toward the negative and positive vertical directions, respectively. Note that P may have multiple floors and ceilings. Walls are surface patches that are neither floors nor ceilings. The floor and celling patches are determined using an iterative bottom-up clustering approach adopted from the idea of *variational shape approximation* [7]. As in the idea of proxy [7] a critical property that we attempt to maintain during the patch construction process is the difference between the normal directions of the facets in the patch and representative plane (i.e., the proxy) of the in smaller than a user defined value. In our cases the proxy is always a horizontal plane. An example of the decomposition is shown in Fig. 2.

Once the mesh is decomposed in to wall, floor and ceiling patches, the footprints are identified from the *lower boundaries* of the wall patches. It is important to realize that many of these urban objects are created with openings (holes) at the bottom, therefore, it is possible that there are no floors identified or the floor patches may contain holes. For each boundary ∂ from the wall patches except the ceiling boundaries, we determine if ∂ is a footprint by check if ∂ forms a local minimum (i.e., the vertices adjacent to ∂ are higher that those in ∂). The footprint identified using the proposed method is shown in Figs. 2 and 3. Fig. 3(a) shows a group of buildings in Oklahoma City. The floor pathces of the group of buildings is shown in Fig. 3(b). The footprint of the whole group is shown in Fig. 3(c). The footprint of a single building represented by a complex polygon is shown in a zoom-in area in Fig. 3(d).

In addition to produce the city ground plan, these footprint can also be used to modify the urban objects. For example, one can project these footprints to the terrain surface and translate the wall boundaries with the footprints to tightly integrate the urban objects and terrain.



(a) (b) (c) (d)

Fig. 3 Extracting the footprints of the buildings. (a) Group of buildings in down town Oklahoma. (b) Footprints and the buildings. (c) Only the footprints. (d) Detail of the footprint. The ground plan is composed for an outer boundary and four hole boundaries.

5 Building Block Identification

Given a set of footprints $\mathcal{F}$, we would like to identify the overlapping footprints. This is commonly known as the *broad phase* collision detection and can be optimized using ideas such as spatial hashing [24] or plane sweep algorithms [2]. However, our goal is not to identify the intersections but to compute their union.

Our approach is a simple GPU-based computation. First, we rasterize each footprint with a unique color. Then the kernel function executed on each GPU core will connect each colored pixel to all the colored neighbors. Finally, all connected components are extracted from these connection to form blocks $\mathcal{B}$. For each $b \in \mathcal{B}$, a set of overlapping footprints F_b are identified based on the colors in the connected component. The right most pixel is identified as the first seed of the block.

Note that this method may ignore some footprints that are entirely covered by other footprints during the rasterization process. This is in fact a benefit since this footprint will not contribute to the boundary of the final ground plan. Another benefit of using GPU-based approach is that some small boundaries can be automatically eliminated from the ground plans due to rasterization resolution. For example, these small boundaries usually have small effect on the simulation results but consumes a significant amount of computation time. Therefore, the resolution of the rasterization can be a

user defined value to control the desired level of detail. In our implementation, we simply set resolution by ensuring that the bounding box of the smallest footprint is covered by at least 4 pixels.

6 Online Boundary Evaluation

Given a set of overlapping footprints and a seed pixel, our goal here is to extract a well-defined boundary of a ground plan. More specifically, we need to compute the boundary of the union of a set of polygons representing the footprints. The main idea of our approach is to *incrementally* extract the boundary of the arrangement induced from the input polygons. During the extraction process, we repetitively extend the extracted boundary by maintaining its desired topological properties.

Let $F_b = \{P_i\}$ be a set of polygons. Our goal is to compute the boundary of $\partial(\bigcup_i P_i)$ from a seed. To simplify our notation, we let $Q = \partial(\bigcup_i P_i)$. 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., $\mathbf{v_i} = \overrightarrow{p_i\, p_{i+1}}$, and the outward normal $\mathbf{n_i}$.

6.1 Determine the Seed from a Seed Pixel

A seed is a pair of a vertex r and an incident edge e_r of r from $\{P_i\}$ that are guaranteed to be on the boundary of Q. Given a seed pixel s, we can determine the seed by processing the vertices and edges overlapping with s. First, fining r is easy. From all vertices of $\{P_i\}$, we let r be the extreme point in the direction to an empty cell neighboring to s. Without loss of generality, we assume r is the rightmost vertex in b. Next, we let e_r be an edge incidents to r such that e_r's outward normal has the largest x coordinate among all the edges incidents to r. It is simple to show that r must be a vertex of Q and e_r must contribute to one or multiple edges of Q. See Fig. 4(a).

