Computers and Structures 244 (2021) 106456

Contents lists available at ScienceDirect

Computers and Structures

journal homepage: www.elsevier.com/locate/compstruc

A framework to automate the design of digitally-fabricated timber plate structures

Aryan Rezaei Rad [a,c,*], Henry Burton [b], Nicolas Rogeau [a,c], Petras Vestartas [a], Yves Weinand [a,c]

[a] Laboratory for Timber Constructions (IBOIS), École Polytechnique Fédérale de Lausanne (EPFL), EPFL, ENAC, IIC, IBOIS, GC H2 711(Bâtiment GC), Station 18, CH-1015, Lausanne, Vaud, Switzerland

[b] School of Civil and Environmental Engineering, University of California Los Angeles (UCLA), 5731H Boelter Hall, Los Angeles, CA, United States

[c] The National Centre of Competence in Research (NCCR) Digital Fabrication, Swiss Federal Institute of Technology in Zurich (ETHZ), HIB E 25, Stefano-Franscini-Platz 1, CH-8093 Zurich, Zurich, Switzerland

A R T I C L E I N F O

Article history:
Received 25 April 2020
Accepted 4 November 2020

Keywords:
Computer-aided design (CAD)
Computer-aided engineering (CAE)
Macro modeling technique
Timber plates
Integral mechanical attachment technique
Computer programming

A B S T R A C T

The current study uses knowledge from digital architecture, computer science, engineering informatics, and structural engineering to formulate an algorithmic framework for integrated Computer-Aided Design (CAD) and Computer-Aided Engineering (CAE) of Integrally-Attached Timber Plate (IATP) structures. The algorithm is designed to take the CAD 3D geometry of an IATP structure as input and automates the construction and analysis of the corresponding CAE model using a macroscopic element, which is an alternative to continuum Finite Element (FE) models. Each component of the macro model is assigned a unique tag that is linked to the relevant geometric and structural parameters. The CAE model integrity is maintained through the use of the common data model (CDM) concept and object-oriented programming. The relevant algorithms are implemented in Rhinoceros 3D using RhinoCommon, a .NET software development kit. Once the CAE macro model is generated, it is introduced to the OpenSees computational platform for structural analysis. The algorithmic framework is demonstrated using two case structures: a prefabricated timber beam with standard geometry and a free-form timber plate arch. The results are verified with measurements from physical experiments and FE models, where the time needed to convert thousands of CAD assemblies to the corresponding CAE models for response simulation is considerably reduced.

1. Introduction

Spurred by the proliferation of interactive modeling tools and visual programming languages, the Computer-Aided Design (CAD) framework and digital geometry processing have facilitated the design of spatial timber plate structures with complex geometries. To understand the structural performance of such systems, especially in terms of their functionality, numerical simulations are generally employed to simulate the structural response. Numerical models that include the geometrical and mechanical properties of the system under consideration are generally described as Computer-Aided Engineering (CAE) models. By and large, CAD and CAE models for spatial timber plate structures are constructed in different environments. As such, investigations have sought to develop methodologies to convert CAD data into CAE models, which are discussed in Section 1.1. The scope and motivation of the study are then outlined in Section 1.2.

1.1. Spatial timber plate structures and available CAD-to-CAE data exchange tools

Data exchange tools are useful for timber plate structures with a large number of elements because of their ability to considerably reduce the time required to convert thousands of CAD assemblies to the corresponding CAE models. These tools also enable the idealization of CAD models so that the numerical simulations can be readily performed in a CAE platform. The CAD-to-CAE exchange for spatial free-form timber plates was first introduced by the Institute for Computational Design [1] and the Institute of Building Structures and Structural Design [2]. In particular, Li and Knippers [3], Krieg et al. [4], and La Magna et al. [5] developed CAD-to-FE exchange tools to simulate the behavior of segmented timber plate shells. The CAD model, which is constructed in Rhinoceros 3D [6]

* Corresponding author at: EPFL, ENAC, IIC, IBOIS, GC H2 711(Bâtiment GC), Station 18, CH-1015 Lausanne, Switzerland.

E-mail addresses: aryan.rezaeirad@epfl.ch (A. Rezaei Rad), hvburton@seas.ucla.edu (H. Burton), yves.weinand@epfl.ch (Y. Weinand).

URL: http://www.dfab.ch/ (N. Rogeau).

https://doi.org/10.1016/j.compstruc.2020.106456
0045-7949/© 2020 The Authors. Published by Elsevier Ltd.

using the visual programming environment, Grasshopper 3D [7], consists of three-dimensional (3D) brick elements to represent the timber plates. The data transfer from the CAD to CAE platforms was done either statically via import/export of the corresponding dwg file or dynamically through a parametric modeling interface. A centralized repository is generated to encompass the datastructure associated with the CAE geometry, mechanical properties, and analysis handlers. The generated data is then mapped to an appropriate FE platform(s) such as SOFiSTiK [8] and/or ANYSIS® [9], where two-dimensional (2D) plane shell elements and one-dimensional (1D) springs are used to simulate the behavior of the timber panels and joints, respectively.

Recently, the Laboratory for Timber Construction (IBOIS) at École Polytechnique Fédérale de Lausanne (EPFL) [10] has designed free-form Integrally-Attached Timber Plate (IATP) structures [11,12], where elements are joined using the Integral Mechanical Attachment (IMA) technique (Fig. 1a). The defining feature of IMAs is that the connection between timber plates is established solely through their geometry without additional connectors such as nails, screws, dowels, and adhesives [11,12]. The design process for IATP structures begins with the definition of the target surface established in a CAD environment (Fig. 1b). Using ShapeOp [13] along with Grasshopper 3D [7], Robeller et al. [14] introduced an iterative IATP optimization framework to develop and refine the design object, design the pattern, and establish the global form (Fig. 1c). The algorithm was formulated to introduce a new assembly-constrained tiling, and identify the pattern of timber plates, mesh discretization, the density of planar meshes, the curvature and segmentation of the design surface, the angle between adjacent plates, the position of IMAs, and the assembly pattern between the elements. ShapeOp finds the optimum geometry that satisfies fabrication and assembly constraints but does not consider the structural design objectives. The pseudocode associated with the NURBS subdivision used to obtain the quad mesh is provided in Appendix 1.1. The basis of the NURB subdivision algorithm was developed by Robeller et al. [14]. The CAD model was then completed by converting the segmented planes to 3D brick elements (Fig. 1d). The term 3D is used because the plate thickness is explicitly represented in the model. A data exchange algorithm was then introduced to convert the CAD model to the Computer-Aided Manufacturing (CAM) model [14]. The CAD-to-CAM algorithm provided the information about the orientation of the fabrication tool, which was described with tool center points and two Euler angles. Generating the upper and lower contours for each plate, the exchange algorithm outputs the corresponding G-code using a loft-like offset of the contours. The G-code is then used for 5-axis digital fabrication. The pseudocode associated with the generation of the CAD 3D model and the fabrication outlines is provided in Appendix 1.2. The CAD-to-CAM algorithm was developed by Robeller et al. [14].

Recently, Nguyen et al. [15] developed an automated numerical tool that uses a parametric modeling framework to convert the CAD data associated with IATP structures to the corresponding FE numerical model (Fig. 1e). The CAD model was idealized using pla-

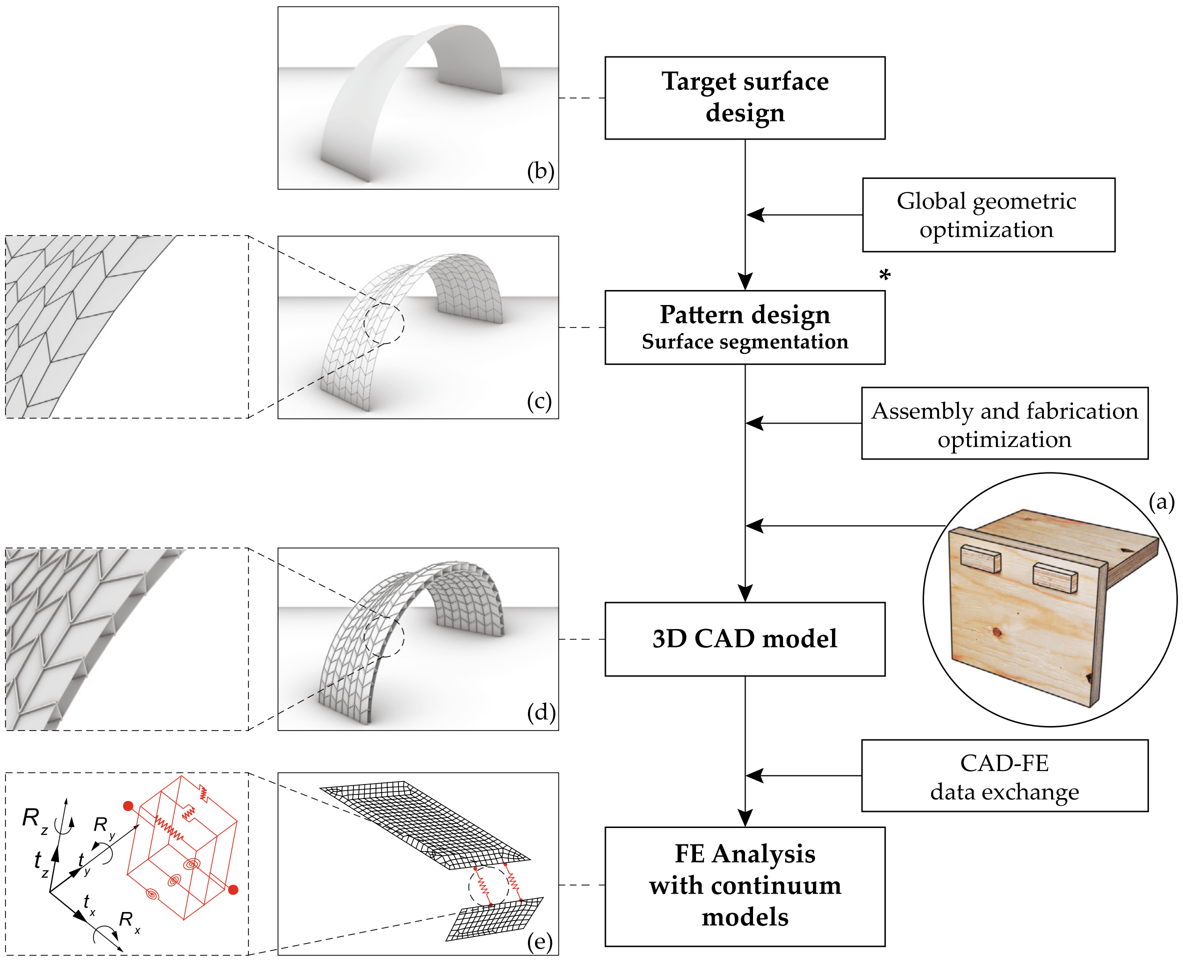


Fig. 1. (a) Integral Mechanical Attachment (IMA), (b) target surface, (c) Quad pattern, (d) CAD 3D Integrally-Attached Timber Plate (IATP) structure, (e) FE model.

nar surfaces to represent timber plates, and line elements for the IMAs. The idealization process includes transformation algorithms such as detail removal [16], dimensional reduction [17], and non-manifold topological modeling [18]. The heterogeneous data exchange methodology [19] was employed to export the data associated with the model geometry from the CAD environment. Using the RhinoCommon library [20], plane-plane and line-plane intersection methods were used to generate the idealized geometry. The Initial Graphics Exchange Specification (IGES), consisting of only mathematical descriptions of the model geometry, was used as the neutral data format. The idealized geometry was then introduced to the FE modeling software ABAQUS^{TM} using the scripting interface. Finite strain S4R shell elements were employed to simulate the behavior of planar surfaces and springs were used to simulate the behavior of the IMAs (Fig. 1e).

1.2. Motivation and scope of the current study

The structural design of large-scale IATP structures requires several steps: (1) Understanding the behavior of IMAs under different load cases. For this step, physical experiments have been conducted by Rezaei Rad et al. [12,11,21] to gain insight into the behavior of joints under tensile, edgewise, and flatwise loads, respectively. (2) Introducing a computationally-efficient mechanical model, where the level of detail associated with the design object is adapted to simplify the design process. For this step, a macroscopic model that employs only beam and spring elements was introduced by Rezaei Rad et al. [22] to simulate the behavior of timber shells with IMAs. (3) Introducing an interface to automatically convert the design CAD geometry associated with hundreds of timber plates with thousands of IMAs to the corresponding macro model. The details of this step have not been formulated and is not available in typical building information modeling (BIM) tools and is a major challenge in the design and performance assessment of large-scale IATP structures. Therefore, the primary objective of the current study is to develop a CAD-to-CAE algorithmic data exchange framework that translates the CAD data associated with a custom-defined IATP structure to the corresponding CAE macro model. The proposed framework is demonstrated by evaluating the performance of large-scale IATP prototypes that have recently been constructed.

Establishing an automatic link between the CAD and CAE macro models has multiple advantages. It considerably reduces the time required to prepare a CAE model. Furthermore, repetitive manual operations are removed from the design process and human errors are minimized. Moreover, instead of working with three or more software packages, the model integrity is maintained in one platform by using object-oriented programming. In addition, the proposed framework enhances the design flexibility of IATPs since multiple variants can be considered in the design process while the associated structural performance can be evaluated with minimal computational expense. The modeling and computational simulation of complex structures is also improved through the use of an open-source CAE platform.

The parametric CAD-to-CAE exchange integrates data analysis and transformation, geometrical simplification, and structural analyis of timber plates with edgewise IMAs. Furthermore, while the concept and methodology of the proposed CAD-to-CAE data exchange algorithm are independent of the computational platform, the current study employs (1) the Rhinoceros .NET Software Development Kit (SDK) called RhinoCommon [20] and Rhino. Python [23] to construct the plugins and components for the CAD-to-CAE data transformation, and (2) The Open System for Earthquake Engineering Simulation, OpenSees [24], to perform the structural analysis. The data exchange framework is designed in the CAD environment while the output data generated by the framework is formulated such that it follows the syntax of OpenSees. The overall workflow of the current study is shown in Fig. 2.

The details of the macroscopic mechanical model for the structural analysis of spatial timber plates are reviewed in Section 2. The algorithmic CAD-to-CAE data exchange methodology and design workflow are outlined in Section 3. The integrative design approach, which encompasses the geometry generation and structural analysis, is also explained. Details of the exchange framework and the indexing algorithm used to define the macro model components are also presented. Section 4 applies the proposed framework to two design cases: prefabricated timber beams with standard geometries and a free-form IATP structure. The conclusions are summarized in Section 5. The pseudocode associated with the IATP design modules is provided in Appendix 1.3.

2. Macro modeling technique for timber plate structures

The analysis of structural systems governed by plate/shell action generally requires model simplifications such as dimensional reduction and detail removal to reduce the computational time and complexity of the model [25,26,27]. For IATP structures, a macroscopic modeling technique has been accordingly introduced by Rezaei Rad et al. [22], where a series of one-dimensional elements (i.e. springs and beam elements) are adopted as an alternative to the detailed continuum FE models (i.e. shell or brick elements).

