BIM Voxelization Method Supporting Cell-Based Creation of a Path-Planning Environment

Qiankun Wang[1]; Weiwei Zuo, Ph.D.[2]; Zeng Guo, Ph.D.[3]; Qianyao Li[4]; Tingting Mei, Ph.D.[5]; and Shi Qiao, Ph.D.[6]

Abstract: Cell-based path planning is studied and applied worldwide. Building information models (BIMs) can be used as the environment for path planning. However, studies providing an applicable and detailed approach to converting a BIM to cells are rarely reported. This study proposes a voxel-based method that can automatically convert a BIM to cells. This technique, verified for accuracy and applicability, can quickly and conveniently create a cell-based path-planning environment based on a given BIM before testing and applying path-planning methods. With this method, researchers and practitioners can focus more on research and the application of path planning instead of the creation of a cell-based environment, which is time-consuming and tedious. **DOI: 10.1061/(ASCE)CO.1943-7862.0001864.** *© 2020 American Society of Civil Engineers.*

Author keywords: Building information modeling; Cell-based path planning; Space discretization; Voxelization introduction.

Background

Numerous studies have been conducting path planning, which has been widely applied in the architectural, engineering, and construction (AEC) industry. Research related to path planning includes construction vehicle moving path simulation (Zhang et al. 2007), in-site material transport planning (Park et al. 2012; Wang et al. 2019), indoor moving-path navigation (Lin et al. 2013; Yuan and Schneider 2010), evacuation simulation and evaluation (Cheng et al. 2018; Marzouk and Daour 2018), and equipment disassembly simulation (Tan et al. 2017). Prior to a path-planning analysis, the path-planning space must be properly depicted, which can be addressed by a cell-based representation. A cell-based representation is an environment consisting of numerous cells. It can suitably simulate and analyze the two-dimensional (2D) or near 2D moving patterns of trucks, mobile cranes, and people (ElNimr et al. 2016; Park et al. 2012; Zhang et al. 2007). By distributing the cells in a three-dimensional (3D) space, the 3D moving patterns can also be depicted, including laborers' stair climbs and materials being transported by elevators or tower cranes (Lin et al. 2013; Wang et al. 2019). In addition, by appending semantic information to cells, other tasks can be supported, such as pedestrian simulation (Hsu and Chu 2014) and a fire-evacuation analysis (Shi et al. 2009), or other nonpath-planning tasks, such as workspace planning (Elmahdi et al. 2011) and workspace soft clash detection (Moon et al. 2014).

To create a cell-based path-planning environment, two tasks have to be completed: a cell allocation and cell navigability test. The former allocates cells in the path-planning space, whereas the latter determines if a cell can be navigated by a moving object. Voxelization, which is a common method for approximating 3D objects with cubics (voxels), can potentially create a cell-based path-planning environment because when voxelization finishes, the upper face of each voxel forms a cell (Fig. 2). In addition, voxelization can potentially facilitate the cell navigability test because navigability depends on whether a cell is intersected by obstacles and whether the voxel-based intersection test is efficient (Echegaray and Borro 2012). Building information models (BIMs), which contain both physical and functional information in 3D or nD digital models, provide a preferable virtual environment for path planning. However, in most situations, the models are not created in a voxel-based manner. Therefore, an applicable method to automatically convert BIM models to voxels has to be developed. With this method, researchers and practitioners can focus more on the research and the application of path planning instead of the creation of a cell-based environment, which is time-consuming and tedious.

Related Research

Research on voxelization is mainly conducted in computer graphics (Stolte and Kaufman 2001; Yang et al. 2017a; Zhang et al. 2018; Zhao et al. 2004). In the AEC industry, voxel-assisted techniques are also widely-applied, including 3D point cloud processing (Krijnen and Beetz 2017; Shirowzhan et al. 2018), structure recognition (Sun et al. 2018), a morphological analysis of building component materials (Yang et al. 2017b), and a finite-element analysis (Castellazzi et al. 2015). These studies use two voxel generation

[1]Professor, School of Civil Engineering and Architecture, Wuhan Univ. of Technology, Wuhan City, Hubei Province 430070, China. Email: wangqk@whut.edu.cn

[2]School of Civil Engineering and Architecture, Wuhan Univ. of Technology, Wuhan City, Hubei Province 430070, China. Email: 542814313@qq.com

[3]Lecturer, School of Civil Engineering and Architecture, Wuhan Univ. of Technology, Wuhan City, Hubei Province 430070, China (corresponding author). ORCID: https://orcid.org/0000-0001-6633-0035. Email: zeng.guo@qq.com

[4]Ph.D. Candidate, School of Civil Engineering and Architecture, Wuhan Univ. of Technology, Wuhan City, Hubei Province 430070, China. Email: qianyao.li@qq.com

[5]Lecturer, School of Civil Engineering and Architecture, Wuhan Institute of Technology, Wuhan City, Hubei Province 430205, China. Email: 22243571@qq.com

[6]Post-Doctor, School of Civil Engineering and Architecture, Wuhan Univ. of Technology, Wuhan City, Hubei Province 430070, China. Email: dinojoe@whut.edu.cn

Note. This manuscript was submitted on March 22, 2019; approved on January 16, 2020; published online on May 11, 2020. Discussion period open until October 11, 2020; separate discussions must be submitted for individual papers. This paper is part of the ***Journal of Construction Engineering and Management***, © ASCE, ISSN 0733-9364.

strategies: identical-sized voxelization and self-adapting voxelization (Fig. 1 and Table 1).

The identicalsized voxelization strategy generates voxels with an identical size to represent the voxelized objects/spaces. Voxels with identical sizes can be generated using two methods: surface-based voxelization and slice-based voxelization. In surface-based voxelization, the surface of the objects is first voxelized and then the generated voxels are used to define others. For instance, to extract the free indoor spaces of a CityGML model, Xiong et al. (2016) voxelized the surface of these components and passed semantic information from the surface voxels to their neighbors to cluster them as various indoor spaces. Sun et al. (2018) proposed a compositional building structural recognition method that also starts with building surface voxelization. To voxelize a boundary-representation model, Young and Krishnamurthy (2018) proposed a method in which the boundary of the model is first divided into triangle meshes, followed by the application of a two-level process to identify the boundary and inner voxels. In the slice-based method, the objects are sliced into sections; voxels are first generated on those sections and then merged as the voxelized object. Bandi and Thalmann (1998) proposed a method that uses several horizontal planes equidistant to one another to cut the 3D environment (model).

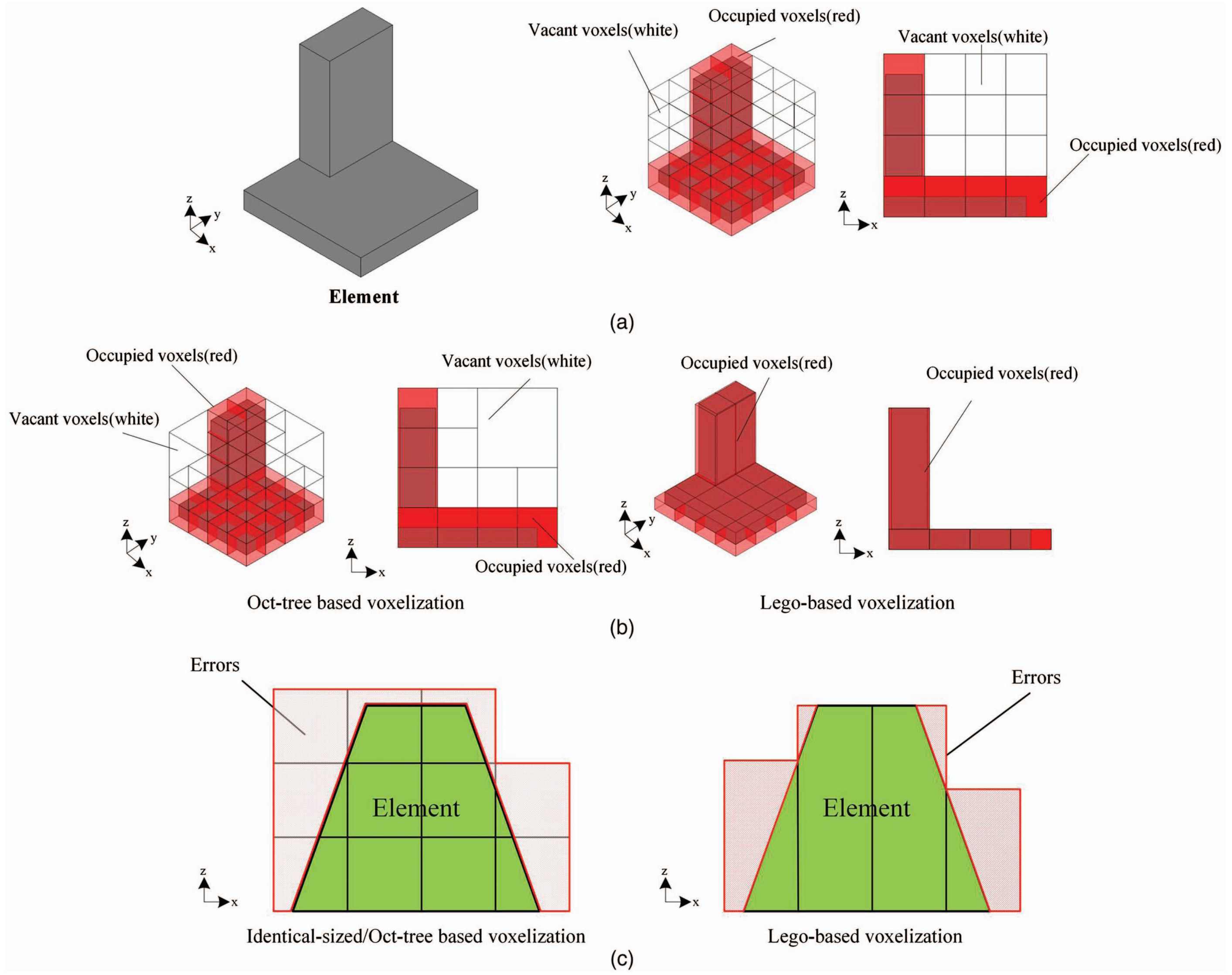


Fig. 1. (Color) Identical-size and self-adapting representation: (a) identical-sized voxelization strategy; (b) self-adapting voxelization strategy; and (c) voxelization errors of the identical-sized/oct-tree based voxelization versus LEGO-based voxelization.

Table 1. Comparison of voxelization strategies

	Voxel width and depth		Voxel height	
Voxelization strategy	Occupied voxels	Vacant voxels	Occupied voxels	Vacant voxels
Identical-sized voxelization	Identical	Identical	Identical	Identical
Self-adapting voxelization				
Octree-based	Identical	Changeable	Identical	Changeable
LEGO-based	Identical	No voxel	Changeable	No voxel

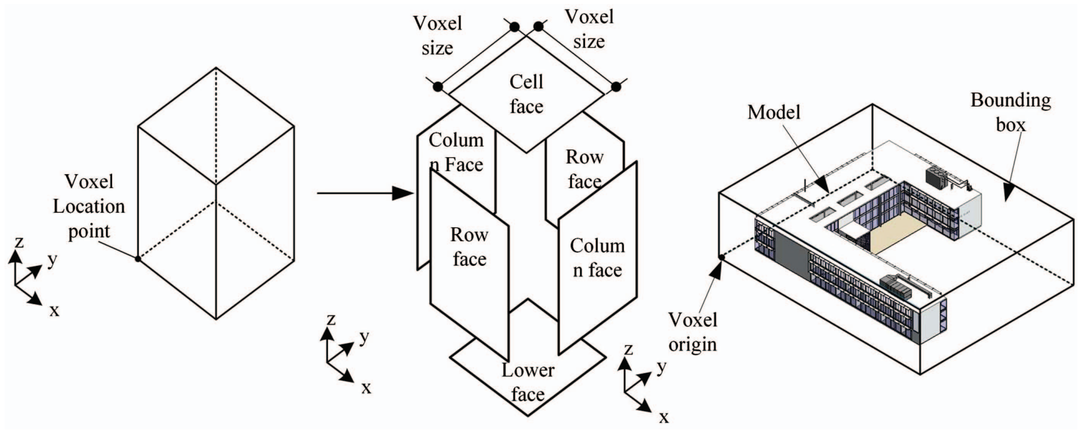


Fig. 2. (Color) Some parameters of a LEGO-based voxel.

