Volume xx (*200y*), *Number z, pp. 1–17*

Enhancing Urban Façades via LiDAR based Sculpting

Jiju Peethambaran[†1] and Ruisheng Wang[‡1,2]

[1]Geospatial Intelligence Lab, Department of Geomatics Engineering, Schulich School of Engineering, University of Calgary, Canada
[2] Beijing Advanced Innovation Center for Imaging Technology, Capital Normal University

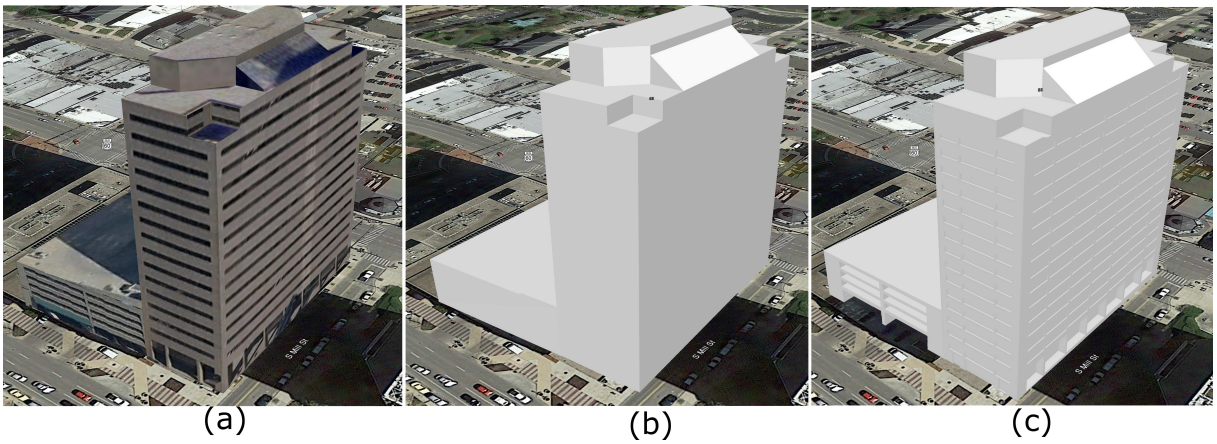

Figure 1: *An example showing the detail enhancement of a coarse model taken from Google earth (GE). (a) Photo-textured model from GE (b) Coarse model without textures and, (c) The model enhanced using the proposed framework. We used the coarse model of world trade center, Lexington, Kentucky and its parkade for the illustration.*

Abstract

Buildings with symmetrical façades are ubiquitous in urban landscapes and detailed models of these buildings enhance the visual realism of digital urban scenes. However, a vast majority of the existing urban building models in web-based 3D maps such as Google earth, are either less detailed or heavily rely on texturing to render the details. We present a new framework for enhancing the details of such coarse models, using the geometry and symmetry inferred from the LiDAR scans and 2D templates. The user defined 2D templates, referred to as coded planar meshes (CPM), encodes the geometry of the smallest repeating 3D structures of the façades via face codes. Our encoding scheme, take into account the directions, type as well as the offset distance of the sculpting to be applied at the respective locations on the coarse model. In our approach, LiDAR scan is registered with the coarse models taken from Google earth 3D or Bing maps 3D and decomposed into dominant planar segments (each representing the frontal or lateral walls of the building). The façade segments are then split into horizontal and vertical tiles using a weighted point count function defined over the window or door boundaries. This is followed by an automatic identification of CPM locations with the help of a template fitting algorithm that respects the alignment regularity as well as the inter-element spacing on the façade layout. Finally, 3D boolean sculpting operations are applied over the boxes induced by CPMs and the coarse model, and a detailed 3D model is generated. The proposed framework is capable of modeling details even with occluded scans and enhances not only the frontal façades (facing to the streets) but also the lateral façades of the buildings. We demonstrate the potentials of the proposed framework by providing several examples of enhanced Google Earth models and highlight the advantages of our method when designing photo-realistic urban façades.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based Modeling

1. Introduction

Reconstruction of urban scenes from large-scale 3D point clouds acquired through LiDAR (light detection and ranging) technol-

† jijupnair2000@gmail.com
‡ ruiswang@ucalgary.ca

ogy is an active area of research among computer graphics, computer vision and photogrammetry and remote sensing communities. Driven by an increasing demand for the virtual representations of urban areas in several applications such as urban planning, real-time emergency response training, digital mapping of mobile devices or vehicles, computer games and virtual tourism, the topic invites significant theoretical as well as commercial interests as underlined in a recent survey [MWA*13]. Huge public acceptance of web-based commercial applications such as Google earth 3D, Bing maps 3D and Apple maps, further motivates the researchers to focus on even the minute details pertaining to urban modeling and enhance the quality of the visualization without compromising on the algorithmic scalability and computational performance.

Visualizations in web-based 3D applications such as Google earth at large or medium scale are limited to relatively coarse building models, which consist of roof structures and planar façades. Most of these models have been created using airborne data, which, in most cases, cover the building façades only partially due to the viewpoint restrictions. As a consequence, many finer details of façades such as windows, balconies, doors, eaves and other concave portions are found to be either distorted or completely lost (See Figure 1(b)). While these visualizations are suitable for larger areas from elevated view points, a ground based presentation often demands a high degree of realism from a pedestrian view point. Moreover, 3D navigation applications within urban environments require fully interpreted urban scenes with the knowledge of building façades (doors and windows), roads, sidewalks, trees or parking spaces. In this context, generation of 3D city models, with rich details at ground level, represents one of the most desirable topics for pursuit, from an academic as well as commercial perspectives.

Contributions. In this paper, we propose a new framework to enhance the geometric details of the existing coarse building models (available in GE) with the help of corresponding LiDAR scan and 2D templates. We mainly focus on urban symmetric façades showing global rectilinear grid pattern [MZWVG07] shown in Figure 1 (a). In urban façades, symmetric relationships, in particular, the translational symmetry between the floors is quite common. Detecting the repeating patterns on the urban façades allow us to recover the occluded parts in the scan via replication and further helps in synthesizing urban model by replacing the basic geometric patches with the corresponding 3D templates created by the user [MPWC13]. We find an alternative to the traditional 3D template models in planar meshes that encodes the geometry of the architectural constructs commonly found on urban façades. The proposed framework combines the capabilities of both, physically based modeling (using LiDAR) and the procedural modeling, two prominent approaches to architectural reconstruction. Following are the specific contributions of this work.

- The concept of coded planar meshes along with its encoding scheme-a technique to encode different 3D structures commonly found on urban façades into planar meshes.
- A procedure for splitting the symmetric façades into floors and tiles based on a weighted point count function. Mutual information based floor replication procedure to replicate the most robust floors to the occluded parts is also developed and integrated into the splitting scheme.
- A template fitting procedure, that respects the alignment regularities as well as the inter-element spacing of façade layout, to automatically identify the locations of CPMs on the coarse model.

2. Related Work

There exists a vast body of literature on architectural and façade modeling. A survey by Musialski et al. [MWA*13] provides a comprehensive overview of automatic and interactive urban reconstruction techniques which use imagery or LiDAR scans acquired either from the ground or from the air. In this section, we focus only on works most relevant to our topic, in particular those on automatic, interactive and procedural techniques for architectural modeling and methods dealing with coarse model enhancements.

Automatic modeling. In general, there exist two approaches to create building models from LiDAR data: first by fitting pre-defined structural patterns such as planes, primitives or grammars and the second, data-driven methods such as 2.5D dual contouring [ZN11]. Poullis and You [PY09, PY11] introduced an automatic system that creates lightweight and water-tight polygonal 3D models from airborne LiDAR based on the statistical analysis of the geometric properties of the input data. In [ZN11], 2.5D modeling of buildings from LiDAR point cloud with topology control has been proposed, where 2.5D building topology has been defined as a set of roof, wall and point features. A few other automatic reconstruction methods adopt Manhattan-world assumptions [VAB12, VAB10] consisting of rectilinear structures. In [PV09], the authors exploit the knowledge about the feature's size, positions, orientations, and topology to recognize façade elements and subsequently construct the façades from terrestrial scans without any human intervention.

Interactive modeling. In their seminal work, Debevec et al. [DTM96], introduced a multi-view interactive method that combines geometry-based modeling with image-based modeling into one pipeline. Later, Sinha et al. [SSS*08] presented an interactive method to generate textured 3D models of architectures from a collection of unordered photos based on camera parameters and point clouds obtained through SfM. Recently, more methods for interactive reconstruction from laser scan have been proposed such as optimization based snapping [ASF*13], the snapping of smartboxes [NSZ*10] to LiDAR scan, procedural extrusions from building foot prints [KW11], semantic decomposition and reconstruction of low rise building from mobile LiDAR [LGZ*13] as well as a semi-automatic method which fuses laser scan data with photographs [LZS*11]. However, such interactive techniques are rather time consuming, expensive and labor intensive, requiring personnel with extensive training and experience to aid the modeling.

Procedural modeling. There has been a lot of research on procedural modeling of urban façades [WWSR03, MWH*06, MZWVG07, SM15], which focus on encoding and generating façade layouts using synthetic rules or grammars. A related concept, called inverse procedural modeling, deals within ferring a set of procedural grammars from photos of typical façades and using these as priors to aid further reconstruction efforts [WYD*14, TKS*13]. Downside of procedural modeling is that, it needs expert specifications of the rules and may be limited in the realism of resulting models and their variations. Furthermore, it is very difficult to formulate the required rules to construct exact existing buildings.

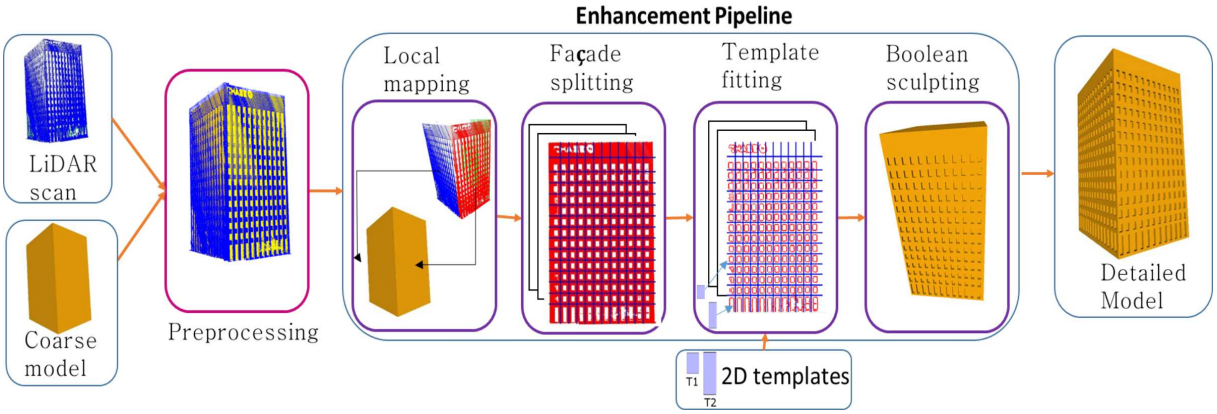

Figure 2: *Overview of the framework. Registered coarse model and LiDAR scan is fed into a pipeline consisting of four stages: (i) local mapping, (ii) façade splitting, (iii) template fitting; and (iv) boolean sculpting, and a detailed 3D model is generated.*

Coarse model refinement. Recently, fusion of laser scans or images with coarse building models found reasonable and efficient in modeling high quality building façades. A prominent work [NJGW15] in this direction reconstructs building details by automatically assembling 3D templates on coarse textured building models with the help of images. Xiao et al. [XFT*08] present an automatic approach for façade modeling that use images captured along streets. A façade is decomposed into rectilinear patches, and each patch is then augmented with a depth value optimized using the structure-from-motion depth data. In 2015, Verdie et al. [VLA15] introduced an approach to reconstruct urban scenes in the form of level of details (LOD) from coarse surface meshes. Integrated processing of LiDAR and image data [BH07] has been found to be useful to recover the building details. Another recent approach exploits geometric and radiometric point cloud information to enrich the façade details of LOD2 building models [TH15]. In summary, with the increasing availability of coarse models (thanks to the recent advances in MVS workflows such as Acute3D, Autodesk or Pix4D) in the web based applications, there exists a huge scope for refining the quality of such coarse meshes using the information embedded in other widely available input sources such as laser scans or images.