Three key techniques are used to develop the macro model: feature-based simplification, topology-based modification, and virtual topology adaptation. Geometric reduction, which is a part of feature-based simplification, is used to convert the 3D timber plate elements (Fig. 3a) to 1D macro elements (Fig. 3b). The macro model is valid for quadrilateral timber plates with edgewise connections subjected to linear elastic behavior. The quadrilateral timber plates with non-orthonormal edges can be also simulated using the macro model. The model consists of inner beams, boundary elements, two shear springs, two-node link elements, and uniaxial tension-compression springs distributed along the boundary elements.

The inner beams, which are oriented parallel (Fig. 3c) and perpendicular to the fiber orientation of the plate (Fig. 3e), provide out-of-plane flexural stiffness. A set of distributed uniaxial tension-compression springs is used along the edges of the quadrilateral macro model to simulate direct axial and in-plane flexural loads (Fig. 3d, f). Using a pair of uniaxial springs in the fiber-parallel (Fig. 3h) and fiber-perpendicular directions (Fig. 3i), the shear behavior of the timber plate is simulated. As part of the topology-based modification, the connection area is replaced with a two-node link element. Six springs are embedded in the link element, which is used to simulate the kinematic response of the joint (Fig. 3g). Because the connection area is simulated using the two-node link elements, the primary role of the boundary beams is to stabilize the macro model, and therefore, are modeled as rigid (Fig. 3j). These rigid boundary elements are pin-ended, enabling rotation about their in-plane and out-of-plane axes. The boundary elements also facilitate the distribution of force among the inner beams. Consequently, the behavior of the macro model is independent of the cross-section dimensions of the boundary element. It is worth noting that introducing the zero-length rotation-free springs at the end of the boundary elements is an example of the virtual topology adaptation.

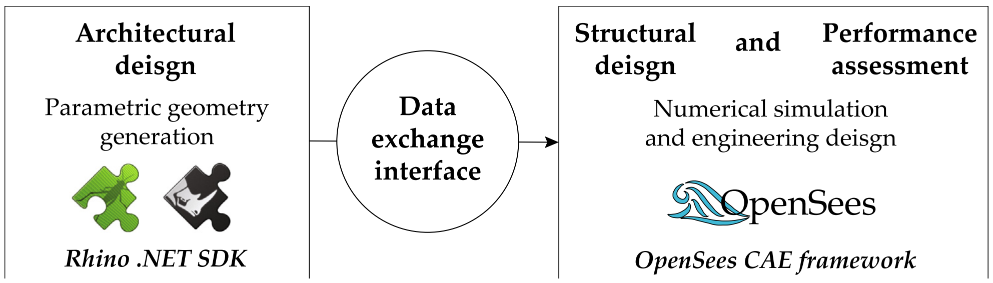


Fig. 2. CAD-to-CAE data transformation using RhinoCommon .NET SDK and OpenSees, and performance assessment of IATP structures.

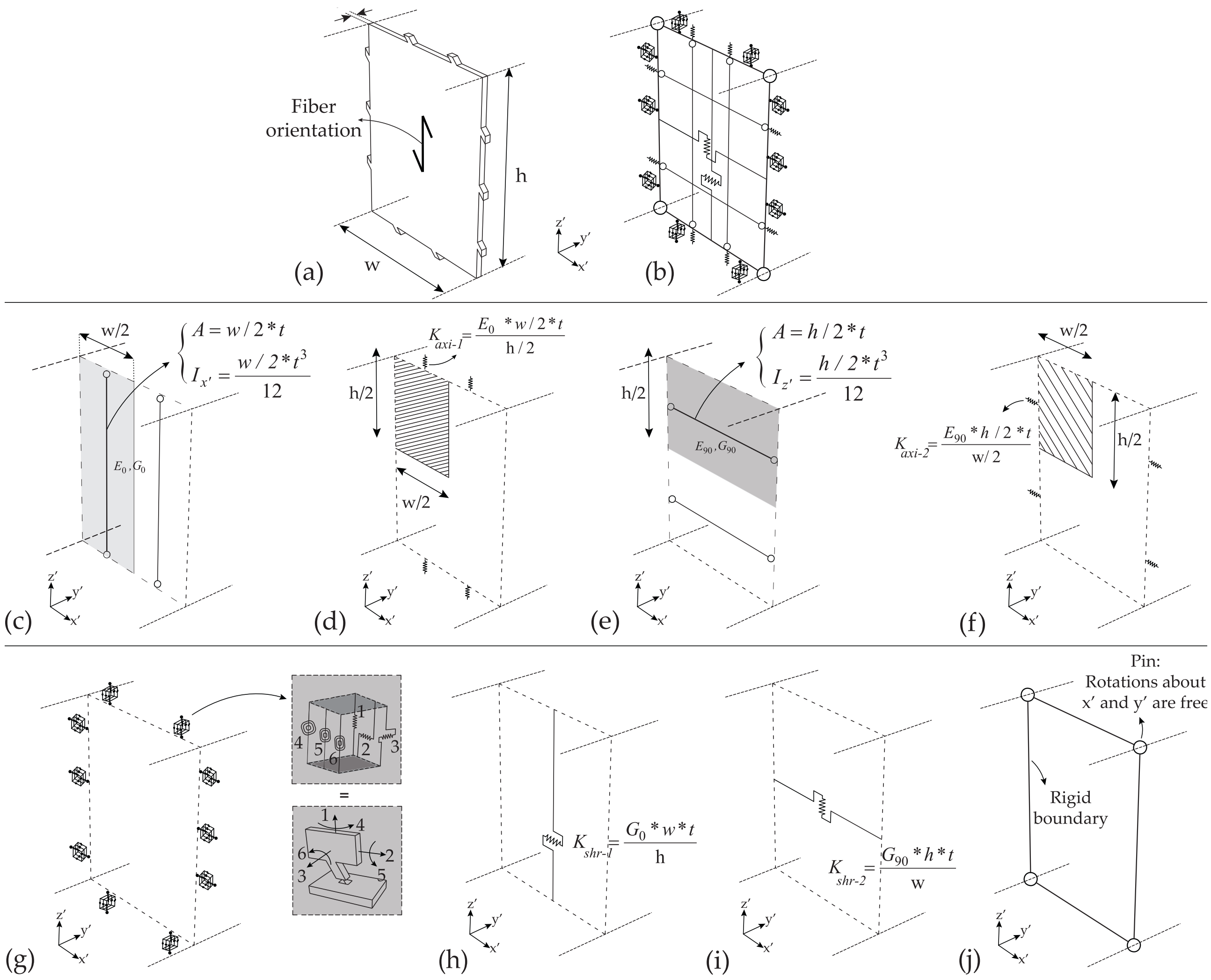

Fiber orientation
h
w
(a)
(b)
w/2
$A = w/2*t$
$I_{x'} = \frac{w/2*t^3}{12}$
E_0, G_0
(c)
$K_{axi-1} = \frac{E_0 * w/2*t}{h/2}$
h/2
w/2
(d)
$A = h/2*t$
$I_{z'} = \frac{h/2*t^3}{12}$
E_{90}, G_{90}
h/2
(e)
w/2
$K_{axi-2} = \frac{E_{90}*h/2*t}{w/2}$
h/2
(f)
1 2 3 4 5 6
=
(g)
$K_{shr-1} = \frac{G_0*w*t}{h}$
(h)
$K_{shr-2} = \frac{G_{90}*h*t}{w}$
(i)
Pin: Rotations about x′ and y′ are free
Rigid boundary
(j)
z′ y′ x′

Fig. 3. Isometric view of (a) the real geometry, (b) the macro model of an IATP element. (c) fiber-parallel beam, (d) uniaxial fiber-parallel spring, (e) fiber-perpendicular beam, (f) uniaxial fiber-perpendicular spring, (g) IMAs, (h) fiber-parallel shear spring element, (i) fiber- perpendicular shear spring, (j) boundary elements (credit: Rezaei Rad et al. [22]).

3. Methodology: automatic CAD-to-CAE data exchange framework

3.1. Description of the IATP system

An IATP system includes the assembly of multiple boxes denoted as C_i in Fig. 4, where i = 1, 2, ..., 8. Each box consists of top, bottom, cross longitudinal, and cross transverse plates, which are labeled as T_i, B_i, CL_i, and CT_i in Fig. 4, respectively. The tenons are located along the perimeter of the T_i, and B_i plates and the slots are located in the perimeter of the CL_i, and CT_i plates. The cross plates are first connected via dovetail shaped IMAs and along the vector $\mathbf{u_i}$ (i = 8 in Fig. 4). The T_i, and B_i plates are then simultaneously connected to the corresponding cross plates along the internal assembly vector $\mathbf{V_{i,int}}$ to form a 4-sided box. This is illustrated in Fig. 4, which shows T_8 and B_8 being connected to CL_8 and CT_8 along the vector $\mathbf{V_{8,int}}$. Each box is then connected to its neighbor with only the TT joints and along the external assembly vector, $\mathbf{V_{i,ext}}$ (i = 8 in Fig. 4).

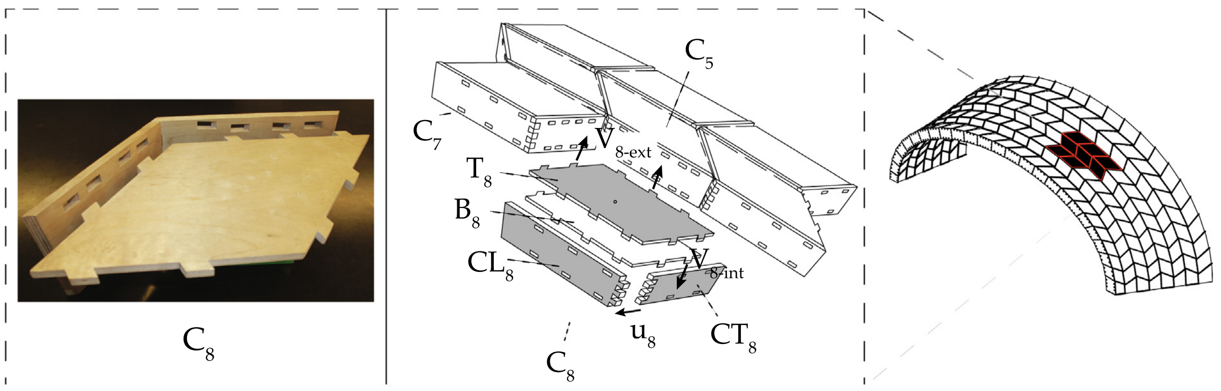


Fig. 4. Description of the Integrally-Attached Timber Plate (IATP) system.

3.2. CAD-to-CAE macro model exchange

The current investigation aims to add another layer to the design of IATPs by incorporating the macroscopic models in the structural analysis of large-scale structures. Toward this goal, an algorithmic framework for generating the CAE macro model from a custom-defined CAD 3D model is introduced. The framework consists of multiple steps. First, an algorithm is introduced to compute the perimeter polygon associated with the mid-surface of each quadrilateral 3D timber plate element, convert the plate from a 3D solid to a 1D polygon, and assign a unique tag to each polygon (Fig. 5a-to-b). The details of this step are presented in Section 3.2.1. Next, by retrieving the neighboring polygons from the mesh, 1D line elements are defined and used to represent the connection between adjacent polygons. Section 3.2.2 discusses the details of this step. The data associated with the polygons and connections and their ID tags are then stored as a list. In the next step, a new algorithm is formulated to generate the components of the macro model from the list of data generated in the previous steps (Fig. 5b-to-c). The formulation of this algorithm is detailed in Section 3.2.3. Since a typical IATP structure consists of hundreds of timber plates with thousands of joints, assigning the relevant mechanical properties to each beam/spring element of the macro model would not be possible without a systematic identification protocol. Therefore, Section 3.2.4 formulates an indexing algorithm to assign a unique tag to each component of the macro model.

According to the proposed workflow, and given the CAD 3D geometry of an IATP structure, a data structure is designed to encapsulate the geometry of the macro model components with the corresponding ID tags, and their mechanical properties. In other words, the data structure consists of the coordinates of beam and spring nodes, each of which is assigned a unique index. The material and cross-sectional propertiesassociated with each beam and spring are also included in this data structure. The output provided for the CAE macro model (Fig. 5c) follows the syntax of OpenSees, an object-oriented open-source software for structural modeling and response simulation. As such, the data structure formulated within the CAD environment can generate the data

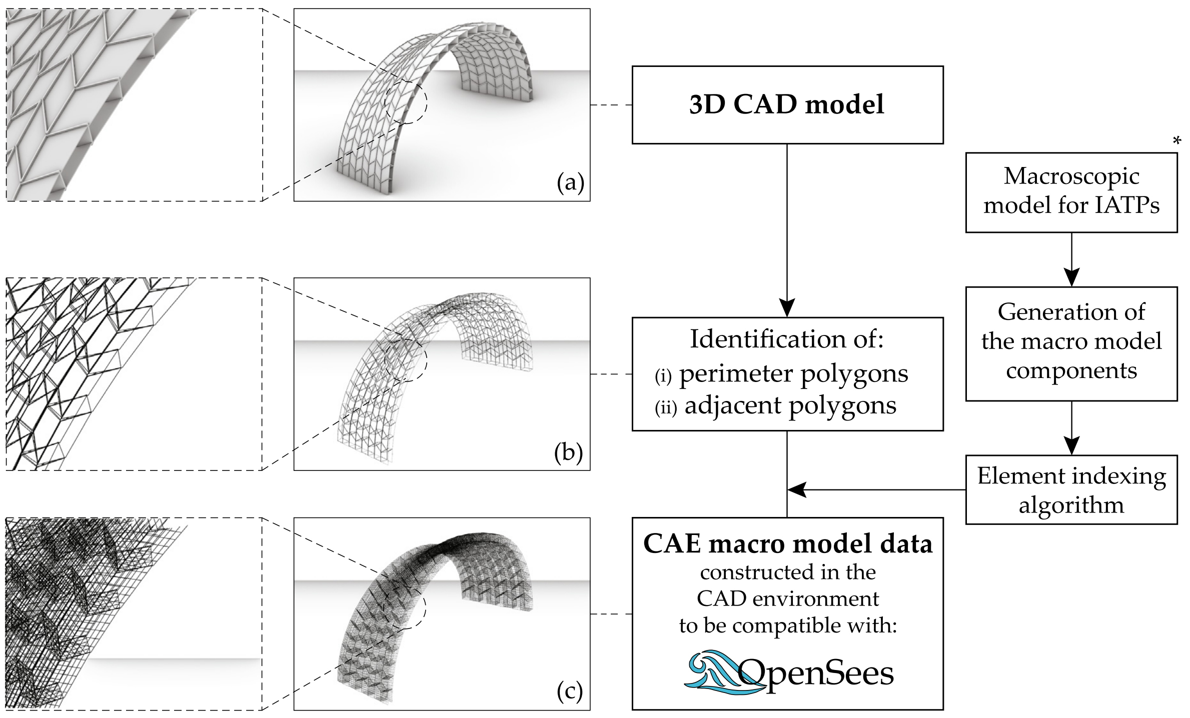


Fig. 5. An overview of the automatic CAD-to-CAE data exchange framework for IATPs (Figure note: * The macroscopic model for IATPs is originally developed by Rezaei Rad et al. [22]).

required for structural analysis. This, in particular, enables the user to work in the CAD modeling environment while generating the data required for a CAE platform.

The identification of the perimeter polygons and IMAs and associated indexing algorithms (Fig. 5b), as well as the automatic generation of the macro model components and associated indexing (Fig. 5c), are introduced to Rhinoceros 3D [6] as plug-ins using Application Programming Interfaces (APIs). While APIs within Rhinoceros 3D are available in different programming languages such as C++, C#, VB, Python, and VBScript, they use the same .NET RhinoCommon library [20]. Grasshopper 3D [7], which is a graphical algorithm editor within Rhinoceros 3D that enables visual programming, was used to edit the plug-ins and components.

