Scan4Facade: Automated As-Is Façade Modeling of Historic High-Rise Buildings Using Drones and AI

Yuhan Jiang
North Carolina A&T State University

Sisi Han
Marquette University

Yong Bai
Marquette University, yong.bai@marquette.edu

Follow this and additional works at: https://epublications.marquette.edu/civengin_fac

Part of the Civil Engineering Commons

Recommended Citation
Jiang, Yuhan; Han, Sisi; and Bai, Yong, "Scan4Facade: Automated As-Is Façade Modeling of Historic High-Rise Buildings Using Drones and AI" (2022). Civil and Environmental Engineering Faculty Research and Publications. 374.
https://epublications.marquette.edu/civengin_fac/374
Scan4Façade: Automated As-Is Façade Modeling of Historic High-Rise Buildings Using Drones and AI

Yuhan Jiang  
Assistant Professor, Department of Built Environment, North Carolina A&T State University, Greensboro, NC  
Sisi Han  
Graduate Student, Department of Civil, Construction and Environmental Engineering, Marquette University, Milwaukee, WI  
Yong Bai  
McShane Chair and Professor, Department of Civil, Construction and Environmental Engineering, Marquette University, Milwaukee, WI

Abstract  
This paper presents an automated as-is façade modeling method for existing and historic high-rise buildings, named Scan4Façade. To begin with, a camera drone with a spiral path is employed to capture building exterior images, and photogrammetry is used to conduct three-dimensional (3D)
reconstruction and create mesh models for the scanned building façades. High-resolution façade orthoimages are then generated from mesh models and pixelwise segmented by an artificial intelligence (AI) model named U-net. A combined data augmentation strategy, including random flipping, rotation, resizing, perspective transformation, and color adjustment, is proposed for model training with a limited number of labels. As a result, the U-net achieves an average pixel accuracy of 0.9696 and a mean intersection over union of 0.9063 in testing. Then, the developed twoStagesClustering algorithm, with a two-round shape clustering and a two-round coordinates clustering, is used to precisely extract façade elements’ dimensions and coordinates from façade orthoimages and pixelwise label. In testing with the Michigan Central Station (office tower), a historic high-rise building, the developed algorithm achieves an accuracy of 99.77% in window extraction. In addition, the extracted façade geometric information and element types are transformed into AutoCAD command and script files to create CAD drawings without manual interaction. Experimental results also show that the proposed Scan4Façade method can provide clear and accurate information to assist BIM feature creation in Revit. Future research recommendations are also stated in this paper.

Introduction
Scan-to-BIM is the process of using three-dimensional (3D) measurement tools [e.g., 3D laser scanner or light detection and ranging (LiDAR)] to scan existing buildings or construction sites and then create building information modeling (BIM) objects over the captured 3D reality data (e.g., point cloud) (Bassier et al. 2016; Higgins 2022a; Park and Cho 2022; Perez-Perez et al. 2021). Generally, Scan-to-BIM is labor-intensive, costly, and time-consuming work, which requires manual interactions in the following procedures: setting up terrestrial laser scanner at multiple interior and exterior locations of a building to acquire completed and overlapped scans at first; then, registering scans (point clouds) to a unified point cloud; next, drafting two-dimensional (2D) drawings over the point cloud in AutoCAD (Autodesk, Mill Valley, California); and finally, tracing the drawings to create BIM features in Revit (Autodesk) (Abdelgawad et al. 2017; Bassier et al. 2016; Higgins 2022a). Moreover, recent studies showed advantages of artificial intelligence (AI, including machine learning and deep learning) in point cloud segmentation, object detection, and classification (Agapaki and Brilakis 2020; Koo et al. 2021; Ma et al. 2020; Park and Cho 2022; Perez-Perez et al. 2021; Xu et al. 2021). However, the automatic modeling objects are the same as the commercial software programs, such as EdgeWise Building and MEP (ClearEdge3D, Superior, Colorado), which are limited to indoor spaces, structures, and mechanical, electrical, and plumbing (MEP) systems (Abdelgawad et al. 2017; ClearEdge3D 2021a, b, c).

The demand for building façade inspection continuously increases in the United States and around the world. For example, in Cleveland, the initial deadline for buildings 30–50 years old was June 6, 2018, and the deadline for next inspection is June 6, 2023 (periodic inspection every 5 years) (City of Cleveland 2022); and in San Francisco, buildings five or more stories tall are required to conduct periodic façade inspection every 10 years, and the initial inspection deadline was December 31, 2021, for buildings built before 1910 (City and County of San Francisco 2021). During the inspection, façade sketches are essential documents for recording new distress conditions (i.e., locations and dimensions), marking changes in previously observed distress conditions, and compositing inspection reports (Mohammadi 2021; Shi and Ergan 2020). To achieve high accuracy in documentation, using up-
to-date as-is façade drawings is much better than using rough sketches. Previous research only showed evidence of ground-based LiDAR point cloud in low-rise building façade modeling (Bassier et al. 2017; Dore and Murphy 2014; Pérez et al. 2021; Wang et al. 2015; Xia and Wang 2019b), while to acquire scans of high-rise building façades, the LiDAR sensor would need to be mounted on aerial vehicles to access higher floors. Although the DJI’s product promotion shows that its latest LiDAR drone can 3D-map a high-rise building construction project (DJI 2022), there is no published work evaluating its performance. In contrast, a recent study (Jacob-Loyola et al. 2021) showed evidence for using a drone photogrammetric mesh model for high-rise construction progress monitoring in Navisworks (Autodesk) with as-designed BIM models. Previous research also indicates that close-range photogrammetry has an accuracy of 5 cm (Han et al. 2022; Jiang and Bai 2021; Takahashi et al. 2017), which is much the same as the expensive LiDAR drone that has a vertical accuracy of 5 cm and a horizontal accuracy of 10 cm at 50 m (DJI 2022). Therefore, using a cheaper camera drone for building façade scanning is more affordable, and low-rise and high-rise buildings can be accurately 3D-reconstructed via structure from motion (SfM) like the examples shown in Karachaliou et al. (2019) and Pix4D (2018), respectively. Consequently, orthoimages of the building façades can represent all façade elements in 2D (Jiang et al. 2021b; Karachaliou et al. 2019).

Therefore, to address the limitations of previous studies and advance the automatic level of as-is façade modeling and inspection of existing or historic buildings (especially high rise), this study proposes a cost-effective and accurate approach, named Scan4Façade, which integrates drone, AI, and BIM in scanning, processing, and modeling with the following features:

1. In the scanning phase, due to the height, a camera drone is used to access higher floors (tower) for capturing images of high-rise building; while for lower stories and indoor spaces, ground-based LiDAR approaches could be used as supplements (which are not discussed in this paper).
2. In the processing phase, SfM photogrammetry is used for 3D reconstruction with the captured façade images; AI technologies of supervised deep learning and unsupervised clustering are used to automatically extract and classify coordinates, sizes, and types of façade elements from façade orthoimages.
3. In the modeling phase, 2D as-is CAD drawings of façades are automatically drafted using AutoCAD commands and scripts; next, 3D as-is BIM models can be easily created by tracing the 2D drawings.

The remainder of this paper is organized as follows. The section “Background and Related Works” reviews related works in this area; the section “Scan4Façade: The Proposed As-Is Façade Modeling Method” introduces the proposed Scan4Façade method; the section “Experimental Results” presents the experimental results of automated as-is façade modeling of the Michigan Central Station (MCS) (office tower), in which 868 windows were used as examples in façade element extraction and modeling; finally, the sections “Discussion” and “Conclusion,” follow.

Background and Related Works

The National Fire Protection Association (NFPA) defines a high-rise building as a building higher than 23 m (75 ft) or about seven stories high (Hall 2011). In Chicago, the Home Insurance Building [originally 10 stories with a height of 42.1 m (138 ft), completed in 1885 and demolished in 1931] was one of the first
steel-frame skyscrapers (very tall high-rise buildings) built in the United States. Existing and historic high-rise buildings in the Midwest were influenced by the Chicago School-Architecture style, in which the arrangement of windows on the façade typically creates a grid pattern, like the buildings shown in Fig. 1. From 1860 to 2005, 419 buildings (five or more stories tall) were built in Milwaukee, more than half of which are NFPA-defined high-rise buildings (City of Milwaukee 2008). According to the histogram shown in Fig. 2, high-rise buildings initially bloomed from 1885 to 1935, with the tallest one, Hilton Milwaukee City Center (28 stories), shown in Fig. 1(a). Then, the second blooming started from 1945 to 2005, with the highest building, US Bank Center (43 stories), shown in Fig. 1(b). All of those buildings are more than 15 years old that were subjected to an initial critical exam and periodic façade inspections to ensure that they are in a safe condition, according to city building façade inspection ordinance (City of Milwaukee 2022). Because the as-is façade drawings can reflect the current façade conditions during the inspection period and are useful in recording new distress conditions, marking changes in previously recorded distress conditions, and composing façade inspection report, the remainder of this section summarizes and discusses the existing façade modeling–related technologies, techniques, and strategies.