3. Overview

We consider urban scenes containing 3D coarse models (Google earth 3D model) and the mobile LiDAR acquired from the corresponding streets. A building and its laser scan from both, the urban scene and the mobile LiDAR, can be located and extracted with the help of geodetic information or using the information provided by the freely available digital maps such as Google maps or OpenStreet Maps of the corresponding street. Like many other urban reconstruction works [VAB12, FJZ05, VAB10], we adopt Manhattan world assumptions, where the buildings are made of rectilinear structures and exhibit extensive symmetry. It is to be noted that such buildings can be increasingly seen in the downtown areas of modern cities.

Preprocessing. Preprocessing stage is meant to align the model and the laser scan with each other. Like in [NSZ*10], an input GE model is first aligned with the three orthogonal axes in the Cartesian coordinate system such that the model is erected on XY-plane and the ground up-vector is aligned to the Z-axis. Triangular meshes are converted into a polygonal mesh by merging all the possible coplanar triangles. This apparently, give rise to polygonal faces representing façade walls and considerably facilitates the subsequent mapping and sculpting operations. The coarse model is represented using face-vertex lists, consisting of a list of vertices and a list of faces that points to its vertices. Each face is defined by a list of counter clock-wise oriented vertices. To align the model with the data, we first randomly sample the faces of coarse model and then apply ICP [BM92] registration technique, which considerably reduces the positional shift between the inputs.

Both, the laser scan and the coarse model are complementary to each other in terms of building attributes, i.e. while laser scan contains structural information of façade entities, the building outline is embedded in the coarse model. In other words, both the inputs carry partial information about the building, which adversely affects the accuracy of ICP registration. In particular, liDAR scan contains voids representing windows or doors of the building whereas the corresponding sampled coarse model lacks such holes. Consequently, feature points in the window/door regions of the model will falsely correspond to the features in the laser scan, leading to an incorrect alignment of the inputs as well as unnecessary computation of the algorithm in terms of correspondence finding. Defect-laden laser scan with missing data, noises and outliers, further affects the accuracy of registration. Considering these aspects, we expect the user to fine tune the alignment between ICP registered model and the LiDAR scan. Registration refinement can be performed using open source software such as Cloudcompare [clo].

Pipeline. Inputs to our façade refinement framework consist of a polygonal model (denoted by $\mathcal{M}$) of an urban building and the corresponding laser scan (denoted by $\mathcal{D}$), already registered with $\mathcal{M}$. The inputs pass through a refinement pipeline consisting of four stages as illustrated in Figure 2. The pipeline starts by decomposing the LiDAR scan into dominant wall segments and then establishing

an injective mapping between the decomposed segments and the coarse model faces. Each wall segment is then split into floors and tiles using horizontal and vertical splitting lines, respectively. To compensate for occlusions and sparse sampling, we replicate the most robust floors to the occluded areas, where the defective areas are determined using a heuristic formulated on the confident floor size. We employ a template fitting procedure on the split façade to find out the respective locations of user defined templates on the coarse model. Finally, boolean union and difference operations are applied on the boxes formed out of 2D templates and the coarse model to produce fine detailed building models. Each stage of the refinement pipeline is elaborated in Sections 4.1-4.4.

4. Façade Enhancement Framework

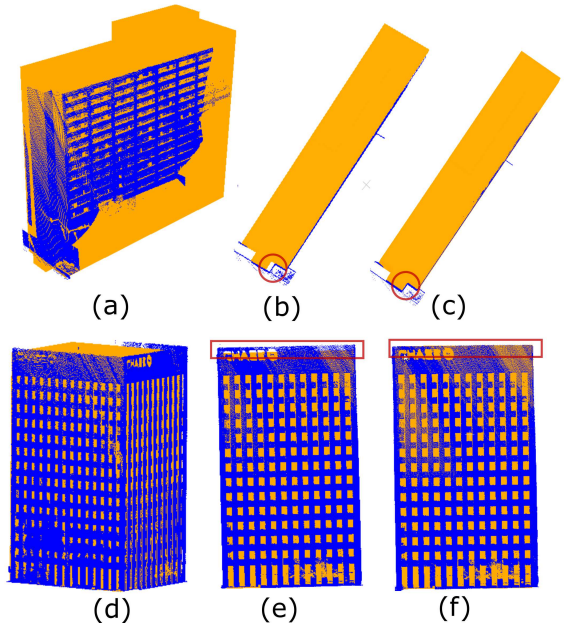

Figure 3: ***Motivation for local mapping and model rectification****. (a) & (d) Registered GE model and LiDAR scan of two buildings. Top views before (b) and after (c) model rectification, and front views before (e) and after (f) the height rectification. In Figures 3(b) & 3(e), deviation of the GE model (the encircled and the boxed-in regions) from the original dimensions (captured in LiDAR) is evident. Such GE models, unfaithful to the original dimensions of the buildings, can fail distance-to-model filtering based mapping and hence, clearly calls for an alternate mapping and model rectification algorithms.*

4.1. Local Mapping

Urban façades are often characterized by dominant planes with windows, balconies and doors. To extract meaningful façade entities from the LiDAR scan, it is essential to decompose the scan into planar segments, each representing frontal or lateral walls of the building. An accurate mapping of the decomposed segments to their corresponding polygonal faces is extremely important for transfering the extracted details to the coarse model. We observe that ICP registration and a simple *distance-to-model* filtering thereafter, are not adequate to segment the LiDAR scan due to various reasons. Mainly, the dimensions of less detailed and manually created building models (such as the models in Google Earth) often deviate from the actual measurements of the building, as shown in Figure 3. So, a distance based segmentation can lead to wrong classification of points existing at the boundary regions, i.e. LiDAR points corresponding to one polygonal face may be incorrectly assigned to the segment of its neighboring face. This will affect the whole process of detail transfer, especially when the wrongly assigned points carry valuable information about the façade. Therefore, we need a dedicated procedure to segment the raw point cloud and assign each segment to the polygons of the model.

For segmentation, we employ a Kernel Hough Transform (KHT) based technique [LO15], that generates segments consisting of approximately coplanar clusters of samples. The method has the advantage of being reasonably fast for large point cloud and works reasonably well for noisy data. The mapping algorithm first chooses the dominant planar segments from the set of segments obtained from KHT algorithm based on their sizes, deviation to Z-vector (0,0,1) and a user supplied segment count. The segment count represents the number of walls that are reasonably scanned in the data. For instance, a typical value of segment count for mobile LiDAR scan of a building with four dominant planar walls is three as the rear walls of the buildings are often unscanned in MLS. In our experiments, we used the values 1-3 for the segment count and this can be decided by the user based on the scanning quality and availability. Large k segments whose plane normals are nearly orthogonal to the Z-vector are chosen for further processing, where k is the segment count.

Let $S = \{s_1, s_2, ..s_k\}$ be the set of chosen segments. We are interested in the vertical walls representing the façades, and hence, only the coarse model faces which are orthogonal to the ground plane are considered for the correspondence finding. Let $F = \{f_1, f_2, .., f_n\}$, denotes the set of all such faces. To establish an injective mapping between the elements of S and F, we compute the misalignment scores between each face and segment by adding their respective deviations and proximity. The deviation among the face and segment planes, is quantified through the corresponding normals deviation. The proximity is defined as the spatial distance between segment and face planes. It is measured by the least Euclidean distance between the segment points and the face plane. A segment s_i is mapped to a face f_j that returns the lowest misalignment score.

Model rectification. Many coarse models deviate from the original buildings measurements, e.g. see Figure 3(b). The inconsistencies mainly happen in terms of their height, width or length. To rectify the width and length accuracies, we identify the faces that lie at a certain distance (0.2m) from their corresponding LiDAR segment and extrude them towards the LiDAR segment horizontally, i.e. along the normal direction of the segment plane (refer to Figure 3 (c)). Depending on the actual position of the face, the extrusion can be a push or pull operation [LWM14]. However, due to the acquisition constraints and sparse scanning of the top floors, it is extremely difficult to infer the correct building height from the LiDAR scan. Hence, the building height is altered only if

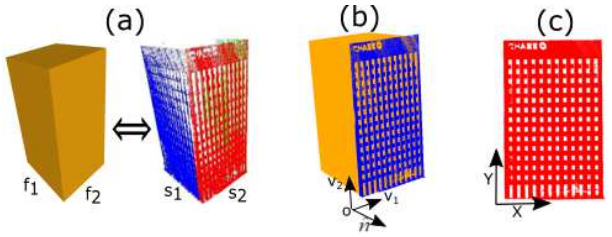

Figure 4: ***3D to 2D transformation****. The algorithm finds the correspondence between the decomposed segments (S) and the vertical polygonal faces (F) as shown in (a), then each segment, s_i is projected onto a 3D plane containing the corresponding face, f_j using the local coordinate frame as illustrated in (b), and the 3D points on the plane are transformed into 2D domain (c). For the sake of brevity, we have illustrated the transformation for only one segment in the figure.*

the heights of the mapped adjacent faces are less than the height of the corresponding segment bounding boxes. In such case, we extrude the common face at the higher elvation to the desired distance, as shown in Figures 3(e)-3(f). Otherwise, we leave the height unaltered. Our rectification procedure is simple, yet reasonable under Manhattan assumptions. Once the mapping and rectification is completed, we eliminate all the points lying inside $\mathcal{M}$ from each segment. This step is meant to remove the outliers due to laser reflection from the window curtains or the walls behind the windows, which otherwise will affect the hole detection and splitting stage.

2D conversion. Each segment, s_i is projected onto a 3D plane containing its f_j and then transformed to 2D domain. This is achieved by defining arbitrary 2D axes (v_1 and v_2) orthogonal to the f_j normal, $\hat{n}$ and an origin in the 3D plane (o) and using these axes to transform each 3D point to the corresponding 2D point (refer to Figure 4) as follows. $q_x = \langle (p-o), v_1 \rangle$ and $q_y = \langle (p-o), v_2 \rangle$, where p and q are the 3D and 2D points, respectively and $\langle . \rangle$ represents the inner product. We use the point-in polygon checking to eliminate the outliers lying outside the face boundaries. Final output of local mapping is a map data structure (denoted by $\mathcal{Z}$) consisting of model faces F and the corresponding 2D segments, and a list of 3D to 2D transformation parameters such as 2D axes and origin, $\mathcal{P}_i = \{o_i, v_{i1}, v_{i2}\}$, for each segment $s_i \in S$.