6.2 Extracting Boundary from a Seed

Traditionally, $\{P_i\}$ contains only two elements, and the boundary of Q is determined by computing the arrangement of the edges of $\{P_i\}$, which is a subdivision of the space into vertices, edges and faces (cells) from a set of line segments. One way to extract the boundaries from such an arrangement is by finding all the faces that have positive winding numbers [11,40]. However, computing the arrangement can be time consuming, i.e., $O(n^2)$ for n line segments. Our method skips arrangement computation and find the boundary by computing the intersections on the fly. To bootstrap the extraction process, we start from the seed (r, e_r). Our method then proceeds by incrementally discovering the vertices and edges of Q from r and e_r. To abuse the notation a little bit, we let r be the latest vertex of Q discovered, and let e_r be an edge of $\{P_i\}$ that contains an edge of Q. Therefore, in every incremental step, our method will need to (1) identify which portion of e_r belongs to Q, and (2) identify the next r and e_r until all edges of Q are discovered.

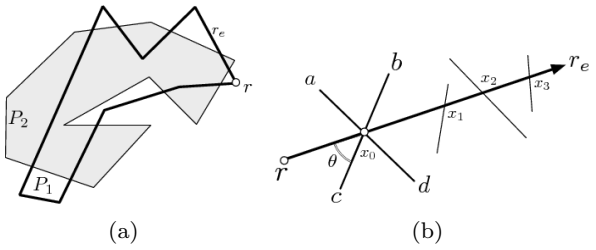


Fig. 4 (a) The union of two polygons P_1 and P_2. The vertex r is the rightmost vertex of P_1 and P_2, and e_r is the edge incident to r whose outward normal has the largest x coordinate among all the edges incidents to r. The pair r and (a subset of) e_r must be on $\partial(\bigcup_i P_i)$. (b) Given the last vertex r and a potential edge e_r discovered in the extraction process, the segment $\overline{rx_0}$ must be an edge of Q and the next r is x_0 and the next r_e is the edge containing $\overline{x_0 c}$.

To identify the portion of e_r that contributes to Q, let x_j be a sorted list of intersections between e_r and other line segments $e_j \neq e_r$ in $\{P_i\}$. The intersections x_j are sorted in non-decreasing order using the distance to r. Therefore x_0 is the intersection closest to r. Now we claim that x_0 must be a vertex of Q and the segment $\overline{rx_0}$ between r and x_0 must be an edge of Q. This observation is proved in Lemma 1

Lemma 1 *Let x_0 the the closest intersection to r. We say that x_0 must be a vertex of Q and the segment $\overline{rx_0}$ between r and x_0 must be an edge of Q.*

Proof Assuming that x_0 is not on ∂Q. Then x_0 must be interior to Q. Since we know that r is a vertex of Q, when we move from r to x_0, there must be a point $x' \in \partial Q$ before we reach the interior of Q. If we wish to remain on the boundary of Q, we must move the another edge of Q at x'. Therefore, x' must be an intersection of e_r and another segment from $\{P_i\}$. However, we know that x_0 is the intersection closest r. This means x_0 cannot be interior to Q, and in fact x_0 and x' must be the same point and the segment $\overline{rx_0}$ must be on ∂Q.

Therefore, $\overline{rx_0}$ is an edge of Q and x_0 becomes the next r, i.e., the last vertex discovered. In the second

step, we need to find out which edge of $\{P_i\}$ (that is not e_r) incident to x_0 will contain an edge of Q. Let $S = \{s_j\} \setminus e_r$ be a set of line segments incident to x_0 excluding e_r. Then to compute the next e_r, we solve the following:

$$\arg\min_{s_j \in S} \Theta(e_r, s_j) \ ,$$

where Θ is a function measuring the clockwise angle between e_r and s_j. Intuitively, the next e_r will be a line segment that makes the largest right turn from the current e_r at x_0. Now, with r and e_r updated, we repeat the process until a closed loop is found. See Fig. 4(b).

There are many advantages over the existing approach. First, in contrast to the traditional boolean operation approach, the proposed method can handle arbitrary number of elements in $\{P_i\}$ at once. 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. Thrid, the proposed method can handle degenerated cases easily, i.e., two polygons touch at a single vertex or a line. The proposed approach can even handle non-simply polygon, whose edges may self intersect, and polychain, which does not form a loop or enclose an area.