Fig. 1. Downtown Milwaukee, Wisconsin: (a) Milwaukee River west; and (b) Milwaukee River east.

Fig. 2. Milwaukee buildings, built from 1860 to 2005: (a) histogram; and (b) bar chart.
As-Is BIM Workflow: Scanning, Processing, and Modeling

BIM is a system of digital building representation created for sharing information, conducting simulation, and performing automation control of the built environment during its life cycle (including design, construction, operation and maintenance, and deconstruction phases) (Czerniawski and Leite 2020; Higgins 2022b; Karunaratne and Dharmaratna 2022; Motalebi et al. 2022; Plaseckienė 2022). Consequently, in different stages of the building life cycle, BIM models can be classified as as-designed (before construction begins), as-constructed (at any time during construction), as-built (after construction has been completed), and as-is (existing conditions at some point after construction has been completed—often many years later) models (Higgins 2022c). As-is BIM for existing buildings gains ground toward sustainable facility management, which updates and maintains the information about current conditions of a building and any deformation or damage to the building (Dore and Murphy 2014; Higgins 2022b; Karachaliou et al. 2019; Wang et al. 2015). In addition, in contexts of building inspection, maintenance, renovation, addition and repair, the building owners and facility managers must provide the architects, engineers, and contractors with the as-built/as-is data (i.e., CAD drawings and BIM models) (Andrich et al. 2022; D’Angelo et al. 2022; Durdyev et al. 2022). However, some of the existing and historic buildings (e.g., buildings in Figs. 1 and 2) may lack as-built/as-is data because: (1) as-built drawings were not done as it was not required at that time; (2) as-built drawings are missing/damaged; and (3) as-built drawings may differ significantly from the real conditions of the buildings due to unrecorded changes (Higgins 2022c).

Currently, in Scan-to-BIM practice, data acquisition of indoor spaces is heavily reliant on the 3D laser scanning devices (LiDAR), for example, terrestrial laser scanners (terrestrial LiDAR) and mobile mapping systems [mobile LiDAR or simultaneous localization and mapping (SLAM)], which can take millions of measurements and collect 3D point clouds about the built environment (Abdelgawad et al. 2017; Czerniawski and Leite 2020; Higgins 2022a; Tang et al. 2010). Registering scans together is required for terrestrial LiDAR (ClearEdge3D 2021c). Then, an as-built/as-is BIM model could be created for the built environment based on the scanned point clouds via manual tracing (user models directly over scans), manual extraction (user selects a feature in scans, tools analyze feature geometry, and make a best-guess modeling decision), and automated extraction (tools analyze a whole section of the scans and return a geometric model) (Abdelgawad et al. 2017; Chen et al. 2019; Czerniawski and Leite 2020; Higgins 2022a; Park and Cho 2022; Perez-Perez et al. 2021; Xia and Wang 2019a). The automated extraction tools, such as EdgeWise Building and MEP (ClearEdge3D, Superior, Colorado), could save a lot of time in modeling regular shaped architectural and structural elements (e.g., walls, beams, and columns), MEP systems, and floor plans as Revit-friendly polygons; then, users could build new features directly on top of the polygons in Revit (Abdelgawad et al. 2017; ClearEdge3D 2021a, b, c).

Additionally, in Scan-to-BIM academic research, semiautomated and automated approaches have been developed to generate geometric models from point clouds, such as pipelines (Lee et al. 2013), indoor spaces (Hong et al. 2015; Macher et al. 2017), indoor structures (Jung et al. 2014, 2016), steel structures (Yang et al. 2020), and low-rise building façades (Dore and Murphy 2014; Wang et al. 2015). Moreover, recent research took advantage of AI (including machine learning and deep learning) and improved the automatic level of ground-based LiDAR point cloud segmentation, object detection and
classification via support vector machines (Bassier et al. 2017), convolutional neural networks (CNNs) (Perez-Perez et al. 2020, 2021; Xu et al. 2021), and PointNet/PointNet (Agapaki and Brilakis 2020; Chen et al. 2019; Koo et al. 2021; Ma et al. 2020; Park et al. 2020; Park and Cho 2022). However, the previously explored automatic modeling objects are the same as the commercial tools limited to indoor spaces, indoor structures, and MEP systems; and users may have to remodel numerous parts manually (Abdelgawad et al. 2017; Higgins 2022a).

Furthermore, the aforementioned ground-based (or human and robotics carried) LiDAR approach is limited to scanning low-rise building façades (Dore and Murphy 2014; Pérez et al. 2021; Xia and Wang 2019b). To scan high-rise building façades, the LiDAR sensor needs to be mounted on aerial vehicles to access higher floors. Previous research used airborne LiDAR for 3D city modeling, while it was limited to the building footprints and roofs (Huang et al. 2018; Park and Guldmann 2019; Resop et al. 2019; Zhang et al. 2021b). The drone-carried LiDAR approach has been used for 3D mapping (3D reconstruction) of pavement (Li et al. 2019), vegetation (Kellner et al. 2019; Resop et al. 2019), bridges (Bolourian and Hammad 2020), riverscape topography (Resop et al. 2019), and urban regions and communication towers (Alsadik and Remondino 2020). The DJI’s product promotion shows that its Matrice 300 RTK and Zenmuse L1 (integrates a LiDAR module and an RGB camera) can 3D-map a high-rise building construction project (DJI 2022). The flight planning method for tower 3D mapping (Alsadik and Remondino 2020) has the potential for high-rise buildings, while no existing research evaluated the drone LiDAR in high-rise building façade modeling. In contrast, a previous research (Karachaliou et al. 2019) showed evidence of using drone photogrammetry to generate façade orthoimages to assist low-rise historic building as-is BIM modeling; a previous study (Jacob-Loyola et al. 2021) showed the feasibility of using a drone photogrammetric mesh model to monitor the progress of a high-rise building construction project with as-designed BIM and Navisworks. Moreover, the DJI’s camera drone Phantom 4 Pro V2.0 ($1,599) and the surveying version Phantom 4 RTK ($6,600) are much cheaper than the LiDAR drone ($28,000) (DJI 2022). Previous research also indicates that close-range photogrammetry has an accuracy of error within 5 cm (Han et al. 2022; Jiang and Bai 2021; Takahashi et al. 2017), which is much the same as the expensive LiDAR drone that has a vertical accuracy of 5 cm and a horizontal accuracy of 10 cm at 50 m (DJI 2022). Therefore, using a cheaper camera drone for high-rise building façade scanning is more affordable, and façades can be accurately 3D-documented like the example in Pix4D (2018).

AI-Assisted Drone Photogrammetry (3D Reality) Data Processing

Drone Photogrammetry is the process of using a drone to capture a series of highly overlapped images, then extracting and matching feature points from overlaps to determine spare points’ geometrical data, and using dense matching to generate point clouds (Han et al. 2022). The commercial photogrammetry software, such as Metashape (Agisoft LLC, St. Petersburg, Russia), ReCap Photo (Autodesk), Pix4Dmapper (Pix4D, Prilly, Switzerland), and so on, can generate reality data in formats of 2D orthophotos, 3D point cloud, and 3D mesh model. AI is powerful technology that has been tested in 3D reality data acquisition (Choe et al. 2021; Han et al. 2022; Knyaz et al. 2017; Luo et al. 2016) and most widely used in 3D reality data processing. Section “As-Is BIM Workflow: Scanning, Processing and Modeling” showed evidence of AI-assisted 3D point cloud processing for automated object extraction, while the most employed architecture, engineering, and construction (AEC) cases are AI-assisted 2D/3D image processing for automated object detection. Previous studies showed the feasibility of
representing AEC objects’ features of color texture and elevation via a large resolution photogrammetric orthophoto (Dadrasjavan et al. 2019; Jiang et al. 2021b), range image (Zhou and Song 2020a, b), 3D pavement image (Hsieh and Tsai 2020), surface height plot (Edmondson et al. 2019), or depth map (Roberts et al. 2020). Then, CNNs with the sliding window scheme were employed for AEC object detection from small and thin cracks (Ali et al. 2019; Jiang et al. 2021a; Maniat 2019; Protopapadakis et al. 2019; Zhou and Song 2020a) to large vegetation areas and linear roadways (Jiang et al. 2020, 2021a; Kussul et al. 2017; Liu et al. 2018; Liu and Abd-Elrahman 2018).