4.2. Façade Splitting

In this stage, we detect the general structure in a façade and subdivide it into smaller tiles, each holding a window, door or balcony (see Figure 5 for an illustration). The input is the map data structure, $\mathcal{Z}$ obtained from the local mapping stage (Section 4.1). For each 2D segment, the splitting function generates a set of horizontal and vertical lines subdividing the façade into floors and tiles. For brevity, we restrict the following discussions to a single 2D segment namely $s \in \mathcal{Z}$, since the same process is applicable to all the remaining segments in $\mathcal{Z}$. Let the corresponding face of s be $f \in \mathcal{Z}$.

Splitting locations of the façade are determined based on some weak architectural principle, put forward by Muller et al. [MZWVG07]: "horizontal splitting lines are included where vertical boundary lines are rare and horizontal boundary lines are dense, and vertical splitting lines in the opposite case". To incorporate such prior knowledge into our splitting function, we enumerate all the potential boundary lines from s. A key observation to identify the window or door boundaries is that laser pulses penetrate transparent surfaces such as window glasses, thereby creating holes in the LiDAR. Consequently, holes in LiDAR scan are reasonable representations of façade features such as windows. In principle, our splitting algorithm detects the holes in s and include the splitting lines (both horizontal as well as vertical) at locations where the hole boundaries are rare. An overall approach to our splitting is shown in Figure 5.

We compute the boundary of the s using the well known α-shape [EKS83] method. Outer boundary of s seldom contributes to the subdivision. Therefore, we remove all the boundary points, that lie at certain distance d from the convex hull edges as shown in Figure 5(c). While setting the d value, we should make sure that it does not remove interior points representing the façade features. Since many urban buildings exhibit rectangular outlines for the façades and convex-hulls of such rectangular façades often converge to the corresponding boundary polygons, smaller values of d (0.5 m in our experiments), are quite reasonable for the boundary points removal.

4.2.1. Splitting function

To meaningfully split the façade, we employ a sweep line algorithm that scans the façade points, s in two principal directions in 2D. Our splitting algorithm assumes that the façade features (windows, doors etc.) are well aligned with the respective 2D axes of s (defined in Section 4.1). In point set domain, a few façade splitting strategies have been proposed [SHFH11, SB03], especially for 3D points. Inspired from these functions, we propose a function which takes into account the weighted point count [SB03] of boundary points to the splitting lines. Weighted point count indicates that the points contribution is inversely proportional to its distance from the sweep line. However, unlike the real valued splitting functions in [SHFH11,SB03], we use a combination of real valued and binary valued functions for splitting. Let L_H denotes a horizontal line, then the horizontal splitting function is defined as in Equation 1.

$$F(L_H) = \sum_i 1 - \frac{|y_{L_H} - y_i|}{h} \tag{1}$$

In Equation 1, h denotes the vertical distance between adjacent elements (usually 1-2.5 meters for urban buildings) and y_i denotes the y-coordinate of each boundary point that lie within a distance $\frac{h}{2}$ from the splitting line L_H. Let $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ denote the bounding rectangle of the 2D projected vertices of face, f. Line sweeping algorithm evaluates the function $F(L_H)$ at each location y_L in the range, $[y_{min}, y_{max}]$. The corresponding y-coordinates $\{y_{L_1}, y_{L_2}, ..\}$ of the local minima are then identified as the horizontal splitting locations as shown in Figure 5(d). To correctly determine the splitting locations, we sample the entire $[y_{min}, y_{max}]$ with an interval size of 0.01 meters. Usually, horizontal floors are at least 2 meters high and hence, meaningless slices of smaller sizes (<2m) are eliminated and will not be considered for further processing.

The local minima of real functions are sensitive to the scanning

quality and outliers. There are high chances that a sparsely scanned element boundary may induce local minima at wrong places (for eg. splitting the horizontal edges of the element) and subsequently, lead to incorrect splitting. This can be alleviated by using a binary function (Equation 2) for vertical splitting. In the binary form, we no longer look for the local minima, instead, employ a median based method on '0' intervals to find the splitting locations. This has a clear advantage, i.e. even if the real function attains a local minimum value>0 over the poorly scanned element boundary, the function is still evaluated to 1 and avoids any chances of being picked as a splitting location.

Let L_V denotes a vertical line at location $x_{L_V} \in [x_{min}, x_{max}]$, the binary splitting function is formulated as in Equation 2.

$$F(L_V) = \begin{cases} 1 & \text{if } \sum_i 1 - \frac{|x_{L_V} - x_i|}{w} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

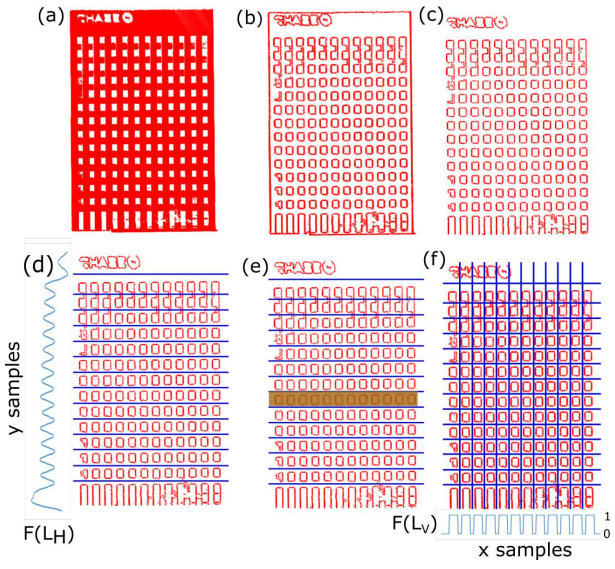

Figure 5: *Illustration of façade splitting. (a) Façade points, s (b) boundary points extracted using α-shape, (c) interior boundary points after removing boundary points close to the convex hull of s, (d) Horizontal splitting lines inserted at local minima of $F(L_H)$, (e) Most confident tile highlighted in orange color; and (f) Vertical splitting lines included at the median of zero intervals of $F(L_V)$.*

In Equation 2, w denotes the horizontal distance between two adjacent elements (0.2-0.5 meters in our experiment) and x_i denotes the x-coordinate of each boundary point that lie within a distance $\frac{w}{2}$ from the splitting line L_V. The function $F(L_V)$ outputs a series of ones and zeros intervals as illustrated in Figure 5(f). Local minima, i.e., output intervals consisting of one or more zeros imply the absence of façade entities. Consequently, if there is a 1-0 transition, the algorithm records all the splitting locations till a 0-1 transition occurs and then use the median of these splitting locations as the desired candidate splitting line for that specific interval. All the median x-coordinates $\{x_{L_1}, x_{L_2}, ..\}$ of the zero intervals constitute the locations for vertical splitting lines. It should be noted that only zero intervals lying between '1-0' and '0-1' transitions are considered as the splitting candidates, leaving out the zero intervals at the two ends (see Figure 5(f)).

On most of the high-rise urban façades, translational symmetry between the floors is very evident in vertical direction [MPWC13] than horizontal direction. Consequently, vertical splitting on any of the horizontal floors suffices to the vertical splitting of the entire façade. However, rather than splitting a random horizontal floor, we prefer to split the most confident horizontal floor, which is free of outliers, occlusions and noises. This leads to a robust and accurate vertical splitting. Most confident floors are identified according to the procedure detailed in the subsequent paragraphs. The splitting lines obtained on the most confident floor are extended to both directions till y_{min} and y_{max} to form the final vertical splitting lines. At this juncture, we would like to point out that our confident floor based splitting is mainly designed for façades with monotonous repititive structures and hence not suitable for façades with interlaced grids [SHFH11].

4.2.2. Floor replication

Due to the occlusions by trees or other physical objects while scanning, a few windows or doors may be completely missed in the LiDAR scan. Such missing data affect the overall splitting procedure, thereby leading to incorrect template fitting. Hence, it is essential to recover occluded windows or doors from the already detected horizontal patterns. Most of the urban façades are composed of repetitive elements and hence it seems reasonable to replace the occluded areas with a copy of the most confident element.

We define the confidence of each floor T, $Conf(T)$ in terms of its similarity to the neighboring floors and the floor coverage by the points (Equation 3).

$$Conf(T) = Sim(T) + Coverage(T) \quad (3)$$

In Equation 3, the similarity *Sim*, is measured by the average of the mutual information of T with its adjacent floors. For an extreme floor, we take the mutual information with its single neighboring floor as the similarity. In information theory and statistics, MI is a quantity that measures the relationship between two random variables, X and Y. Essentially, it quantifies the Kullback-Leibler distance [Kul59] between the two joint distributions, P(X = x,Y = y), and the product of their marginal distributions, P(X = x) and P(Y = y) as given in Equation 4

$$MI(X,Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x).P(y)} \quad (4)$$

Several applications including image based procedural modeling [MZWVG07] use MI as a similarity measure on image intensities. We adapt MI on intensities to the point set domain with the help of gridding (2D alternative to the 3D voxelization). Instead of image intensities, we consider cell intensity, which is defined as the number of points in a grid cell. To compute the similarity between two slices, T_i and T_j, we first align them together using ICP registration [BM92] by restricting to translational transformation in Y direction and then create grids for T_i and T_j, denoted by $G(T_i)$ and

$G(T_j)$, respectively. The grid resolution is set to 0.05 meters in X and Y directions. Marginal probability distributions of cell intensities are computed in terms of the ratio of number of cells with a particular intensity to the total number of grid cells in $G(T)$. For joint distributions, we use the intensity values of the corresponding cell pairs, for eg. $G(T_i)[m,n]$ and $G(T_j)[m,n]$. Intensity values ranging from 0 to the largest of $G(T_i)$ and $G(T_j)$ are considered for MI calculation.

In Equation 3, floor coverage, $Coverage(T)$ is defined as the ratio of scanned area to the total area of the floor. This ratio is computed based on the number of occupied grid cells (cells with at least one point) and the total number of grid cells. Coverage terms compensates for possible effects on the confident floor detection due to the occlusions caused by the same object as shown in Figure 6. If the floor coverage by the LiDAR points is not accounted, it is highly likely that an occluded floor, with its unoccluded part showing higher similarity than a fully scanned floor with relatively lower similarity, will be falsely detected as the confident floor, thereby affecting the whole model enhancement process. Both the terms, $Sim(T)$ and $Coverage(T)$ in Equation 3 are normalized to their respective largest values. After computing the confidence of all the horizontal floors, we identify the floor with the highest confidence value as the most confident tile, denoted by H_{conf}.