The sliced volumes are then projected to the horizontal planes with grid cells. If some cells are covered by the projections, they are either unnavigable or surmountable depending on the elevation of the volumes. Although not explicitly stated in this method, the combination of the cell and the distance between the planes determine a voxel.

The self-adapting voxelization strategy generates voxels by considering the size of the voxelized objects. The voxels generated during voxelization are not identical in size. Self-adapting voxel methods come in two forms: the octree-based method and the LEGO-based method. The octree-based method adopts octrees to voxelize objects/models. An octree is a hierarchical tree structure with each of its internal nodes consisting of eight children and is used to partition a 3D space by recursively subdividing it into eight octants (Chen and Huang 1985). The advantage of octree-based voxelization is that it is efficient in voxelizing a not fully-occupied 3D space because the vacant space can be quickly excluded from further voxelization. This method has also been used in some studies for voxelization: Bischoff et al. (2005) used a hierarchical octree data structure to generate voxels to assist polygon model restoration; de Queiroz and Chou (2016) used an octree to voxelize point-cloud data. However, the disadvantage of this method is that the neighboring relationship of different voxels is hard to determine because of the tree structure. To address this concern, Mu et al. (2010) proposed a pedigree coding method that can rapidly determine the neighboring relationship of two voxels from an octree. The LEGO-based method divides the objects/models into LEGOs. A LEGO is a building block developed by the LEGO company in Denmark (Min et al. 2018). Yuan and Schneider (2010) proposed a LEGO-based method for indoor path planning. They divided the indoor spaces into LEGO blocks and considered the upper face of the LEGOs as the cells for navigation. The advantage of this method is that the heights of the LEGOs are self-adapting, and a 3D element can contain considerably fewer LEGO voxels, compared with the equal-sized or octree-based techniques, facilitating calculation in path planning.

Summary

Current research provides valuable voxelization strategies applicable in various areas. However, the aforementioned strategies have some limitations in representing a path-planning environment: the identical-sized voxels cannot fit the objects or models well unless the size of each individual voxel is reduced, which then leads to an increase in the number of voxels and extends the elapsed time for voxelization. The octree-based method from the self-adapting voxelization strategy can efficiently detect voxels that are not occupied by any element within a few iterations; however, the method is not efficient enough to detect voxels occupied by elements because the iteration does not stop until the voxel size reaches its minimum. The LEGO-based approach is suitable for representing a path-planning environment because it can approximate the element more accurately than the two aforementioned approaches, as shown in Fig. 1(c); however, neither an executable and detailed approach to the LEGO-based voxelization of BIM models nor the performance of this method is introduced. To address the limitations in a LEGO-based voxelization, this study elaborates on the voxelization algorithm in the context of BIM and conducts tests to verify the efficiency and applicability of the method.

The structure of this paper is arranged as follows: after the introduction of the background and related research in the introduction part, a four-step BIM model voxelization method is elaborated in the "BIM Model Voxelization Method" section. A validation test is then conducted to test the correctness, speed, and applicability of the proposed method in the "Validation Test" section, and the results of the test are illustrated and discussed in the "Result and Discussion" section. Finally the conclusion, contribution, and limitation of this study are summarized in the "Conclusion" section.

BIM Model Voxelization Method

Given a right-handed Cartesian coordinate system whose x-, y-, and z- axis points are forward, right, and upward, respectively, a LEGOlized voxel is an axis-aligned bounding box (AABB) with a fixed width (x-scale), depth (y-scale), and variable height (z-scale). To represent the location of the voxel, several parameters are introduced: (1) the location point, which is the voxel vertex with minimum x, y, and z (left, Fig. 2); (2) the column faces, which are the two voxel faces perpendicular to the x-axis (middle, Fig. 2); (3) the row faces, which are the two voxel faces perpendicular to the y-axis (middle, Fig. 2); (4) the cell face, which is the upper face of the voxel representing a cell for path planning (if the cell can be used by the moving object); (5) the lower face, which is the bottom face of the voxel (middle, Fig. 2); and (6) the voxel origin, which is the location of the first voxel, and is often set as the minima of the AABB enveloping the entire BIM models (right, Fig. 2). To facilitate the voxelization process, each column face and row face is indexed by an integer. Given a voxel vox with the location point $lc(x_{lc}, y_{lc}, z_{lc})$, the voxel origin is $o(x_o, y_o, z_o)$, and the cell size is cs; therefore, the index of the column and row face intersecting at lc is calculated using Eq. (1), as follows:

$$ci = \left\lfloor \frac{x_{lc} - x_o}{cs} \right\rfloor$$
$$ri = \left\lfloor \frac{x_{lc} - x_o}{cs} \right\rfloor \quad (1)$$

where ci = index of the aforementioned column face; and ri = index of the aforementioned row face.

A voxel also has an index. The voxel index is determined by the face indexes of the column and row faces intersecting at the location point. For example, the aforementioned voxel vox has an index denoted by $[ci, ri]$. The first element of the voxel index is called a column index, and the second element is called a row index.

Given a BIM model consisting of numerous elements, the voxelization of the elements is conducted one by one. The method of voxelizing a BIM element consists of four steps: element solid distraction, solid face triangulation, triangle mesh voxelization, and mesh voxel merging (Fig. 3).

Element Solid Distraction

In many BIM applications, the space occupation of their elements are represented by solids, and the solids can be accessed by the corresponding application program interfaces (APIs). In this step, the solids of a given element are directly obtained by APIs for further process.

Solid Face Triangulation

After the solids are generated from the given elements, the surface of each solid is converted into triangle meshes. A triangle mesh is a 3D mesh consisting of numerous triangular facets. In computer graphics, triangle meshes can represent the approximation of complex surfaces (Dunn and Parberry 2005). Some studies cited in "Related Research" used triangle meshes to generate voxels (Sun et al. 2011; Xiong et al. 2016; Young and Krishnamurthy 2018). In the present study, triangle meshes are also used as inputs to voxelize the corresponding solid. Some 3D computer-aided designs or BIM software applications provide built-in functions for triangulation and thus can be applied directly for this task.

In this study, each triangular facet from the triangle mesh exhibits two properties: (1) the coordinates of the three vertices, and (2) a normal of the facet pointing outward from the corresponding solid. The first property can be obtained after triangulation, but the second property cannot always be obtained. In this situation, the face hosting the meshes can be used to determine the normal. Given a triangular facet with three vertices—$v_1(x_1, y_1, z_1)$, $v_2(x_2, y_2, z_2)$, and $v_3(x_3, y_3, z_3)$—its normal is determined as follows. First, a cross product between the vectors $v_{12} = v_2 - v_1$ and $v_{13} = v_3 - v_1$ is created. The result (vec_{cp}) is a vector orthogonal to the triangle with the direction either the same as or opposite that of the correct triangle normal. Second, the coordinate of the triangle centroid is calculated. Third, the centroid is projected to the face, and the normal of the projection (n_p) is determined by using the related API function—for instance, the project method of Revit API version 2017. Finally, the dot product between n_p and vec_{cp} is calculated. If the value is larger than 0, which means that the angle between n_p and vec_{cp} is less than $\pi/2$, vec_{cp} is the correct facet normal; otherwise, the inverse of vec_{cp}, which is $-1 \times \text{vec}_{cp}$, is the correct facet normal.

Triangle Mesh Voxelization

This step, which converts the triangle mesh of each solid to voxels, consists of two substeps: grid-point generation and voxel generation.

A Grid-Point Generation

A grid point (GP) is the potential intersection between a voxel and a triangular facet. It determines the location and shape of the voxel. There are four types of GPs: the vertex GP (VGP), column

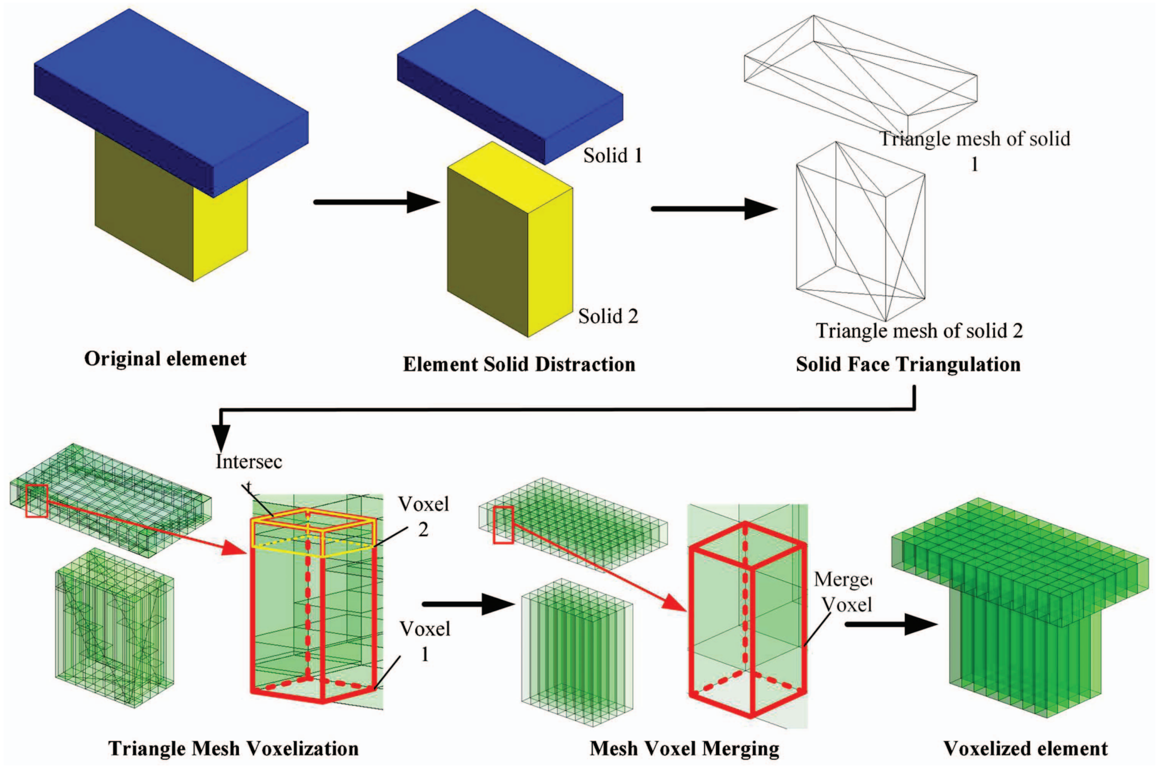


Fig. 3. (Color) Process of element voxelization.