Similarly, each building façade surface is a relatively flat plane, and the façade elements have a much simpler geometrical shape compared with structural and mechanical elements. As a result, the pixel coordinate of an orthoimage of the building façade can represent all façade elements’ 2D locations (Jiang et al. 2021b; Karachaliou et al. 2019). Previous studies show the possibility of using a 2D GIS spatial model (via stitching drone-based close-range images) in façade inspection (Chen et al. 2021a), the feasibility of using point clouds’ orthographic views of RGB and normal features in window detection (Jiang et al. 2021b), and the evidence of using drone photogrammetry to record historic building façades and generate orthoimages to assist in historic building information modeling (Karachaliou et al. 2019). Like in Fig. 1, most of the existing or historic high-rise building façades are vertical surfaces and can be represented as 2D flat planes alternative to complicated 3D indoor spaces. It is unnecessary to conduct AI-assisted point cloud segmentation to extract façade elements from point clouds. In contrast, applying AI-assisted image segmentation methods could further simplify the as-is façade modeling in the critical processes of façade element detection, like the wall and door detection in BIM models using multi-view CNN (Koo et al. 2021), the window detection in aerial images using fast-RCNN (Lippoldt 2019), and the window detection in façade images using semantic-synchronized GAN (Cai et al. 2021). As a result, façade elements, such as windows, can be represented as 2D bounding rectangles alternative to the 3D bounding boxes in point cloud (Chen et al. 2019).

Moreover, for precisely extracting a building façade object’s boundary, a convolutional encoder–decoder-based pixelwise segmentation is required, such as FCN (Alipour et al. 2019; Shelhamer et al. 2017), U-net (Liu et al. 2019; Ronneberger et al. 2015), SegNet (Badrinarayanan et al. 2017; Kearney et al. 2020), and DeepLabv3+ (Chen et al. 2018; Ji et al. 2020), which were designed for general image semantic segmentation tasks, and have been adopted in AEC for object detection with 2D images. Generally, convolutional encoder–decoder models use convolutional and deconvolutional layers to generate and explain feature maps; use max-pooling and up-sampling layers to resize feature maps and keep the main features after convolutional and deconvolutional layers; use rectified linear unit (ReLU) activation function in hidden layers for faster model training; use dropout layers and early stopping to prevent overfitting (Jiang et al. 2021b, 2022a). Also, they use merging layers to combine the feature maps (tensors) from two different layers as a new feature map (tensors) (Chollet 2020a), such as the element-wise addition layers used in FCN (Shelhamer et al. 2017) and channel concatenation layers used in U-net (Ronneberger et al. 2015). For image segmentation: (1) using Sigmoid activation function in the end convolutional layer can directly generate the numerical value prediction for each pixel; and (2) using SoftMax activation function in the end convolutional layer can get the probability of each pixel belonging to the predefined classes, such as crack and noncrack (Badrinarayanan et al. 2017; Song et al. 2020).
Additionally, the U-net (designed for biomedical image binary segmentation) has the advantage of achieving a higher accuracy with smaller training data sets (images and ground truth labels) (Jiang et al. 2021b; Liu et al. 2019; Zhang et al. 2021a) and shows a good performance in thin object detection (Majidifard et al. 2020), that is, when applied to pavement crack detection, it achieved a pixel accuracy of 0.9892 and an intersection over union (IoU) of 0.4850 (Augustaukas and Lipnickas 2019). In addition, previous evaluation concluded that performances are not significantly different among CrackSeg (Song et al. 2020), DeepCrack (Zou et al. 2019), DeepLabv3+, PSPNet (Zhao et al. 2017), U-Net, and SegNet for crack/noncrack binary segmentation on flat pavement surfaces (Song et al. 2020). Therefore, in this research, U-net was selected as an example AI model for target façade object/nontarget binary segmentation over façade orthoimages.

Furthermore, the AI model–generated pixelwise segmentation results need to be further processed for individual façade object extraction with coordinates and sizes and then for the classification of the extracted façade objects into different categories and types. Previous research (Jiang et al. 2021b) shows that drone photogrammetry–generated point clouds are sparse in some places, especially in windows, and could result in the created mesh models containing deformations at those places. Increasing overlaps could not fix this issue, because photogrammetry has limitations in poorly textured surfaces and reflective materials, and these uniform surfaces were unable to generate sufficient key points due to low contrast and variation (Jiang and Bai 2021). As a result, the extracted façade objects would have different dimensions, even though they are the same type objects. Therefore, adding clustering to correct the extracted façade object results is necessary, because the dimensions are significantly different between the different types of façade objects but have slight changes within each singular type (Lippoldt 2019). In addition, the deformations could also result in coordinate offset for façade objects. As shown in Fig. 1, the arrangement of windows on a high-rise building façade is in a grid pattern, which means the as-designed and as-built windows are aligned to the same elevation for each floor and the same column crossing multiple floors. Then, the created as-is windows should agree with the same rules if no significant damage or deformation occurred on the façade. Therefore, the extracted façade objects should be further clustered to find a unified elevation for each floor and a unified axis for each column. In this research, k-means clustering (able to set a specific number of clusters to generate that number of centroids) was selected as an example clustering approach for façade object-type clustering and coordinate alignment.

Scan4Façade: The Proposed As-Is Façade Modeling Method

This section introduces the proposed as-is façade modeling method, Scan4Façade, as shown in Fig. 3. There are four main procedures, namely, (1) drone photogrammetry and façade orthoimage generation; (2) AI-assisted façade orthoimage segmentation; (3) façade element extraction, classification, and alignment; and (4) façade reconstruction and as-is modeling in AutoCAD and Revit. Details of each procedure are discussed in the following.
A spiral path is recommended to capture overlapped images of high-rise buildings for drone photogrammetry (Fig. 3). This research employed SfM photogrammetry software ReCap Photo for 3D reconstruction and model visualization, in which the generated 3D mesh model can be viewed in nonperspective (orthographic) mode. Assuming a building has four façades, and all are flat planes, then, the orthographic left, right, front, and back views of the 3D mesh model represent the left, right, front, and back façades, respectively. For each orientation, a 2D façade orthoimage could be captured on screen or exported from ReCap Photo manually. In addition, the authors also developed a pointcloud2orthoimage tool to generate façade orthoimages with any designated scale from photogrammetry or LiDAR point cloud; the code and demo are available in Jiang (2022) and Jiang et al. (2022b). In photogrammetry, the scale is named ground sampling distance (GSD), which means that a pixel's length stands for a physical length of ground in centimeters (Han et al. 2022). Examples of façade orthoimages are shown in Fig. 3 and available in Jiang (2022).

Additionally, on each building façade, a façade element can be uniquely defined using the 2D coordinates of \( X \) (column, horizontal distance to the left edge) and \( Y \) (elevation, vertical distance to ground) and the Type (with specific dimensions of width and height). The \( X \) and \( Y \) coordinates could be converted from the pixel coordinates of the corresponding façade orthoimage with a known GSD. Consequently, the façade element extraction task can be simplified as a 2D image processing task.

AI-Assisted Façade Orthoimage Segmentation

Data Set Creation and Data Augmentation

The first stage is preparing images and labels for deep learning model training if the target façade element is a new one. According to the discussions in the section “AI-Assisted Drone Photogrammetry (3D Reality) Data Processing,” the U-net (Ronneberger et al. 2015) was selected for façade orthoimage...
pixelwise segmentation because it can reach higher levels of accuracy with fewer model training images and labels (Jiang et al. 2021a, b; Liu et al. 2019; Zhang et al. 2021a). In addition, it is better to prepare separate labels and train separate U-net models for different kinds of façade elements because U-net was designed for binary segmentation tasks, such as using 0 for windows and 1 (or 255 in 8-bit grayscale) for non-windows. Using a smart tablet and digital pen to create labels in a digital drawing application, such as SketchBook (Autodesk), is recommended [Fig. 4(a)]. Then, the exported image format (*.PNG) labels can be further processed by Label-App (Jiang et al. 2020, 2021a) to generate spreadsheet format (*.CSV) labels [Fig. 4(b)]. Orthoimages are not necessary in a training data set. Images can be snapped from a mesh model in perspective mode, and the regions of target façade elements need to be marked on separate layers in SketchBook [Fig. 4(a)].

![Fig. 4. Label creation: (a) SketchBook; and (b) Label-App.](image)

The exported image and label (from a 12.9-inch iPad Pro [Apple, Cupertino, California]) have a dimension of 2,732 × 2,048 pixels, which is too large to be processed in a deep learning workstation (with 4 × 11 GB memory GeForce RTX 2080 Ti GPUs [Nvidia, Santa Clara, California]) for U-net model training and prediction. Therefore, the authors employed a previously developed input disassembling and output assembling algorithm to supply U-net in model training and prediction stages, in which 50% overlap in both width and height (Jiang et al. 2020, 2021a). The small-patch size of 128 × 128-pixel was selected because the previous comparative analysis results confirmed that it was better than 256 × 256 and 512 × 512-pixel in AEC applications of onsite path detection, pavement cracking detection, highway slope detection, and building façade window detection (Jiang et al. 2021b) and also suitable in façade crack inspection with patch-based CNN and U-net (Chen et al. 2021b).