6.3 Robust and Efficient Nearest Intersection

As we have seen earlier, the main step that we used to determine the external boundary of the union 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 [28] and can be inefficient if exact arithmetic is used to overcome the numerical problems.

In this section, we will discuss our approaches to handle this problem. We will show that the 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 [8]. 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 of the 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 closest to p than other segments in $\mathcal{S}$. That is, given s, p and $\mathcal{S}$, we try to solve the following:

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

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. 6.3 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 $\overrightarrow{qr} \cdot \mathbf{t_n}$, where $r \in t$ is a point on t and $\mathbf{t_n}$ is the normal direction of t. If v is greater than zero, we say q is visible by t. If v is zero, the q is on t. Otherwise, we say q is on t.

Now, our goal is to find a point $q \in s$ that is invisible by a segment but is visible by 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 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 segments then that means that all segments in $\mathcal{O}$ intersects s at q and are equidistance 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 determine in the 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 segment in $\mathcal{S}$. This happens when the size of $\mathcal{S}$ is greater than one while the length of the search range collapse 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 ap-

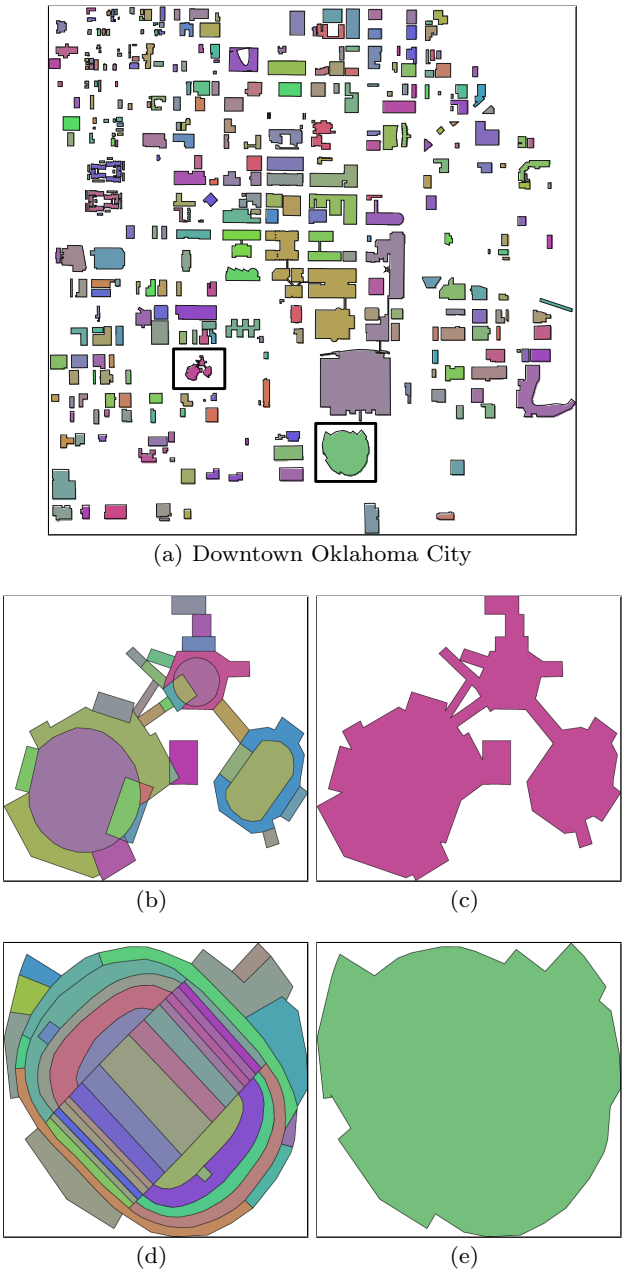
proach that computes the (parameterized) intersection between two line segments will require two divisions, 14 multiplications and 10 summations to compute the parameterizations of the intersection. As a result much higher precision is needed for the traditional approach. More importantly, the traditional 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.

7 Results

Our framework is implemented in C++ and NVIDIA CUDA C. The exact number type uses MPFR [12]. We use three examples shown in Figs. 1, 5, and 6. These are areas from Downtown Oklahoma City and New York City. Several interesting regions in these maps are highlighted to show complicated overlapping footprints (e.g., Fig. 1(b) and 5(d)), non-manifold ground plans (e.g., Fig .5(c) and 6(c)), and narrow error-prone features (see Fig. 6(e)). As shown in these examples, our new approach takes only seconds on the same dataset, and successfully handles degenerated cases.