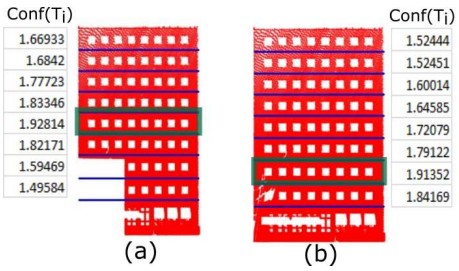

Figure 6: *Effect of uniform occlusions in confident floor detection. Confident floor (highlighted using green rectangle) detected on (a) an occluded façade and (b) an unoccluded façade. The occlusion has been manually introduced to analyze the performance of confident floor detection. Confident values identified by our algorithm is reported along with the façades. In the presence of an occlusion, the coverage term plays a significant role in finding the most robust floor.*

One of the major difficulties in dealing with occlusions in our context is the accuracy to which we can distinguish occlusions from true holes that represent windows or doors. There exist algorithms [DR12] which focus on consolidating LiDAR scans using occlusion filling algorithms borrowed from image domain, such as *inpainting* techniques. However, in our context, inpainting techniques may lead to undesirable outputs, especially by filling true holes and subsequently affecting the accuracy of the template fitting procedure (Section 4.3). We therefore employ a simple heuristic to classify the occluded floors, which works well in practice. In our framework, all the floors with its size less than 70% of the size of H_{conf} are considered to be either occluded or sparse. All such floors are replaced with the H_{conf} by simple translations in y direction.

We construct a two dimensional matrix to represent the split façade, denoted by $\mathcal{FG}$ (abbreviates façade grid). $\mathcal{FG}$ is a $(m+1)\times(n+1)$ matrix with entries g_{ij}, where m and n are the number of horizontal and vertical splitting lines. Each row of $\mathcal{FG}$ stores the boundary clusters in the respective floors along with the indices to access them. Each element in $\mathcal{FG}$ is a three tuple, i.e $g_{ij} =< p,q,C_{pq} >$, where p takes values y_{min} and the locations of horizontal splitting lines, $\{y_{L_1}, y_{L_2}, ..y_{L_m}\}$ and q takes values x_{min} and the locations of vertical splitting lines, i.e $\{x_{L_1}, x_{L_2}..,x_{L_n}\}$. The third element C_{pq} is the set of boundary points enclosed in the region defined by $[p,q]\times[p^{'},q^{'}]$, where $p^{'}$ and $q^{'}$ are the splitting locations of $g_{i+1,j+1}$. On reaching $i=(m+1)$ and $j=(n+1)$, $p^{'}$ and $q^{'}$ takes the values y_{max} and x_{max}, respectively. Recall that $[x_{min},y_{min}]\times[x_{max},y_{max}]$ denotes the bounding rectangle of the 2D projected vertices of f corresponding to the point set s.

To deal with diverse patterns of ground or top floors [MZWVG07], the user has the provision to specify the range of values for p via flag variables. For example, to exclude ground floor from further computations, i.e. if the $flag_{ground}$ is set to 0, the algorithm ignores the ground floor (y_{min}) while constructing $\mathcal{FG}$ and the boundary clusters of the ground floor are stored in another matrix with a similar struture. In a subsequent iteration, the stored matrix consisting of the ground floor are then modeled separately, in which it undergoes vertical splitting and all other subsequent operations. Diversely built top floors are handled in a similar manner using $flag_{top}$ variable.

4.3. Template Fitting

LiDAR scans are often too coarse in practice, contaminated with noises and occlusions and apparently, inferring accurate geometry of 3D façades from such point clouds becomes almost infeasible. A reasonable alternative would be to employ user defined templates representing façade entities for the modeling. Recently, Nan et al. [NJGW15] used 3D templates to enhance the coarse model with the help of images. The method produces realistic and detailed buildings. However, creating 3D templates can be time consuming and expensive as it demands an advanced level of experience in 3D modeling. With the popularity of web based 3D maps, urban modeling, is actively researched and followed by a wider audience with varied backgrounds, ranging from computer graphics through photogrammetry and remote sensing to urban planning. Aimed at users with basic knowledge on polygon creation, we propose an alternative to 3D templates in *coded planar meshes (CPM)* (See Figure 7 for examples). A coded planar mesh is an efficient way of encoding 3D architectural information into 2D by means of face codes. Dimensions of the planar meshes including the depths of the 3D façade entities can be directly estimated from the point clouds. Our template fitting module takes planar meshes representing the façade elements as inputs, find their locations on the coarse model and replicate them to their respective locations. The potential template locations are defined by the entries of the matrix, $\mathcal{FG}$.

4.3.1. Coded planar mesh (CPM)

In computer graphics, polygonal meshes are defined as a collection of vertices, edges and faces used to approximate complex geometric objects. Contrary to this conventional use, we investigate on an

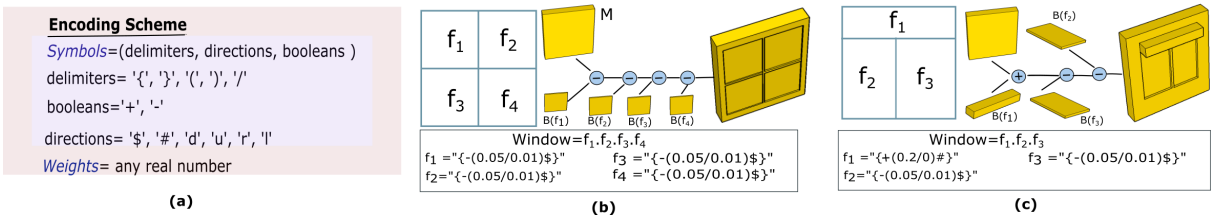

Figure 7: *Examples demonstrating the encoding scheme used in our framework. (a). Symbols and weights used in our encoding scheme, (b)-(c) CPMs, face codes and the conceptual CSG tree leading to the respective 3D structures. The terminals of each CSG tree are the boxes induced by the CPM faces and the coarse model.*

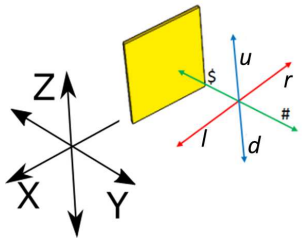

Figure 8: *Representations of directions used in our encoding scheme. The directions are illustrated with respect to a coarse model face, where the green axis lies along the face normal.*

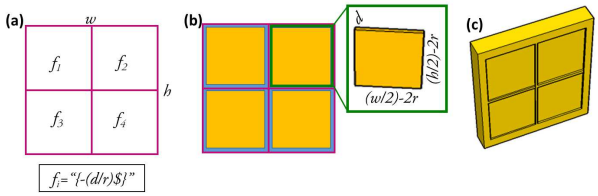

Figure 9: *Role of offset distance in the creation of sculpting box. (a) Template along with the face codes (given right below the template) (b) Template with boxes embedded in the respective faces, and (c) Window modeled using values $d = 0.05$ and $r = 0.01$ in the face code. In Figure 9 (b) blue border areas represent the offset region and a box corresponding to a face is zoomed in. Resizing of height and width values using the offset distance, r is mainly intended to model window frames (Figure 9(c)) or grills.*

interesting possibility of polygonal meshes to encode orthogonal 3D archetypal elements using the concept of coded planar meshes. Basically, CPM is a planar polygonal mesh with face codes $\mathfrak{C}$. Face codes for CPM are built on symbols and weights. Symbols consists of delimiters, booleans and directions as illustrated in Figure 7(a). While delimiters separate different codes and encapsulates the weights, booleans indicate the sculpting operation to be applied. We use boolean difference ('-') and union ('+') in our coding scheme. Directional symbols (shown in Figure 8) represent the six directions defined by the three orthogonal axes, where one of the axes is parallel to the normal of face F and the other two axes are defined accordingly. The symbols $ and # indicate the two opposite directions along the face normal axis. Since the models are erected on XY-plane in the pre-processing stage, symbols *u* (up) and *d* (down) specify the direction of sculpting along the original Z-axis. The symbols *l* (left) and *r* (right) are pointed along the third axis orthogonal to face normal axis and Z-axis.

Basically, the algorithm constructs boxes from the face codes and these boxes are later used as the inputs for sculpting. Therefore, besides various symbols, face codes also consist of weights to create the sculpting boxes. There are two types of weight values separated by symbol /. The value preceding / specifies the actual depth of the box in the direction pointed by the succeeding symbol and the one after / indicates an *offset* distance from the border of the face. The *offset* distances are introduced to recover window frames (shown in Figure 7(b))or balustrades (Figure 10) of balconies. If the *offset* is 0, then the two dimensions of the box excluding the depth would be the dimensions of the corresponding face (refer to Figure 9 (a)). Here, the face could be an original face from CPM or the new faces created by the predecessor face codes (see Figure 10). An illustration of the basic idea is presented in Figures 7 (b)-7 (c), which show different types of CPMs, corresponding conceptual CSG tree leading to the 3D structures.

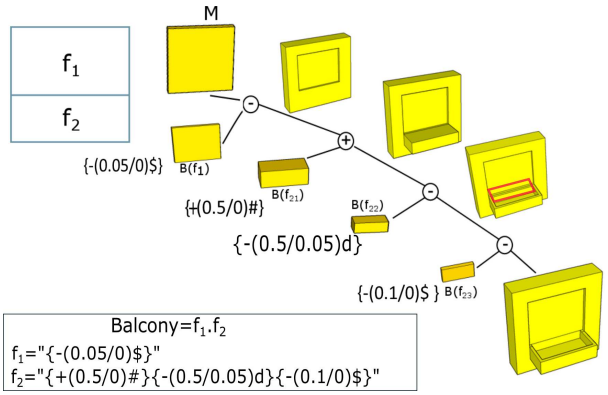

Figure 10: *A step wise illustration of sculpting. Red colored rectangle encapsulates the protrusion generated after second sculpting, which is subsequently removed in the next step.*

To bring more clarity into the encoding scheme, we elaborate on various semantic interpretations of the face codes using an example (Figure 10). Consider the face code of f_2 given in Figure 10.

$$f_2 = \{+(0.5/0)\#\}\{-(0.5/0.05)d\}\{-(0.1/0)\$\}$$

It consists of three codes, each separated by curly braces. The first code creates a box $B(f_{21})$ with depth 0.5 meters from the coarse model face, keeping the other two dimensions equal to that of f_2 (as the offset distance is 0). The box is then attached to the coarse model in the sculpting stage (Section 4.4), which give rise to five new ***sculptable faces*** with respect to f_2. A ***sculptable face*** is a newly generated face on the coarse model as a result of an immediate preceding sculpting operation and represents a potential face for future sculpting. Possibly many new faces may be generated on the model M due to a sculpt operation, however all of which are not considered to be sculptable in our context. Only the faces which are completely defined by newly introduced vertices in a particular iteration are referred to as ***sculptable faces***. Such faces can be identified by keeping track of the face and vertex lists of the model M.

The second code instructs the algorithm to create another box, $B(f_{22})$ having a depth of 0.5 meters in vertical direction along the Z-axis (as specified by the d), with respect to the sculptable faces induced by $B(f_{21})$. As there are five new sculptable faces, we combine *direction* and *type* of current and previous boolean operations to identify the appropriate sculptable face. First, the algorithm searches for the sculptable faces whose normals are parallel to the Z axis. We get two such sculptable faces in the downward direction, one at the top and one at the bottom. Since the current boolean type is a subtract and the previous boolean type was a union, we check for the top most sculptable face by determining the spatial locations of the faces with respect to each other, i.e. the sculptable face having vertices with higher Z coordinates is considered to be the top face. This is quite reasonable as the model is already aligned with the three orthogonal axes in the Cartesian coordinate system. The rationale behind selecting the top face for sculpting is straightforward. The previous boolean operation, which was a union, would have attached a box to the base model, thereby creating sculptable faces on the attached box. Hence, to reflect the sculpting on the attached box which is a part of the coarse model, a subsequent subtract operation in downward direction should work on top sculptable face rather than the bottom face. However, a union operation in downward direction for such a type configuration works with the bottom sculptable face. A similar procedure is followed in other directions as well.