Furthermore, the following combined data augmentation (DA) strategies were designed to prepare the 128 × 128-pixel small-patch data sets:

1. Randomly flip image and label in one of the following options: horizontal flipping [Fig. 5(a)], vertical flipping, horizontal and vertical flipping [Fig. 5(b)], or nonflipping.
2. Randomly resize the flipped image and label in the range of [0.5, 1.5].
3. Either randomly conduct the perspective transformation of the image and label [keep left, right, top, or bottom edge the same, see Figs. 5(c–e)], or not.
4. Cut blank margins (in left, right, top, and bottom edges) from the transformed image and label.
5. Pad the remaining image and label to be multiples of 128 pixels.
6. Randomly adjust the padded image’s brightness, color, contrast, or sharpness in the range of [0.5, 1.5]; adjustments are not applied to the label [Figs. 5(d and e)].
7. Rotate the adjusted image and label by 0°, 90°, 180°, and 270°.
8. Crop the four sets of rotated image and label into 50% overlapped 128 × 128-pixel small-patches by moving a 128 × 128-pixel slide window with a stride of 64-pixels in both width and height directions.
9. Discard blank image and label small patches, but keep only textured samples for reducing the overall size of data sets.

**Fig. 5.** Data augmentation: (a) horizontally flipped; (b) horizontally and vertically flipped; (c) perspective transformed (left fixed); and (d and e) brightness adjusted.

By repeating Steps 1–9 with multiple rounds of DA (recommend about 20–30 rounds, which skips the random processing Steps 1–3 and 6 at the first round for keeping the original image and label, and then, run all Steps 1–9 for the remaining rounds), the created model training data sets will have enough differences in size, shape, color, orientation, and perspective view of the target façade element; then, the trained U-net will be able to detect the façade elements on arbitrary views (which is discussed subsequently).

**Model Training and Model Usage**

In this research, U-net model was set up in Keras 2.3.1 (TensorFlow-GPU 1.14, Google Brain Team, Mountain View, California); model architecture and code can be found in Chollet (2020b) and Zhi (2019). The following configurations are recommended for training:

1. `model.compile(optimizer="adam,"loss="binary_crossentropy,"metrics=["accuracy"]` which is suitable for binary segmentation of window and nonwindow.
2. `earlystopping=EarlyStopping(patience=10,monitor="val_loss,"mode="min")` which stops model training when the validation loss does not reduce for 10 epochs. It is important to avoid model overfitting.
3. `checkpointer=ModelCheckpoint(filepath="Unet_model.h5,"save_best_only=True,monitor="val_loss,"mode="min")` which saves the model only if the validation loss is reduced in an epoch.
4. `model.fit(batch_size=256,epochs=100,validation_split=0.05,callbacks=[earlystopping,checkpointer])` which means using 256 samples in one forward/backward pass to avoid insufficient GPU
memory, stopping model training at the 100th epoch, and using 5% samples for model validation.

Additionally, due to the trained U-net model having the fixed dimensions of 128 × 128-pixel, the following operations are recommended to process a large-sized image to generate a large-sized label image:

1. Pad a large dimension image to an integral multiple of 128-pixel with blank right and bottom margins.
2. Disassemble the padded image into 50% overlapped 128 × 128-pixel small patches, which is the same as Step 8 in data augmentation.
3. Use the well-trained U-net to generate label prediction for each disassembled small patch.
4. Stitch small-patch label predictions via 50% overlaps in both width and height directions.
5. Change the pixel value in the range of [0, 15] to 0 as the target element and [240, 255] to 255 as nonelements.
6. Remove the padded right and bottom margins to return an original-dimension pixelwise-segmented label image for the inputted large-dimension image.

Step 5 is necessary because the original U-net predictions (values in the range of [0, 1]) are multiplied by 255 to return the 8-bit grayscale label images in which pixel values are in the range of [0, 255]. In future practice, the binary filtering is an alternative option, which updates all pixels with value ≤255/2 to 0 to indicate façade elements and replaces others with 255 to represent nonelements.

Façade Element Extraction, Classification, and Alignment

Element Extraction

Façade elements in drone photogrammetric mesh models may have irregular shapes like the distorted windows in Fig. 6(a), resulting in pixelwise-segmented label images having the same deformation in Fig. 6(b). The authors proposed a sawtooth() algorithm (pseudocode shown in Fig. 7) to trim labels and generate sawtooth labels shown in Fig. 6(c), where the irregular edges of the original labels were trimmed to nearly straight edges, and small pieces of noises were broken into fragile pieces as well.

![Figure 6](image_url)

**Fig. 6.** (a) Original image; (b) U-net labels; and (c) sawtooth-labels overlap the image.
sawtooth(Label):
    Mask[:,] = 0  # initial Mask, which has the same size as Label
    Mask[Label == 0] = 1  # update Mask for generating horizontal sawtooth
    for v in range(0, imgHeight, stride):  # moving row to row
        for u in range(0, imgWidth, stride):  # recommended stride = 5
            win = Mask[v - 1:v + 1, u - winSize1:u + winSize1]  # pick up a window
            num = sum(win)  # count the number of façade element pixels
            if num < threshold:  # recommended threshold 10 or 15
                Label[v - 1:v + 1, u - winSize2:u + winSize2] = 255  # update the label
        Mask[:,] = 0  # initial Mask for the vertical sawtooth
    Mask[Label == 0] = 1
    for u in range(0, imgWidth, stride):  # moving col to col
        for v in range(0, imgHeight, stride):
            win = Mask[v - winSize1:v + winSize1, u - 1:u + 1]  # recommended winSize1 = 5
            num = sum(win)
            if num < threshold:  # recommended threshold 10
                Label[v - winSize2:v + winSize2, u - 1:u + 1] = 255  # recommended winSize1 = 5 * 2
    return Label

Fig. 7. Pseudocode of the sawtooth() algorithm.

Additionally, a combinedElementExtraction() function was designed to extract the contours and straight rectangles of all individual façade elements that occurred in the label images. This function combined findContours() and boundingRect() functions in Open Source Computer Vision Library (OpenCV 2020). The findContours() returns a list of contours in the label image, no matter their sizes. This means the fragile pieces are also included in the list. Therefore, a filter of area (or width and height) is used to skip thin pieces of noise while using boundingRect() to find straight bounding rectangles for all façade elements. The façade element extraction results are numbered and annotated like in Fig. 8(a) and returned as a Data-Frame (df_Element) with columns = [x, y, w, h] for representing extracted façade elements. The obtained properties include the coordinates of x and y and the dimensions of width (w) and height (h), which are in pixel coordinates, and the coordinate origin is the top-left corner like in Fig. 8(a). In addition, each window’s detailed profile (cropped from the façade orthoimage) can be easily generated on the basis of a pixelwise label for future reference in modeling the detailed shape and material.
Shape Classification and Space Alignment

Mesh models’ deformation can result in various dimensions of width and height that are extracted for the same-type façade element. The authors developed a *twoStagesClustering()* algorithm (pseudocode shown in Fig. 9) to automatically classify extracted façade elements. Examples of processing status and results are shown in Fig. 8. In this algorithm, the input parameter *stories* can be either manually inputted or automatically determined as the number of windows in a line (column) (Lippoldt 2019), which is not discussed in this paper.
twoStagesClustering(df_Element, stories):
    num_elements = len(df_Element)
    num_yCoord = int(stories * 1.2 - 1) # number of class and centroid for y coordinate clustering
    num_xCoord = int(num_elements/stories * 2 + 1) # number of class and centroid for x clustering
    num_size&ele = stories + 17 # number of class and centroid for 1st shape clustering
    df_Element, (centroid_size&ele, centroid_w&h, centroid_x, centroid_y)
        = elementClustering(df_Element, (num_size&ele, 0, num_xCoord, num_yCoord)) # 1st clustering
    Mask[:] = 255 # has the same dimension as the Label
    for i in range(num_elements):
        c = int(df_Element['size&ele_label'][i]) # get the 1st clustering type
        w = centroid_size&ele[c,0] # width of the clustering centroid
        h = centroid_size&ele[c,1] # height of the clustering centroid
        x = centroid_x[int(df_Element['xlabel'][i])] # get the 1st coordinates clustering type
        y = centroid_y[int(df_Element['ylabel'][i])] # and get the centroid of x and y
        rectangle(Mask, (x, y), (x+w, y+h)) # draw the current element in the Mask with a filled rectangle
    df_Element = combinedElementExtraction(Mask) # extract elements in the Mask/Modified label — image
    num_size = stories - 5 # number of class and centroid for 2nd shape clustering
    df_Element, (centroid_size&ele, centroid_w&h, centroid_x, centroid_y)
        = elementClustering(df_Element, (0, num_size, num_xCoord, num_yCoord)) # 2nd clustering
    for i in range(num_elements):
        c = int(df_Element['w&h_label'][i]) # get the 2nd clustering type
        w = centroid_w&h[c,0]
        h = centroid_w&h[c,1]
        x = centroid_x[int(df_Element['xlabel'][i])] # get the 2nd coordinates clustering type
        y = centroid_y[int(df_Element['ylabel'][i])] # and get the centroid of x and y
        coordinateConversion(w, h, x, y, GSD)

Fig. 9. Pseudocode of the twoStagesClustering() algorithm.