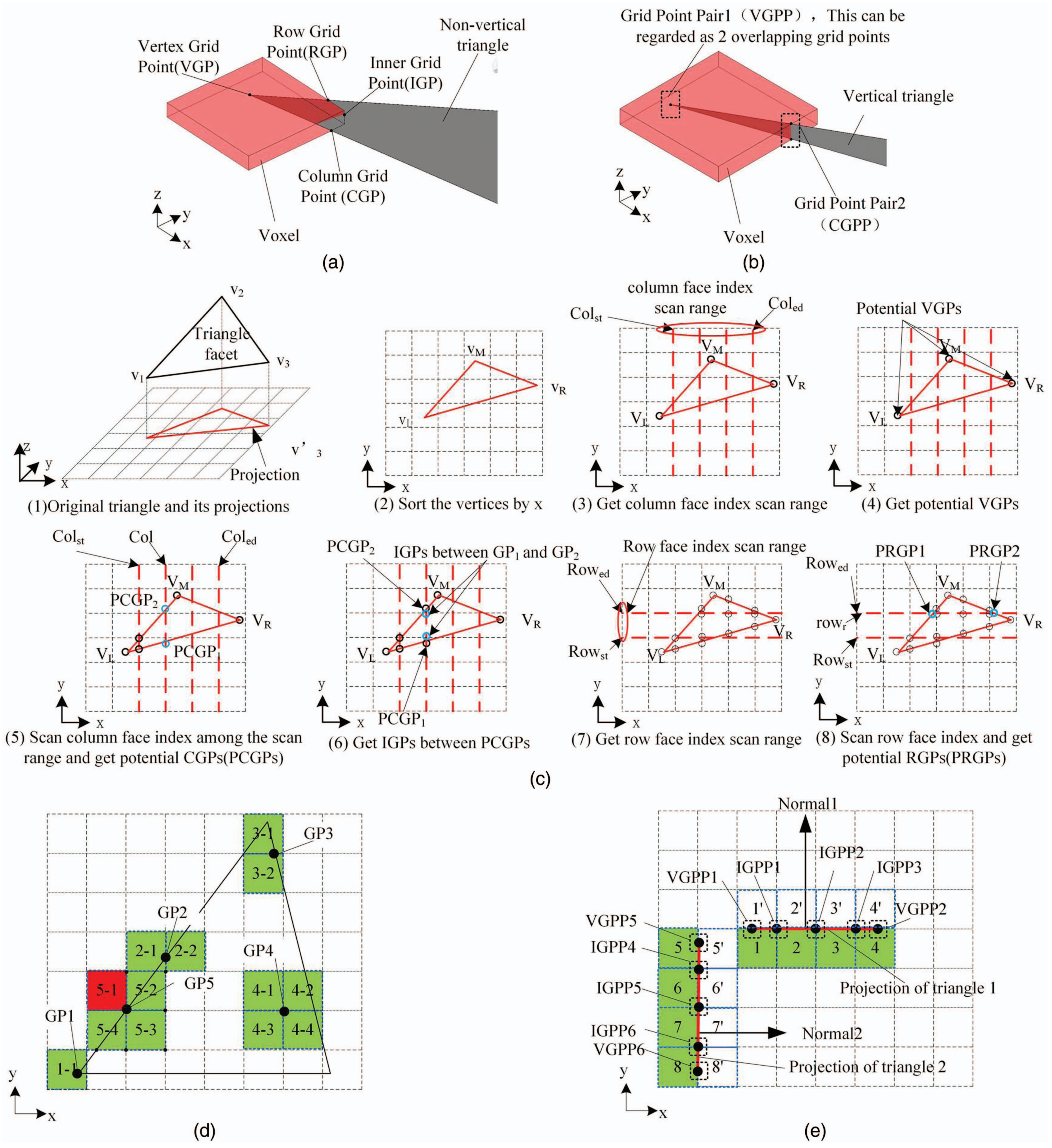

Vertex Grid Point(VGP)
Row Grid Point(RGP)
Non-vertical triangle
Inner Grid Point(IGP)
Column Grid Point (CGP)
Voxel
(a)
Grid Point Pair1（VGPP），This can be regarded as 2 overlapping grid points
Vertical triangle
Grid Point Pair2（CGPP）
Voxel
(b)
(1)Original triangle and its projections
(2) Sort the vertices by x
(3) Get column face index scan range
(4) Get potential VGPs
(5) Scan column face index among the scan range and get potential CGPs(PCGPs)
(6) Get IGPs between PCGPs
(7) Get row face index scan range
(8) Scan row face index and get potential RGPs(PRGPs)
(c)
GP1
GP2
GP3
GP4
GP5
(d)
Normal1
Normal2
Projection of triangle 1
Projection of triangle 2
(e)

Fig. 4. (Color) Triangle-based element voxelization: (a) grid points on nonvertical triangle; (b) grid points on vertical triangle; (c) grid point generation for nonvertical triangle facets; (d) determining the affection range of each GPs; and (e) affection range of each GPs of vertical triangle in odd situations.

GP (CGP), row GP (RGP), and inner GP (IGP) [Fig. 4(a)]. A VGP is the intersection between one of the facet vertices and the voxel; a CGP is the intersection between one column face of a voxel and an edge of the facet; an RGP is the intersection between one row face of a voxel and an edge of the facet; and an IGP is the intersection of a column face and a row face of a voxel and the facet. A GP may have multiple types. However, in this study, a GP is allowed to have only one type. The GP type with x and y values of x_{gp} and y_{gp}, respectively, is listed in Table 2.

Table 2. Grid type of a vertex grid with multiple types

	x_{gp}-x_o	
y_{gp}-y_o	Divisible by s	Not divisible by s
Divisible by s	IGP	RGP
Not divisible by s	CGP	VGP

Note: x_o, y_o = the x and y coordinates of the origin of the voxel space; s = the voxel size; IGP = inner grid point; RGP = row grid point; CGP = column grid point; and VGP = vertex grid point.

```
Input:
(1) The 3 vertice of the triangle facet v1($x_1$,$y_1$,$z_1$), v2($x_2$,$y_2$,$z_2$) and v3($x_3$,$y_3$,$z_3$)
(2) Voxel origin:o($x_o$,$y_o$,$z_o$)
(3) Cell size: cs
BEGIN
    Create a collection $GP_{coll}$ to collect all the GPs;
    Sort v1,v2 and v3 by x coordinate as
    vLeft($x_L$,$y_L$,$z_L$), vMid($x_M$,$y_M$,$z_M$), vRight($x_R$,$y_R$,$z_R$)
    where $x_L \le x_M \le x_R$;
    Get the column face index ranging from $col_{st}$ to $col_{ed}$ by equation(2)
    as is shown in (3) of Fig. 4(c);
    FOR EACH v in {$v_{Left}$,$v_{Mid}$,$v_{Right}$} DO
        Assign the type of v by Table 2;
        Add v to $GP_{coll}$;
    END
    FOR column face index col from colst to coled DO
        Get the Coordinates of the 2 potential CGPs
        PCGP1（$x_{c1}$,$y_{c1}$,$z_{c1}$），PCGP2（$x_{c2}$,$y_{c2}$,$z_{c2}$）
        on the column face col by Equation (3) and (4);
        For EACH PCGP in{PCGP1,PCGP2} Do
            Assign the type of PCGP by Table 2;
            If PCGP is IGP And PCGP $\notin$GPcoll THEN
                Add PCGP as IGP to GPcoll;
            ELSE
                Add PCGP as CGP to GPcoll;
            END
        END
        Get the row face index between PCGP1 and PCGP2 ranging from rowst to rowed by equation (5);
        FOR EACH row face index rowi from $row_{st}$ to $row_{ed}$ DO
            Get the IGP ($x_i$, $y_i$, $z_i$) on the row faces indexed rowi by equation (6);
            Add the IGP to $GP_{coll}$;
        END
    END
    Sort $v_1$,$v_2$ and $v_3$ by y coordinate as
    $v_{Bottom}$($x_B$,$y_B$,$z_B$), $v_{Intermediate}$($x_I$,$y_I$,$z_I$), $v_{Upper}$($x_U$,$y_U$,$z_U$)
    where $y_B \le y_I \le y_U$;
    Get the potential RGP row face index ranging from rowst to rowed by equation (7);
    FOR row face index rowr from rowst to rowed DO
        Get the Coordinates of the 2 potential RGPs
        PRGP1($x_{r1}$,$y_{r1}$,$z_{r1}$), PRGP2($x_{r2}$,$y_{r2}$,$z_{r2}$)
        on the row face indexed rowi by Equation (8) and (9);
        FOR EACH PRGP in { PRGP,PRGP2} DO
            Assign the type of PRGP by Table 2;
            IF PRGP is IGP AND GP$\notin GP_{coll}$ THEN
                Add PRGP as IGP to $GP_{coll}$;
            ELSE
                Add PRGP as RGP to $GP_{coll}$;
            END
        END
    END
    RETURN $GP_{coll}$
END
```

Fig. 5. Pseudocode of nonvertical triangular facet grid-point generation.

Grid-Point Generation for Nonvertical Triangular Facets. A nonvertical triangular facet is a facet with the z-value of its normal not equal to zero. The process of determining the GP is presented in Fig. 4(c). Its pseudocode, presented in Fig. 5, includes Eqs. (2)–(9)

$$\mathrm{col}_{st} = \left\lfloor \frac{x_L - x_o}{cs} \right\rfloor + 1$$
$$\mathrm{col}_{ed} = \left\lceil \frac{x_R - x_o}{cs} \right\rceil - 1 \tag{2}$$

$$x_{c1} = col \times cs + x_o$$
$$x_{c2} = x_{c1}$$
$$y_{c1} = \frac{y_M - y_c}{x_M - x_c} \times (x_{c1} - x_c) + y_c$$
$$y_{c2} = \frac{y_R - y_L}{x_R - x_L} \times (x_{c2} - x_L) + y_L$$
$$z_{c1} = z_L - \frac{x_n(x_{c1} - x_L) + y_n(y_{c1} - y_L)}{z_n}$$
$$z_{c2} = z_L - \frac{x_n(x_{c2} - x_L) + y_n(y_{c2} - y_L)}{z_n} \tag{3}$$

$$(x_c, y_c) = \begin{cases} (x_L,\ y_L), \mathrm{col}_{st} \le \mathrm{col} \le \left\lceil \dfrac{x_M - x_o}{cs} \right\rceil - 1 \\ (x_R, y_R), \max\left(\left\lceil \dfrac{x_M - x_o}{cs} \right\rceil, \mathrm{col}_{st} \right) \le \mathrm{col} \le \mathrm{col}_{ed} \end{cases} \tag{4}$$

$$\mathrm{row}_{st} = \left\lfloor \frac{\min(y_{c1}, y_{c2}) - x_o}{cs} \right\rfloor + 1$$
$$\mathrm{row}_{ed} = \left\lceil \frac{\max(y_{c1}, y_{c2}) - x_o}{cs} \right\rceil + 1 \tag{5}$$

$$x_i = x_{c1}$$
$$y_i = \mathrm{row}_i \times cs + y_o$$
$$z_i = z_L - \frac{x_n(x_i - x_L) + y_n(y_i - y_L)}{z_n} \tag{6}$$

$$\mathrm{row}_{st} = \left\lfloor \frac{y_B - y_o}{cs} \right\rfloor + 1$$
$$\mathrm{row}_{ed} = \left\lceil \frac{y_U - y_o}{cs} \right\rceil - 1 \tag{7}$$

$$
\begin{aligned}
x_{r1} &= \frac{y_M - y_r}{x_M - x_r} \times (y_{r1} - y_r) + x_r \\
x_{r2} &= \frac{y_U - y_B}{x_U - x_B} \times (y_{r2} - y_B) + x_B \\
y_{r1} &= \text{row}_r \times cs + y_o \\
y_{r2} &= y_{r1} \\
z_{r1} &= z_B - \frac{x_n(x_{r1} - x_B) + y_n(y_{r1} - y_B)}{z_n} \\
z_{r2} &= z_B - \frac{x_n(x_{r2} - x_B) + y_n(y_{r2} - y_B)}{z_n}
\end{aligned} \tag{8}
$$

$$
(x_r, y_r) = \begin{cases} (x_B y_B), \text{row}_{st} \le \text{row}_r \le \max\left(\left\lceil \frac{y_I - y_o}{cs} \right\rceil, \text{row}_{st}\right) \\ (x_U, y_U), \max\left(\left\lceil \frac{y_I - y_o}{cs} \right\rceil, \text{row}_{st}\right) \le \text{row}_r \le \text{row}_{ed} \end{cases} \tag{9}
$$

Grid-Point Generation for Vertical Triangular Facets. A vertical triangular facet is a facet with the z-value of its normal equal to zero. The horizontal projection of this kind of facet is a line section so that two different GPs may have the same x- and y-coordinates. In this study, the two GPs with the same x- and y-coordinates in a vertical triangle form a GP pair (GPP) [Fig. 4(b)]. A GPP also comes in four types: vertex GPP (VGPP), column GPP (CGPP), row GPP (RGPP), and inner GPP (IGPP), and its type is determined by any of the GP forming it. Notably, the triangular facet may have a VGP with x- and y- coordinates different from those of any other GPs; in this case, the VGP can be regarded as two overlapping potential VGPs to confer only two GPs to any GPP.