The other two dimensions of $B(f_{22})$ are resized according to the offset value, 0.05. Note that such a resizing will introduce a width of 0.05m on the balustrade, which is essential for a realistic modeling (refer to the balcony of Figure 10). This offset distance of 0.05, however leads to a small protrusion on the interior face of balcony after the second sculpting as highlighted in the rectangular (red) region of Figure 10. To remove this protrusion, we rely on the third code, that use a box of depth 0.1 m (0.05 m due to the sculpting by f_1+0.05 m due to the offset value in second sculpting of $B(f_{22})$) along the face normal axis to realize the task.

4.3.2. CPM assignment

A key step in our framework is to correctly assign the CPMs to their corresponding boundary clusters present in the façade grid, FG. Each cell in the grid g_{ij} holds a set of boundary points (as shown in Figure 5(f)), denoted as C_{ij} capable of defining a potential façade element and we are equipped with a set of 2D templates, $\mathcal{T} = \{T_1, T_2.., T_k\}$, where $k \geq 1$. Template assignment problem deals with correctly assigning a C_{ij} with its template, T_m. To maximize the correct template assignment, we define a fitting score, F_{score} between each cluster C_{ij} and template T_m, where F_{score} depends on the box alignment and edge fitting terms as given in Equation 5.

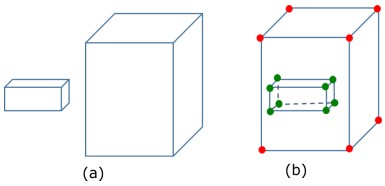

Figure 11: *Illustration of sculptable faces. (a) The model and a sculpting box before (a) and after (b) the union operation. The operation give rise to 5* **sculptable faces**, *each consists of only the newly introduced vertices (green) of the model $\mathcal{M}$.*

$$F_{score}(C_{ij}, T_m) = \lambda AE(B(C_{ij}), B(T_m)) - (1-\lambda)EF(T_m, C_{ij}) \quad (5)$$

Equation 5, the weighting parameter $\lambda \in [0,1]$ balances the alignment and edge fitting terms. Essentially, the fitting aims at lowering the box alignment error, while maximizing the edge-to-point fitting.

The **alignment error** term , **AE** measures the deviation between the bounding boxes (denoted by $B(.)$) of the cluster and the template and is determined based on the symmetric difference between them (Equation 7).

$$AE(B(C_{ij}), B(T_m)) = \frac{Area(B(T_m) \setminus B(C_{ij})) \cup Area(B(C_{ij}) \setminus B(T_m))}{Area(B(T_m))} \quad (6)$$

We consider only the bounding box of the CPM faces representing holes for the alignment as well as edge fitting. Such faces are characterized by their face code, i.e. it always start with a code having directional symbol '$' with a negative sign for the boolean operation.

The **edge fitting** term, **EF** estimates how well the edges e of T_m fits to the boundary points p of C_{ij}, i.e.

$$EF(C_{ij}, T_m) = \sum_{e,p | d(e,p) < \mu} 1 - \frac{d(e,p)}{\mu} \quad (7)$$

where $d(.)$ measures the Euclidean distance between an edge e of T_m and a boundary point p of C_{ij}. We consider only the points within a distance μ (default value of 0.1 m) to the edges e. Note that, while **AE** term favors the correct positional alignment of T_m over C_{ij}, **EF** term encourages the edges of CPM to lie close to the potential edges in C_{ij}.

User creates the 2D templates such that one of its corner vertices coincides with the origin and the edges are aligned with the two principal axes of XY-plane (more on template creation is presented in Section 5). For computing the F_{score}, the fitting algorithm first translate T_m from the origin to the grid cell that holds the cluster C_{ij}. Translation in X and Y directions are performed so as to align the centroid of $B(T_m)$ to the centroid of $B(C_{ij})$ (denoted by C_c). It then

search for the lowest fitting score by placing T_m at various locations (both in x and y directions) in a box of size 0.1 meters, centered at C_c. The location with the lowest F_{score} is recorded as the potential location of T_m for C_{ij}. For each template T_m, its lowest fitting score and the translational parameters, T_{xy} are recorded. Finally, from $\mathcal{T}$, the template that returns the lowest F_{score} is assigned to the cluster, C_{ij}.

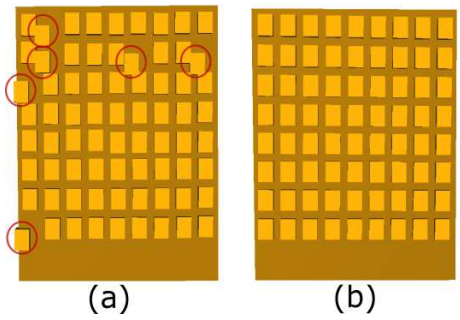

Figure 12: *Effect of regularization. Templates before (a) and after (b) the regularization. The templates violating the regularity are encircled.*

Layout regularization. A few 2D templates assigned to the clusters may still violate the rectilinear arrangement of the façade as shown in Figure 12(a). This is mainly due to the false positive boundary points (from small occlusions or sparse sampling) that affect the template assignment in the respective grid cell. A recent study on layout regularization [JNY*16] reveals the effect of template alignment, spacing and size constraints on the final layouts. Inspired from their study, we address the regularization problem by enforcing the alignment regularity and the optimal inter-element spacing on the façade. Specifically, we minimize the row and column wise deviations among the template bounding boxes subjected to equal spacing constraints, thus leading to the following minimization problem.

$$\text{minimize} \quad \Sigma_i^{cols}\Sigma_j^{rows}\|x_i^* - x_{ij}\|^2 + \Sigma_j^{rows}\Sigma_i^{cols}\|y_j^* - y_{ji}\|^2$$

subject to

$$\begin{aligned} x_{i+1}^* - \tfrac{w_{i+1}}{2} - (x_i^* + \tfrac{w_i}{2}) &= x_i^* - \tfrac{w_i}{2} - (x_{i-1}^* + \tfrac{w_{i-1}}{2}), \\ \forall i, 1 < i \leq cols - 1, & \\ y_{j+1}^* - \tfrac{h_{j+1}}{2} - (y_j^* + \tfrac{h_j}{2}) &= y_j^* - \tfrac{h_j}{2} - (y_{j-1}^* + \tfrac{h_{j-1}}{2}), \\ \forall j, 1 < j \leq rows - 1, & \\ x_{min} \leq x_i^* \leq x_{max}, \quad y_{min} \leq y_j^* &\leq y_{max} \end{aligned} \tag{8}$$

where *cols* and *rows* are the number of columns and rows of the façade grid and each bounding box is denoted by its center, (x_{ij}, y_{ij}). In Equation 8, h_j denotes the height of the bounding box in $\mathcal{FG}[j,1]$ and w_i denotes the width of the bounding box in $\mathcal{FG}[1,i]$. The proposed linear least square minimization model refines the column-wise and row-wise lines, i.e. x_i^* and y_j^* fitted to the bounding box centers, and finally generates a 2D grid with constant inter-node spacing as shown in Figure 12 (b). In each iteration, bounding box locations are updated with its respective grid node locations.

Algorithm 1: Sculpting($\mathcal{M}$, $\mathcal{L}$)

Input: Coarse model, $\mathcal{M}$ and the list of CPMs, $\mathcal{L} = \{l_{ij}\}$
Output: Enhanced 3D model, $\mathcal{M}$

```
for l_ij ∈ L do
    for each face f of l_ij do
        Extract the face code, fc of f ;
        Determine the curly brace pair number, #cb;
        prev_token = ϕ;
        while fc is not empty do
            Parse fc to retrieve the code, e encapsulated by
            the first pair of parentheses, '{ }';
            Create the box corresponding to e, taking into
            account the prev_token;
            if boolean token of e is '-' then
                M = M − b_i;
                if #cb>1 then
                    Create the list of sculptable faces, SF;
                    prev_token ='-';
                end
            end
            else if boolean token of e is '+' then
                M = M + b_i;
                if #cb>1 then
                    Create the list of sculptable faces, SF;
                    prev_token ='+';
                end
            end
            Delete e from f_i;
            Decrement #cb;
            Empty the list, SF;
        end
        prev_token = ϕ;
    end
end
return Enhanced model, M
```

4.4. Boolean Sculpting

In sculpting stage, first we construct 3D boxes from the replicated CPMs. Then, we employ boolean operations on the coarse model and the CPM boxes to actually model the details on the façade. The list of all CPMs for each façade, along with its face codes are obtained from template fitting stage. We extract the faces of each CPM and parse the face codes to identify the boolean type, depth, offset and the direction of the corresponding sculpting box. Using the transformation parameters, $\mathcal{P}_i = \{o_i, v_{i1}, v_{i2}\}$, where $1 \leq i \leq segmentcount$, determined in the first stage (Section 4.1), we convert each 2D vertex of CPMs, q to the corresponding 3D point, p as follows: $p = o_i + q_x \cdot v_{i1} + q_y \cdot v_{i2}$. Once all the 3D vertices of a CPM face are obtained, then the algorithm determines the remaining four corner vertices of the sculpting box by replicating its vertices at the specified *depth* from the face, along the given direction. More specifically, we determine a plane, $P_{parallel}$ parallel to the ***sculptable face*** at a distance equal to the *depth* and then

project the CPM vertices onto $P_{parallel}$, thus creating all the eight corner vertices of the required box.

To apply boolean operations, both $\mathcal{M}$ and the CPM boxes are converted to 3D Nef_polyhedra, which are closure of half spaces under boolean operations [HKM07]. We prefer Nef_polyhedra because the problems due to floating-point arithmetic and degeneracies are taken care in Nef_polyhedral complexes. In general, for each CPM face, the sculpting algorithm cycles between the box creation and sculpting (refer to the lines 5-23 of Algorithm 1), thus enabling the detail synthesis in any of the six directions mentioned in Figure 8. By repeating the whole procedure (Sections 4.2-4.4) for each façade, we get a more detailed 3D model of the urban building.

5. Results

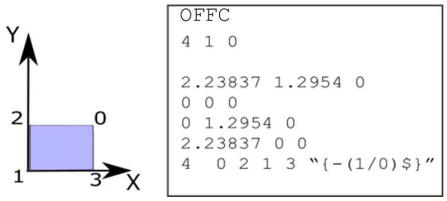

Figure 13: *An example of a coded planar mesh along with its file description. In the file description, following the OFFC keyword, the number of vertices, faces and edges are given, respectively. Vertex coordinates follow. The mesh is created in XY plane and hence the zero Z-coordinates. Vertex coordinates are followed by face description (last line) that consists of vertex indices in counter-clockwise direction. Face codes are added along with the face description.*

We implemented our façade enhancement algorithm (depicted in Figure 2) in C++ using the functions and geometric predicates available in CGAL library. We tested our framework on real world buildings having different styles and complexity, taken from the downtown areas of Lexington, Kentucky and Toronto. Due to the unavailability of LiDAR scans to demonstrate certain capabilities of the framework, we have also used synthetic scans generated by Poisson disk sampling from existing models in standard web libraries (for eg. thingiverse.com [tv]). All the experiments were performed on a machine having 3.6 GHz processor with 16 GB memory.