In Fig. 9, elementClustering() returns a modified df_Element with added columns of size&elelabel, w&hlabel, xlabel, and ylabel to represent K-means clustering results via the combined façade element properties of w, h, y, w, h, x, and y, respectively. The first shape clustering considers the extracted elements’ elevations and dimensions of width and height; then, the elements are grouped within each floor (story), and the results are numbered and annotated like in Fig. 8(b). The first clustering also creates a modified label image (Mask) to reflect the shape clustering results like in Fig. 8(c). In the Mask, elements are regular rectangles with the dimensions and coordinates from the first clustering results, and the sawtooth is not applied. Then, façade elements are extracted via the combinedElementExtraction() function again, and the results are numbered and annotated like in Fig. 8(c). The second shape clustering considers only the elements’ dimensions to group the same-sized elements crossing different stories, and the results are numbered and annotated like in Fig. 8(d).

Additionally, the mesh models’ deformation may also result in various coordinates for the extracted façade elements at the same horizontal position (column) or/and the same vertical position (story). The authors also employed the two-stages coordinate clustering of x and y along the two-stage shape clustering. The x and y values of the clustering centroids will be assigned to the x and y of each façade.
element according to its coordinate clustering types. The two-stage space alignment results are numbered and annotated like in Figs. 8(b and d).

Facade Reconstruction and As-Is Modeling

The extracted facade elements’ coordinates and dimensions are in pixel coordinates (origin is the top-left corner of each orthoimage) and are different from the AutoCAD/Revit software’s coordinates and units. In case facade orthoimages are generated from the mesh model, the actual dimension of a facade element can be measured on a jobsite or obtained from the 3D mesh model; then, the GSD can be determined by GSD = actualSize/pixelSize. When using the pointcloud2orthoimage tool (Jiang 2022), the GSD is a designated value. A coordinateConversion() function was designed to automatically convert pixel coordinates x and y to real coordinates X and Y for each extracted facade element; the pseudocode is shown in Fig. 10. The coordinate conversion results are numbered and annotated like in Fig. 11(a). In the global relation of each facade orientation, the pixel coordinate origin (x₀ = 0, y₀ = 0) has the real coordinate (X₀ = 0, Y₀ = Roof Height), and y and Y have reversed directions. In addition, in the local relation of each extracted facade element, the converted real coordinate Y refers to the bottom-left corner alternative to the top-left one in pixel coordinate y.

coordinateConversion(w, h, x, y, GSD)
W = w × GSD #convert w, h to real — world size W, H
H = h × GSD
X = pixel2real(x) #convert x, y to real coordinate X, Y
Y = pixel2real(y) − H #change the local coordinate origin from the top — left corner to bottom — left of each element
C = ‘Element’ + str(c) #element type name
writeCommand2File([str(X), \t, str(Y), \t, str(W), \t, str(H), \t, str(C), \n]) #in lines, X, Y, W, H and C are separated with the delimiter of \t tab

Fig. 10. Pseudocode of the coordinateConversion() function.

In Fig. 10, the writeCommand2File() function exports the extracted facade elements’ properties of X, Y, W (width), H (height), and C (type) to a command file like in Fig. 11(b), where the physical unit is millimeter. This command format is designed to fit with the Rectangle and Text functions in AutoCAD. The authors first proposed an approach to simulate keyboard typing and pressing to automatically draw individual elements and annotate their types via AutoCAD command. In addition, an accelerated version transformed the command file to an AutoCAD script file to run it automatically. Video recording of both drafting processes is available in Jiang (2021a). The drafted results will look like Fig. 11(c), where the origin is marked and annotated as well.
Experimental Results

The MCS is a historic former intercity passenger rail depot in Detroit. It was added to the National Register of Historic Places in 1975. The roof height of the MCS is 70 m, and the office tower has a typical floor plan shown in Fig. 12(b), which was reproduced from the original floor plan presented in Seiler (2013). In this research project, 90 of the 739 images of the MCS were collected from a public access online drone photogrammetry demo (Pix4D 2018). These 90 images were imported into ReCap Photo to generate a 3D mesh model in 1.5 h. In Fig. 12(a), the train station was clipped, and only the office tower was kept.

![Fig. 12. Michigan Central Station (office tower): (a) mesh model with camera positions; (b) typical floor plan; and (c) point cloud section.](image)

Façade Segmentation Model Training and Validation

The image and label shown in Fig. 4 were used to train the U-net model for window detection. The image contains a few windows that snapped from the mesh model in perspective mode, and the image and label have the dimension of 2,732 × 2,048 pixels and are available in Jiang (2021b). By running DA 30 times, 102,975 image and label small patches (128 × 128-pixel) were generated for model training and validation. Then, in each epoch, the U-net was trained on 97,826 samples and validated on 5,149 samples, which had the plots of model training and validation loss and accuracy, as shown in Fig. 13. The typical training epoch spent 212 s with 4 × 11 GB memory GeForce RTX 2080 Ti GPUs. The model training was completed at the 100th epoch, which means that the early stopping function was not activated.

![Fig. 13. U-net training and validation: (a) loss; and (b) accuracy.](image)
The best model occurred and was saved at the 100th epoch, which had the lowest validation loss of 0.0316 and a validation accuracy of 0.9922. The training and validation accuracy in Fig. 13 were measured by Keras accuracy, which calculates how often predictions equal labels for the 128 × 128-pixel small patches (Chollet 2020c). Moreover, compared with the manually created label image in Fig. 4, the assembled large-sized label image predication has a pixel accuracy of 0.9993, a nonwindow IoU of 0.9992, and a window IoU of 0.9964. These numeric evolutions are better than using point cloud RGB + Normal feature images with a pixel accuracy of 0.991048, a nonwindow IoU of 0.99008, and a window IoU of 0.898452 (Jiang et al. 2021b). In addition, these numeric evolutions confirmed that the U-net was well trained with DA, and the applied image disassembling and label assembling algorithm did not impact U-net’s performance as in the previous studies (Jiang et al. 2020, 2021a).

Window Extraction, Classification, and Alignment

Four orthoimages (GSD = 23 mm/pixel) of the left, right, front, and back of the MCS were exported from the mesh model via ReCap Photo and then processed by the developed method for window extraction, classification, and alignment. The finalized windows of the left and right façades are shown in Figs. 8 and 11, respectively. The front and back façades are shown in Figs. 14(a and b), in which the deformations of both orthoimages were clipped to avoid the U-net detecting them as windows, and a distorted window (around \(X = 82,000\) mm and \(Y = 65,000\) mm) in the back façade was removed as well. Figs. 14(a and b) show that the window space alignment was well executed.

Due to the deformations, the ground truth pixelwise labels were hard to create for the four façade orthoimages, and the numerical values of pixel accuracy and IoU were not evaluated. However, it is obvious that all annotated windows precisely represent the true windows (true positive pixels are over 50%). Moreover, the extracted windows are compared with the actual numbers in Table 1, where only two small windows in the front façade (around \(X = 18,000\) mm and \(Y = 22,000\) mm) were skipped.
These two windows had very narrow U-net predictions, which were discarded by the filter in the `combinedElementExtraction()` function. The 865 extracted windows accounted for 99.77% of the 867 total windows (excluding the removed one on the back façade).

**Table 1. Detected windows**

<table>
<thead>
<tr>
<th>Façade</th>
<th>Window</th>
<th>Detected</th>
<th>Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>97</td>
<td>97</td>
<td>100.00</td>
</tr>
<tr>
<td>Right</td>
<td>103</td>
<td>103</td>
<td>100.00</td>
</tr>
<tr>
<td>Front</td>
<td>350</td>
<td>348</td>
<td>99.43</td>
</tr>
<tr>
<td>Back</td>
<td>317</td>
<td>317</td>
<td>100.00</td>
</tr>
<tr>
<td>Total</td>
<td>867</td>
<td>865</td>
<td>99.77</td>
</tr>
</tbody>
</table>

As-Is Modeling in AutoCAD and Revit

Four command files of the left, right, front, and back façades were created and saved in *Notepad* (*.TXT) format, as in the example shown in Fig. 11(b). Next, the four drawings of the left, right, front, and back façades were automatically drafted in AutoCAD via the corresponding command and script files; see the video recording in Jiang (2021a). Each drawing has an annotated origin in the bottom-left corner, see Fig. 11(c), which is designed to register the four façade drawings to the four corners of floor plans to assemble a 3D wireframe for the building like in Fig. 15(b).