3.2.1. CAD-to-CAE Exchange, Module 1: Identification of perimeter polygons and associated indexing

The identification of the perimeter polygon associated with each timber plate is the primary step in the CAD-to-CAE data exchange framework. This step is central to the CAE macro model development because the inner beams are generated by dividing the perimeter polygons into equal segments along the fiber-parallel and fiber-perpendicular directions and adding inner lines at the segment interfaces. From the CAD 3D model (Fig. 6a), the mid-surface plane of each timber plate is recalled (Fig. 6b). Next, using fabrication contours available from the CAD 3D model, the basic mesh edges are retrieved (Fig. 6c). Using the *Intersection.LinePlane* method available in the RhinoCommon library [20], the edges are intersected with the mid-surface planes. The four corner nodes of the quadrilateral mid-surface are identified (Fig. 6d). Accordingly, a polygon element crossing at these four nodes is generated (Fig. 6e). The polygon element is represented by segmented lines.

The 3D CAD model assigns a unique identifier index to the following components: strips, boxes, and plates, which are schematically shown in Fig. 7. The most complex IATP structures are those with free-form geometries (Fig. 7a). In these situations, the structure is constructed by assembling multiple strips (Fig. 7b). Each strip is an assembly of multiple boxes (Fig. 7c) and each box consists of four IMA-connected timber plates (Fig. 7d). This data-structure enables the user to readily find the position of a timber plate by concatenating three numbers. Therefore, each polygon is recognized in 3D space with the following concatenated three number sequences: $S_iB_jP_k$, where S_i, B_j, and P_k, correspond to ith strip, jth box, and kth plate. The algorithm can also accommodate simpler geometries such as beams with hollow cross sections, in which case, the first two types of components (strips and boxes) are not used and only the plate identifier is employed. This will be discussed in Section 4.1.

After identifying the perimeter polygon corresponding to each timber plate, the associated identifier is retrieved from the CAD model (Fig. 6f), and both the polygon geometry and the identifier are collected in a sorted list format using Grasshopper DataTree [28] (Fig. 6g). The list is then used in the module described in Section 3.2.3 to construct the macro model components. The pseudocode associated with this module is provided in Appendices 1.2 and 1.3.

3.2.2. CAD-to-CAE exchange, Module 2: Identification of IMAs and associated indexing

The second module of the CAD-to-CAE exchange uses 1D line elements to represent the connection between neighboring polygons. These lines serve as mechanical joints in the macro model according to Fig. 3g. To simplify the IMA geometry, each connection is represented by two points, one in the plate containing the tenon and the other in the plate containing the slot. To compute these points, the mid-surface associated with timber plates is recalled from the CAD 3D model (Fig. 8a, b). Also, the data associated with the neighboring plates are retrieved from the mesh. Next, a plane perpendicular to the mid-surface of each plate is introduced to the middle of the tenon, and the intersection point is identified (Fig. 8c). This point represents the tenon of the IMA. Then, by using the *Line.ClosestPoint* method available in the RhinoCommon library [20], the tenon's mate, which is located in the neighboring polygon, is identified (Fig. 8d). Next, the line element is defined by connecting the two polygons (Fig. 8e). This modeling technique is in accordance with the assumption made for the macroscopic model where each IMA is simulated using a two-node link.

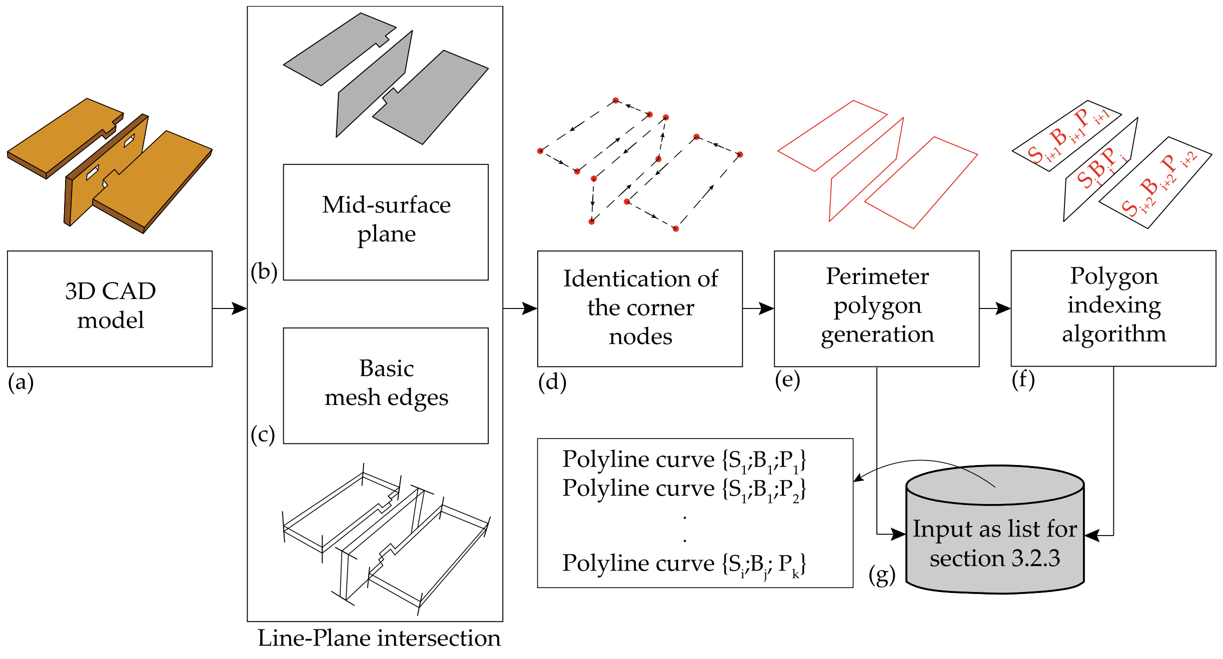

3D CAD model
(a)
Mid-surface plane
(b)
Basic mesh edges
(c)
Line-Plane intersection
Identication of the corner nodes
(d)
Perimeter polygon generation
(e)
Polygon indexing algorithm
(f)
S_{i+1}B_{i+1}P_{i+1}
S_iB_iP_i
S_{i+2}B_{i+2}P_{i+2}
Polyline curve {S_1;B_1;P_1}
Polyline curve {S_1;B_1;P_2}
Polyline curve {S_i;B_j; P_k}
Input as list for section 3.2.3
(g)

Fig. 6. Perimeter polygon identification algorithm.

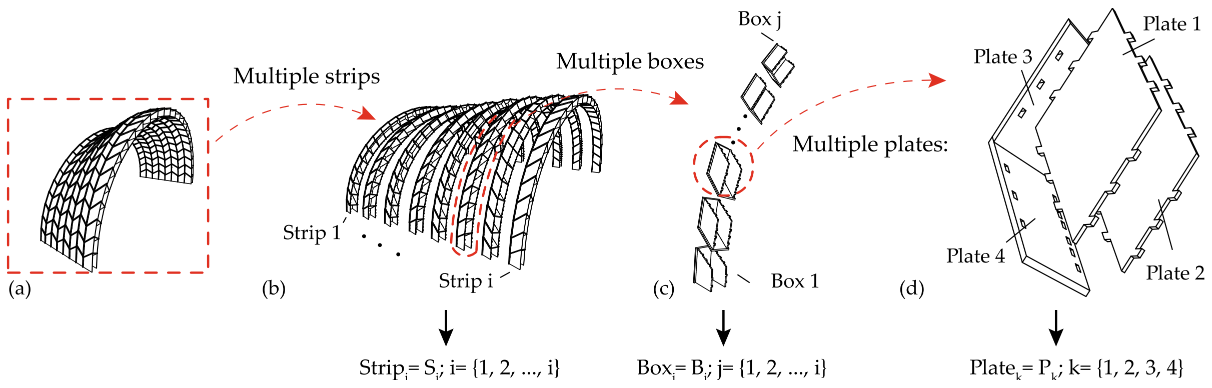


Fig. 7. A schematic representation of the polygon indexing algorithm.

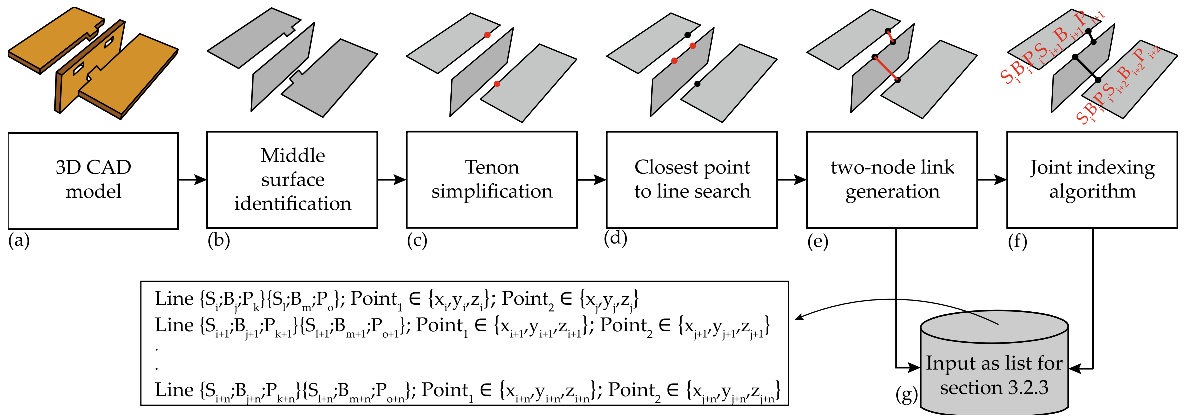

3D CAD model
Middle surface identification
Tenon simplification
Closest point to line search
two-node link generation
Joint indexing algorithm
(a)
(b)
(c)
(d)
(e)
(f)
(g)
Input as list for section 3.2.3
$S_iB_jP_kS_lB_mP_o$
$S_{i+1}B_{j+1}P_{k+1}$
$S_{i+2}B_{j+2}P_{k+2}$
Line {S_i;B_j;P_k}{S_l;B_m;P_o}; Point$_1$ ∈ {x_i,y_i,z_i}; Point$_2$ ∈ {x_j,y_j,z_j}
Line {S_{i+1};B_{j+1};P_{k+1}}{S_{l+1};B_{m+1};P_{o+1}}; Point$_1$ ∈ {x_{i+1},y_{i+1},z_{i+1}}; Point$_2$ ∈ {x_{j+1},y_{j+1},z_{j+1}}
.
.
Line {S_{i+n};B_{j+n};P_{k+n}}{S_{l+n};B_{m+n};P_{o+n}}; Point$_1$ ∈ {x_{i+n},y_{i+n},z_{i+n}}; Point$_2$ ∈ {x_{j+n},y_{j+n},z_{j+n}}

Fig. 8. Identification of wood-wood connections and the corresponding simplifcation process using 1D line elements.

After identifying the perimeter polygon, an indexing algorithm is introduced to assign a unique identifier to each 1D line element (Fig. 8f). Thousands of joints exist in a typical IATP structure and each joint has specific mechanical characteristics based on its geometry and fiber orientation. Therefore, assigning a unique ID to each joint is central to the construction of the corresponding numerical model.

The most convenient way to assign a unique identifier to each 1D line element is to build a new layer of information using the existing perimeter polygon IDs. Given that a joint primarily serves to connect two adjacent polygons, the indexing algorithm is formulated to include the indices of the two connected polygons. Additionally, since two neighboring polygons can be associated with more than one joint, another counter is added to the joint identifier. In other words, the indexing algorithm includes a seven number sequence where the first six identify the strip, box, and plate number of the first and second polygons and the 7th number, Jnt_m, identifies the mth joint. Accordingly, the index of the mth joint element which connects two adjacent polygons in 3D space becomes $S_iB_jP_kS_xB_yP_zJnt_m$. Fig. 9 shows the logic of the indexing algorithm.

Finally, the geometry of the generated lines and associated indices are collected in a list format using Grasshopper DataTree (Fig. 8g) and used in the module described in Section 3.2.3. The pseudocode associated with the generation of the 1D lines is provided in Appendix 1.4.

Generally, the orientation of timber fibers is parallel to that of the length of each polygon (vector $\mathbf{e_1}$ in Fig. 10). The local coordinate system for each joint is shown in Fig. 10. Given the ith timber plate, the direction of the plate edge and the vector normal to the plate are defined as $\mathbf{e_1}$ and $\mathbf{n_i}$ in Fig. 10, respectively. Accordingly, the orientation of each joint is determined by computing the cross product of the vectors $\mathbf{e_1}$, and $\mathbf{n_i}$ ($\mathbf{e_1}\times\mathbf{n_i}$).

3.2.3. CAD-to-CAE exchange, Module 3: Generation of the macro model components

A generic algorithm, shown in Fig. 11, is introduced in this section to generate the components of the macro model associated with a custom-defined timber plate within an IATP structure (Fig. 3). The main steps are as follows:

(1) Using the list of input data containing the perimeter polygons and associated indices derived from Section 3.2.1, the four corner nodes are retrieved (Fig. 11b in green) and two additional nodes are generated at each corner (Fig. 11b in red). These additional nodes have the same coordinates as the corresponding corner node, and they will be used to construct zero-length rotationally-free elements in the macro model. This is in accordance with the modeling requirement of the boundary elements in the macro model (Fig. 3j).

(2) The inner beams of the macro model are constructed by introducing lines with equal distances along the fiber-parallel (Fig. 11c) and fiber-perpendicular directions (Fig. 11d). A virtual polyline connecting the four corner nodes is defined and the length and width of each timber plate are determined. Recalling that the fiber-parallel and fiber-perpendicular directions correspond to the length and width of the timber plate, respectively, these two directions are divided into equal segments. Line elements are then added at the segment interfaces to serve as the inner

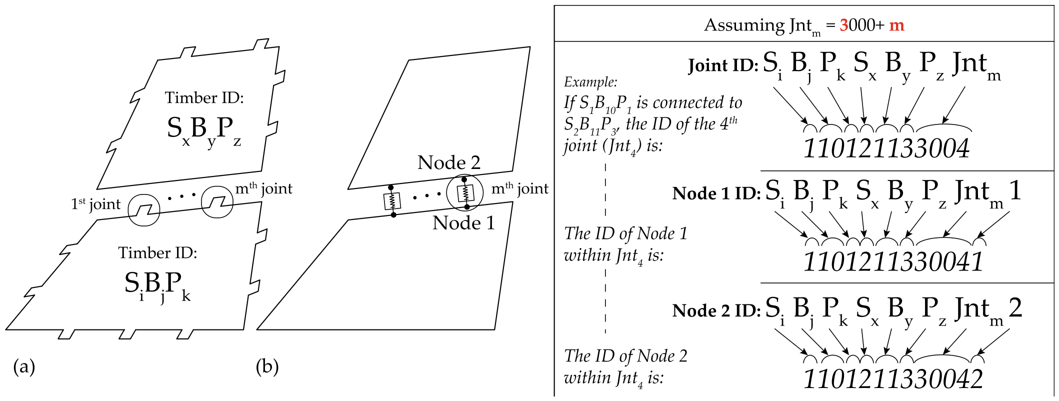

Timber ID:
$S_xB_yP_z$
1st joint
mth joint
Timber ID:
$S_iB_jP_k$
(a)
Node 2
Node 1
mth joint
(b)
Assuming Jnt_m = 3000+ m
Joint ID: $S_i\ B_j\ P_k\ S_x\ B_y\ P_z\ Jnt_m$
Example:
If $S_1B_{10}P_1$ is connected to $S_2B_{11}P_3$, the ID of the 4th joint (Jnt_4) is:
110121133004
Node 1 ID: $S_i\ B_j\ P_k\ S_x\ B_y\ P_z\ Jnt_m\ 1$
The ID of Node 1 within Jnt_4 is:
1101211330041
Node 2 ID: $S_i\ B_j\ P_k\ S_x\ B_y\ P_z\ Jnt_m\ 2$
The ID of Node 2 within Jnt_4 is:
1101211330042

Fig. 9. Joint indexing algorithm.