Template creation. For each building model, we create and use a set of coded planar meshes, each representing a repeating element of the building. If the dimensions of the façade elements are known a priori, these details can be used for the template creation. Otherwise, the user has to define the templates based on the information available from the point clouds. The dimensions of elements can be measured from the point cloud using 3D open source software such as Meshlab [mes]. The planar meshes are created in 3D software such as sketch up (base version) and are later exported in an extended .OFF format called .OFFC. We embed the face codes for the templates along with the face description in .OFFC files. An example of a planar mesh along with its .OFFC file is shown in Figure 13. Compared to other interactive building modeling tools [NSZ*10] the user interaction in our framework is confined to template creation and pre-processing. Once the user provides the registered coarse model and LiDAR scan along with a set of 2D templates designed for the building, the four stage enhancement pipeline generates the detailed 3D model of the building with no additional user assistance.

5.1. Splitting Comparison

We qualitatively compare the splitting procedure employed in our framework with two other related works, including adaptive aprtitioning [SHFH11] and a weighted point count based splitting [SB03] in Figure 14. The corresponding parameters for each algorithm were fine tuned so as to achieve the best results on the experimental data. Adaptive partitioning generates a hierarchial representation of 3D façades and performs well for façades with interlaced grids. However, the proposed splitting produced better results for the occluded and sparse façades with monotonous repetitions as shown in Figure 14.

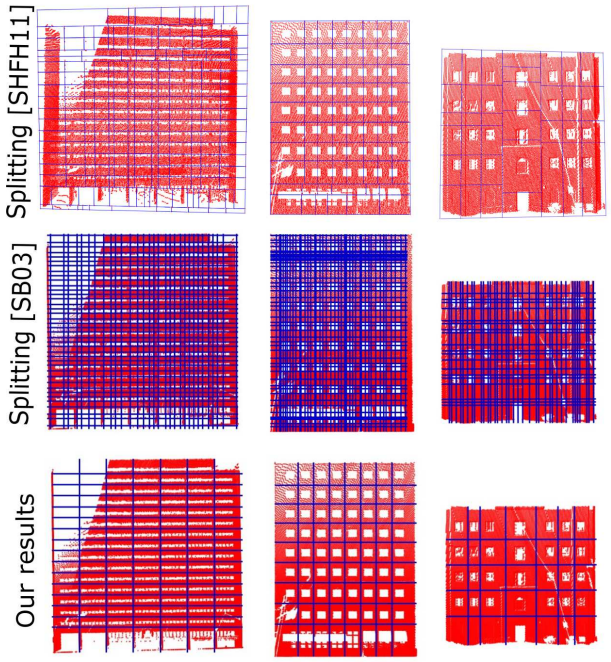

Figure 14: ***Qualitative comparison:*** *Even for occluded (column 1) and sparse (column 2) façades, our function generates a reasonable splitting, in which each tile holds a potential façade element. First row ([SHFH11]) features 3D splitting results whereas the other two methods perform splitting in 2D.*

While we define the weighted point counts on the boundary points determined by α-shapes, the function in [SB03] is defined over the entire façade points. We aim at including lines between elements whereas the objective of [SB03] is to extract the window edges from the splitting lines. To this end, they merge multiple, neighboring tiles having similar point densities to single tile regions. Compared to the function in [SB03], which subdivides the

façade into a raster of irregular, rectangular tiles, our method performs noticeably better while partitioning occluded (column 1, Figure 14) and sparse (column 2, Figure 14) façades into meaningful tiles.

Table 1: *Keeping in mind the reproducibility of the results, we report 2D template descriptions of a few models used in our experiment. Template dimensions and the weights in face codes are reported in meters. One alternative to the user-defined weights (representing the sculpting depth) would be to use standard depth dimensions in the architectural domain.*

Model	Templates(with face#)	Dimensions		Face codes
		length	width	
Fig. 15	T1	1.3	2.3	"{-(1/0)$}"
	T2	1.3	5.3	"{-(1/0)$}"
Fig. 16	T1	8.6	1.4	"{-(0.5/0)$}"
	T2	4.3	1.4	"{-(0.5/0)$}"
	T3	6.4	1.4	"{-(0.5/0)$}"
	T4	8.6	6.2	"{-(2/0)$}"
	T5	6.4	6.2	"{-(2/0)$}"
	T6	8.6	3.6	"{-(0.5/0)$}"
	T7	2.3	3.6	"{-(0.5/0)$}"
Fig. 17	T1->f1	5.3	4.0	"{-(0.4/0)$}"
	T1->f2	0.7	4.0	"{+(0.4/0)$}"
	T2	0.4	1.2	"{-(0.4/0)$}"

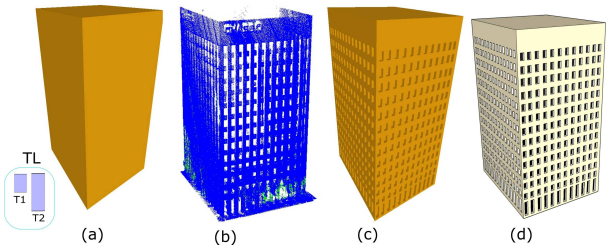

Figure 15: ***Symmetric façade****. An example of high rise building with many number of façade elements exhibiting extensive symmetry. We have shown the coarse model taken from Google earth (a), the corresponding LiDAR scan (b), enhanced model generated by the proposed framework (c) and a manually textured model (d). The proposed framework modeled all the façade details located on the frontal and two lateral walls of the building. The model use only two templates (given in the template library (TL), bottom left) to model 520 façade elements.*

5.2. Enhanced models

Symmetry. Figures 15-20 show the coarse models geometrically enhanced using our method. In each figure, we have provided the coarse model, LiDAR scan, enhanced model and a manually textured model and the set of templates used for enhancement. An example of a building with a large number of façade elements is shown in Figure 15. Considering the frontal and two lateral walls of the building, our method succeeds in detecting and enhancing all the façade entities (including windows and doors). It is to be noted that, for façades exhibiting extensive symmetries (Figures 15 & 20(a)), the framework demands only a minimal user involvement to construct the CPM from the point clouds.

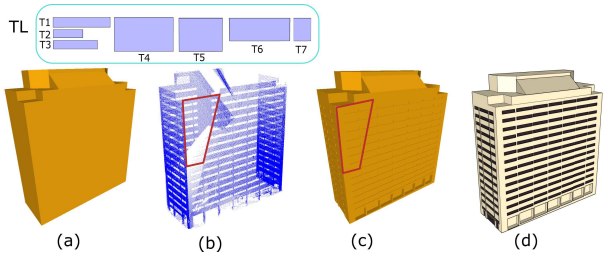

Figure 16: ***Modeling from occluded scans****. Note that, occluded region, marked using red boundary in Figure 16 (b), has been successfully reconstructed (Figure 16 (c)) by our framework. Dimensions and face codes of the templates(given in TL, top left) for this model have been reported in Table 1.*

Occlusions. The proposed framework performs well under considerable occlusions (approximately 25% portion of the frontal façade points are occluded in Figure 16). In Figure 16, the method replicates the details from the confident tile and accurately reconstruct the details over the occluded regions. Figure 20(b) features a building with heterogeneous elements on the same wall (frontal wall). Our method accurately reconstructs all the elements at their respective locations. However, since the bottom part of the lateral wall is not scanned well (data is completely missing, see the LiDAR scan in Figure 20 (b)), the splitting function is unable to identify the floors and subsequently, floor replication is not performed leading to a non-enhanced bottom part on the side wall.

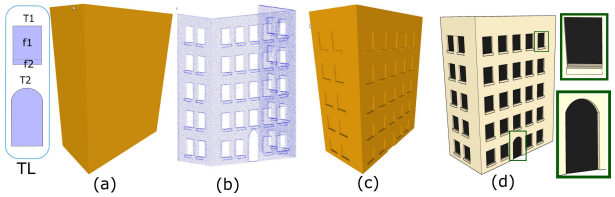

Figure 17: ***A model with protruding structures and arch door****. This example clearly indicates the potentials of coded planar meshes in modeling non-rectangular structures such as arch doors (shown in inset).*

Non-rectilinearity. Coded planar meshes are also capable of modeling non-rectilinear structures such as arch doors as shown in Figure 17. A realistic arch door has been modeled on the building with the help of a fine-grained polygonal approximation to the arch (T2 in template library of Figure 17). The proposed framework not only uses the cuboids, but also the polyhedra (for eg. induced from the polygonal approximation of the arch) for sculpting. The example in Figure 17 also shows the modeling of protruding structures such as window ornaments on the façade.

Heterogeneity. Figure 18 shows another building with multiple façade elements (2 types of windows and a balcony to the interior). In the inset of Figure 18, one can observe the extended parapet on the side walls of the balcony. This is a consequence of the *offset* distance used in the face weights of the coding scheme. Please note that, without such an *offset* distance, the parapet would not have

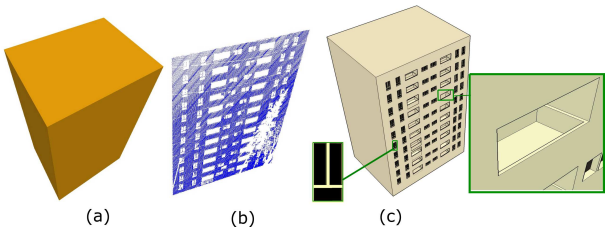

Figure 18: ***A model with balcony****. A façade with globally rectilinear grid pattern consisting of heterogeneous elements. Due to the uavailability of GE model, we have used a manually created box for enhancement. Reconstruction of the occluded regions (Figure 18 (b)) with the correct window type can be seen Figure 18 (c).*

been modeled, instead only a debossing on the wall to represent the entire balcony would be generated. The proposed framework can also enhance façades with a hetrogeneous pattern of elements (Figure 20(c)), via appropriate $flag$ variables as discussed in Section 4.2.2.

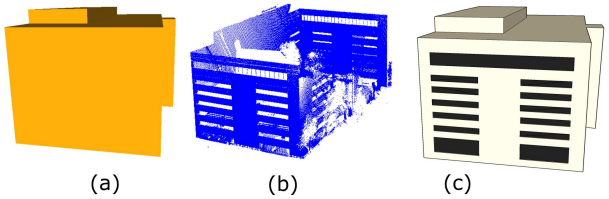

Figure 19: ***An example of a façade with diverse floor pattern.*** *The top floor is entirely different from the rest of the floors. The framework successfully reconstructs such façades by setting flag variables ($flag_{top}$ in this case).*

To summarize, the framework along with the coded planar meshes is capable of modeling various types of façade elements such as plane windows (Figure 15& 20(b)), ribbon windows (Figure 16) and window subdivisions and frames (Figures 18 & 20(a)-(b)), rectangular and arch doors (Figure 17), protruding artifacts (Figure 17) and balconies (Figure 18) and parking floors (Figure 20(d)), all of which are ubiquitous in urban façades.