Determining the GPPs of a vertical triangular facet is similar to determining the GPs of a nonvertical triangular facet. The pseudocode with Eqs. (10)–(13) is shown in Fig. 6

$$
\begin{aligned}
x_{c1} &= \text{col} \times cs + x_o \\
x_{c2} &= x_{c1} \\
y_{c1} &= \frac{y_M - y_c}{x_M - x_c} \times (x_{c1} - x_c) + y_c \\
y_{c2} &= \frac{y_R - y_L}{x_R - x_L} \times (x_{c2} - x_L) + y_L \\
z_{c1} &= \frac{z_M - z_c}{x_M - x_c} \times (x_{c1} - x_c) + z_c \\
z_{c2} &= \frac{z_R - z_L}{x_R - x_L} \times (x_{c2} - x_L) + z_L
\end{aligned} \tag{10}
$$

$$
(x_c, y_c, z_c) = \begin{cases} (x_L, y_L, z_L), \text{col}_{st} \le \text{col} \le \left\lceil \frac{x_M - x_o}{cs} \right\rceil - 1 \\ (x_R, y_R, z_R), \max\left(\left\lceil \frac{x_M - x_o}{cs} \right\rceil, \text{col}_{st}\right) \le \text{col} \le \text{col}_{ed} \end{cases} \tag{11}
$$

```
INPUT:
(1) The 3 vertices of the triangle facet v1($x_1$,$y_1$,$z_1$), v2($x_2$,$y_2$,$z_2$) and v3($x_3$,$y_3$,$z_3$)
(2) Voxel origin($x_o$,$y_o$,$z_o$)
(3) Cell size: cs
BEGIN
    Create a collection $GPP_{coll}$ to collect all the GPPs;
    Sort v1,v2 and v3 by x coordinate as
    $v_{Left}$($x_L$,$y_L$,$z_L$), $v_{Mid}$(xM,yM,zM), $v_{Right}$($x_R$,$y_R$,$z_R$)
    where $x_L \le x_M \le x_R$;
        FOR EACH v in {$v_{Left}$,$v_{Mid}$,$v_{Right}$} DO
        Assign the type of v by Table 2;
        Create a GPP whose type is the same as that of v;
        Add v to GPP;
        Add GPP to $GPP_{coll}$;
    END
    Get the potential CGP column face index ranging from $col_{st}$ to $col_{ed}$ by equation (2);
    FOR column face index col from $col_{st}$ to $col_{ed}$ DO
        Get the cooridnates of the2 potential RGPs:
        PRGP1($x_{c1}$,$y_{c1}$,$z_{c1}$), PRGP2($x_{c2}$,$y_{c2}$,$z_{c2}$)
        on the column face indexed col by Equation (10) and (11);
        FOR EACH PCGP in{PCGP1,PCGP2} DO
            Define the type of PCGP by Table 2;
            Create a GPP whose type is the same as that of PCGP;
            Add PCGP1 and PCGP2 to this GPP;
            If GPP is IGPP And GPP $\notin GPP_{coll}$ THEN
                Add GPP as IGPP to $GPP_{coll}$;
            ELSE
                Add GPP as CGPP to $GPP_{coll}$;
            END
        END
    END
    Sort v1,v2 and v3 by y coordinate as
    $v_{Bottom}$($x_B$,$y_B$,$z_B$), $v_{Intermediate}$($x_I$,$y_I$,$z_I$), $v_{Upper}$($x_U$,$y_U$,$z_U$)
    where $y_B \le y_I \le y_U$;
    Get the potential RGP row face index ranging from $row_{st}$ to $row_{ed}$ by equation (7);
    FOR row face index $row_r$ FROM $row_{st}$ TO $row_{ed}$ DO
        Get the Coordinates of the 2 potential RGPs
        PRGP1($x_{r1}$,$y_{r1}$,$z_{r1}$), PRGP2($x_{r2}$,$y_{r2}$,$z_{r2}$)
        on the row face indexed rowi by Equation (12) and (13)
        FOR EACH PRGP in {PRGP1, PRGP2} DO
            Assign the type of PRGP by Table 2;
            Create a GPP whose type is the same as that of GP;
            Add PRGP1 and PRGP2 to GPP
            IF GPP IS IGPP AND GPP$\notin GPP_{coll}$ THEN
                Add GPP as IGPP to $GPP_{coll}$;
            ELSE
                Add GPP as RGPP to $GPP_{coll}$;
            END
        END
    END
    RETURN $GPP_{coll}$
END
```

Fig. 6. Pseudocode of vertical triangular facet grid-point generation.

$$
\begin{aligned}
x_{r1} &= \frac{x_I - x_r}{y_I - y_r} \times (y_{r1} - y_r) + x_r \\
x_{r2} &= \frac{x_U - x_B}{y_U - y_B} \times (y_{r2} - y_B) + x_B \\
y_{r1} &= \text{row}_r \times cs + y_o \\
y_{r2} &= y_{r1} \\
z_{r1} &= \frac{z_I - z_r}{x_I - x_r} \times (x_{r1} - x_r) + z_r \\
z_{r2} &= \frac{z_U - z_B}{x_U - x_B} \times (x_{r2} - x_B) + z_B
\end{aligned} \tag{12}
$$

$$
(x_r, y_r, z_r) = \begin{cases} (x_B y_B, z_B), \text{row}_{st} \le \text{row}_r \le \left\lceil \frac{y_I - y_o}{cs} \right\rceil - 1 \\ (x_U, y_U, z_U), \max\left(\left\lceil \frac{y_I - y_o}{cs} \right\rceil, \text{row}_{st}\right) \le \text{row}_r \le \text{row}_{ed} \end{cases} \tag{13}
$$

Voxel Generation

Voxel generation relies on previously generated GPs or GPPs. Before elaborating on this step, the index of a GP/GPP requires an introduction. Given a GP (x_{gp}, y_{gp}, z_{gp}), the cell size cs, and the voxel origin (x_o, y_o, z_o), its index, denoted by $[\text{col}_{gp}, \text{row}_{gp}]$, is calculated using Eq. (14). The index of a GPP is the same as that of any of its GP

Table 3. Indices of voxels affected by GP [col, row]

VGP	CGP	RGP	IGP
[col,row-1]	[col,row], [col-1,row]	[col,row], [col,row-1]	[col,row], [col-1,row] [col,row-1], [col-1,row-1]

Note: IGP = inner grid point;RGP = row grid point;CGP = column grid point; and VGP = vertex grid point.

$$\mathrm{col}_{gp} = \left\lfloor \frac{x_{gp} - x_o}{cs} \right\rfloor$$
$$\mathrm{row}_{gp} = \left\lfloor \frac{y_{gp} - y_o}{cs} \right\rfloor \quad (14)$$

Voxel Generation for Nonvertical Triangles. For a voxel from a nonvertical triangle, the generated GPs exert different effects: VGP affects the voxels whose indexes are the same as that of the VGP [e.g., GP1 in Fig. 4(d) affects Voxels 1-1]; CGP/RGP affects the voxels that share the same column/row faces as it does [e.g., GP2/GP3 in Fig. 4(d) affects Voxels 2-1, 2-2/3-1, and 3-2]; and IGP affects the voxels horizontally around it [e.g., GP4 in Fig. 4(d) affects Voxel 4-1, 4-2, 4-3, and 4-4]. The indexes of voxels affected by GP [col, row] are listed in Table 3. For a nonvertical triangle, a voxel is valid only if it is affected by three GPs or more because at least three GPs are present in the intersection area between the triangular facet and the voxel. For instance, Voxels 5-2, 5-3, and 5-4 in Fig. 4(d) are valid because they are affected by three GPs, whereas Voxel 5-1 is invalid because only one GP (GP5) affects it. The elevation of the lower face and that of the cell face of a voxel are the minimum and the maximum z-values of the GPs affecting it, respectively. The collection of all valid voxels forms the voxelization of the triangular facet.

Voxel Generation for Vertical Triangles. For a voxel from the vertical triangular facet, there are two special situations: (1) the facet overlaps with a column plane, and the x-value of the facet is positive; and (2) the facet overlaps with a row plane, and the y-value of the facet is positive. In the first, the column index of each voxel affected by GPP is that of the GPP–1; that is, the GPPs only affect the voxels on their left. To illustrate, in Fig. 4(e), VGPP 5 affects Voxel 5 instead of Voxel 5', while IGPP 4 affects only Voxels 5 and 6 instead of Voxels 5, 6, 5', and 6'. In the second, the GPPs only affect the voxels below them. To illustrate, in Fig. 4(e), VGPP 1 affects Voxel 1 instead of Voxel 1', while IGPP 1 affects Voxels 1 and 2 instead of Voxels 1, 2, 1', and 2'. In the situation otherwise, the voxels affected by GPPs are the same as those affected by GPs. Table 4 lists the voxel indexes affected by various GPPs. A voxel is valid only if it is affected by two or more GPPs because the intersection between the voxel and a vertical triangle contains at least two GPPs. The elevation of the lower and cell face of a voxel is determined in a manner similar to that for nonvertical triangular facets, and the collection of all voxels forms the voxelization of the vertical triangular facet.

Triangle Voxel Merging

Voxel Merging in a Solid

The voxels from the previous step are only on the surface of the solid, and some of them may intersect with one another; therefore, the inner part of the solid needs to be filled, and the intersections among the voxels need to be removed. Given a solid and its voxels, voxel merging is conducted as follows: first, the voxels are grouped by their indexes; and second, the voxels are merged group by group. Voxel merging within a group is depicted in Fig. 7 and is explained as follows:

1. Two types of voxels—entrance voxels and exit voxels—are found in (1) in the figure. To inteprete them, imagine that there is a bulk of rays coming from an area under the solid. The rays can enter the solid via the lower faces of a voxel hosted by a vertical triangular facet or a nonvertical triangular facet with a z-component of its normal pointing downwards, hence the two voxels are entrance voxels. Meanwhile, the rays start to leave the voxel only when it reaches the lower face of the voxels hosted by nonvertical triangular facets with the z-component of its normal pointing upward—hence, the name, exit voxels.
2. The sorting rules in (2) in Fig. 7 are as follows. The voxels are ordered by their lower face elevations in ascending order. If two voxels have the same lower face elevation, the entrance voxel should be ordered before the exit voxel to ensure that the aforementioned rays pass the entrance voxels first before entering the solid.
3. In (3) in Fig. 7, two pointers—the main pointer and the subpointer—are identified. The voxel pointed by the main pointer is the main voxel, whereas that pointed by the subpointer is the subvoxel. The subvoxel is the potential merging target of the main voxel. If the current subvoxel can be merged by the main voxel, the subpointer subsequently moves to the next voxel to create a new subvoxel, and the main pointer remains unchanged. If the current subvoxel cannot be merged by the main voxel, the main pointer moves to the subvoxel to make it a new main voxel, meanwhile the subpointer moves to the voxel next to the new main one and that voxel becomes the new sub voxel. This process continues until the main pointer points to the last voxel in the group.
4. Voxel merging in (4) of Fig. 7 starts at the first voxel in the group. Given a main voxel and subvoxels $\mathrm{vox}_{\mathrm{main}}$, $\mathrm{vox}_{\mathrm{sub}}$, with lower and cell face elevations of l_1, l_2, and c_1, c_2, respectively. After merging, the main voxel has a lower face elevation of min (c_1, c_2) and a cell face elevation of max (c_1, c_2). The type of main voxel inherits that of the voxel with a larger cell face elevation. For instance, if max $(c_1, c_2) = c_2$, then the new main voxel is similar in type to that of $\mathrm{vox}_{\mathrm{sub}}$; otherwise, the main voxel type remains unchanged.
5. Voxels in (4.1) and (4.2) are merged but not the voxels in (4.3) in Fig. 7. The reasons are as follows: in (4.1) in Fig. 7, Voxels v8 (main voxel) and v7 (subvoxel) are disconnected. The intermediate zone between v8 (an entrance voxel) and v7 (an exit voxel) indicates that imaginary rays have entered the solid

Table 4. Indices of voxels affected by GPP [col,row]

	VGPP	CGPP	RGPP	IGPP
Triangular facet overlapping with a column face and $x_n > 0$	[col-1,row]	—	—	[col-1,row], [col-1,row-1]
Triangular facet overlapping with a row face and $y_n > 0$	[col,row-1]	—	—	[col,row-1], [col-1,row-1]
None of the above	[col,row-1]	[col, row], [col-1,row]	[col,row], [col,row-1]	[col,row], [col-1,row] [col,row-1], [col-1,row-1]

Note: x_n, $y_n = x$ and y values of the normal of the triangular facet;VGPP = vertices grid point pair;CGPP = column grid point par; RGPP = row column grid pair; and IGPP= inner grid point pair.