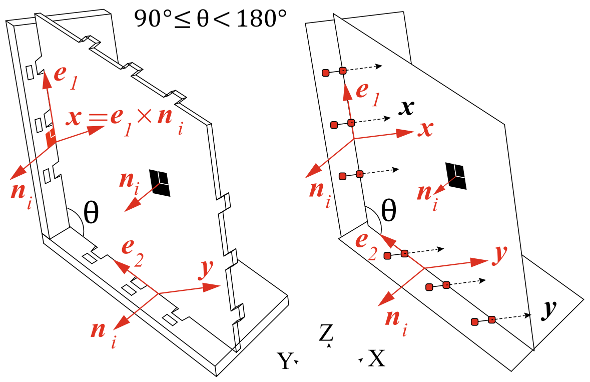


Fig. 10. Identifying the local coordinate system for timber joints (IMAs).

beams for the macro model, each with a unique local coordinate system. The cross-sectional properties of each inner beam element are computed based on its tributary area, and the material properties are assigned based on its fiber orientation.

(3) Computing the geometric transformation of the elements (Fig. 11e) and using the wood orthotropic properties (Fig. 11f), the data associated with the fiber-parallel (Fig. 11g) and fiber-perpendicular beam elements and nodes (Fig. 11h) is stored in a text file with .tcl format using the OpenSees syntax.

(4) While defining the beams and springs using the OpenSees syntax, the algorithm assigns each element a unique identification index. The logic behind the indexing algorithm is explained in Section 3.2.4.

(5) Using the list of input data containing the joints and associated indices derived from Section 3.2.2 (Fig. 11i), the end nodes of the joints are retrieved (Fig. 11j). The mechanical properties of the IMAs derived from physical experiments [11,12,21] are assigned to the joints (Fig. 11m). Using the OpenSees element library, the *two-node link element* with six discrete uncoupled springs is employed to simulate the behavior of the IMAs (Fig. 11n). The elements are stored in a text file with .tcl format and the OpenSees syntax (Fig. 11o), each with the unique identification tag explained in Section 3.2.2.

(6) The boundary elements of the macro model are generated by connecting the corner nodes, corner release nodes, end nodes of the fiber-parallel and fiber-perpendicular inner beams, and end nodes of the joints (Fig. 11k). A uniaxial elastic material with infinite stiffness is assigned to the boundary elements. These elements are then stored in a text file with .tcl format using the OpenSees syntax (Fig. 11l). While constructing the boundary elements, the algorithm assigns each element the unique identification index explained in Section 3.2.4.

(7) The .tcl files generated in the previous steps are compiled, and the CAE macro model is generated. Given that a systematic identification protocol is established, the elements that are subjected to loads are parametrically defined. Analysis handlers are specified, and the CAE model is used for structural analysis in OpenSees (Fig. 11p).

3.2.4. CAD-to-CAE exchange, Module 4: Nodes and elements indexing

The algorithm in this module automatically generates the components of the macro model associated with an IATP element and assigns them a unique identification index. This is central to the CAD-to-CAE data transformation framework because to construct the OpenSees numerical model, each element of the macro model and the corresponding nodes should be indexed with an identifier. In the algorithm, (1) Crn_m is the tag that corresponds to the mth corner node shown in Fig. 11b in green, (2) $Pcrn_m$ is the tag that corresponds to the mth corner node used to construct the rotationally-free zero-length element (Fig. 11b in red), (3) Trs_m is the tag that corresponds to the mth fiber-perpendicular inner beam and its associated nodes (Fig. 11c),(4) Lng_m is the tag that corresponds to the mth fiber-parallel inner beam and its associated nodes (Fig. 11d), and (5) Cnt_m is the tag that corresponds to the mth boundary element. These tags are then linked to the ones developed for each polygon.

For each plate, the first three numbers are associated with the polygon tag, described in Section 3.2.1, and the fourth number represents the macro model component type, as discussed in the previous paragraph. Accordingly, the index of a corner node and the nodes used for the rotationally-free zero-length element becomes $S_iB_jP_kCrn_m$ (Fig. 12a) and $S_iB_jP_kPcrn_m$ (Fig. 12b), respectively. The index of the fiber-parallel inner beam, fiber-perpendicular inner beam, and boundary element of the macro model becomes $S_iB_jP_k$-Lng_m (Fig. 12c), $S_iB_jP_kTrs_m$ (Fig. 12d), and $S_iB_jP_kCnt_m$ (Fig. 12e), respectively. The pseudocode associated with the generation of the macro model components and indexing algorithm is provided in Appendix 1.4. The algorithm is written in the Rhino3D CAD environment using the *RhinoScriptSyntax* library [29] which allows the user to create, access, and manipulate a list of points, vectors, lines, surfaces, and non-uniform rational basis spline (NURBS).

Assigning a unique tag to each component of the macro model within a timber plate has multiple advantages. First, it minimizes the number of manual operations and allows the rapid identification of each component . Given that a typical IATP structure con-

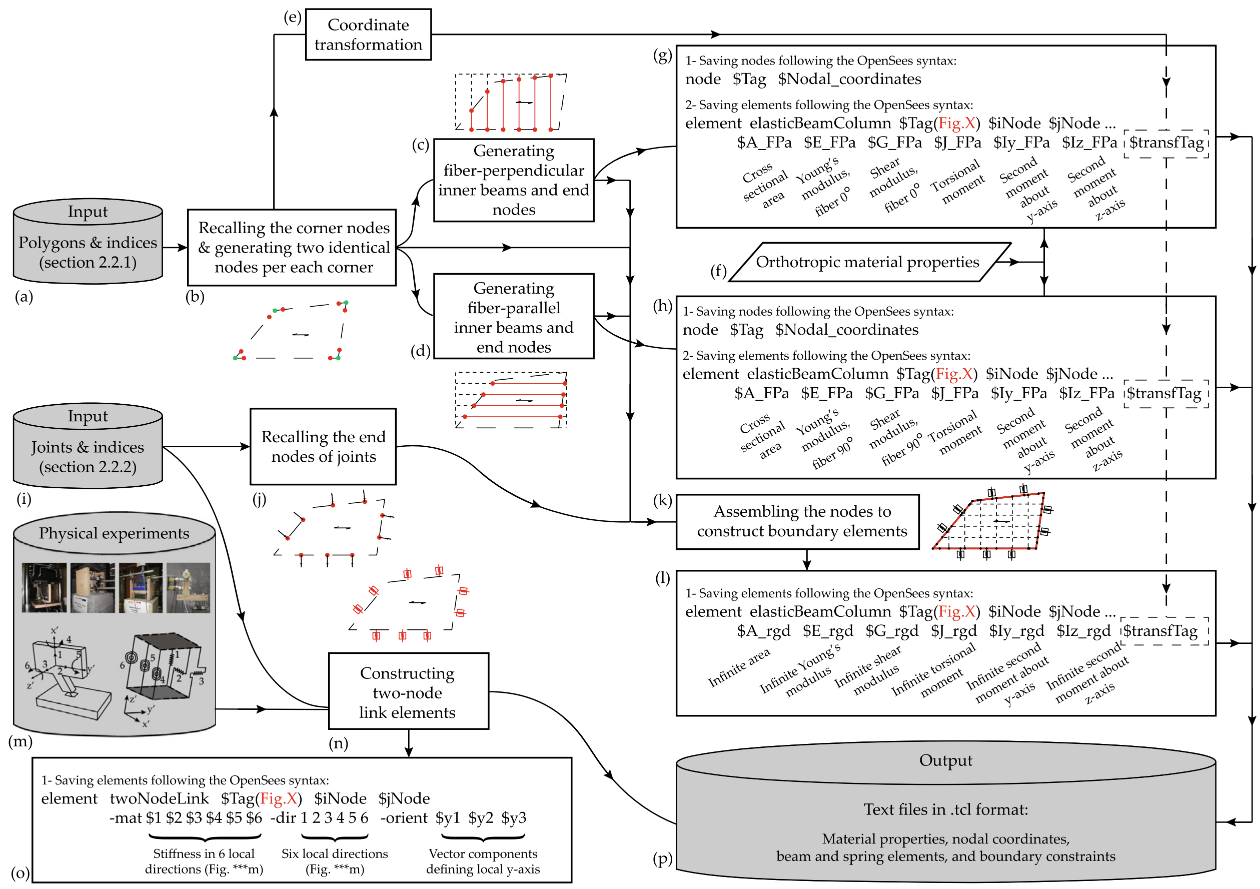

(e) Coordinate transformation
(g) 1- Saving nodes following the OpenSees syntax:
node $Tag $Nodal_coordinates
2- Saving elements following the OpenSees syntax:
element elasticBeamColumn $Tag(Fig.X) $iNode $jNode ...
$A_FPa $E_FPa $G_FPa $J_FPa $Iy_FPa $Iz_FPa $transfTag
Cross sectional area
Young's modulus, fiber 0°
Shear modulus, fiber 0°
Torsional moment
Second moment about y-axis
Second moment about z-axis
(c) Generating fiber-perpendicular inner beams and end nodes
Input
Polygons & indices (section 2.2.1)
(a)
(b) Recalling the corner nodes & generating two identical nodes per each corner
(f) Orthotropic material properties
(d) Generating fiber-parallel inner beams and end nodes
(h) 1- Saving nodes following the OpenSees syntax:
node $Tag $Nodal_coordinates
2- Saving elements following the OpenSees syntax:
element elasticBeamColumn $Tag(Fig.X) $iNode $jNode ...
$A_FPa $E_FPa $G_FPa $J_FPa $Iy_FPa $Iz_FPa $transfTag
Cross sectional area
Young's modulus, fiber 90°
Shear modulus, fiber 90°
Torsional moment
Second moment about y-axis
Second moment about z-axis
Input
Joints & indices (section 2.2.2)
(i)
(j) Recalling the end nodes of joints
(k) Assembling the nodes to construct boundary elements
Physical experiments
(m)
(l) 1- Saving elements following the OpenSees syntax:
element elasticBeamColumn $Tag(Fig.X) $iNode $jNode ...
$A_rgd $E_rgd $G_rgd $J_rgd $Iy_rgd $Iz_rgd $transfTag
Infinite area
Infinite Young's modulus
Infinite shear modulus
Infinite torsional moment
Infinite second moment about y-axis
Infinite second moment about z-axis
(n) Constructing two-node link elements
(o) 1- Saving elements following the OpenSees syntax:
element twoNodeLink $Tag(Fig.X) $iNode $jNode
-mat $1 $2 $3 $4 $5 $6 -dir 1 2 3 4 5 6 -orient $y1 $y2 $y3
Stiffness in 6 local directions (Fig. ***m)
Six local directions (Fig. ***m)
Vector components defining local y-axis
(p) Output
Text files in .tcl format:
Material properties, nodal coordinates, beam and spring elements, and boundary constraints

Fig. 11. Generation of the macro model components and data storage.

sists of hundreds of timber plates with thousands of joints, assigning the relevant mechanical properties to each beam/spring element of the macro model would not be possible without a systematic identification protocol. Furthermore, using the indexing algorithm, the compatibility between the CAD and CAE data structures is maintained in a single platform. Moreover, after assigning the indices to the elements and nodes, the loaded elements within a structure and boundary conditions are readily identified.

4. Verification and case studies

In this section, the algorithm is used to construct the CAD-to-CAE macro model and analyze prefabricated IATP structures with standard (Section 4.1) and free-form (Section4.2) geometries. FE models as well as the results from recent experiments performed on full-scale prototypes are used to verify the proposed framework.

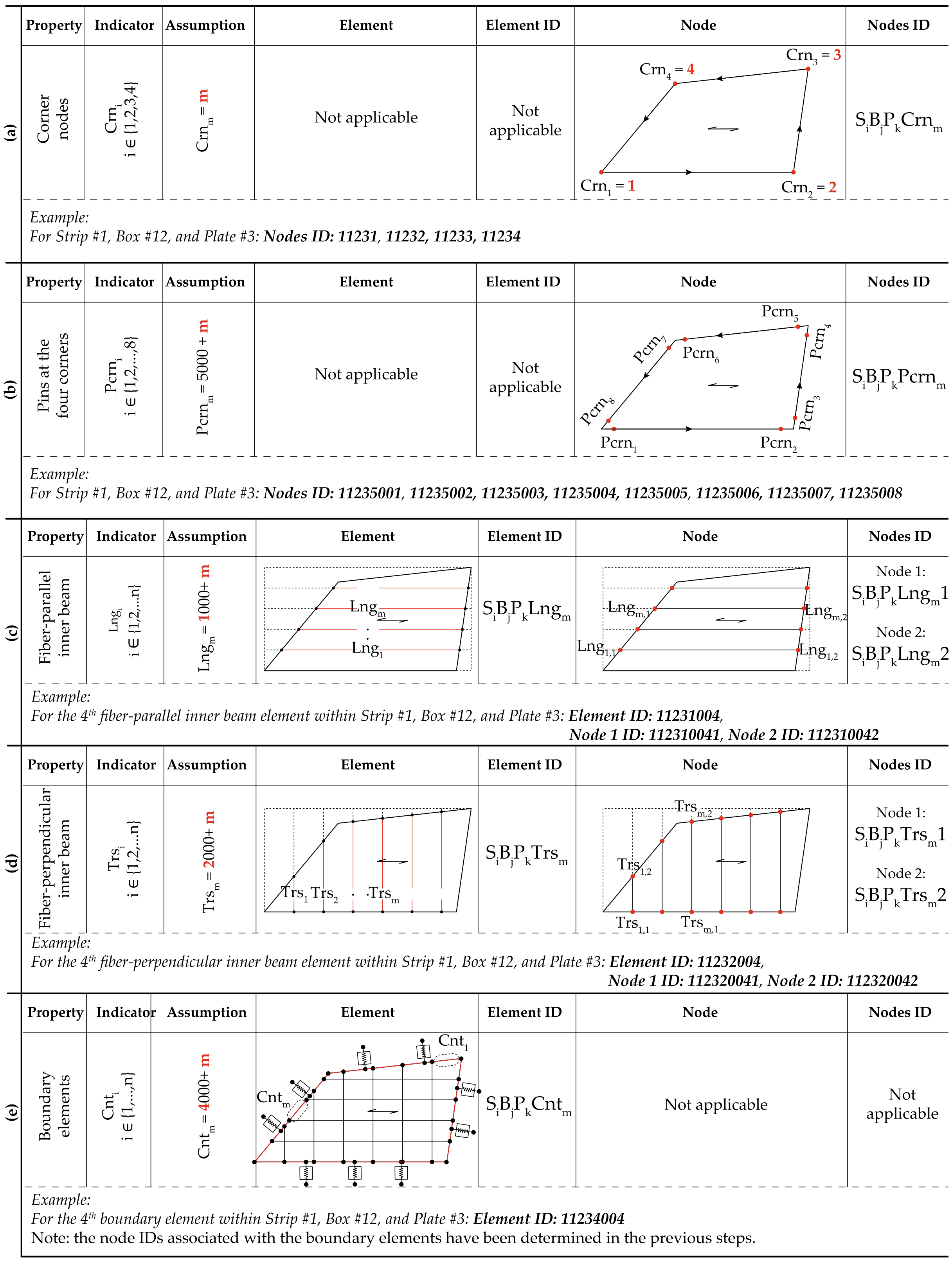
4.1. Standard prefabricated IATPs

Using the IMA technique, prefabricated timber plates have been used to construct structural components with standard shapes. Gamerro et al. [30] introduced the construction of large span standard beams with hollow cross sections using segmented timber elements. The structural element consists of either one or two layers of timber plates for each web or flange. Fig. 13a, b shows single- and double-layer assemblies with hollow cross sections, respectively.