The total running time for generating the ground plans in Fig. 1 is 54 milliseconds. For generating the ground plans in Fig. 5, our framework takes 124 milliseconds. Finally, Fig. 6 takes 339 milliseconds. All these experiments are performed on a dual core 2.54 GHz Intel CPU and NVIDIA GeForce 9400M (with 16 cores). To show the significance of our results, we have attempt to compute the union of all the buildings in our Oklahoma city dataset using the union functionality provided by ArcGIS. We found that ArcGIS takes hours to complete the computation and requires specific types of overlaps between the components in order to generate successful unions. Therefore, our proposed approach is designed to tackle these serious defects to provide both robustness and efficiency.

Application: Dispersion Simulation. We used the identified ground plans to merge the seamless surface of the buildings and 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 [26]. 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. Fig. 7 shows snapshots of air pollutant dispersion simulation in the integrated Oklahoma City model using the ground plans in Fig. 5.



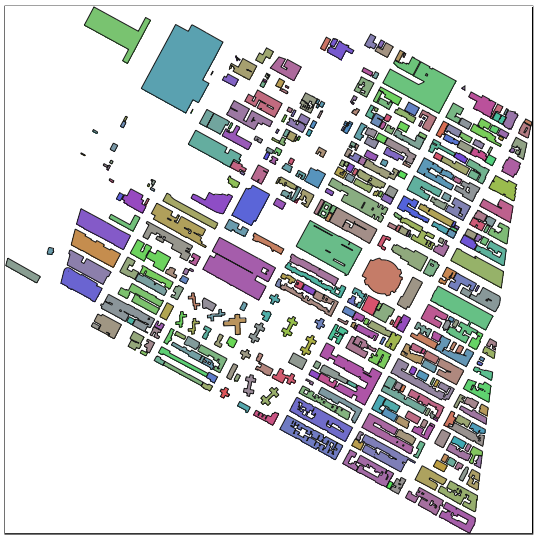
(a) Downtown Oklahoma City

(b) (c) (d) (e)

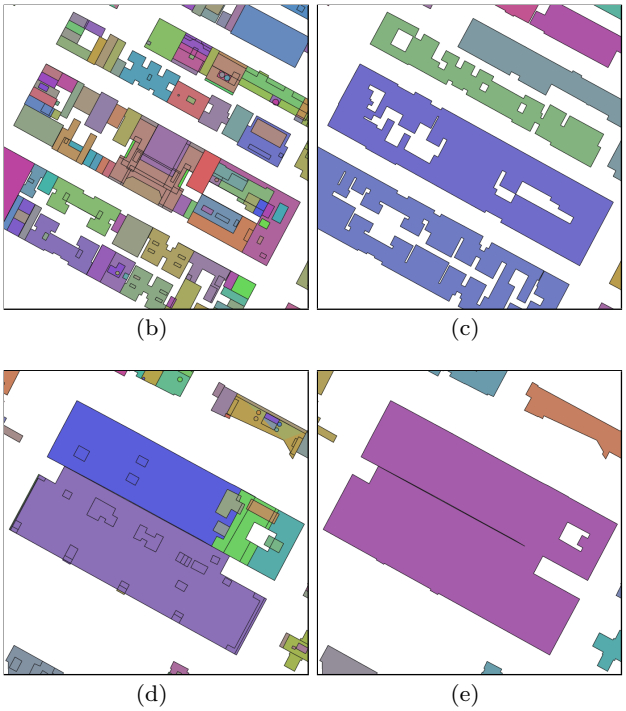
Fig. 5 In total 358 ground plans are represented in this image. There ground plans are created from 1454 footprints. Two interesting regions are highlighted.