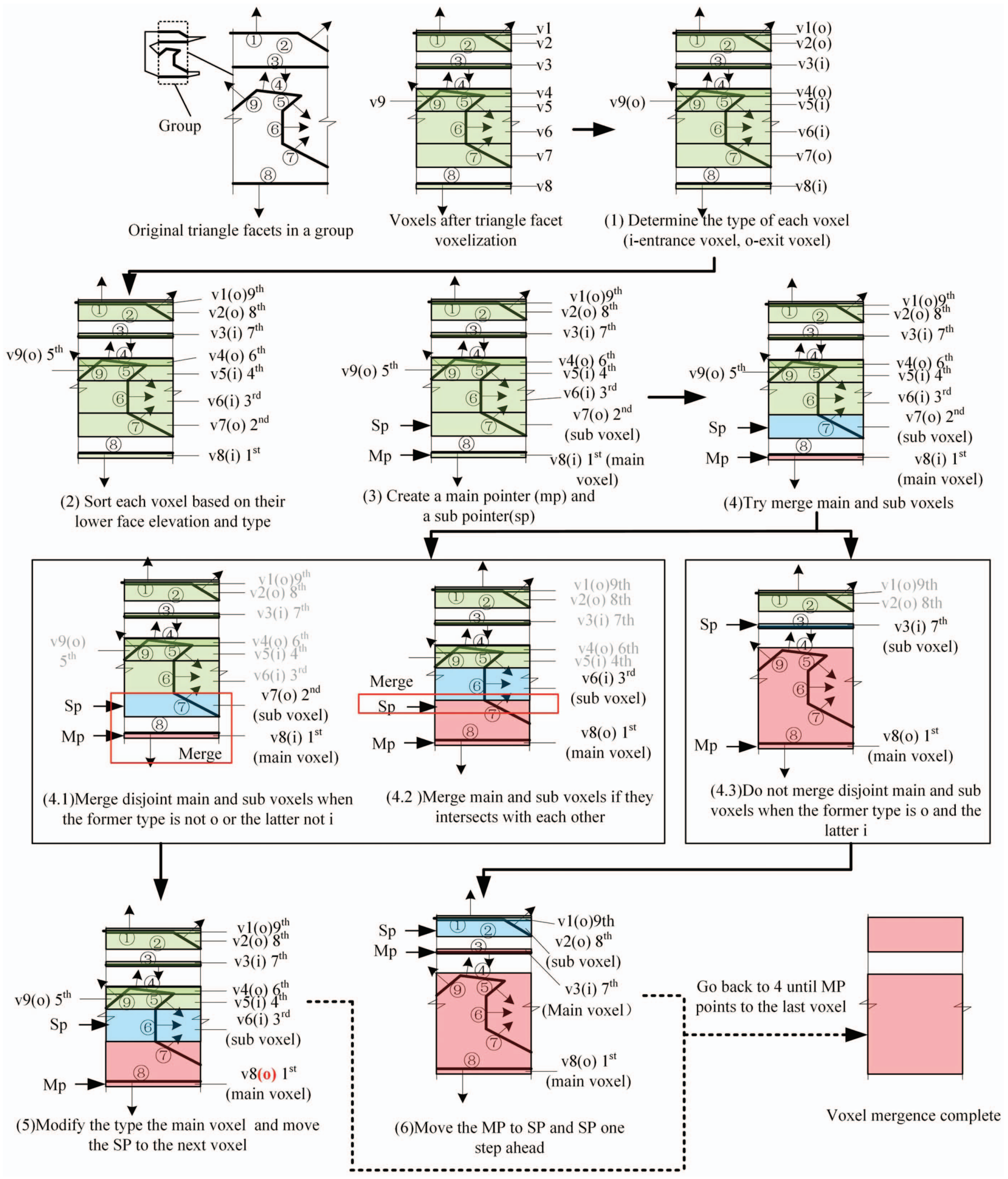

Group
Original triangle facets in a group
Voxels after triangle facet voxelization
(1) Determine the type of each voxel (i-entrance voxel, o-exit voxel)
(2) Sort each voxel based on their lower face elevation and type
(3) Create a main pointer (mp) and a sub pointer(sp)
(4)Try merge main and sub voxels
(4.1)Merge disjoint main and sub voxels when the former type is not o or the latter not i
(4.2)Merge main and sub voxels if they intersects with each other
(4.3)Do not merge disjoint main and sub voxels when the former type is o and the latter i
(5)Modify the type the main voxel and move the SP to the next voxel
(6)Move the MP to SP and SP one step ahead
Go back to 4 until MP points to the last voxel
Voxel mergence complete

Fig. 7. (Color) Merging of triangular voxels.

but have not left yet, suggesting that this zone is the inner part of the solid. Thus, the two voxels should be merged to fill this zone. The reason for merging v8 (currently v8 is merged with v7) and v6 in (4.2) in Fig. 7 is that they intersect each other, and the reason for not merging v8 (currently v8 has merged v7 and v4) and v3 in (4.3) in Fig. 7 is that v8 is currently an exit voxel. Its next voxel, v3, is an entrance voxel. The rays coming from v8 remain outside the voxel until they reach v3. Therefore, the region between the two voxels is outside the solid and should not be merged. The rule for voxel merging between two disjoint voxels is summarized in Table 5.

Voxel Merging Across a Solid

An element may contain multiple solids. After the voxels within a solid are merged, voxels from a different solid also need to be merged. The two voxels from different solids need to be merged only where they intersect with each other.

Table 5. Voxel merging strategy

Voxel under (main voxel)	Voxel upper (subvoxel)	
	Entrance voxel	Exit voxel
Entrance voxel	Merge	Merge
Exit voxel	Not merge	Merge

Validation Test

To validate the proposed method, a validation test is conducted using three subtests: a correctness test, which evaluates the accuracy of the proposed method; a speed test, which measures the elapsed time of the method; and an applicability test, which determines and verifies the advantage of the method. All subtests are run on a laptop with an Intel i7 7700HQ CPU (4 cores 2.8GHz) and 16GB RAM. The API is from Autodesk Revit 2017.

Correctness Test

The correctness test focuses on the accuracy of voxel generation. This test runs on a model created by Autodesk Revit, as shown in Fig. 8(a). The test model contains a floor slab and six types of objects: an axis-aligned cubold, a tilt cubold, concave polyhedrons, gate-like shapes, a hexahedron, and a sphere. These objects represent axis-aligned elements, nonaxis-aligned elements, irregular elements, and elements with curved surfaces at a construction site.

For the purpose of the accuracy test, the proposed method in this study is compared with two other voxelization techniques: the identical-size strategy (Strategy 1) and octree-based voxelization strategy (Strategy 2). Given the element, the identical-sized strategy is as follows: the bounding box of the given element is first obtained, and the box is divided into several voxels with identical heights, widths, and depths. The voxels are then scanned individually, and *ElementIntersectsSolidFilter* is used to test if the current voxel intersects the element. All intersecting voxels are then collected as the voxelization of the given element; in the octree-based strategy, the bounding box is first chosen as the root node of the tree and then divided into eight boxes (voxels) of equal sizes as their child nodes. Each subnode is then checked to determine whether it intersects the element. The node that intersects the element is further divided until the size is smaller than or equal to the threshold set by the user. The voxels intersecting the element are considered as the output. For convenience of expression, the voxels generated using the proposed method are referred to as test voxels, and those generated using Strategies 1 and 2 are referred to as base voxels.

The accuracy of the proposed method is determined by the discrepancies between the total volumes of the test voxels and of the two base voxels of each element. The discrepancies, denoted by dv_{elem}, are calculated using Eq. (15), where m and n are the number of test and base voxels in the element; vt_i is the volume of the ith test voxel; and vb_j is the volume of the jth base voxel. The cell size of a path-planning environment is often given by the user, so therefore, the base voxels and the test voxel share the same width and depth. In this test, the width and depth of any voxel is set as 300×300 mm. In this situation, the size of the voxel is only affected by its height. As the voxelization accuracy increases with the decrease of the voxel size, the height of the base voxels starts at 300 mm and decreases to 100 mm with a step of 100 mm. During this process, the dv_{elem} as well as the elapsed time of the proposed method and Strategies 1 and 2 are measured sequentially. To minimize occasion errors, at each voxel height, the elapsed time of the three methods are recorded 10 times to get the averages. The level of detail of the triangle is set to the highest that Revit can reach to minimize the errors caused by element triangulation

$$dv_{\mathrm{elem}} = \frac{\sum_{i=1}^{m} vt_i - \sum_{j=1}^{n} vb_j}{\sum_{j=1}^{n} vb_j} \tag{15}$$

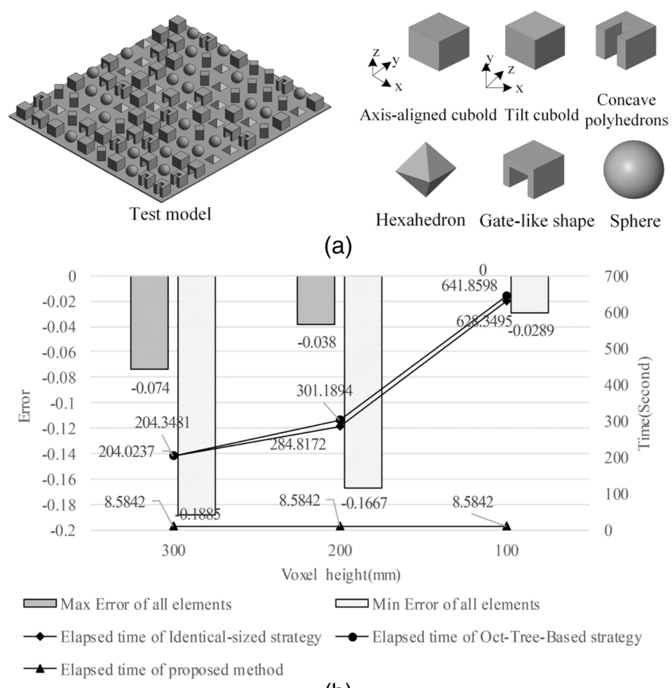


Fig. 8. Correctness test result: (a) test model for correctness test; and (b) test result.

Speed Test

This test focuses on the elapsed time of the method on various triangular facets. In this test, the model *rme_advanced_sample_project*, one of the sample projects of Revit, is chosen for the speed test because it contains numerous mechanical facilities with nonplanar surfaces, as shown in Fig. 9(a). The number of triangular facets can be modified by adjusting the level of detail of the triangulation, enabling the evaluation of the performance of the voxelization method under various numbers of triangular facets.