Detailed urban synthesis. To further illustrate the potentials of the proposed framework in large-scale and detailed urban scene generation, we constructed a portion of Lexington downtown, Kentucky (Figure 21) by adding enhanced models to the Google earth (GE) scene. From Figure 21, it is apparent that the scene consisting of detailed models considerably augments the visual quality and provides more information for virtual navigation.

5.3. Performance Statistics

Fitting accuracy. The proposed fitting is predominantly data driven. To assess the quality of fitting, we conducted an experiment on an example model by taking a few similar 2D templates (created by resizing the true template), as illustrated in Figure 22. There may

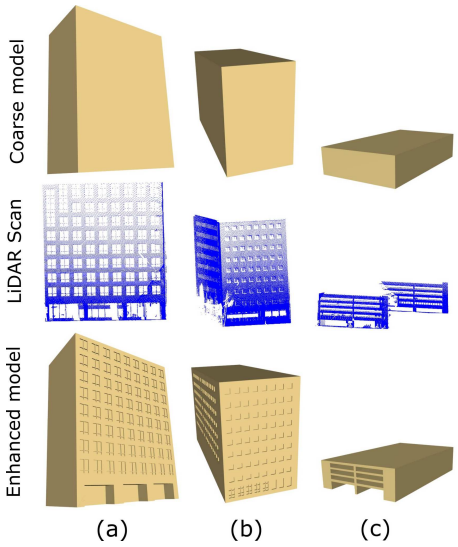

Figure 20: *Result gallery of enhanced building models showcasing different types of façade patterns.*

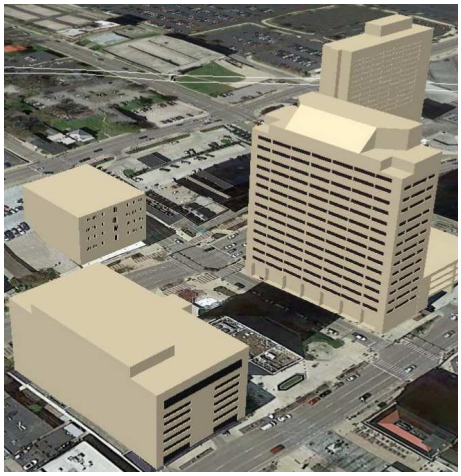

Figure 21: *An example of an urban scene with detailed buildings. A portion of Lexington, Kentucky, United States.*

Table 2: *Performance statistics. #FE and #S are abbreviations for number of detected façade elements and segment count.*

Model	# points	# FE	# S	Time (Minutes)				Parameters	
				Map	Split	Fit	Sculpt	h	w
Fig. 15	4736K	520	3	0.874	69.05	1.59	3.068	2.4	0.2
Fig. 16	410K	189	3	0.169	7.395	0.93	1.115	2	0.4
Fig. 17	57.9K	45	3	0.025	0.922	0.953	0.525	3.5	1

be dimensional variations among the templates designed by different users and it is important to demonstrate that the fitting algorithm always favors the most accurate template. Fitting was performed for various scans with synthetic noises. We introduced Gaussian noise in the scan with varying standard deviations as shown in Figure 22. We utilized the metric, average root mean square error (RMSE)

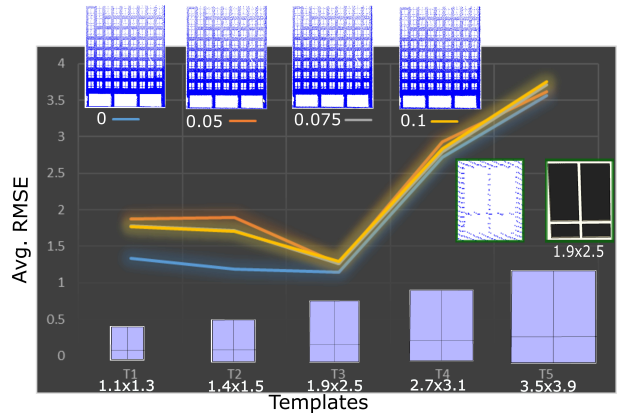

Figure 22: *Fitting quality in the presence of noise. Average RMSE for the true template and its variants for a scan with varying noise levels. The legends show scans having Gaussian noise amounting to 50% of the original scan size and is varied by different standard deviations. The dimensions (width × height) of each template is shown right below them. An accurately modeled window (extreme right) along with a representative point cloud is also shown for clarity.*

between the templates and the ground truth boundary clusters (obtained from the original LiDAR scan) to quantify the geometric accuracy of the template fitting. Let S_i and C_i respectively denote a boundary cluster and its assigned CPM. Then, RMSE of template fitting on a façade with n CPMs is defined as follows.

$$Avg.RMSE = \sqrt{\frac{\sum_{i=1}^{n} \frac{\sum_{\forall s \in S_i} \|C_i - s\|^2}{|S_i|}}{n}} \quad (9)$$

Template fitting on the original scan always incurred the lowest RMSE. As the standard deviation of the noise model increases, more points tend to spill out in the hole regions, and consequently, identified as boundary points. This, in turn increases the overall fitting error. Among various templates, the true template achieved the lowest RMSE. As can be seen from Figure 22, the fitting accuracy to the point cloud with noise 0.05 is low for a few scaled templates. However, it achieves a desirable fitting accuracy for the true template. This is more clear from the RMSE values of various point clouds for the true template, as given below: 1.175 (original cloud), 1.257 (noise with $\sigma = 0.05$), 1.263(noise with $\sigma = 0.075$) and 1.297(noise with $\sigma = 0.1$). In general, due to the outlier removal strategy presented in Section 4.1, the proposed farmework is tolerant to noises induced due to the reflections from window curtains and interior walls. A reconstructed model using the original scan and true template is shown in Figure 20(a).

Timing. Table 2 reports the CPU times (in minutes) incurred due to different stages of the pipeline for various models. Splitting stage is computationally the most expensive stage, especially because it consists of alpha shape extraction, line sweeping and confident tile detection, which in turn depends on a series tasks such as floor to floor ICP registration, voxelization and MI computation, all of which are costly for large-sized scans. Relatively, other stages, i.e. mapping, fitting and box creation incurs nominal computational time and hence the overall time is predominantly determined by the façade splitting.

Parameters. The major parameters (Table 2) include the value of α and the height (h) and width(w) parameters in splitting for which we use the values from the default range settings in all the examples. The parameter, α is inversely related to the point density of the building scan. A low-density scan requires a larger value of α in order for the boundary points to be connected to each other. The density is highly variable along the building height and also depends on the target distance from the scanner. While the top floors have low point density, floors towards the ground are densely scanned. Similarly, high dense LiDAR scan can be generated for a target which is close to the scanner. The value of α should account for such variabilities. Nowadays, LiDAR systems are capable of generating scans with point densities ranging from 10 to 1000 pts/m^2. According to a standard point density to sample spacing conversion [Ols13], sample spacings of 0.03 to 0.3 meters are achievable for mobile scanners and therefore, possible values for α would lie in this range. In our experiment, values between 0.05-0.15 meters found to produce satisfactory results.

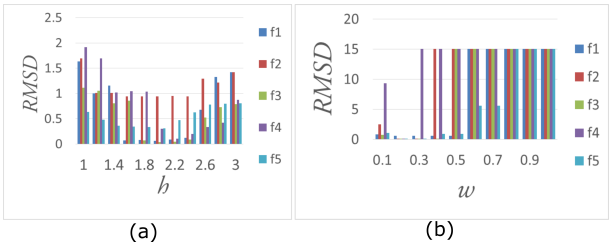

Figure 23: *Effect of the parameters h (a) and w (b) on the accuracy of splitting. Apparently, h values in the range [1.4-2.4] meters found to split the façades with low RMSD from the manually created reference lines. For some façades, $w \geq 0.3$ did not generate any splitting lines. We have used RMSD value of 15 to indicate such instances.*

To arrive at the range of values provided for the parameters h and w, we conducted a splitting experiment on different façades (Figures 15-20) taken from the buildings of Lexington, Kentucky. For each façade, we manually included the most plausible horizontal and vertical splitting lines, referred to as *reference* lines. Then, we generated the *splitting* lines for various values of $h \in [1m - 3m]$ and $w \in [0.1m - 1m]$. To quantify convergence between splitting and reference lines, we used the root mean square deviation (RMSD). In vertical direction, RMSD is computed as follows(Equation 10).

$$RMSD = \sqrt{\frac{\Sigma_i \| y_{r_i} - y_{s_i} \|^2}{\#\text{ splitting lines}}} \quad (10)$$

Each splitting line (y_s) considers its spatially closest reference line (y_r) for computing the deviation. Figure 23(a) captures the effect of the parameter h for horizontal splitting. Based on our experiment (Figure 23) on the available data, we observe that values lying in 1.4-2.4 meters would possibly leads to a near optimal splitting. Since the floor height of most buildings lie in 3m-5.5m [bh, MZWVG07], the window height has to be lower than

the floor height and hence, the suggested range of values seem reasonable and conform to the standards. The parameter, w should be more than the window frame width and less than the inter-window spacing in horizontal direction. Based on our experiment on various façades (Figure 23(b)), we find that a value in [0.2-0.4] m is a reasonable choice for vertical splitting. It is also possible to measure these values during the template creation step.

5.4. Discussion

3D templates Vs CPM. As opposed to our LiDAR based sculpting pipeline, most of the previous works [NJGW15, MZWVG07] rely on images to locate the template locations on the coarse model. Both, the works [NJGW15, MZWVG07] employ pre-built 3D template libraries for the enhancement. On the contrary, the proposed method works on 2D templates, which greatly reduces the burden on the user and is a cheaper alternative. Professional 3D modelers are required to design the 3D template libraries, whereas any user with basic knowledge on 3D software (for measuring the façade dimensions) can create 2D templates, which are essentially planar graphs with codes. Moreover, the encoding scheme, mainly consisting of directions, booleans operations and the sculpting depths (weights) is fairly simple and self-explanatory. We believe that encoding the basic units of repeating façade patterns is relatively easy compared to a much more involved design of procedural rules [MZWVG07] for the entire façade (comprises its floors, tiles and window structures).

User interaction. Like many other techniques [NJGW15, MZWVG07], we consider the template creation as a separate task, i.e.it does not affects the algorithmic running time. Hence, the real user interaction in our framework happens only in the pre-processing stage, that too for the registration refinement. User interaction in the pre-processing stage is quite common in urban modeling pipeline, for eg. the method in [NJGW15] requires the user to interactively create a coarse model from the point cloud obtained through SfM. LiDAR based smartboxes technique [NSZ*10] demands a more involvement from the user during the modeling process. The user needs to define the region of interest (ROI) in the screen space for smartbox creation and drag-and-drop the smartboxes to create its multiple instances. Heavily corrupted data demands such user interaction while reconstructing. In procedural modeling technique [MZWVG07], the user interactively work on the automatically modeled façade to introduce the depths to the façade elements to the rest of the region.