Additionally, Revit was used to create the as-is BIM model via the following steps:

1. Insert the typical floor plan, Fig. 12(b), into Level 1 (elevation = 0) and roof (elevation = 70,000 mm).
2. Insert the left, right, front, and back drawings into the left, right, front, and back elevations, respectively.
3. Align the marked origins to corresponding corners like in Fig. 15(a).
4. Place walls along the typical floor plan and connect them to the roof.
5. Insert windows according to the positions and the annotated window types. The Revit has the *Copy* (shortcut CO) and *Array* (shortcut AR) functions to accelerate the reduplicative operations of inserting the same type of windows.
The created as-is BIM model of the MCS (office tower) is shown in Fig. 15(c), in which all 868 windows were precisely placed in their positions. Using the automatically drafted 2D façade drawings as the reference information provides more clarity, more accuracy, and more information than directly tracing point clouds or the mesh model, like in Fig. 15(d), to create as-is BIM models for high-rise buildings.

**Discussion**

**Efficiency of Data Augmentation**

In the early stages of this research, the authors trained the U-net with four pairs of images and labels with dimensions of $2,732 \times 2,048$ pixels, which are available in Jiang (2021b). The conducted DA contains $0^\circ$, $90^\circ$, $180^\circ$, and $270^\circ$ rotations of the images and labels. The trained U-net was used for on-screen window extraction directly over the ReCap Photo, a video recording is available in Jiang (2020), which indicates that the U-net can detect windows accurately when a window has a similar size and view angle to training datasets; however, the U-net’s performances significantly dipped when switching view angles and adjusting the mesh model sizes. In other words, the U-net is sensitive to a target object’s size. This is reasonable because the U-net was originally developed for biomedical image segmentation problems, in which target objects have similar sizes among a stack of images (Ronneberger et al. 2015). Therefore, randomly resized images and labels in a range of $[0.5, 1.5]$ were added in Step 2 of DA to improve the performance with different-sized windows. In addition, to successfully detect windows from different view angles, Step 3 of DA contains random perspective transformations of the images and labels. Moreover, brightness, color, contrast, and sharpness adjustments were conducted in Step 6 of DA, because the different façade faces of a high-rise building would have different lighting environments like in examples shown in Fig. 14.

By conducting the proposed DA, the well-trained U-net model has an average testing pixel accuracy of 0.9696 and a mean IoU of 0.9063 (a nonwindow IoU of 0.9614 and a window IoU of 0.8512) in the four
pairs of images and labels. The relative lower performance of window IoU resulted from the ground truth labels being not accurate on window boundaries. The well-trained U-net model successfully passed the testing with the 3D mesh model’s arbitrary views in both orthographic [Fig. 16(a)] and perspective [Fig. 16(b)] modes and also passed the testing with an image of high-rise buildings on Wall Street, New York [Fig. 16(c)]. Fig. 16 shows that the windows are well extracted and annotated with contours. Therefore, the developed DA can improve U-net’s façade element detection performance on arbitrary views, while using a limited number of labels in model training. In addition, the well-trained U-net model with the disassembling and assembling algorithm can be used for window detection with other image sources and other buildings without size limitation, while the previous research could only produce label images with a size of 256 × 256-pixel using the same GPU GeForce RTX 2080 Ti (Cai et al. 2021). The façade orthoimage segmentation could be further improved by adding more training datasets, such as the low-rise building façade database, eTRIMS, and LabelMeFacade (Frohlich et al. 2010).

Effectiveness of Clustering for Classification

The classified window types of each façade are listed in Table 2. According to the mesh model, the windows of Left-W-3, Right-W-1, Front-W-1, and Back-W-0 are the main types of windows in the MCS; however, the extracted sizes were not equal to one another. The extracted widths were [87, 88, 89] pixels, and the heights were [115, 116, 118, 119] pixels. Therefore, the authors conducted the multivariate analysis of Cluster Observations (linkage method: complete; distance measure Euclidean; number of clusters: 12) with the extracted 865 windows by using statistical software Minitab (LLC, State College, Pennsylvania). The third shape clustering results are shown in Table 3, and analysis follows.

Table 2. Classified windows via two-stage-clustering

<table>
<thead>
<tr>
<th>Façade Type</th>
<th>Num.</th>
<th>W (mm)</th>
<th>H (mm)</th>
<th>w (px)</th>
<th>h (px)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-W-0</td>
<td>4</td>
<td>2,369</td>
<td>2,622</td>
<td>103</td>
<td>114</td>
<td>Cluster6</td>
</tr>
<tr>
<td>Left-W-1</td>
<td>7</td>
<td>2,461</td>
<td>5,980</td>
<td>107</td>
<td>260</td>
<td>Cluster10</td>
</tr>
<tr>
<td>Left-W-2</td>
<td>4</td>
<td>851</td>
<td>2,461</td>
<td>37</td>
<td>107</td>
<td>Cluster3</td>
</tr>
<tr>
<td>Left-W-3</td>
<td>36</td>
<td>2,001</td>
<td>2,714</td>
<td>87</td>
<td>118</td>
<td>Cluster5</td>
</tr>
<tr>
<td>Location</td>
<td>W-4</td>
<td>4</td>
<td>2,024</td>
<td>1,725</td>
<td>88</td>
<td>75</td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
<td>-----</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Left</td>
<td>W-5</td>
<td>6</td>
<td>851</td>
<td>1,817</td>
<td>37</td>
<td>79</td>
</tr>
<tr>
<td>Left</td>
<td>W-6</td>
<td>18</td>
<td>851</td>
<td>2,691</td>
<td>37</td>
<td>117</td>
</tr>
<tr>
<td>Left</td>
<td>W-7</td>
<td>4</td>
<td>1,978</td>
<td>2,001</td>
<td>86</td>
<td>87</td>
</tr>
<tr>
<td>Left</td>
<td>W-8</td>
<td>2</td>
<td>874</td>
<td>1,242</td>
<td>38</td>
<td>54</td>
</tr>
<tr>
<td>Left</td>
<td>W-9</td>
<td>4</td>
<td>1,817</td>
<td>2,760</td>
<td>79</td>
<td>120</td>
</tr>
<tr>
<td>Left</td>
<td>W-10</td>
<td>4</td>
<td>2,001</td>
<td>2,576</td>
<td>87</td>
<td>112</td>
</tr>
<tr>
<td>Left</td>
<td>W-11</td>
<td>4</td>
<td>2,415</td>
<td>2,806</td>
<td>105</td>
<td>122</td>
</tr>
<tr>
<td>Right</td>
<td>W-0</td>
<td>3</td>
<td>920</td>
<td>1,311</td>
<td>40</td>
<td>57</td>
</tr>
<tr>
<td>Right</td>
<td>W-1</td>
<td>36</td>
<td>2,001</td>
<td>2,737</td>
<td>87</td>
<td>119</td>
</tr>
<tr>
<td>Right</td>
<td>W-2</td>
<td>22</td>
<td>874</td>
<td>2,668</td>
<td>38</td>
<td>116</td>
</tr>
<tr>
<td>Right</td>
<td>W-3</td>
<td>6</td>
<td>2,369</td>
<td>1,863</td>
<td>103</td>
<td>81</td>
</tr>
<tr>
<td>Right</td>
<td>W-4</td>
<td>10</td>
<td>2,392</td>
<td>2,898</td>
<td>104</td>
<td>126</td>
</tr>
<tr>
<td>Right</td>
<td>W-5</td>
<td>1</td>
<td>3,795</td>
<td>2,852</td>
<td>165</td>
<td>124</td>
</tr>
<tr>
<td>Right</td>
<td>W-6</td>
<td>4</td>
<td>2,047</td>
<td>1,771</td>
<td>89</td>
<td>77</td>
</tr>
<tr>
<td>Right</td>
<td>W-7</td>
<td>5</td>
<td>897</td>
<td>1,725</td>
<td>39</td>
<td>75</td>
</tr>
<tr>
<td>Right</td>
<td>W-8</td>
<td>4</td>
<td>1,840</td>
<td>3,036</td>
<td>80</td>
<td>132</td>
</tr>
<tr>
<td>Right</td>
<td>W-9</td>
<td>4</td>
<td>2,070</td>
<td>2,001</td>
<td>90</td>
<td>87</td>
</tr>
<tr>
<td>Right</td>
<td>W-10</td>
<td>4</td>
<td>2,300</td>
<td>2,668</td>
<td>100</td>
<td>116</td>
</tr>
<tr>
<td>Right</td>
<td>W-11</td>
<td>4</td>
<td>2,047</td>
<td>2,530</td>
<td>89</td>
<td>110</td>
</tr>
<tr>
<td>Front</td>
<td>W-0</td>
<td>17</td>
<td>2,461</td>
<td>2,599</td>
<td>107</td>
<td>113</td>
</tr>
<tr>
<td>Front</td>
<td>W-1</td>
<td>207</td>
<td>2,024</td>
<td>2,668</td>
<td>88</td>
<td>116</td>
</tr>
<tr>
<td>Front</td>
<td>W-2</td>
<td>19</td>
<td>782</td>
<td>2,622</td>
<td>34</td>
<td>114</td>
</tr>
<tr>
<td>Front</td>
<td>W-3</td>
<td>40</td>
<td>2,093</td>
<td>1,978</td>
<td>91</td>
<td>86</td>
</tr>
<tr>
<td>Front</td>
<td>W-4</td>
<td>5</td>
<td>966</td>
<td>2,116</td>
<td>42</td>
<td>92</td>
</tr>
<tr>
<td>Front</td>
<td>W-5</td>
<td>29</td>
<td>2,415</td>
<td>2,806</td>
<td>105</td>
<td>122</td>
</tr>
<tr>
<td>Front</td>
<td>W-6</td>
<td>23</td>
<td>2,047</td>
<td>2,507</td>
<td>89</td>
<td>109</td>
</tr>
<tr>
<td>Front</td>
<td>W-7</td>
<td>2</td>
<td>736</td>
<td>1,541</td>
<td>32</td>
<td>67</td>
</tr>
<tr>
<td>Front</td>
<td>W-8</td>
<td>6</td>
<td>2,001</td>
<td>1,679</td>
<td>87</td>
<td>73</td>
</tr>
<tr>
<td>Back</td>
<td>W-0</td>
<td>216</td>
<td>2,047</td>
<td>2,645</td>
<td>89</td>
<td>115</td>
</tr>
<tr>
<td>Back</td>
<td>W-1</td>
<td>13</td>
<td>2,208</td>
<td>2,093</td>
<td>96</td>
<td>91</td>
</tr>
<tr>
<td>Back</td>
<td>W-2</td>
<td>12</td>
<td>2,507</td>
<td>2,599</td>
<td>109</td>
<td>113</td>
</tr>
<tr>
<td>Back</td>
<td>W-3</td>
<td>12</td>
<td>2,116</td>
<td>2,001</td>
<td>92</td>
<td>87</td>
</tr>
<tr>
<td>Back</td>
<td>W-4</td>
<td>11</td>
<td>2,392</td>
<td>2,875</td>
<td>104</td>
<td>125</td>
</tr>
<tr>
<td>Back</td>
<td>W-5</td>
<td>26</td>
<td>2,116</td>
<td>2,553</td>
<td>92</td>
<td>111</td>
</tr>
<tr>
<td>Back</td>
<td>W-6</td>
<td>6</td>
<td>2,024</td>
<td>1,725</td>
<td>88</td>
<td>75</td>
</tr>
<tr>
<td>Back</td>
<td>W-7</td>
<td>5</td>
<td>2,254</td>
<td>2,323</td>
<td>98</td>
<td>101</td>
</tr>
<tr>
<td>Back</td>
<td>W-8</td>
<td>16</td>
<td>2,346</td>
<td>2,645</td>
<td>102</td>
<td>115</td>
</tr>
</tbody>
</table>