8 Conclusion

As 3D geometric models describing urban objects, such as buildings and bridges, are becoming widely available via virtual global platforms, using these urban models to perform large scale simulations can provide significant benefit to many communities. In this paper,



(a) A larger NYC example



(b) (c) (d) (e)

Fig. 6 A residential area in NYC. There are 409 ground plans represented in this image. These ground plans are generated from 5996 footprints. Two interesting regions are highlighted.

we described the first known framework for extracting ground plans from urban models. Our method first extracts footprints from individual meshes and then iteratively determines the ground plans by alternating the computation between CPUs and GPUs. In the core of our framework is an online boundary evaluation method that is theoretically guaranteed to produced correct boundary. The numerical robustness and efficiency is

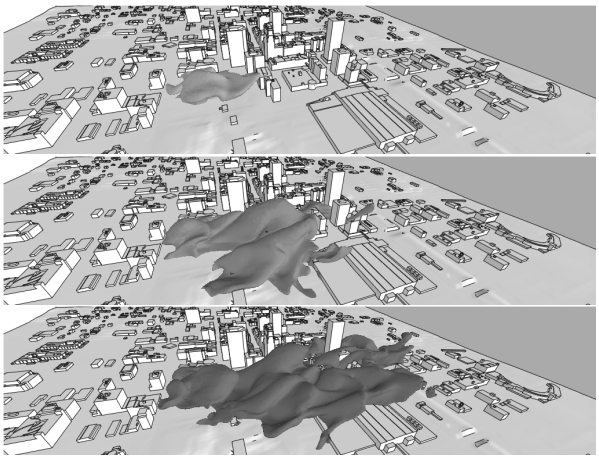


Fig. 7 A cloud is depicted at 3 different time instances in the integrated Oklahoma City model. The cloud is transported and diffused by the effects of the wind and turbulence. Clouds at 100 (top), 250 (mid) and 500 seconds (bottom) from the beginning of the release.

provided by an segment intersection algorithm based on the idea of dynamic precision. Finally, we have empirically shown that the results of our method can be readily used for integrating the building meshes to DEM terrain models, which is a process that usually requires weeks if not months of laborious manual process.

Limitations. Our method is designed to handle meshes of different qualities. However, our current implementation does not work well if the mesh is made of triangle soup, which is sometimes found in Google Earth. It will require an addition mesh processing step to discover the mesh connectivity in order to distinguish wall, ceiling and floor surface patches. Moreover, although our method provides some degree of simplification based on the rasterization resolution, there still exist small features, such as narrow gaps and deep but narrow cavities (see Fig. 6(e)), in the final ground plans. As we have mentioned, existing works on urban model simplification either focus on application in visualization or on the simplification of a single building. Further research is needed to produce simplified ground plans for city scale simulations.

References

1. Agarwal, P., Pach, J., Sharir, M.: 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 pp. 9–48 (2008)
2. Bentley, J.L., Ottmann, T.A.: Algorithms for reporting and counting geometric intersections. IEEE Trans. Comput. **C-28**(9), 643–647 (1979)
3. Camelli, F.: 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)