Assembling Vs Sculpting. Modeling in our framework relies on boolean sculpting operations as opposed to the template assembling found in [NSZ*10, NJGW15]. The sculpting approach has a unique advantage. By coupling sculpting depths with façade structure, the proposed framework is capable of reconstructing even the entire floors of the parkades as shown in Figure 20(c). This is an interesting and useful outcome of our framework, not seen in any of the previous methods. Smartboxes [NSZ*10] are designed to fit the point data and consequently, the reconstruction of the interior parts of the parking floors, which are mostly unscanned by the LiDAR system is difficult or almost infeasible. Approaches such as template assembly [NJGW15] or procedural modeling [MZWVG07], focus on assembling 3D templates over the coarse model. No sculpting is applied to the coarse model and consequently, reconstruction of the entire floor including its interior region seems to be impractical.

Limitations. There are a few limitations reflecting on the current implementation of the framework. First, our splitting function is mainly designed for façades with globally rectilinear grid structure. It is not powerful enough to subdivide interleaved façades [SHFH11]. One way to address interleaved façades is to introduce the horizontal floors grouping before vertical splitting and perform vertical splitting on each floor groups as described in [SHFH11]. This will considerably make the pipeline more generic, of course at the expense of increased complexity in the splitting stage. In our current implementation, we use $flag$ variables to handle buildings with diversely designed ground and top floors. Alternately, one can try to integrate the splitting scheme proposed in [SHFH11] to the pipeline. Second, the framework relies on the assumption, "holes in the scan represent façade elements". While this is true for most of the urban buildings with window glasses or doors and balcony doors at a depth from the wall (the points at certain depth from the main wall are removed in the first stage, thereby creating holes in the 2D points), there are other features such as corbels that supports main floor cross beams that go undetected in our framework. Third, the whole framework is now designed under the Manhattan-world assumptions, especially for planar façades, leaving out façades assuming conic or cylindrical shapes. This considerably simplifies the façade element detection in two dimensions. We believe that, this assumption can still be considered as a minor limitation because a vast majority of the man-made buildings still conform to Manhattan structures.

6. Conclusions

We presented a new framework for enhancing symmetric façades using terrestrial laser scan and 2D templates. The introduction of coded planar meshes (2D templates and the associated encoding scheme) as representations of 3D architectural constructs found on urban façades has greater implications for the future of building modeling arena. It not only relieves the user of tedious task of 3D modeling, but also allows us to operate and design algorithms in 2D domain. We have provided several examples of GE models enhanced using our framework and all of these detailed models reiterate the practical potentials of the framework. We observe that fine detailed façades are more informative, conveys the architectural styles and above all, enhances the realism in the virtual tours being offered by most of the web-based application. Considering these facts, the proposed framework is quite relevant, especially in the current scenario where we see a proliferation of web-based 3D maps and related applications.

Acknowledgements

The authors thank Dr. Ruigang Yang for providing the liDAR scan of Lexington, Kentucky and Dr. Chao-Hui Shen for providing the source code of adaptive partitioning [SHFH11]. We thank the anonymous reviewers for their valuable comments and feedback for improving the paper.

References

[ASF*13] ARIKAN M., SCHWÄRZLER M., FLÖRY S., WIMMER M., MAIERHOFER S.: O-snap: Optimization-based snapping for modeling architecture. *ACM Trans. Graph. 32*, 1 (Feb. 2013), 6:1–6:15. 2

[bh] Council of tall buildings and urban habitat. last accessed: 25 July, 2016. URL: http://www.ctbuh.org/TallBuildings/HeightStatistics/. 14

[BH07] BECKER S., HAALA N.: Refinement of building facades by integrated processing of lidar and image data. In *ISPRS Workshop on Photogrammetric Image Analysis* (2007), pp. 1–6. 3

[BM92] BESL P. J., MCKAY H. D.: A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 2 (Feb 1992), 239–256. doi:10.1109/34.121791. 3, 6

[clo] Cloudcompare. last accessed on 22 July, 2016. URL: http://www.danielgm.net/cc/. 3

[DR12] DORIA D., RADKE R. J.: Filling large holes in lidar data by inpainting depth gradients. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (June 2012), pp. 65–72. 7

[DTM96] DEBEVEC P. E., TAYLOR C. J., MALIK J.: Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 11–20. 2

[EKS83] EDELSBRUNNER H., KIRKPATRICK D., SEIDEL R.: On the shape of a set of points in the plane. *IEEE Transactions on Information Theory 29*, 4 (Jul 1983), 551–559. 5

[FJZ05] FRUEH C., JAIN S., ZAKHOR A.: Data processing algorithms for generating textured 3d building facade meshes from laser scans and camera images. *International Journal of Computer Vision 61*, 2 (2005), 159–184. 3

[HKM07] HACHENBERGER P., KETTNER L., MEHLHORN K.: Boolean operations on 3d selective nef complexes: Data structure, algorithms, optimized implementation and experiments. *Computational Geometry 38*, 1-2 (2007), 64 – 99. Special Issue on {CGAL}. 11

[JNY*16] JIANG H., NAN L., YAN D. M., DONG W., ZHANG X., WONKA P.: Automatic constraint detection for 2d layout regularization. *IEEE Transactions on Visualization and Computer Graphics 22*, 8 (Aug 2016), 1933–1944. 10

[Kul59] KULLBACK S.: *Information Theory and Statistics*. Wiley, New York, 1959. 6

[KW11] KELLY T., WONKA P.: Interactive architectural modeling with procedural extrusions. *ACM Trans. Graph. 30*, 2 (Apr. 2011), 14:1–14:15. 2

[LGZ*13] LIN H., GAO J., ZHOU Y., LU G., YE M., ZHANG C., LIU L., YANG R.: Semantic decomposition and reconstruction of residential scenes from lidar data. *ACM Trans. Graph. 32*, 4 (July 2013), 66:1–66:10. 2

[LO15] LIMBERGER F. A., OLIVEIRA M. M.: Real-time detection of planar regions in unorganized point clouds. *Pattern Recognition 48* (2015). 4

[LWM14] LIPP M., WONKA P., MÜLLER P.: Pushpull++. *ACM Trans. Graph. 33*, 4 (July 2014), 130:1–130:9. 4

[LZS*11] LI Y., ZHENG Q., SHARF A., COHEN-OR D., CHEN B., MITRA N.: 2d-3d fusion for layer decomposition of urban facades. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (Nov 2011), pp. 882–889. doi:10.1109/ICCV.2011.6126329. 2

[mes] Meshlab. last accessed on 22 July, 2016. URL: http://meshlab.sourceforge.net/. 11

[MPWC13] MITRA N. J., PAULY M., WAND M., CEYLAN D.: Symmetry in 3d geometry: Extraction and applications. *Computer Graphics Forum 32*, 6 (2013), 1–23. 2, 6

[MWA*13] MUSIALSKI P., WONKA P., ALIAGA D. G., WIMMER M., VAN GOOL L., PURGATHOFER W.: A survey of urban reconstruction. *Computer Graphics Forum 32*, 6 (2013), 146–177. 2

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. *ACM Trans. Graph. 25*, 3 (July 2006), 614–623. 2

[MZWVG07] MÜLLER P., ZENG G., WONKA P., VAN GOOL L.: Image-based procedural modeling of facades. *ACM Trans. Graph. 26*, 3 (July 2007). 2, 5, 6, 7, 14, 15

[NJGW15] NAN L., JIANG C., GHANEM B., WONKA P.: Template assembly for detailed urban reconstruction. *Computer Graphics Forum 34*, 2 (2015), 217–228. 3, 7, 15

[NSZ*10] NAN L., SHARF A., ZHANG H., COHEN-OR D., CHEN B.: Smartboxes for interactive urban reconstruction. *ACM Trans. Graph. 29*, 4 (July 2010), 93:1–93:10. 2, 3, 11, 15

[Ols13] OLSEN M.: *Guidelines for the Use of Mobile LIDAR in Transportation Applications*. Report (National Cooperative Highway Research Program). Transportation Research Board, 2013. 14

[PV09] PU S., VOSSELMAN G.: Knowledge based reconstruction of building models from terrestrial laser scanning data. *{ISPRS} Journal of Photogrammetry and Remote Sensing 64*, 6 (2009), 575 – 584. 2

[PY09] POULLIS C., YOU S.: Photorealistic large-scale urban city model reconstruction. *IEEE Trans. Vis. Comput. Graph. 15*, 4 (2009), 654–669. 2

[PY11] POULLIS C., YOU S.: 3d reconstruction of urban areas. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 International Conference on* (May 2011), pp. 33–40. 2

[SB03] SCHINDLER K., BAUER J.: A model-based method for building reconstruction. In *Proceedings of the First IEEE International Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis* (Washington, DC, USA, 2003), HLK '03, IEEE Computer Society, pp. 74–. 5, 11

[SHFH11] SHEN C.-H., HUANG S.-S., FU H., HU S.-M.: Adaptive partitioning of urban facades. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH ASIA 2011* (2011). 5, 6, 11, 15

[SM15] SCHWARZ M., MÜLLER P.: Advanced procedural modeling of architecture. *ACM Trans. Graph. 34*, 4 (July 2015), 107:1–107:12. 2

[SSS*08] SINHA S. N., STEEDLY D., SZELISKI R., AGRAWALA M., POLLEFEYS M.: Interactive 3d architectural modeling from unordered photo collections. *ACM Trans. Graph. 27*, 5 (Dec. 2008), 159:1–159:10. 2

[TH15] TUTZAUER P., HAALA N.: FaÇade Reconstruction Using Geometric and Radiometric Point Cloud Information. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (Mar. 2015), 247–252. 3

[TKS*13] TEBOUL O., KOKKINOS I., SIMON L., KOUTSOURAKIS P., PARAGIOS N.: Parsing facades with shape grammars and reinforcement learning. *IEEE Trans. Pattern Anal. Mach. Intell. 35*, 7 (2013), 1744–1756. 2

[tv] Thingiverse. last accessed: 7 December, 2016. URL: http://www.thingiverse.com/search/page:1?sort=relevant&q=apa 11

[VAB10] VANEGAS C. A., ALIAGA D. G., BENES B.: Building reconstruction using manhattan-world grammars. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (June 2010), pp. 358–365. 2, 3

[VAB12] VANEGAS C. A., ALIAGA D. G., BENES B.: Automatic extraction of manhattan-world building masses from 3d laser range scans. *IEEE Transactions on Visualization and Computer Graphics 18*, 10 (Oct. 2012), 1627–1637. 2, 3

[VLA15] VERDIE Y., LAFARGE F., ALLIEZ P.: Lod generation for urban scenes. *ACM Trans. Graph. 34*, 3 (May 2015), 30:1–30:14. 3

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM Trans. Graph. 22*, 3 (July 2003), 669–677. 2

[WYD*14] WU F., YAN D.-M., DONG W., ZHANG X., WONKA P.: Inverse procedural modeling of facade layouts. *ACM Trans. Graph. 33*, 4 (July 2014), 121:1–121:10. 2

[XFT*08] XIAO J., FANG T., TAN P., ZHAO P., OFEK E., QUAN L.: Image-based faÇade modeling. *ACM Trans. Graph. 27*, 5 (Dec. 2008), 161:1–161:10. 3

[ZN11] ZHOU Q.-Y., NEUMANN U.: 2.5d building modeling with topology control. In *CVPR* (2011), IEEE, pp. 2489–2496. 2