Applicability Test

This test aims to verify that the proposed voxel-based representation can increase the efficiency of a cell navigability analysis. In this test, the proposed method is compared with four other methods: the AABB method, the ray-based method, the cut-project method, and the solid-based method. The AABB method considers the AABB of a BIM element as its space occupation (Chavada et al. 2012). The navigability of a cell can be evaluated by generating an AABB resembling the moving object above it and checking whether it intersects with the AABBs of the other elements. This method has been used in numerous studies (Lin et al. 2013; Mirzaei et al. 2018; Moon et al. 2014; Tan et al. 2017). The ray-based method proposed by ElNimr et al. (2016) allows each cell to project rays to its neighbors. If a ray hits a component before reaching any neighbor, the cell is unnavigable. The solid-based method by Wang et al. (2018a, b) generates a solid based on a cell. If the solid intersects any element, the cell is inaccessible (unnavigable). The cut-project method introduced by Wang et al. (2019) includes three steps: first, two cutting planes parallel to the supporting surface of the movable object are used to cut all BIM elements; second, the solids are projected between the two planes to the cells on the supporting surface; third, the cells intersected with the projections are regarded as unnavigable.

In this test, the moving object is considered as a labor crew whose space occupation at any time interval while the object is in motion is a $600 \times 600 \times 2{,}000$ mm bounding box (Chinese National Standard 1988). Because the bottom elevation of the feet is $\mathrm{elev}_{\mathrm{foot}}$, the moving object can surmount obstacles with top elevations lower than $\mathrm{elev}_{\mathrm{foot}} + 400$ mm and can bend lower to move across the obstacles with bottom elevations above $\mathrm{elev}_{\mathrm{foot}} + 1{,}600$ mm. At any time interval while the object is in motion, the object keeps the vertical center axis of the bounding box aligned with the location point of a cell from a support element. In this test, the support elements are designated as floor slabs. Two test models are identified: (1) the correctness test model appending several walls with a height

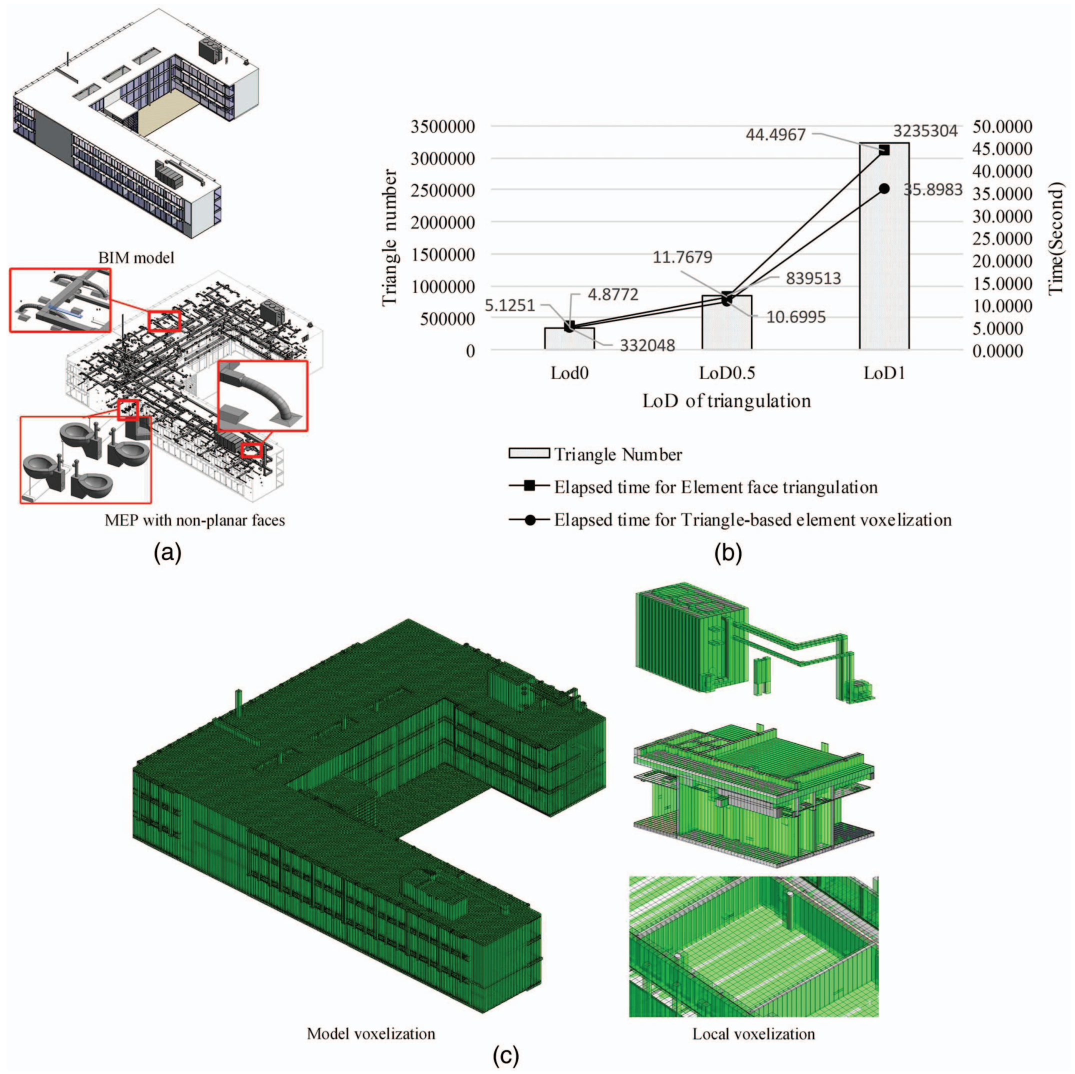


Fig. 9. (Color) Speed test modeling and results: (a) model for speed test; (b) result of speed test; and (c) model after voxelization.

of 300; and (2) the speed test model. The first test model is used to evaluate the performance of the five methods (the proposed method and the four aforementioned methods) in a simple environment, and the second test model is used to further test their performances in a complicated environment.

The unnavigable cells are detected by converting the BIM elements to *C*-obstacles introduced by Lei et al. (2013). An obstacle with a height of more than 400 mm is expanded as a *C*-obstacle, depending on the size of the moving object. The advantage of this strategy is that it can markedly decrease the number of cells for the unnavigable test, as shown in Fig. 10(a). For obstacles less than 400 mm in height (surmountable obstacles), they are not expanded because the moving object can go over it. Therefore, the surmountable cells are those only covered by the surmountable obstacles, as shown in Fig. 10(b).

Result and Discussion

Result and Discussion of the Correctness Test

The test result is presented in Fig. 8(b). As shown in the figure, as the voxel height decreases, the accuracy of the base voxel increases, thereby decreasing the discrepancies between the volumes of the test voxels. This trend proves the accuracy of the proposed method. The identical-sized and octree-based methods require a longer elapsed time to achieve accuracy, whereas the proposed method requires considerably less time to achieve the same level of accuracy.

Result and Discussion of Speed Test

The results of the test are presented in Figs. 9(b and c). In Fig. 9(b), the following results can be found: (1) the element face triangulation uses up most of the total elapsed time; (2) as the number of triangular facets increase, the time required for voxel triangulation increases as well; and (3) the elapsed time for the voxel generation increases with the number of voxels.

Result and Discussion of the Applicability Test

The test result for Project 1 is presented in Fig. 11. Notable results include the following:

- In the AABB method, no free voxel is found because this method regards the AABB of the element as its space occupation. The walls on the floor slab diagonals have AABBs covering all floors; therefore, the cells, are surmountable if not unnavigable. In addition, based on Fig. 11(b), all elements above the

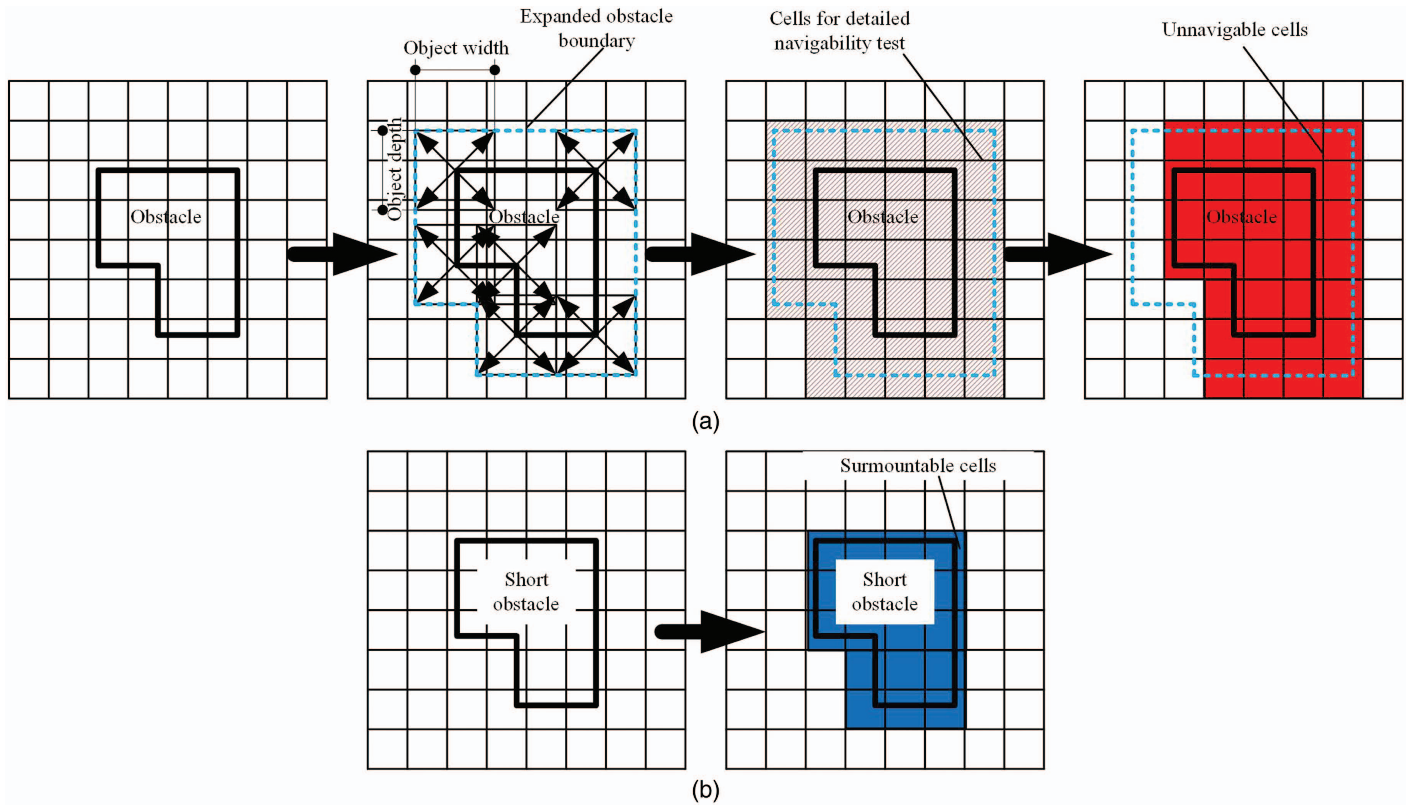


Fig. 10. (Color) Detection of unnavigable and surmountable cells: (a) unnavigable cells detection; and (b) surmountable cells detection.

floor slab have the same layout as those of the unnavigable cells because all have the same size as that of AABB.

- With reference to Fig. 11(b), ray casting seems relatively inaccurate because evaluating the navigability of a cell by using only several rays yields misleading results. In Fig. 11(f), the two rays from a cell can reach the neighbor cells without hitting the obstacle. On the basis of ray casting, it is navigable; however, the cell clearly intersects the obstacle and is unnavigable. Although more rays can be cast to increase accuracy, the errors cannot be eliminated because the rays cannot seamlessly fill the 3D space.
- The cut-project, solid-based method, and the proposed method generate the same results, which are verified by observing the layout of the cells. Thus, in this test, the three methods generate the correct result.
- On the basis of the elapsed time of the methods, the voxel-based method requires the shortest time (0.1094 s) and generates the correct result. Therefore, this method is desirable for a cell-navigability analysis.