4.1.1. Generation of the CAD/CAE macro model

The algorithmic CAD-to-CAE data exchange framework is used to generate the CAE macro model corresponding to an assembly of timber plates with standard geometry, which is then verified against the results of a four-point bending test performed on a full-scale assembly. Three replicates of the assembly, shown in Fig. 14a, were tested by Gamerro et al. [30]. Similar to Fig. 13b, the assembly consists of two layers of timber plates for each flange and web. Oriented Strand Board (OBS) is used for the two webs. For the flanges, the inner and outer layers consist of Kerto Laminated Veneer Lumber (LVL) and OSB boards, respectively. The dimensions of the timber plates and the test setup designed by Gamerro et al. [30] are shown in Fig. 14a–f. The geometry of the CAD model is re-generated in this study using the details provided in Figure A1 in [30], however, the discontinuity between the timber plates within each flange or web is not considered to simplify the CAD model. Furthermore, the small transvere plates used in the reference model [30] were not included in the re-generated CAD model.

Within the context of the CAE macro model generation framework shown in Fig. 5, the assembly shown in Fig. 14a is analogous to a box component in the previously described polygon identification algorithm. As per Sections 3.2.1 and 3.2.2, the perimeter polygons associated with the rectangular plates are first identified, a unique tag is assigned to each one, and the connections between the timber plates are then defined. To simplify the modeling process, the two webs are merged into a single layer with equivalent cross-section properties. Next, and according to Sections 3.2.3 and 3.2.4, the components of the macro model are generated for each polygon, a unique tag is defined for each component, and the associated mechanical properties are computed. Fig. 14g–n shows the corresponding CAE macro model.



	Property	Indicator	Assumption	Element	Element ID	Node	Nodes ID
(a)	Corner nodes	Crn_i $i \in \{1,2,3,4\}$	$Crn_m = m$	Not applicable	Not applicable		$S_iB_jP_kCrn_m$

Example:
For Strip #1, Box #12, and Plate #3: ***Nodes ID: 11231, 11232, 11233, 11234***

	Property	Indicator	Assumption	Element	Element ID	Node	Nodes ID
(b)	Pins at the four corners	$Pcrn_i$ $i \in \{1,2,\ldots,8\}$	$Pcrn_m = 5000 + m$	Not applicable	Not applicable		$S_iB_jP_kPcrn_m$

Example:
For Strip #1, Box #12, and Plate #3: ***Nodes ID: 11235001, 11235002, 11235003, 11235004, 11235005, 11235006, 11235007, 11235008***

	Property	Indicator	Assumption	Element	Element ID	Node	Nodes ID
(c)	Fiber-parallel inner beam	Lng_i $i \in \{1,2,\ldots n\}$	$Lng_m = 1000 + m$		$S_iB_jP_kLng_m$		Node 1: $S_iB_jP_kLng_m1$ Node 2: $S_iB_jP_kLng_m2$

Example:
For the 4th fiber-parallel inner beam element within Strip #1, Box #12, and Plate #3: ***Element ID: 11231004,***
Node 1 ID: 112310041, Node 2 ID: 112310042

	Property	Indicator	Assumption	Element	Element ID	Node	Nodes ID
(d)	Fiber-perpendicular inner beam	Trs_i $i \in \{1,2,\ldots n\}$	$Trs_m = 2000 + m$		$S_iB_jP_kTrs_m$		Node 1: $S_iB_jP_kTrs_m1$ Node 2: $S_iB_jP_kTrs_m2$

Example:
For the 4th fiber-perpendicular inner beam element within Strip #1, Box #12, and Plate #3: ***Element ID: 11232004,***
Node 1 ID: 112320041, Node 2 ID: 112320042

	Property	Indicator	Assumption	Element	Element ID	Node	Nodes ID
(e)	Boundary elements	Cnt_i $i \in \{1,\ldots,n\}$	$Cnt_m = 4000 + m$		$S_iB_jP_kCnt_m$	Not applicable	Not applicable

Example:
For the 4th boundary element within Strip #1, Box #12, and Plate #3: ***Element ID: 11234004***
Note: the node IDs associated with the boundary elements have been determined in the previous steps.

Fig. 12. Automatic indexing algorithm for the macro model components.

For each timber plate used in the flanges, the inner beam elements of the corresponding macro model are spaced at 400.0 mm and 150.0 mm along the fiber-parallel and fiber-perpendicular directions, respectively. For the web plates, the inner beams of the corresponding macro model are spaced at 150.0 mm along both the fiber-parallel and fiber-perpendicular directions. After computing the tributary area associated with each inner beam, the cross-sectional properties are determined. The

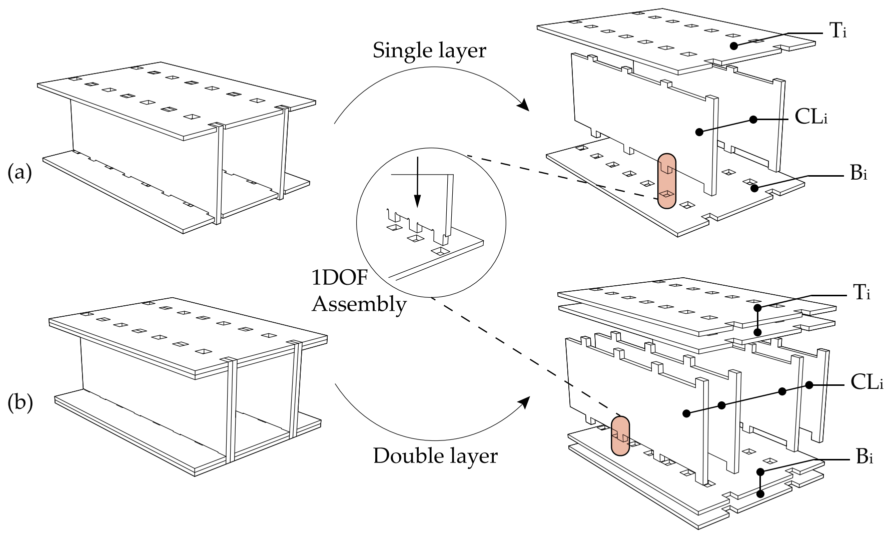


Fig. 13. Standard beams using single and double-layer timber plates with IMAs.

mechanical properties of the timber material (Table 1) are assigned to each element based on its fiber orientation. Additionally, the properties of the shear and tension-compression springs are computed and assigned to the respective elements. Moreover, the load-deformation response of the IMAs, investigated by Gamerro et al. [31], is assigned to the two-node link elements. Fig. 14k–l shows the section size of the inner beam elements, as well as the stiffness values for the uniaxial tension-compression, shear, and the IMA springs.

Using the proposed CAD-to-CAE data exchange, the mean force values and the corresponding stresses of a given cross section can be computed. Given the timber plate cross-section (i.e. section D-D in Fig. 14l), the corresponding mean force is computed by averaging the forces of the inner beams. Additionally, the corresponding mean stress is computed by averaging the stress values of each inner beam element. This is detailed in Fig. 14m.

4.1.2. Results and discussions

The assembly shown in Fig. 14a was subjected to a static load at its mid-span [30]. The vertical mid-span displacement of the assembly was recorded using a Linear Variable Differential Transducer (LVDT). The corresponding CAE macro model is generated and subjected to the same loading protocol used in the physical experiment. The load-displacement curves obtained from the macro model and the physical specimen are shown in Fig. 15. The performance of the assembly using the macro model is evaluated up to the peak strength. This corresponds to the maximum strength of the physical assembly, and according to Fig. 15, it is equal to 33.2 kN.

The results in Fig. 15 show that the responses obtained from the macro model are similar to the experimental load-deflection behavior. The initial stiffness obtained from the macro model and the physical specimen is computed from the associated load-deflection curves in Fig. 15. The average difference between the stiffness from the macro model ($K_{Macro} = 730.46N/mm$) and the physical experiment ($K_{Test} = 822.4$N/mm) is 11.1%, which represents an acceptable degree of accuracy. The adequate performance of the macro-model is attributed to (i) the symmetry of the beam assembly and the regular configuration of the constituent timber plates (which are rectangular), and (ii) the fact that the overall performance of the system is primarily governed by the IMAs. More specifically, since the mechanical properties of the IMAs have been carefully characterized in Gamerro et al. [30][31] and used in the macro model, the predicted load-deformation behavior is comparable to the measurements from the physical experiment. The accuracy of the model can be improved by including the compression and embedment behavior of the IMAs in the macroscopic behavior. Moreover, according to Table 4 in Gamerro et al. [30], a brittle tensile failure was observed in the outer layer of the bottom flange, where the maximum tensile stress was 5.36 MPa. According to the results obtained from the macro model, the mean tensile stress at the maximum strength, which corresponds to the failure state, is 4.93 MPa in the same outer layer of the bottom flange. This represents an acceptable degree of accuracy in computing the state of stress in the timber elements using the macro model.

Using the proposed CAD-to-CAE data exchange, the CAD geometry is translated to the corresponding CAE macro model with an Intel Core i7-4800MQ CPU @2.7 GHz and 16 GB of RAM machine. The time required to generate the CAE macro model is approximately 3 s with 15% of the CPU and 280 MB of memory utilized. The CAE model analysis runtime is approximately 2 s, using 0.4% of the CPU and 7.8 MB of memory.

4.2. Spatial free-form IATPs

4.2.1. Description of the system and generation of the CAD/CAE macro model

Using the IMA technique, prefabricated timber plates are assembled to construct free-form large-scale IATP structures. Detailed documentation of the assembly and construction processes and geometry generation can be found in Rezaei Rad et al. [11,12], Nguyen et al. [15], and Robeller et al. [14]. In this section, the proposed algorithmic framework is used to generate and analyze the CAE macro model associated with a double-layered double-curved prototype that was constructed by ANNEN SA in Manternach, Luxembourg [34]. The entire structure consists of 23 free-form and separate IATP arches, each with a specific and custom-defined geometry. The span and height of the arches range between 22.5 to 53.7 m, and 6.0 to 9.0 m, respectively. Fig. 16a shows the 23 arches along with a schematic representation of the design target surface (denoted as "A" and shown for arches 1 to 5), and the design pattern and the surface discretization (denoted as "B" and shown for arches 6 to 10). The CAD 3D model (denoted as "C" and shown for arches 11 to 15), the CAD 2D model with mid-surface elements (denoted as "D" and shown for arches 16 to 19), and the CAE macro model associated with arches 20 to

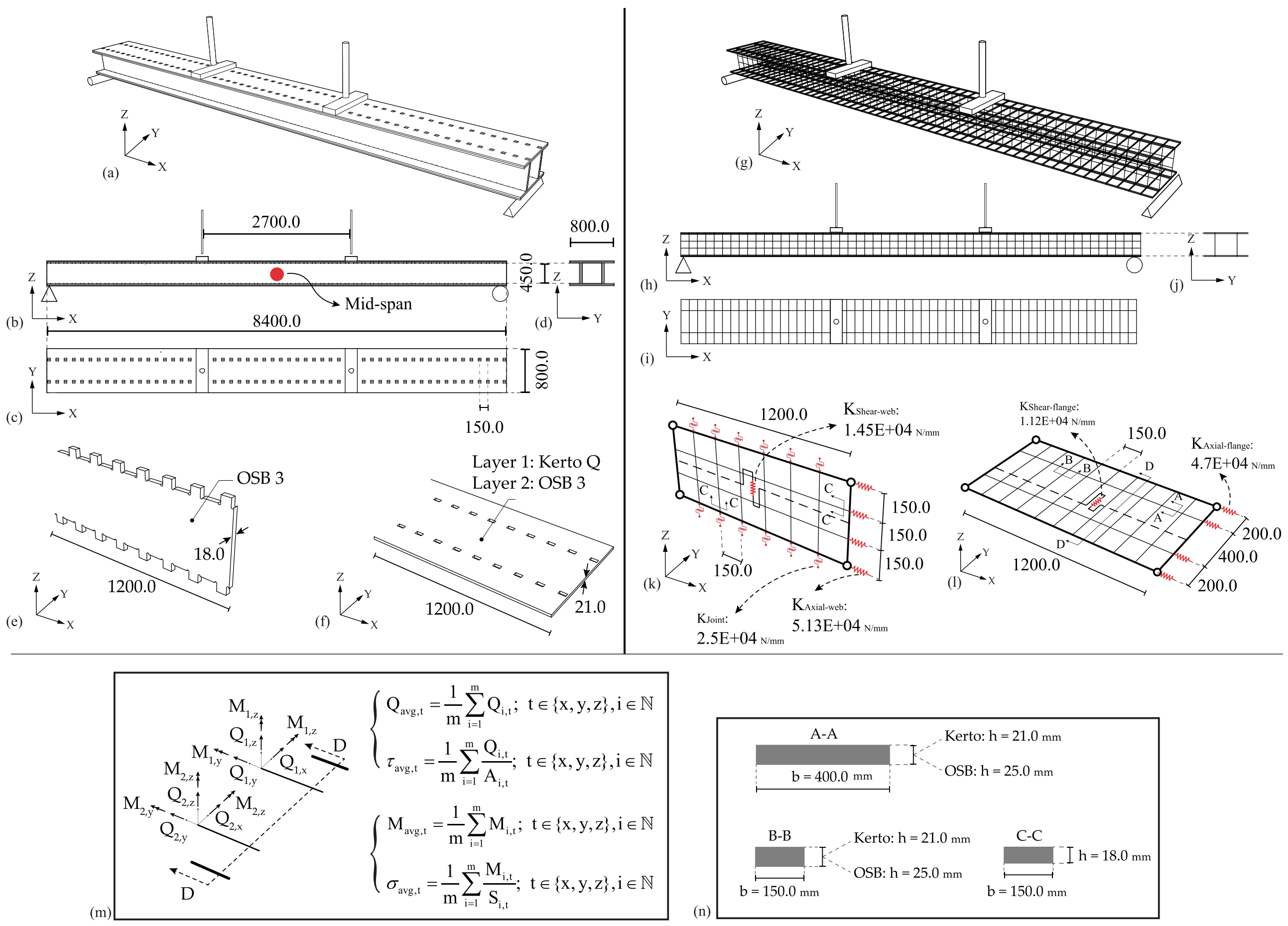

(a)
2700.0
800.0
Mid-span
450.0
8400.0
(b)
(c)
(d)
800.0
150.0
OSB 3
18.0
1200.0
(e)
Layer 1: Kerto Q
Layer 2: OSB 3
1200.0
21.0
(f)
(g)
(h)
(i)
(j)
1200.0
K_Shear-web: 1.45E+04 N/mm
150.0
150.0
150.0
150.0
K_Joint: 2.5E+04 N/mm
K_Axial-web: 5.13E+04 N/mm
(k)
K_Shear-flange: 1.12E+04 N/mm
150.0
K_Axial-flange: 4.7E+04 N/mm
1200.0
200.0
400.0
200.0
(l)
$Q_{avg,t} = \frac{1}{m}\sum_{i=1}^{m} Q_{i,t};\ t \in \{x,y,z\}, i \in \mathbb{N}$
$\tau_{avg,t} = \frac{1}{m}\sum_{i=1}^{m} \frac{Q_{i,t}}{A_{i,t}};\ t \in \{x,y,z\}, i \in \mathbb{N}$
$M_{avg,t} = \frac{1}{m}\sum_{i=1}^{m} M_{i,t};\ t \in \{x,y,z\}, i \in \mathbb{N}$
$\sigma_{avg,t} = \frac{1}{m}\sum_{i=1}^{m} \frac{M_{i,t}}{S_{i,t}};\ t \in \{x,y,z\}, i \in \mathbb{N}$
(m)
A-A
Kerto: h = 21.0 mm
OSB: h = 25.0 mm
b = 400.0 mm
B-B
Kerto: h = 21.0 mm
OSB: h = 25.0 mm
b = 150.0 mm
C-C
h = 18.0 mm
b = 150.0 mm
(n)

Fig. 14. Prefabricated beams: (a) 3D perspective view of the CAD model (re-generated from [30]), (b–d) side and top views, (e) dimensions of a typical web, (f) dimension of a typical flange, (g) 3D perspective view of the CAE macro model, (h-j) side and top views of the macro model, (k) macro model for the web, (l) macro model for the flange, (m) computation of force and stress in section D-D, (n) dimensions of sections A-A, B-B, and C-C.