4. Chang, R., Butkiewicz, T., Ziemkiewicz, C., Wartell, Z., Ribarsky, W., Pollard, N.: Legible simplification of textured urban models. IEEE Computer Graphics and Applications **28**(3), 27 (2008)
5. Cheng, L., Gong, J., Chen, X., Han, P.: Building boundary extraction from high resolution imagery and lidar data. ISPRS08, p. B3b **693** (2008)
6. Cignoni, P., Di Benedetto, M., Ganovelli, F., Gobbetti, E., Marton, F., Scopigno, R.: Ray-Casted BlockMaps for Large Urban Models Visualization. In: Computer Graphics Forum, vol. 26, pp. 405–413. Wiley Online Library (2007)
7. Cohen-Steiner, D., Alliez, P., Desbrun, M.: Variational shape approximation. ACM Trans. Graph. **23**(3), 905–914 (2004)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge, MA (2001)
9. ESRI, E.: Shapefile Technical Description. ESRI. INC, http://www. esri. com (1998)
10. Fabri, A., Giezeman, G., Kettner, L., Schirra, S., Sch "onherr, S.: On the design of CGAL a computational geometry algorithms library. Software Practice and Experience **30**(11), 1167–1202 (2000)
11. Flato, E.: Robust and E cient Construction of Planar Minkowski Sums. Ph.D. thesis, Tel-Aviv University (2000)
12. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software (TOMS) **33**(2), 13 (2007)
13. Frueh, C., Zakhor, A.: 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, vol. 2. IEEE (2003)
14. Hammoudi, K., Dornaika, F., Paparoditis, N.: Extracting building footprints from 3D point clouds using terrestrial laser scanning at street level. ISPRS/CMRT09 **38**, 65–70 (2009)
15. Hanna, S., Brown, M., Camelli, F., Chan, S., Coirier, W., Hansen, O., Huber, A., Kim, S., Reynolds, R.: Detailed simulations of atmospheric flow and dispersion in downtown Manhattan. Bulletin of the American Meteorological Society **87**(12), 1713–1726 (2006)
16. Haunert, J., Wolff, A.: Optimal simplification of building ground plans. In: Proceedings of XXIst ISPRS Congress Beijing 2008, IAPRS Vol. XXXVII (Part B2), pp. 372–378 (2008)
17. Haverkort, H.: Introduction to bounding-volume hierarchies (2004). Part of the PhD thesis, Utrecht University
18. Hwang, Y.K., Ahuja, N.: Gross motion planning – a survey. ACM Comput. Surv. **24**(3), 219–291 (1992)
19. Kada, M., Luo, F.: Generalisation of building ground plans using half-spaces. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences **36** (2006)
20. Kolbe, T.: Representing and exchanging 3D city models with CityGML. 3D Geo-Information Sciences pp. 15–31 (2009)
21. Laidlaw, D., Trumbore, W., Hughes, J.: Constructive solid geometry for polyhedral objects. ACM SIGGRAPH Computer Graphics **20**(4), 170 (1986)
22. LaValle, S., Gonzalez-Banos, H., Becker, C., Latombe, J.C.: Motion strategies for maintaining visibility of a moving target. In: Proceedings IEEE International Conference on Robotics and Automation, vol. 1, pp. 731–736 (1997)
23. Lee, D.: Geographic and cartographic contexts in generalization. In: ICA Workshop on Generalisation and Multiple Representation, Leicester, UK, August. Citeseer (2004)
24. Lefebvre, S., Hoppe, H.: Perfect spatial hashing. ACM Transactions on Graphics (TOG) **25**(3), 579–588 (2006)
25. Lichtner, W.: Computer-assisted processes of cartographic generalization in topographic maps. GeoProcessing **1**(2), 183–199 (1979)
26. Löhner, R., Cebral, J., Camelli, F., Appanaboyina, S., Baum, J., Mestreau, E., Soto, O.: Adaptive embedded and immersed unstructured grid techniques. Computer Methods in Applied Mechanics and Engineering **197**(25-28), 2173–2197 (2008)
27. Neidhart, H., Sester, M.: Extraction of building ground plans from Lidar data. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences **37**, 405–410
28. O'Rourke, J.: Computational Geometry in C, 2nd edn. Cambridge University Press (1998). URL `http://cs.smith.edu/~orourke/books/compgeom.html`
29. Rainsford, D., Mackaness, W.: Template matching in support of generalisation of rural buildings. In: Advances in Spatial Data Handling, 10th International Symposium on Spatial Data Handling, pp. 137–152 (2001)
30. Ricci, A.: A constructive geometry for computer graphics. The Computer Journal **16**(2), 157–160 (1973)
31. Richards, F.: Areas, volumes, packing, and protein structure. Annual Review of Biophysics and Bioengineering **6**(1), 151–176 (1977)
32. Rodriguez, S., Amato, N.: Behavior-based evacuation planning. In: Robotics and Automation (ICRA), 2010 IEEE International Conference on, pp. 350–355. IEEE (2010)
33. Rossignac, J., Requicha, A.: Constructive non-regularized geometry. Computer-Aided Design **23**(1), 21–32 (1991)
34. Sester, M.: Generalization based on least squares adjustment. International Archives of Photogrammetry and Remote Sensing **33**(B4/3; PART 4), 931–938 (2000)
35. Sourabh Bhattacharya, S.H.: Approximation schemes for two-player pursuit evasion games with visibility constraints. In: Proceedings of Robotics: Science and Systems IV. Zurich, Switzerland (2008)
36. Teschner, M., Heidelberger, B., Müller, M., Pomeranets, D., Gross, M.: Optimized spatial hashing for collision detection of deformable objects. In: Proceedings of Vision, Modeling, Visualization VMV'03, pp. 47–54. Citeseer (2003)
37. Thiemann, F., Sester, M.: 3D-symbolization using adaptive templates. Proceedings of the GICON (2006)
38. Vo, C., Harrison, J.F., Lien, J.M.: Behavior-based motion planning for group control. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (2009)
39. Wang, P., Doihara, T.: Automatic Generalization of Roads and Buildings. In: ISPRS Congress (2004)
40. Wein, R.: Exact and efficient construction of planar Minkowski sums using the convolution method. In: Proc. 14th Annual European Symposium on Algorithms, pp. 829–840 (2006)