Notes: *Italic* indicates the main types of windows in the MCS.  
px = pixel.
<table>
<thead>
<tr>
<th>Cluster</th>
<th>Number of observations</th>
<th>Cluster centroids</th>
<th>Comment</th>
<th>Window type</th>
<th>Corrected num.</th>
<th>Dimension, centroid × GSD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>w (px) h (px)</td>
<td></td>
<td></td>
<td></td>
<td>W (mm) H (mm)</td>
</tr>
<tr>
<td>Cluster1</td>
<td>13</td>
<td>37 76</td>
<td>Narrow windows around 16,000 and 55,000 mm</td>
<td>WN-1</td>
<td>14</td>
<td>851 1,748</td>
</tr>
<tr>
<td>Cluster2</td>
<td>5</td>
<td>42 92</td>
<td>Two narrow windows around 55,000 mm in the front façade; three windows belong to Cluster 3</td>
<td>WN-2</td>
<td>2</td>
<td>966 2,116</td>
</tr>
<tr>
<td>Cluster3</td>
<td>63</td>
<td>36 115</td>
<td>Long and narrow windows; also include two skipped windows in the front façade</td>
<td>WN-3</td>
<td>68</td>
<td>828 2,645</td>
</tr>
<tr>
<td>Cluster4</td>
<td>80</td>
<td>90 83</td>
<td>Nearly square windows, just above/below the main types of windows</td>
<td>WN-4</td>
<td>92</td>
<td>2,070 1,909</td>
</tr>
<tr>
<td>Cluster5</td>
<td>552</td>
<td>88 115</td>
<td>The main types of windows between 18,000 and 55,000 mm; two exceptions belong to Cluster 6</td>
<td>WN-5</td>
<td>550</td>
<td>2,024 2,645</td>
</tr>
<tr>
<td>Cluster6</td>
<td>107</td>
<td>105 119</td>
<td>101 windows located on the top of two rows (higher than 60,000 mm), another six windows in the bottom of the right façade, and one removed window in the back façade</td>
<td>WN-6-1</td>
<td>110</td>
<td>2,415 2,737</td>
</tr>
<tr>
<td>Cluster7</td>
<td>18</td>
<td>97 94</td>
<td>12 windows belong to Cluster 4, and six windows belong to Cluster 6</td>
<td>WN-6-2</td>
<td>6</td>
<td>2,415 2,737</td>
</tr>
<tr>
<td>Cluster8</td>
<td>5</td>
<td>39 56</td>
<td>The smallest four windows in the left and right façades around 65,000 mm; the one exception belongs to Cluster 1</td>
<td>WN-8</td>
<td>4</td>
<td>897 1,288</td>
</tr>
<tr>
<td>Cluster9</td>
<td>8</td>
<td>80 126</td>
<td>Wide double hung windows around 12,000 mm in the left and right façades</td>
<td>WN-9</td>
<td>8</td>
<td>1,840 2,898</td>
</tr>
<tr>
<td>Cluster10</td>
<td>7</td>
<td>107 260</td>
<td>The largest window in the left façade</td>
<td>WN-10</td>
<td>7</td>
<td>2,461 5,980</td>
</tr>
<tr>
<td>Cluster11</td>
<td>6</td>
<td>103 81</td>
<td>Fanlights in the right façade</td>
<td>WN-11</td>
<td>6</td>
<td>2,369 1,863</td>
</tr>
<tr>
<td>Cluster12</td>
<td>1</td>
<td>165 124</td>
<td>The largest fanlight in the right façade</td>
<td>WN-12</td>
<td>1</td>
<td>3,795 2,852</td>
</tr>
<tr>
<td>Total</td>
<td>865</td>
<td></td>
<td></td>
<td></td>
<td>868</td>
<td></td>
</tr>
</tbody>
</table>

Note: px = pixel.
Cluster5 has 552 observations, including 495 of the main types of windows and 57 windows from Left-W-10, Right-W-11, Front-W-6, and Back-W-5 in Table 2. These 57 windows are shorter than the main types of windows because their lintels partially hid them during image capturing (the camera positions shown in Fig. 12 indicate that the camera is always at a downward angle in relation to the MCS). In addition, Fig. 17 also indicates that most Cluster5 observations have elevations between 18,000 and 55,000 mm; the two exceptions are Back-W-5-273 and 274 [Fig. 14(b)], which are distorted windows and should belong to Cluster6.

![Fig. 17. Shape clustering results versus elevations.](image)

Cluster6 has 107 observations; 101 of them have elevations higher than 60,000 mm, including 4 of Left-W-0, 4 of Left-W-11, 4 of Right-W-4, 4 of Right-W-10, 17 of Front-W-0, 29 of Front-W-5, 12 of Back-W-2, 11 of Back-W-4, and 16 of Back-W-8. The six exceptions are in the bottom of the right façade as Right-W-4-96, 97, 98, 99, 100, and 101 in Fig. 11, which should be considered as a different window type.

Cluster4 has 80 observations; 60 of them are around 55,000 mm, which include 4 of Left-W-7, 4 of Right-W-9, 40 of Front-W-3, and 12 of Back-W-3. The other 20 windows are around 16,000 mm, including 4 of Left-W-4, 4 of Right-W-6, 6 of Front-W-8, and 6 of Back-W-6. The 18 observations of Cluster7 are all more than 55,000 mm, including 13 of Back-W-1 and 5 of Back-W-7. Back-W-1-30 and 5 Back-W-7 are distorted windows in the mesh model, which should be the same as Back-W-8 of Cluster6; the other 12 of Back-W-1 should have the same size as Back-W-3 of Cluster4.