23 (denoted as "E") are also shown in Fig. 16. The overall design process, from the target surface generation to the construction of the CAE macro model, is schematically illustrated for arch #22 Fig. 16b.

The performance of arch #22 (Fig. 17a) was experimentally evaluated by Nguyen and Weinand [35]. This arch consists of eight strips, each made from 25 boxes and each box is constructed through the assembly of four timber plates (Fig. 17b). The top and bottom plates have non-orthonormal edges while the cross plates are rectangular. Portions of the structure were loaded with 25 kg cement bags. Fig. 17a shows the top view of arch #22 and the timber boxes that were loaded by the cement bags. The number of the cement bags used for each box is also shown in Fig. 17a. A total load of 55 kN was applied to the structure. Based on the area loaded by the cement bags, the average surface load was 1.5 kN/m^2. An electronic total station was used to measure the displacement at 16 different points on the structure. A 3D terrestrial laser scanning technique using FARO® [36] was also used to measure structural displacements.

The vertical displacements measured by the laser scanner and the location of the 16 points measured by the electronic total station are shown in Fig. 18a and also documented in Table 3. Nguyen and Weinand [35] observed that there is considerable difference between the displacements recorded by the total station device and the 3D laser scanner. They concluded that the cloud-cloud distance computation used in the 3D laser scanner depends on the density of the reference cloud. Furthermore, the accuracy of the point clouds depends on the computational method used to calibrate (register) the laser scanner. Using five spherical targets, the point cloud method leads to errors of 10 mm when computing the displacements. The vertical displacements obtained from a FE model is also shown in Fig. 18b and documented in Table 3. The FE continuum model, constructed in ABAQUS^{TM}, utilizes shell elements to simulate the behavior of the timber plates, and springs to simulate the behavior of the IMAs [15,35].

Using the algorithmic framework, the CAE macro model of arch #22 is generated and analyzed. The top view of the CAE macro model geometry is shown in Fig. 18c. The inner beam elements of the macro model are generated by dividing the plates into four equal segments along the fiber-parallel and fiber-perpendicular directions. The tributary area associated with each inner beam is automatically computed and the cross-section properties are accordingly defined. The mechanical properties of the timber material, which are given in Table 2, are assigned to each inner beam element based on its fiber orientation. The macro model is subjected to the same load used in the physical experiment. The surface loads applied by the cement bags are divided into uniform loads and applied to the beam elements of the macro models.

4.2.2. Results and discussions

Table 3 compares the results obtained from the macro and FE models, as well as the measurements from the physical specimen. The macro model is 21.56%, 30.92%, and 22.31% different when compared to the total station measurements, laser scanning measurements, and the continuum FE model, respectively. Considering the scale of the structure and the differences in responses from the

Table 1
Material Properties of Kerto-Q LVL [32] and OSB 3 [33,30].

	OSB 3		*Kerto Q - LVL*
Thickness (mm)	18	25	21
Position	Flatwise	Edgewise	Flatwise
Density (kg/m^3)	576	600	481
$E_{0,\ mean}$(N/mm^2)	4930	3800	10,000
$E_{90,\ mean}$(N/mm^2)	1980	3000	3300
$G_{0,\ mean}$(N/mm^2)	50	1080	60

two types of physical measurements, the performance of the macro model is deemed reasonable. Furthermore, it is observed that the responses from the macro model are closer to that of the experimental results when compared to the FE model. There are several possible reasons for this observation. First, the FE model is mesh sensitive. More specifically, the size of meshes, especially near the joints, can affect the response. Furthermore, because the 1D wire elements used to simulate the TT joints are connected to the 2D shell elements with high mesh density, the continuum FE model is susceptible to numerical instabilities. Also, depending on the element type, adopted integration scheme and mesh size, the FE models were susceptible to variations in the response. It has also been shown that detailed FE simulations do not always provide more accurate responses when compared to simpler models, especially in large-scale structures [38,39]. In fact, while large number of input parameters are used to construct a continuum FE model, the influence of each parameter on the global behavior is not clearly understood. On the other hand, the macro model uses a relatively small number of input parameters and their role in the global behavior is much better understood. However, it is

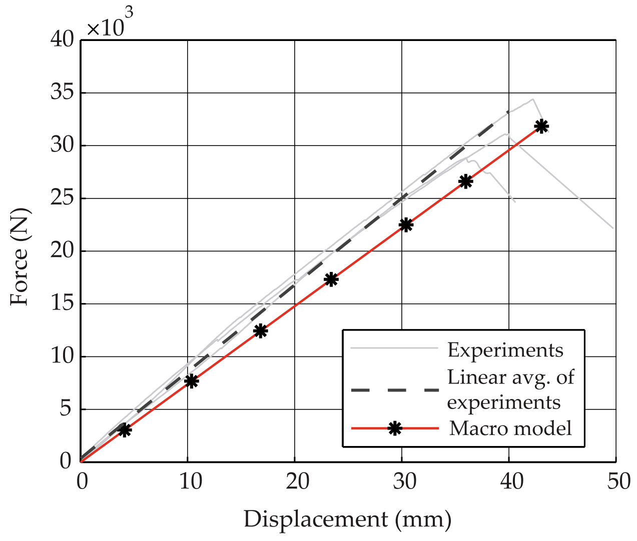


Fig. 15. Force-displacement response from the physical experiment [30] and the macroscopic model developed by the proposed CAD-to-CAE framework.

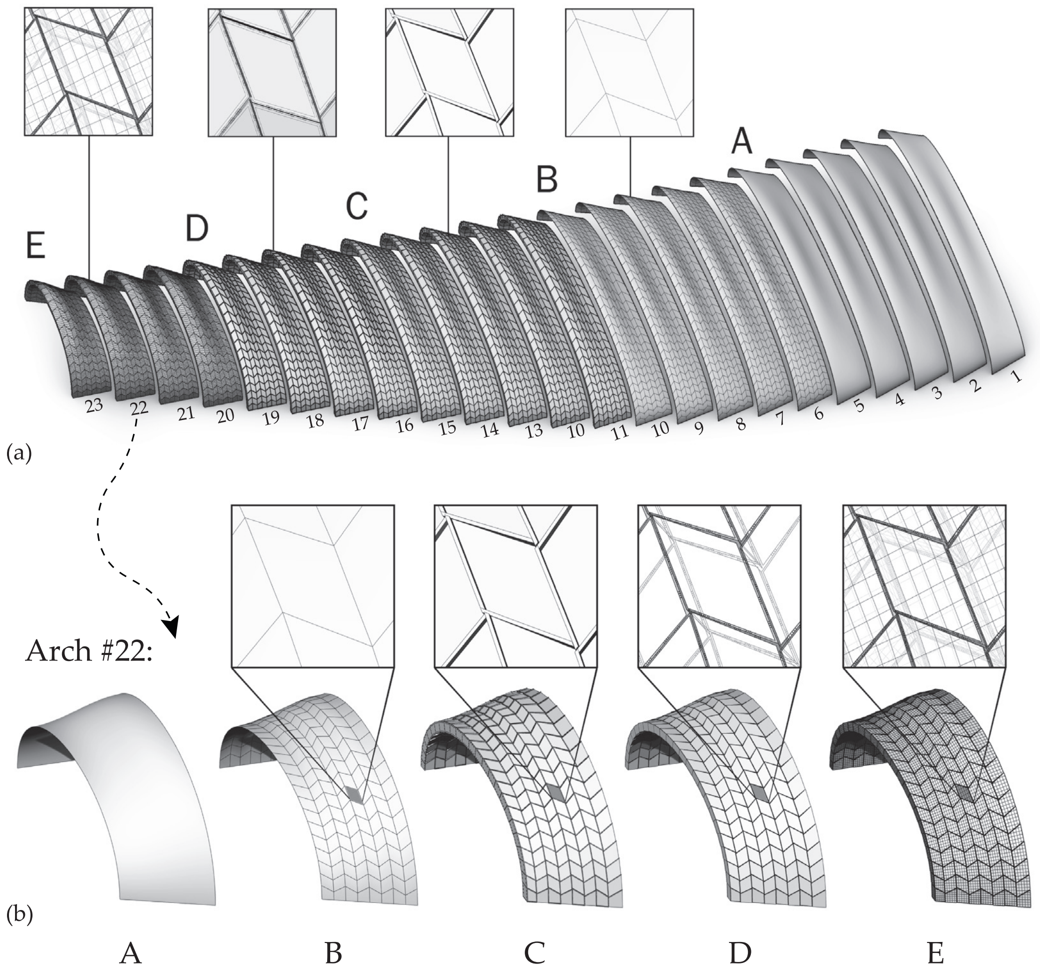
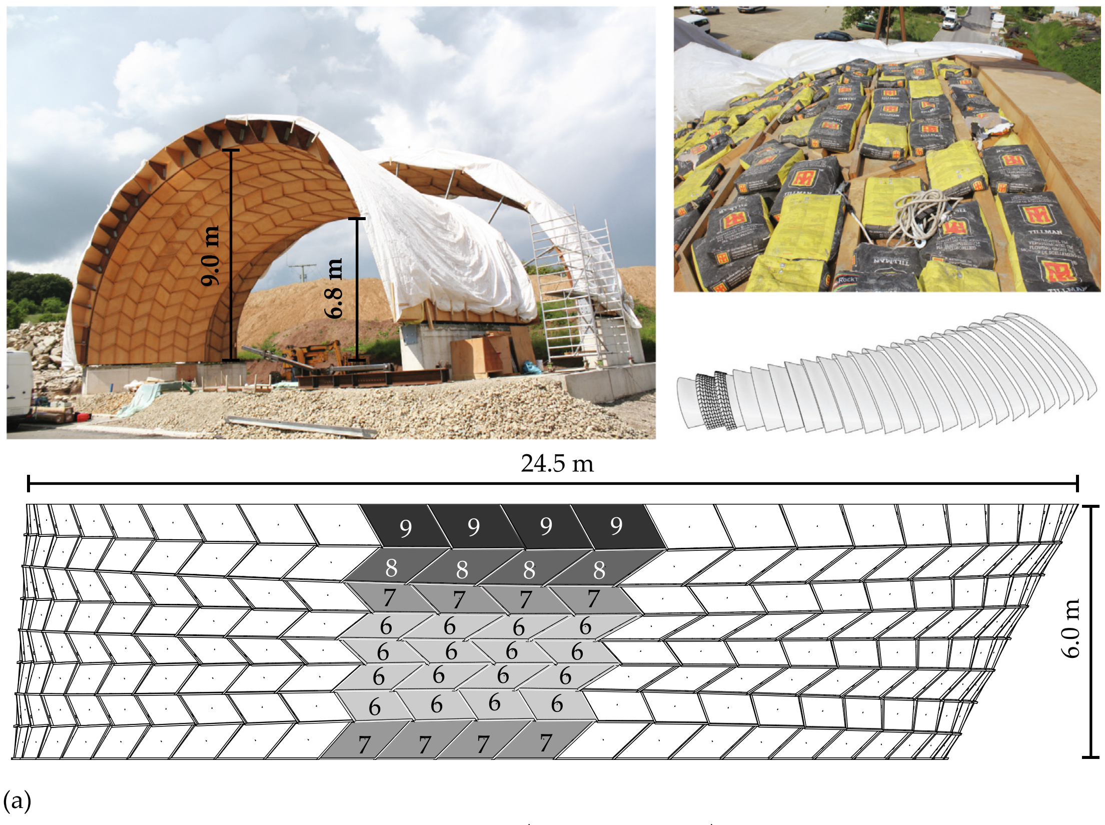


Fig. 16. (a) The geometry of the 23 IATP free-form arches, (b) schematic representation of the target surface (A), design pattern and mesh discretization (B), CAD 3D model (C), CAD 2D model (D), and CAD/CAE macro model (E).



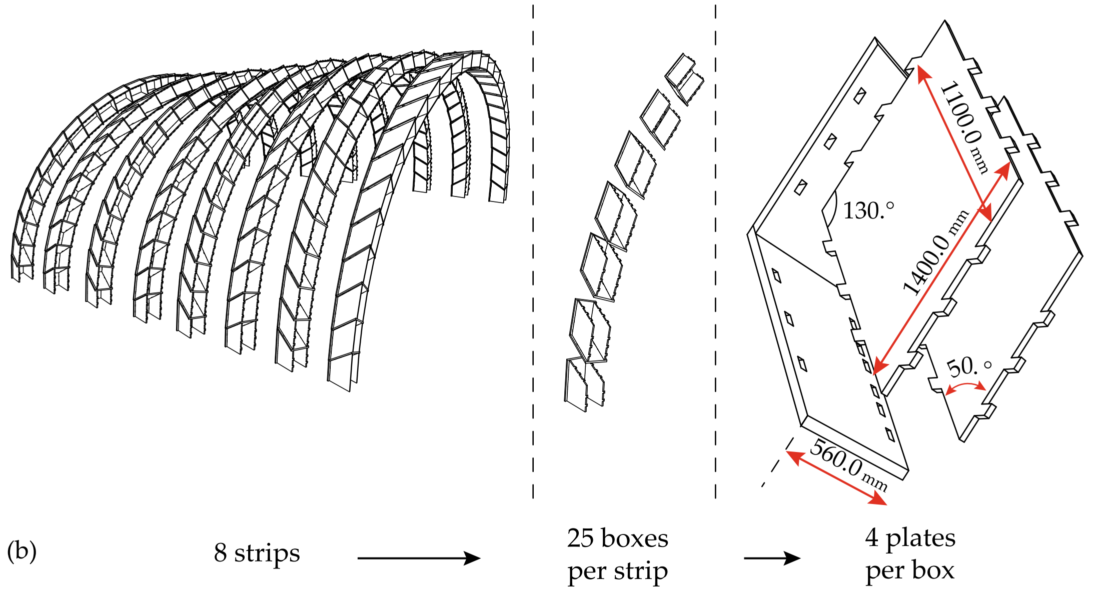


Fig. 17. (a) On-site experimental evaluation of the performance of arch #22 of the IATP structure [35], (b) the process used to assemble the plates, boxes, and strips.

important to note that the accuracy of the macro model in predicting the response of timber plates in the regions that are not directly loaded (i.e. stations 13 and 14) is lower than that of the directly loaded regions. This is attributed to two main factors. First, the scale of the structure is relatively large (25 m $\times$ 6 m $\times$ 9 m) and the global geometry of the structure is asymmetric in plan and elevation. Second, the element formulation in the macro model is such that the rigidity is concentrated in a specific number of discrete and uncoupled beam and springs elements [22], which leads to errors in large-scale IATP structures.