The inaccuracy of the AABB and ray-casting methods is verified in the previous test. Thus, the proposed method, cut-project, and solid-based methods are further tested under Model 2. The test result is presented in Fig. 12. In this test, the results of the proposed and cut-project methods are similar, whereas that of the solid-based method is slightly different. The reason is that the built-in clash detection mechanism of Revit tends to ignore some clashes. In this test, geometry overlaps (clashes) [Fig. 12(c)] between a door and a solid, resembling a moving object, and cannot be detected for reasons that have yet to be determined because the clash detection mechanism of Revit is unknown. By contrast, in the voxel-based and cut-project methods, this error does not occur. Therefore, these two methods are stricter than the solid-based method in the Revit environment.

With regard to speed, the average elapsed time of the voxel-based method is the shortest. The voxel-based method only requires 0.7001 s to complete the cell navigability test, which is considerably shorter than that of the cut-project method (42.1685 s) and that of the solid-based method (665.7224 s) [Fig. 12(d)].

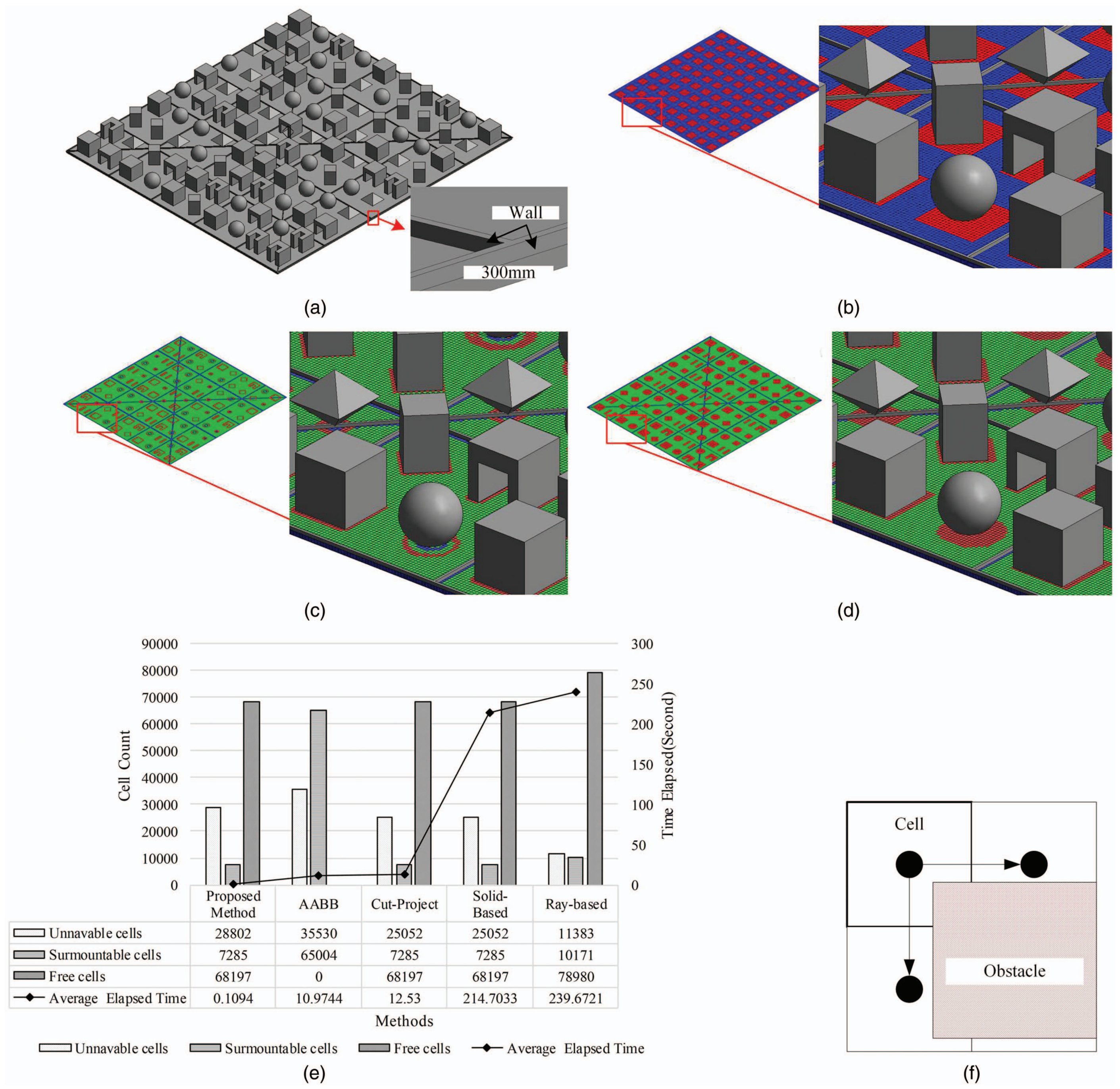
In summary, the voxel-based method exhibits the best performance in accuracy and speed. As such, it is more applicable than the other methods evaluated in this study.

Conclusion

This study proposes a method to discretize a BIM model into a collection of voxels for path-planning tasks. The advantages of this method are as follows:

- By converting the model into voxels, the cells can be automatically generated because the upper surface of a voxel can be regarded as a cell;
- The voxels are reusable in a model so that the voxels are generated only once in the whole path-planning process;
- The elapsed time of a cell-navigability analysis is significantly shortened because the computation-intensive geometry operation is avoided in this stage; and
- The properties of the voxels are extensible so that more customized information can be included for various tasks. In conclusion, this method can increase the efficiency of a path-planning analysis.

This study contributes to the existing body of knowledge by providing an executable BIM model voxelization method to generate a cell-based path-planning environment. With the proposed method, researchers and practitioners can conveniently convert existing models to voxels. Moreover, by designating the BIM elements supporting the moving objects, the cell faces of the voxels hosted by these elements can immediately form a path-planning environment. In addition, a cell navigability test can be conducted quickly and accurately, which is proved in the applicability test.



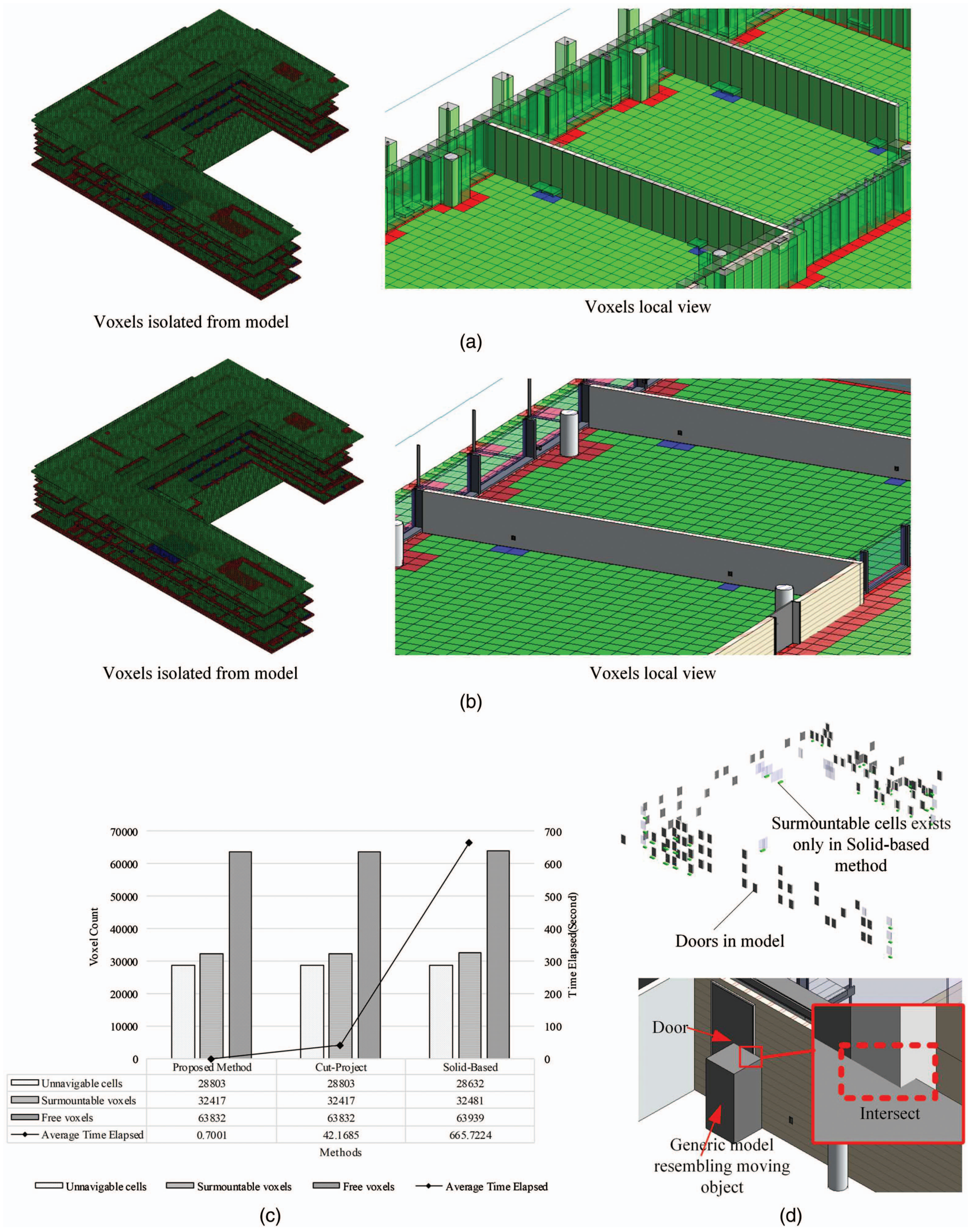
	Proposed Method	AABB	Cut-Project	Solid-Based	Ray-based
Unnavable cells	28802	35530	25052	25052	11383
Surmountable cells	7285	65004	7285	7285	10171
Free cells	68197	0	68197	68197	78980
Average Elapsed Time	0.1094	10.9744	12.53	214.7033	239.6721

Fig. 11. (Color) Test result for Model 1: (a) Revit model of Project 1; (b) result of AABB method; (c) result of ray-based method; and (d) result of cut-project method/solid-based method/proposed method; (e) cell number and time elapsed of each method; and (f) error analysis of ray-casting methods. (a–d) the green, blue, and red areas represent free, surmountable, and unnavigable cells, respectively.

Therefore, the proposed method can facilitate multiple tasks, such as (1) testing various path-search algorithms (Taneja et al. 2016); (2) incorporating cellular automata and agent-based modeling to simulate moving objects (Kneidl et al. 2013; Zhang et al. 2019); (3) other cell-based tasks, such as space usage analysis and optimization (Kumar and Cheng 2015); and (4) indoor people positioning and tracking (Xu et al. 2018).

However, the proposed method still has several limitations:

- The result of the speed test shows that element face triangulation uses most of the total elapsed time because the determination of the correct triangle normal involves time-consuming geometric operations. Accordingly, future studies should include the enhancement of the method for determining the correct triangle normal;
- The realization of voxelization in the prototype adopts a CPU-based sequential process. Parallel processing techniques based on multicore CPU or GPU can potentially induce a significant increase in the speed of voxelization; and
- The approximation of nonplanar faces may affect the accuracy of voxelization. The accuracy of triangulation, as reported by Wang et al. (2018, b), has yet to be done by studies. All aforementioned limitations need to be addressed in further research.



	Proposed Method	Cut-Project	Solid-Based
Unnavigable cells	28803	28803	28632
Surmountable voxels	32417	32417	32481
Free voxels	63832	63832	63939
Average Time Elapsed	0.7001	42.1685	665.7224

Fig. 12. (Color) Test result for Model 2: (a) result of the proposed method; (b) result of solid-based method, cut-project method; (c) cell numbers and elapsed time of each method; and (d) example of clashes Revit cannot detect. The green, blue, and red areas of (a) and (b) represent the free, surmountable, and unnavigable cells, respectively.