Cluster3 has 63 observations, which are long and narrow windows, including 4 of Left-W-2, 18 of Left-W-6, 22 of Right-W-2, and 19 of Front-W-2. The five observations of Cluster2 are Front-W-4, while Front-W-4-2, 22, and 23 are different from Front-W-4-9 and 10 (around 55,000 mm) in Fig. 14(a). In addition, in the front façade [Fig. 14(a)], the skipped two windows and Front-W-4-2, 22, and 23 should belong to Front-W-2 of Cluster3 as well.

The 13 observations of Cluster1 include 6 of Left-W-5, 5 of Right-W-7, and 2 of Front-W-7. The five observations of Cluster8 include two of Left-W-8 and three of Right-W-0, which are the smallest windows on the top of the left and right façades; however, window Right-W-0-2 should be Right-W-7 of Cluster1.

The eight observations of Cluster9 include four of Left-W-9 and four of Right-W-8, which are wide double hung windows. The seven observations of Cluster10 are Left-W-1, which are the largest windows in the bottom of the left façade. The six observations of Cluster11 are Right-W-3, which are
the bottom fanlights of the right façade. Cluster12 is the largest fanlight, window Right-W-5-102, in Fig. 11.

Therefore, based on the aforementioned discussions, classifying the extracted 865 windows into 12 categories is reasonable, which puts similar windows into the correct group regardless of whether the windows are in different façades. The corrected 868 windows (including the two discarded windows on the front façade and the removed one on the back façade) have the types and numbers listed in Table 3.

Comparison of Point Cloud and Mesh Model–Based Orthoimage

The aforementioned sections mainly discussed the photogrammetric mesh model–based façade orthoimage in window extraction and classification. Although several manual processes are required in façade orthoimage generation, the accuracy is worth those extra efforts. This section compares photogrammetric point cloud and mesh model–based façade orthoimages for further improving the automatic level of the proposed method with both types of orthoimages.

In Fig. 18, a right façade orthoimage was generated from the pointcloud2orthoimage tool with a designated GSD = 23 mm/pixel; photogrammetric point cloud and code are available in Jiang (2022). Then, the extracted façade elements in orthoimage pixel coordinates could be accurately converted to the real coordinates without doubt. Moreover, to automatically obtain an accurate GSD for a mesh model–based façade orthoimage, attaching known sized marks, for example, a drone landing pad, to the scanned façades is recommended. Then, an additional U-net model would be used to detect the marks from the façade orthoimages and return the marks’ pixel size for updating the GSDs for each individual façade orthoimage, like in Han et al. (2022) and Jiang and Bai (2021).

Compared with the vividly textured mesh model–based façade orthoimage [Fig. 11(a)], the photogrammetric point cloud–based orthoimage is blurry, as shown in the detailed window example of Fig. 18(b). Even with the blurred orthoimage, the trained U-net successfully detected most windows
and only skipped several windows on the bottom. Given more photogrammetric point cloud–based orthoimages and labels in U-net model training, the segmentation accuracy will be enhanced without doubt. In addition, the proposed sawtooth() algorithm (Fig. 7) successfully trimmed the U-net-generated label and returned the sawtooth labels shown in Fig. 18(c), where small pieces of noise were broken into fragile pieces on the left margin and upper margin of the façade orthoimage. Consequently, 95 of 103 windows were accurately extracted, classified, and aligned over the point cloud–based orthoimage, compared with the mesh model–based orthoimage in Fig. 11. Future practice may attempt to fly the drone closer to the building facades and add more front-facing-view images in 3D reconstruction to increase point cloud density, considering that this research used only 90 images, and each pixel occurred in 3.3 images on average, with a GSD about 35 mm/pixel.

Recommendations for Further Improving Automatic Level

This section discusses the currently unsolved manual intervention processes in the developed method, while also discussing potential approaches to enhance the overall automatic level of the developed method.

The floor plan is a useful and necessary reference to place exterior walls in Revit. In case of missing the original floor plan, inserting either mesh model section or point cloud section into AutoCAD to manually trace exterior walls would be the solution, in which the offset should be considered as half the thickness of the walls. In addition, the previous developed indoor structures modeling method would be considered to automatically extract walls as a polygon (Jung et al. 2016), and then applying the offset would help obtain the central line of the exterior walls as well. However, the photogrammetric point cloud is not as dense and accurate as the ground LiDAR point cloud, which may limit the performance of automatic wall extraction. For example, the distortion on windows resulted in walls that are not straight lines, as shown in the point cloud section of Fig. 12(c). Future research needs to find a method, for example, clustering, to automatically generate typical floor plans for high-rise buildings using multiple sections (floors) of the photogrammetric point cloud.

Moreover, if the building floor plan is not in the shape of a rectangle and has more than the vertical (left and right) and horizontal (front and back) edges in the floor plan, then additional façade orthoimages should be created for each oblique edge on the floor plan. In addition, along the point cloud–based orthoimage generation, a relative depth image could be created with the same GSD; see example in Jiang (2022). The depth image serves as the floor plans for the entire façade, which will benefit the detection and location of protruding components such as columns and decorations. Then, additional orthoimages might be created from the point cloud section to detect the blocked windows.

Additionally, the automatic level of modeling can be further enhanced by transforming the 2D real coordinates to 3D as \((X', Y', \text{elevation})\), in which \((X', Y')\) is a façade element’s projected location. Then, the process of manual registration of façades (elevations) and a floor plan can be skipped. The 3D as-is wireframe model (i.e., polyline) or 3D solid model (i.e., polysolid) can be drafted in AutoCAD with scripts automatically. In addition, saving extracted parameters to the industry foundation classes (IFC) format would save much time and effort in drafting in Revit as well. Furthermore, this research used only windows as an example by switching the target element to others, for example, façade cracks (Chen et al. 2021b), and then façade inspection results would be mapped to the as-is BIM model automatically.
Conclusion
This paper presented an automated as-is façade modeling method, Scan4Façade, for existing and historic high-rise buildings using drone photogrammetry and artificial intelligence (AI). The following developed algorithms, proposed functions, and strategies were fully tested on the office tower of the Michigan Central Station. The significant findings of this research are as follows:

1. The proposed combined DA strategy simplified the manually labeled work and enhanced AI model training efficiency. As the results show, the performance of the U-net pixelwise segmentation improved in distorted, perspective-transformed, and shaded façade element detection, even though the façade elements were from other buildings and shown in arbitrary views. By conducting DA, the U-net achieved an average pixel accuracy of 0.9696 and a mean IoU of 0.9063 (a nonwindow IoU of 0.9614 and a window IoU of 0.8512) in testing.

2. The developed sawtooth() algorithm and twoStageClustering() algorithm could accurately extract the shape and coordinates for each façade element based on façade orthoimages and pixelwise labels. It was quick in processing high-rise buildings and achieved an accuracy of 99.77% in window extraction.

3. The proposed coordinateConversion() function and writeCommand2File() function could automatically obtain information on façade element geometry, save coordinates, dimensions, and types into AutoCAD command and script files, and draft 2D as-is CAD drawings without manual interaction.

Moreover, the Scan4Façade method provides clear and accurate information to create BIM features, for example, layout façade elements on exterior walls, by tracing the automatically generated CAD drawings in Revit. It can save much time and effort in drafting BIM models and is better than the common practice of tracing point clouds to create BIM features. In addition, separate AI models can be trained and used to detect different types of façade defections such as cracks and spalls, and then, these observed façade defections can be mapped to the as-is drawings and BIM models, utilizing the same method as modeling the windows. With hundreds and thousands of high-rise buildings in the United States and around the world, the Scan4Façade method has huge commercial potential because of the deterioration of buildings and infrastructures and the increased demand for building façade inspection and building renovation.

Data Availability Statement
The following data or code generated or used during the study are available in a repository or online:
The training and testing image and label are available in Jiang (2021b). The aerial images of the Michigan Central Station can be accessed via the link in Pix4D (2018). The pointcloud2orthoimage code and demonstration data are available in Jiang (2022) and Jiang et al. (2022b). The U-net code is available in Chollet (2020b) and Zhi (2019).

The following codes generated or used during the study are proprietary or confidential in nature and may only be provided with restrictions: The developed sawtooth() and twoStageClustering() algorithms are presented as pseudocodes in this paper because of the process of patent application.
Acknowledgments
The deep learning workstation was financially supported by the McShane Endowment Fund at Marquette University. The authors are grateful to the reviewers for their valuable comments and feedback.

References


ClearEdge3D. 2021c. “Case study: University science building as-built revit model—EdgeWise building delivers a 55% workflow improvement for scan-to-revit modeling.” Accessed February 20,