One of the major benefits of using the macro modeling approach for large-scale structures is that the resource use, memory footprint, and computational time are considerably reduced. Using the proposed CAD-to-CAE data exchange, the CAD geometry is translated to the corresponding CAE macro model using a machine with an Intel Core i7-4800MQ CPU @2.7 GHz and 16 GB of RAM. The time required to generate the CAE macro model is approximately 1 min and 3 s while using 22% of the CPU and 781 MB of memory. The structural analysis runtime is 11 min and 27 s, using 17.9% of the CPU and 261 MB of memory. This is compared to the computational time and resource use required for the continuum FE analysis of the arch using the same machine. Nguyen et al. [15] reported that the time required to generate the FE model is 64 min and 12 s using 42% of the CPU and 1560 MB of memory. Moreover, the structural analysis took 1864 min and 2 s while using 22% of the CPU and 4482 MB of memory.

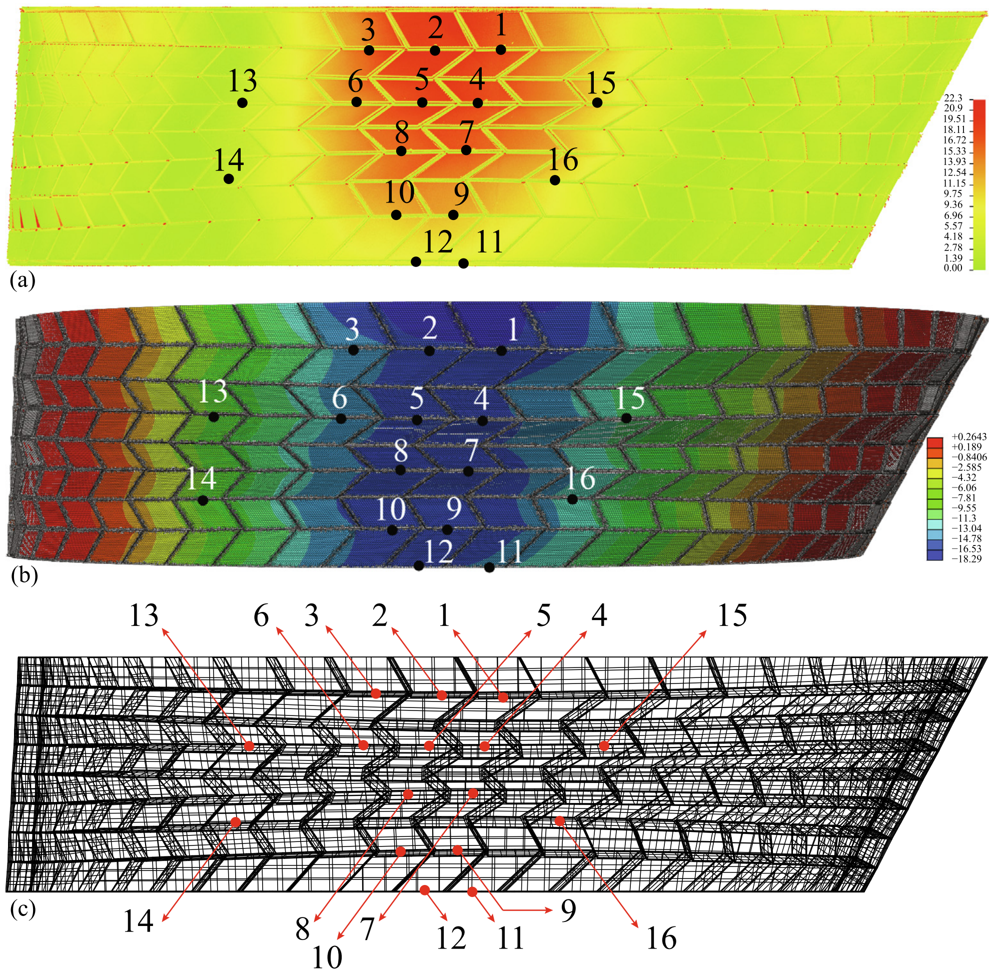


Fig. 18. Top view of Arch #22: The vertical displacements: (a) measured by the laser scanner and the location of the points used in the electronic total station measurement [35], (b) obtained from the FE simulation [35], and (c) the CAE macro model generated by the proposed algorithmic framework for arch #22.

Table 2
Material Properties of beech BauBuche [37].

Symbol	Description	Value (N/mm^2)
E_0	Modulus of elasticity, fiber-parallel	13200.
E_{90}	Modulus of elasticity, fiber-perpendicular	2200.
G_0	Shear modulus, fiber-parallel	820.
G_{90}	Shear modulus, fiber-perpendicular	430.

5. Conclusions

An algorithmic modular framework was developed to take the 3D geometry of a custom-defined Integrally-Attached Timber Plate (IATP) structure as input, compute the mechanical properties and other non-geometric information, and construct and analyze the associated CAE macro model. The key techniques used to convert a 3D CAD model to the macro model include feature-based simplification, topology-based modification, and virtual topology adaptation. An algorithm for identifying the perimeter polygons associated with 3D timber plates was first developed. A simplified line element was used to establish the connection between neighboring elements. An indexing algorithm was then introduced to assign a unique tag to the polygons and connections based on the following three types of IATP components: strip, box, and plate. Next, an algorithm to generate the components of the macro model (inner beams, boundary beams, and spring elements) for each polygon was developed. Another algorithm was formulated to assign a unique tag to the macro model components. The cross-section properties of each inner beam element were determined based on the associated tributary area. Furthermore, the material properties were assigned to the macro model components based on the associated fiber orientation. The CAD-to-CAE data exchange framework was written and compiled in a CAD environment (Rhinoceros 3D) using the RhinoCommon library. The output of the data exchange was designed to follow the syntax of OpenSees, which was used as the structural analysis platform.

The proposed CAD-to-CAE data exchange framework was applied to the following two structures, where the macro models associated with the CAD 3D models were generated: (i) a standard beam element with a hollow cross section, and (ii) a large-scale free-form IATP arch. The proposed data exchange framework was verified against the responses obtained from the physical experiments and continuum Finite Element (FE) models. It is concluded that the centralized repository, which includes the model inputs, analysis options, data management structure, and CAD and macro model databases, is necessary for analyzing complex large-scale IATP structures. Furthermore, since a typical IATP structure consists of hundreds of timber plates with thousands of joints, the development of the CAD-to-CAE exchange framework facilitates a parametric data storage methodology and systematic identifica-

Table 3
Vertical displacements at the identified locations of arch #22.

Station ID	Total station [35]	Laser scanner [35]	Finite Element [35]	Macro	Unit
1	20.5	15	16.58	18.74	mm
2	22.3	17	16.71	20.54	mm
3	18.4	15	14.18	18.41	mm
4	19.8	17	14.81	16.81	mm
5	19.8	18	15.19	18.35	mm
6	15.9	15	13.42	16.47	mm
7	16.5	16	15.32	16.26	mm
8	17.5	15	15.44	17.6	mm
9	12.9	12	15.06	15.14	mm
10	12.0	12	15.0	16.18	mm
11	11.3	7	12.91	13.37	mm
12	11.1	7	14.43	14.63	mm
13	7.6	3	6.96	1.19	mm
14	5.6	1	6.33	1.11	mm
15	10.1	8	9.24	6.78	mm
16	11.8	8	10.76	11.75	mm

tion protocol. The CAD-to-CAE data exchange was shown to minimize the manual operations, and the generation of the CAE macro model becomes deceptively straightforward. Moreover, the proposed framework enables design flexibility in IATPs since it can be applied to different geometries.

Applying the proposed CAD-to-CAE data exchange framework to the prototypes with standard and free-form geometries, it is concluded that the results obtained from the macro model compare well with that of the physical experiments. Moreover, the interpretation of the results is straightforward when the macro modeling technique is used for structural analysis. This, in particular, facilitates seamless global performance evaluation of IATP structures. The total computational time for the macro model, including the creation of the model database and runtime, is significantly less than that of the continuum FE model. As the main conclusion, the proposed CAD-to-CAE data exchange framework can be used by practitioners to design IATP structures. Accordingly, the framework can be adapted by typical BIM tools that support parametric modeling in digital architecture and macro modeling in structural analysis to create, manage, and share the 3D model associated with an IATP structure.

With the proposed CAD-to-CAE data exchange framework, structural design parameters can be included in the design process of IATPs during the architectural design and CAD model development. This would lead to a master model that aims to optimize the design of IATP structures by satisfying not only the assembly and fabrication constraints but also the structural engineering design requirements. Furthermore, the proposed CAD-to-CAE data exchange can facilitate design optimization of large-scale IATP structures by including the non-linear behavior of IMAs. The non-linear behavior, ultimate limit state, and the failure criteria can be easily embedded in the spring elements of the macro model and applied to the thousands of connections in the structure under consideration.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

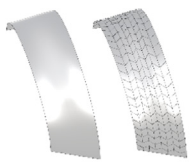
Acknowledgments

The financial support of the National Centre of Competence in Research Digital Fabrication (https://dfab.ch/) funded by the Swiss National Science Foundation is gratefully acknowledged (agreement ID #51NF40141853). The financial support of École Polytechnique Fédérale de Lausanne (EPFL) is also acknowledged with many thanks. The two case studies addressed in Section 4 of the current investigation are parts of technology transfer agreements at the Laboratory for Timber Construction (IBOIS) at EPFL, and the authors wish to sincerely thank the research members at IBOIS for suggestions and discussions.

Appendix 1

1.1. NURBS subdivision and quad mesh generation

```
                          // S - NURBS, UDiv – divisions of rows, VDist – distance of quad in V-direction, T – translation value
                          // Scale – Scaling towards center, Thres – threshold value to stop NURBS divisions towards centre
Mesh SkewedQuadMesh       (Surface S, double UDiv, double VDist, double T, double Scale, double Thres)        // NURBS division to create

    List<MeshFace> quads
    List<Point3d> vertices
    Initialize division parameters: Surface Domain, divisionStep, baseStep, maxStep, shiftParameter, etc.

    for (int i = 0; i < UDiv; i++)                                                                             // Iterate strips of longitudin
      Initialize strip parameters: divisionThreshold, stepV1, ctV                                              //Strip is constructed from 2
      Initialize points Point3d to form quad face
      while (iteration is less than the threshold)                                                             // Iterate until the middle of
        Get surface points to construct mesh faces for the first half of the arch
        Get surface points to construct mesh faces for the second half of the arch

    Construct mesh from vertices and quads
    return quadMesh                                                                                            // Mesh output
```



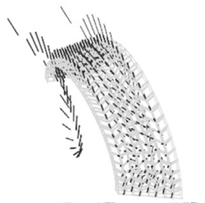
1.2. CAD 3D model generation of IATP structures

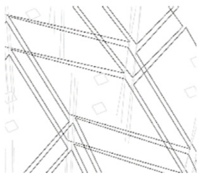
```
                    // QuadMesh – Mesh from SkewedQuadMesh method, Divisions – double strip division,
                    // PlateThickness – Thickness of the plate, TenonDivisions – Distance to Divide Tenons,
                    // ShiftLinear – Box shift along longitudinal axis, ShiftRotation – rotation of side plates to obtain
                    reciprocal like pattern,
                    // MEdgesIDFront, MEdgesIDBottom, MEdgesIDBack – meshes edges for boundary conditions,
                    // CutYCoord0, CutYCoord1, CutZCoord – Coordinates to cut timber plates at gives mesh id
                    // OSDivisions – joint division for OpenSees, OSPlateOffset – Top and bottom plate offset
                    (Mesh QuadMesh, int Divisions, double Offset, double PlateThickness, double TenonDivisions,
MultiData           double ShiftLinear, double ShiftRotation,                                 // Grasshopper Components
DLTPScomp2 :        int MEdgesIDFront, int MEdgesIDBottom, int MEdgesIDBack,                  data for Fabrication and St
GH_Component        double CutYCoord0, double CutYCoord1, double CutZCoord,                   Calculation
                    int OSDivisions, int OSPlateOffset)

                                                                                              //Generic template methods
                    RegisterInputParams (GH_InputParamManager pManager)                       //Register Input Parameters
                    RegisterOutputParams (GH_OutputParamManager pManager)                     //Register Output Paramete
                    override Bitmap Icon                                                      //Create a Component Icon
                    override Guid ComponentGuid                                               //Give a unique index to the

                    SolveInstance(IGH_DataAccess DA)                                          //Main method
                      Fetch all component inputs                                              // Component input
                      Fetch mesh data, including mesh face normal and sorting mesh edges

                      Decide edge divisions                                                   //Initialize Box objects from
                      Create default planes for each edge
                      Initialize BoxPlate objects for each mesh face

                      foreach(BoxPlate box in boxes)                                          //Iterate boxes
                         if box is the even row                                               //Shift boxes (even rows)
                           box.RotateVertCW()
                           box.RotateEdgesCW()
                        box.ProjectBoxPoints()

                      foreach(BoxPlate box in boxes)                                          //Set box and plate adjacenc
                        box.SetBoxNeighbours()

                      foreach(BoxPlate box in boxes)                                          //Series of methods for fabr
                        box.SetDivisionAxes()

                      foreach(BoxPlate box in boxes)
                        box. CreateNewMidContours()

                      foreach(BoxPlate box in boxes)
                        box.CorrectEdgeMidPts()

                      foreach(BoxPlate box in boxes)
                        box.CreateNewPlateContours()

                      foreach(BoxPlate box in boxes)
                        box.CreateTopPlateSlots()
                        box.CreateBottomPlateSlots()
                        box.AssignSlotsToNeighbours()


                                                                                              // Component output

                      foreach(BoxPlate box in boxes)                                          //Fabrication
                         Retrieve Fabrication Outlines

                      foreach(BoxPlate box in boxes)
                        Retrieve data as a DataTree of corner points                          //Perimeter polygons
                        Retrieve data as a DataTree of joint points per plate                 //Connections (IMAs)
                        Retrieve data as a DataTree of ID of the joint start point
                        Retrieve data as a DataTree of ID of the joint end point

                      Output the data to Grasshopper Component Interface

                                                                                              // Output: fabrication conto
                    MultiData: Datatree<Polyline>  Fabrication, Datatree<Point3d>  4Corners,  corners of the plates, joint p
                    Datatree< Point3d > JointPoints, Datatree<int> currentJointStartIndex     each plate, indexing for cur
                    Datatree<int> nextJointStartIndex                                         and next joint id following n
                                                                                              (strip;box;plate;side)
```