Data Availability Statement

Data generated or analyzed during the study are available from the corresponding author by request. Information about the *Journal's* data sharing policy can be found at http://ascelibrary.org/doi/10.1061/%28ASCE%29CO.1943-7862.0001263.

Acknowledgments

The research is supported by "The Fundamental Research Funds for the Central Universities (WUT: 203106006)." The authors are grateful to Mr. Jeremy Tammik who provided valuable suggestions on using the Revit API for element triangulation.

The authors also thank the two anonymous reviewers whose constructive instructions helped improve the research output.

References

Bandi, S., and D. Thalmann. 1998. "Space discretization for efficient human navigation." *Comput. Graphics Forum* 17 (3): 195–206. https://doi.org/10.1111/1467-8659.00267.

Bischoff, S., D. Pavic, and L. Kobbelt. 2005. "Automatic restoration of polygon models." *ACM Trans. Graphics* 24 (4): 1332–1352. https://doi.org/10.1145/1095878.1095883.

Castellazzi, G., A. M. D'Altri, G. Bitelli, I. Selvaggi, and A. Lambertini. 2015. "From laser scanning to finite element analysis of complex buildings by using a semi-automatic procedure." *Sensors* 15 (8): 18360–18380. https://doi.org/10.3390/s150818360.

Chavada, R., N. Dawood, and M. Kassem. 2012. "Construction workspace management: The development and application of a novel nD planning approach and tool." *J. Inf. Technol. Constr.* 17 (13): 213–236.

Chen, H. H., and T. S. Huang. 1985. "Octrees: Construction, representation, and manipulation." In Vol. 579 of *Proc., Cambridge Symp.*, 448–458. New York: International Society for Optics and Photonics. https://doi.org/10.1117/12.950833.

Cheng, J. C. P., Y. Tan, Y. Z. Song, Z. Y. Mei, V. J. L. Gan, and X. Y. Wang. 2018. "Developing an evacuation evaluation model for offshore oil and gas platforms using BIM and agent-based model." *Autom. Constr.* 89 (May): 214–224. https://doi.org/10.1016/j.autcon.2018.02.011.

Chinese National Standard. 1988. *Human dimensions of Chinese adults*. GB10000-1988. Beijing: Standard Press of China.

de Queiroz, R. L., and P. A. Chou. 2016. "Compression of 3D point clouds using a region-adaptive hierarchical transform." *IEEE Trans. Image Process.* 25 (8): 3947–3956. https://doi.org/10.1109/TIP.2016.2575005.

Dunn, F., and I. Parberry. 2005. *3D math primer for graphics and game development (Chinese edition)*. Beijing: Tsinghua University Press.

Echegaray, G., and D. Borro. 2012. "A methodology for optimal voxel size computation in collision detection algorithms for virtual reality." *Virtual Reality.* 16 (3): 205–213. https://doi.org/10.1007/s10055-011-0199-5.

Elmahdi, A., I.-C. Wu, and H.-J. Bargstädt. 2011. "4D grid-based simulation framework for facilitating workspace management." In *Proc., 11th Int. Conf. on Construction Applications of Virtual Reality*, 403–412.

ElNimr, A., M. Fagiar, and Y. Mohamed. 2016. "Two-way integration of 3D visualization and discrete event simulation for modeling mobile crane movement under dynamically changing site layout." *Autom. Constr.* 68 (Aug): 235–248. https://doi.org/10.1016/j.autcon.2016.05.013.

Hsu, J. J., and J. C. Chu. 2014. "Long-term congestion anticipation and aversion in pedestrian simulation using floor field cellular automata." *Transp. Res. Part C* 48 (48): 195–211. https://doi.org/10.1016/j.trc.2014.08.021.

Kneidl, A., D. Hartmann, and A. Borrmann. 2013. "A hybrid multi-scale approach for simulation of pedestrian dynamics." *Transp. Res. Part C* 37 (3): 223–237. https://doi.org/10.1016/j.trc.2013.03.005.

Krijnen, T., and J. Beetz. 2017. "An IFC schema extension and binary serialization format to efficiently integrate point cloud data into building models." *Adv. Eng. Inf.* 33 (Aug): 473–490. https://doi.org/10.1016/j.aei.2017.03.008.

Kumar, S. S., and J. C. P. Cheng. 2015. "A BIM-based automated site layout planning framework for congested construction sites." *Autom. Constr.* 59 (Nov): 24–37. https://doi.org/10.1016/j.autcon.2015.07.008.

Lei, Z., H. Taghaddos, U. Hermann, and M. Al-Hussein. 2013. "A methodology for mobile crane lift path checking in heavy industrial projects." *Autom. Constr.* 31 (May): 41–53. https://doi.org/10.1016/j.autcon.2012.11.042.

Lin, Y.-H., Y.-S. Liu, G. Gao, X.-G. Han, C.-Y. Lai, and M. Gu. 2013. "The IFC-based path planning for 3D indoor spaces." *Adv. Eng. Inf.* 27 (2): 189–205. https://doi.org/10.1016/j.aei.2012.10.001.

Marzouk, M., and I. A. Daour. 2018. "Planning labor evacuation for construction sites using BIM and agent-based simulation." *Saf. Sci.* 109 (Nov): 174–185. https://doi.org/10.1016/j.ssci.2018.04.023.

Min, K., C. Park, H. Yang, and G. Yun. 2018. "Legorization from silhouette-fitted voxelization." *KSII Trans. Internet Inf. Syst.* 12 (6): 2782–2805.

Mirzaei, A., F. Nasirzadeh, M. Parchami Jalal, and Y. Zamani. 2018. "4D-BIM dynamic time–space conflict detection and quantification system for building construction projects." *J. Constr. Eng. Manage.* 144 (7): 04018056. https://doi.org/10.1061/(ASCE)CO.1943-7862.0001504.

Moon, H., V. R. Kamat, and L. Kang. 2014. "Grid cell-based algorithm for workspace overlapping analysis considering multiple allocations of construction resources." *J. Asian Archit. Build. Eng.* 13 (2): 341–348. https://doi.org/10.3130/jaabe.13.341.

Mu, B., M. Pan, and J. Deng. 2010. "Fast large scale voxelization using projection volume and octree." *Geogr. Geo-Inf. Sci.* 26 (4): 27–31.

Park, M., Y. Yang, H. S. Lee, S. Han, and S. H. Ji. 2012. "Floor-level construction material layout planning model considering actual travel path." *J. Constr. Eng. Manage.* 138 (7): 905–915. https://doi.org/10.1061/(ASCE)CO.1943-7862.0000493.

Shi, J. Y., A. Z. Ren, and C. Chen. 2009. "Agent-based evacuation model of large public buildings under fire conditions." *Autom. Constr.* 18 (3): 338–347. https://doi.org/10.1016/j.autcon.2008.09.009.

Shirowzhan, S., S. M. E. Sepasgozar, H. Li, and J. Trinder. 2018. "Spatial compactness metrics and constrained voxel automata development for analyzing 3D densification and applying to point clouds: A synthetic review." *Autom. Constr.* 96 (Dec): 236–249. https://doi.org/10.1016/j.autcon.2018.09.018.

Stolte, N., and A. Kaufman. 2001. "Novel techniques for robust voxelization and visualization of implicit surfaces." *Graphical Models.* 63 (6): 387–412. https://doi.org/10.1006/gmod.2001.0559.

Sun, X., Q. Li, and B. Yang. 2018. "Compositional structure recognition of 3D building models through volumetric analysis." *IEEE Access.* 6: 33953–33968. https://doi.org/10.1109/ACCESS.2018.2842721.

Sun, X., B. Yang, and L. I. Qingquan. 2011. "Structural segmentation method for 3D building models based on voxel analysis." *Acta Geod. Cartographica Sin.* 40 (5): 582–586.

Tan, Y., Y. Song, X. Liu, X. Wang, and J. C. P. Cheng. 2017. "A BIM-based framework for lift planning in topsides disassembly of offshore oil and gas platforms." *Autom. Constr.* 79 (Jul): 19–30. https://doi.org/10.1016/j.autcon.2017.02.008.

Taneja, S., B. Akinci, J. H. Garrett, and L. Soibelman. 2016. "Algorithms for automated generation of navigation models from building information models to support indoor map-matching." *Autom. Constr.* 61 (Jan): 24–41. https://doi.org/10.1016/j.autcon.2015.09.010.

Wang, Q., B. Gao, T. Li, H. Wu, J. Kan, and B. Hu. 2018a. "A triangular mesh generator over free-form surfaces for architectural design." *Autom. Constr.* 93 (Sep): 280–292. https://doi.org/10.1016/j.autcon.2018.05.018.

Wang, Q., Z. Guo, K. Mintah, Q. Li, T. Mei, and P. Li. 2019. "Cell-based transport path obstruction detection approach for 4D BIM construction planning." *J. Constr. Eng. Manage.* 145 (3): 04018141. https://doi.org/10.1061/(ASCE)CO.1943-7862.0001583.

Wang, Q. K., Z. Guo, T. T. Mei, Q. Y. Li, and P. Li. 2018b. "Labor crew workspace analysis for prefabricated assemblies' installation: A 4D-BIM-based approach." *Eng. Constr. Archit. Manage.* 25 (3): 374–411. https://doi.org/10.1108/ECAM-09-2016-0210.

Xiong, Q., Q. Zhu, Z. Du, S. Zlatanova, Y. Zhang, Y. Zhou, and Y. Li. 2016. "Free multi-floor indoor space extraction from complex 3D building models." *Earth Sci. Inf.* 10 (1): 69–83. https://doi.org/10.1007/s12145-016-0279-x.

Xu, W., L. Liu, S. Zlatanova, W. Penard, and Q. Xiong. 2018. "A pedestrian tracking algorithm using grid-based indoor model." *Autom. Constr.* 92 (Aug): 173–187. https://doi.org/10.1016/j.autcon.2018.03.031.

Yang, X., S. Y. Chen, and Z. P. You. 2017a. "3D voxel-based approach to quantify aggregate angularity and surface texture." *J. Mater. Civ. Eng.* 29 (7): 04017031. https://doi.org/10.1061/(ASCE)MT.1943-5533.0001872.

Yang, X., X. Luo, H. Li, X. Luo, and H. Guo. 2017b. "Location-based measurement and visualization for interdependence network on construction sites." *Adv. Eng. Inf.* 34 (Oct): 36–45. https://doi.org/10.1016/j.aei.2017.09.003.

Young, G., and A. Krishnamurthy. 2018. "GPU-accelerated generation and rendering of multi-level voxel representations of solid models." *Comput. Graphics.* 75 (Oct): 11–24. https://doi.org/10.1016/j.cag.2018.07.003.

Yuan, W., and M. Schneider. 2010. "Supporting 3D route planning in indoor space based on the LEGO representation." In *Proc., 2nd ACM SIGSPATIAL Int. Workshop on Indoor Spatial Awareness*, 16–23. New York: Association for Computing Machinery.

Zhang, C., A. Hammad, T. M. Zayed, G. Wainer, and H. Pang. 2007. "Cell-based representation and analysis of spatial resources in construction simulation." *Autom. Constr.* 16 (4): 436–448. https://doi.org/10.1016/j.autcon.2006.07.009.

Zhang, P., N. Li, Z. Jiang, D. Fang, and C. J. Anumba. 2019. "An agent-based modeling approach for understanding the effect of worker-management interactions on construction workers' safety-related behaviors." *Autom. Constr.* 97 (Jan): 29–43. https://doi.org/10.1016/j.autcon.2018.10.015.

Zhang, Y., S. Garcia, W. Xu, T. Shao, and Y. Yang. 2018. "Efficient voxelization using projected optimal scanline." *Graphical Models.* 100 (Nov): 61–70. https://doi.org/10.1016/j.gmod.2017.06.004.

Zhao, D., C. Wei, B. Hujun, Z. Hongxin, and P. Qunsheng. 2004. "Real-time voxelization for complex polygonal models." In *Proc., 12th Pacific Conf. on Computer Graphics and Applications*, 43–50.