1.3. Identification of the perimeter polygons and simplified IMAs

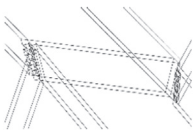
BoxPlate Class

```
// BoxPlate class contains 6 plate objects and main geometry processing method to obtain the overall
geometry
class BoxPlate

  int SID                                                                   //Properties of the
  int ID                                                                    //The mesh data st
  int Neighbor_ab_ID, Neighbor_bc_ID, Neighbor_cd_ID, Neighbor_da_ID        //data structure is
  int EdgeIndex_AB, EdgeIndex_BC, EdgeIndex_CD, EdgeIndex_DA                //BoxPLate includ
  double div_ab, div_bc, div_cd, div_da                                     //and adjacent mes
  double planarity
  double h_top, h_bot,                                                      //the layer and pla
                                                                            including:
  double L_tab                                                              //(1) tenon-mortise
  Plate PT, PB, W0, W1
  Vector3d v                                                                //(2) edge and inse
  Vector3d ad, ba, ab, bc, cd, da
  Vector3d n_ab, n_bc, n_cd, n_da
  Line line_A, line_B, line_C, line_D, line_E, line_F                       //Edges are stored
  Line line_A_offs, line_B_offs, line_B_offs_, line_C_offs, line_D_offs, line_D_offs_
  Line line_A_mid, line_B_mid, line_C_mid, line_D_mid
  Point3d A, B, C, D                                                        //Corner points
  Plane pS
  Plane p_AB, p_BC, p_CD, p_DA                                              //Planes for curren
  Plane p_AB_mid, p_BC_mid, p_CD_mid, p_DA_mid
  Plane p_AB_offs, p_BC_ offs, p_CD_ offs, p_DA_ offs
  Plane PTT2, PTT, PTM, PTB
  Plane PBT, PBM, PBB, PBB2
  Plane TP_da, TP_ab
  string info
  bool DoubleUp_Wall_BC, DoubleUp_Wall_CD                                   //Boolean informa
  bool HasNeighbor_ab, HasNeighbor_bc, HasNeighbor_cd, HasNeighbor_da       //adjacency and bo
  bool HasNeighborPlane_ab, HasNeighborPlane_da
  bool IsBoundary_ab, IsBoundary_bc, IsBoundary_cd, IsBoundary_da
  bool HasPolylines, HasBreps


 void SetNeighbors(MeshTopologyEdgeList mte)                                //Iindexing and adj
   GetConnectedFaces(FaceEdgeIndex)                                         //is transferred to
   if(the face has two neighbours)                                          //BoxPlate data-st
     Set Neighbor_ab_ID, HasNeighborPlane_ab, HasNeighbor_ab
   else
     Set IsBoundary_ab
   Repeat the same step for each edge

 void SetDivAxes (List<BoxPlate> boxes, List<int> eIds1, List<int> eIds2, List<int> eIds3,   //Construct plate e
 double r, double s, double Yb, double Yf, double Zg, int id)              and structural mo
   Create planes for cutting sides and base of the shell, because edges are curved by vertex normals
   Set cut plates per each box by user defined mesh index numbering
   Set edge planes for each mesh edge, considering normal of adjacent mesh faces
   Shift and rotate edges considering user input
   Repeat the same process for each edge
   Get lines for mesh edge plane intersections
   Offset planes and obtain plate line segments from PlanePlane intersection   //Fabrication data
                                                                            //Perimeter polygo
   Create line segments as well for structural calculation considering specifications of Macro model   model generation
  List<Point3d> CreateNewPlateMidContourTop(Plane plane, Line lA, Line lB, Line lC, Line lD,   //Base function to
  Plate Plate)                                                              outlines
    Intersect lines obtained from previous method to plate top plane
    Collect intersection result – Point3d to construct an outline
    Also call Plate.CreateEdgeMidPts method to get middle points of the polygons
    Return list of corner points

  List<Point3d> CreateNewPlateMidContourTop2(Plane plane, Line lA, Line lB, Line lC, Line
lD,
  Plate Plate)
    Repeat same process as in previous method but for bottom outline

  List<Point3d> CreateNewPlateMidContourSides(Plane plane0, Plane plane1, Line l0, Line l1,
Plate Plate)
    Repeat previous method for side plates

  void CreateNewBoxMidContour(Plane plane, Line lA, Line lB, Line lC, Line lD)   //Get edge and ins
    Intersect lines from mesh edge plane intersection with mesh faces planes
    Get polygon from lines points
    Get edge vectors from pair of corner points A,B,C,D
    Computer insertion vector from two edges v = -(ba+da), where ba = B-A and da = D-A
```

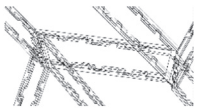




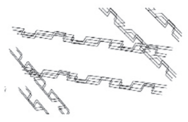
```
void CreateNewMidContours(double femScaleTopPlate)                                        //Get plate polygons
   Index the plates (PT.name, PB.name, W0.name, W1.name)
   Index StripID of each plate
   Index BoxID of each plate
   Index PlateID of each plate
   Compute planes for top and bottom sides of the plates
   Compute outlines from these planes - CreateNewBoxMidContour()
   Create offset outlines for top and bottom plates – CreateNewPlateMidContourTop()        //Fabrication
   Create outlines for top and bottom plates – CreateNewPlateMidContourTop2()              //Fabrication
   Create side outlines                                                                    //Fabrication
   Create middle outlines                                                                  //Perimeter polygon for Ope

void CorrectEdgeMidPts(List<BoxPlate> boxes)                                              //Correct top, bottom edge m
   bool boolNeighbour
   int NeighborID
   for(iterate 4 times)
      check edge neighbours
      if(edge is not boundary)
         foreach (iterate edge mid points)
            if(when the distance between current and neighbor point is less than the user distance)
               Assign the corrected edge middle point by mean value of both points
      repeat the same process for bottom plate
```



```
void CreateNewPlateContours ()                                                            //Assign plate planes and co
   Create top and bottom plate contours – CreateNewPContour()                             //dovetail and tenon-mortise
   Create side plates outlines – CreateNewWContour()
   Assign middle and base planes for the plates
   Create side outlines
   Call List<List<Point3d>>Dovetail() method for the box
   Add corner points to dovetail nested point list
   Assign side plate the middle and base planes
```



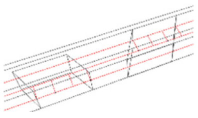
```
Outline CreateNewPContour(Plane plane, Line lA, Line lB, Line lC, Line lD, Plate Plate)   //Create tenons for each con
   Get corner points per plate side (top or bottom)
   Create tenon-mortise joint if edge is not boundary CreateTenons()
   Add joint point collection in correct order of the plate contour
   Add joint sequence to keep track of how many tenons does the plate have
   Return the outline including tenon joints

Outline CreateNewWContour(Plane plane, Line lA, Line lB, Line lC, Line lD)                //Create side outlines witho
   Get corner points by line plane intersections
   Create an outline and assign empty joint sequence
   Return the outline
```



```
List<Point3d> CreateTenons(Point3d p0, Point3d p1, Vector3d v, Plane pT, Point3d CorrMidPt,  //Slots are computed separa
double t_plate, double L_tab, double div)                                                   outline geometry
   Create tenons following the insertion vector
   Move tenon geometry from the corners considering the plate thickness and the insertion angle
   Enumerate points following a modulor of 2
   Add tenon at every second pin

AssignSlots(List<BoxPlate> Boxes)                                                         //Creates hole for side plate
   Iterate top and bottom plate and add slots from PB and PT to side plates W0,W1         //and bottom plate of tenon j
   Slots data is also used for the CAD-to-CAE joinery and indexing
   Append slots also from neighbor plates since side plates are connected to four plates in total
```

```
List<List<>> Dovetail()                                                                   //Create edge outlines repre
   Get corner points of the side plate edge, where the joints have to be added            //Dovetail joints
   Create 4 lines for side plate edges
   for (iterate 8 times)
      Create plane for two neighbor mesh edge directions
      for the first plane add not rotation and for the rest turn plane by 15 degrees
   foreach (infinite line in lines)
      intersect with 6 planes to get the zigzag outline of dovetail
   Sort outlines to match the orientation
   Return nested point lists for each edge of side plates

void RotateVertCW()                                                                       //Shift corner points
   Copy corner points
   Assign corner point shifted to the right

void RotateEdgesCW()                                                                      //Shift edge values
   Get division values
   Get edge indices
   Assign both division and edge values by shifting by one index

void ProjectToPlane()
   Project corner point to plates contour base plane using, by line plane intersection    //Planarize corner points
```

```
void FitPlane()                                                              //Fit corner points
   Get corner points
   Fit corner points to plane
   Move origin of plate to corner points centre
   Check direction of the newly created plane by user define normal
   Flip plane if angle* 180.0 / Math.PI is more than 90 deg

List<Point3d> Sort(List<Point3d> a, List<Point3d> b)                         //Combine two list
  Create and empty Point3d list                                              //joint method
  for(iterate 8 times)
    Add first point to the list
    if(i%2!=0)
      add a[i] and b[i] points to the empty list
    else add in reverse order
    Add last point to the list
    Return sorted list


                  // Plate class that represents top and bottom outlines of the timber plate including slots
Plate Class       class Plate                                                // Class representing timber

                    Outline ContourTop = new Outline(new Polyline());        //Properties of the Plate cla
                    Outline ContourBot = new Outline(new Polyline());        //Top and bottom polygons
                    Plane Base = new Plane();
                    Plane MidPlane = new Plane();
                    List<Point3d> MidCornerPts = new List<Point3d>();        //Corner points
                    List<Point3d> MidCornerPts2 = new List<Point3d>();
                    Polyline outline2 = new Polyline();
                    Brep MidSurface4pts;
                    List<Point3d> MidSrfEdgeMidPts = new List<Point3d>();    //Mid edge points
                    Point3d[] MidSrfEdgeMidPtsCorr = new Point3d[4];
                    Polyline outline = new Polyline();

                                                                             //connection Properties use
                    List<List<Polyline>> Slots = new List<List<Polyline>>(); macro model

                    List<List<Point3d>> SlotsSimplified = new List<List<Point3d>>();
                    List<List<GH_Path>> SlotsSimplifiedNeiID = new List<List<GH_Path>>();
                    List<List<List<Polyline>>> Slots_Out = new List<List<List<Polyline>>>();
                    List<List<List<Point3d>>> Slots_InSimplified= new List<List<List<Point3d>>>();
                    List<List<List<Point3d>>> Slots_OutSimplified = new List<List<List<Point3d>>>();
                    List<List<List<GH_Path>>> Slots_InSimplifiedID = new List<List<List<GH_Path>>>();

                   int StripID;                                              // indexing
                   int BoxID;
                   int PlateID;
                   int hCount;
                   string name = null;
                   bool HasSlots = false;
                   double Thickness = 0;

                   void CreateEdgeMidPts()
                     Get mid-points of corner polygons
                     Define polygon from points

                   void CreateSlots(int FEMDivisions = 1)                    //Iterate joint sequence to g
                     Get top and bottom contours polygons as Point3D arrays  //of the plates (mortise outli
                     Declare slots for nest polygon array for the plate slots as List<List<List<Polyline>>>
                     Declare collection for OpenSees joint representations as List<List<List<Point3d>>>
                     Declare collection for tracking joint indices List<List<List<GH_Path>>>

                    foreach(int num in ContourTop.jointSeq)                  //Iterate joint sequence
                      Create local nested collections to store polygons and OpenSees point data for joints
                                                                             //that defines how many join
                                                                             //plate edge
                      for (int i = 0; i < num; i++)                          //iterate each tenon
                         Create local collections to store polygons and joint data
                         Create rectangle polygon for each tenon and store the data    //Fabrication data
                         Get middle points of the tenons and mortises of current and adjacent plates   //1D lines
                         Find closest-point to four corner polygons since they differ from fabrication model
                         The tenons and mortises has an angle while the macro model contains perpendicular edge joints
                         Two lines are created per each joint
                         Line segments are interpolated depending on the user needs
                         Joint data is stored in a point list together with indexing of StripID, BoxID, PlateID, EdgeID
```



```
                        // pline - polygon, jointSeq – joint order
Outline Class           class Outline                                                    // Class representing polygo
                                                                                         // of timber plate i.e. top or
                          Polyline pline
                          List<int> jointSeq

                          Outline(Polyline outline)
                            pline = new Polyline(outline)


                        // Static helper for geometry processing
Outline Utilities       static class Utilities
Static Geometry Utilities
Class
                          Line TweenLine(Line l0, Line l1)                                                       //Interpolate line
                          List<Point3d> TweenPolylines(List<Point3d> p0, List<Point3d> p1, double t = 0.5)        //Interpolate points of a poly
                          double GetDist(Point3d p0, Vector3d v, Plate P)                                        //Get distance: point to proj
                          List<Point3d> CP(List<Point3d> points, Polyline outline)                               //Get closest points to a poly
                          List< List<Point3d> > CP(List< List<Point3d> > points, Polyline outline)               //Nested closest points to a p
                          void ShiftPoints(List<Point3d> A, int n)                                               //Shift points to the right
                          void ShiftPolyline(Polyline A, int n)                                                  //Shift polygon points to the
                          Polyline TwoPlaneTwoLines(Plane plane0, Plane plane1, Line l0, Line l1)                 //Construct a polygon
                          Line ScaleLn(Line line, double scale = 10)                                             //Scale a line
                          bool IsClockwisePolygon(Polyline polygon)                                              //Check if a polygon is clock
                          InsertPolyline(Polyline x, IEnumerable<Polyline> y)                                    //Insert polygons into a poly
                          Point3d LinePlane(Line line, Plane plane)                                              //Intersect line and plane
                          Line IntPtPln(Point3d origin, Vector3d axis, Plane planeBot, Plane planeTop)           //Cut line by two planes
                          double Lerp(double value1, double value2, double amount)                               //Interpolate numeric values
                          Point3d[] InterpolatePoints(Point3d from, Point3d to, int Steps)                       //Interpolate points by numb
                          Point3d PlanarPoint(Point3d A0, Plane newplane)                                        //Project point to plane by i
                          bool MeasurePlanarity(Point3d A, Point3d B, Point3d C, Point3d D)                      //Measure quad planarity by
```

1.4. Generation of the macro model components

```
Using :          rhinoscriptsyntax library                                                              // basic geometry fu
Input :          C (list of all Cells as polylines), S, B and P (associated Strip, Box and Plate identifiers as list of integers)
Input :          J (list of all Joints as segments), Js and Je (associated list of tags (Strip, Box, and Plate) for the connecting polygons)
Output:          N, E, G (lists of strings for nodes, elements, and geometric transformation)           // following OpenSe
Initializing:    nodesInfo = []                                                                         // temporary storage

Tag                                                                                                     // Calling S_iB_jP_k to e
                 for i = 0 to i = length of C -1
                     tag = 1000*S[i] + 10*B[i] + P[i]

Corner nodes
1.2                  points = corners of C[i]                                                           // Calling the corner
1.3                  for j = 0 to 3
1.3.1                    Add to N: 'node' + (100000*tag + j+1) + x,y,z coordinates of points[j]         // indexing corner n
1.3.2                    Add to nodesInfo: ((100000*tag + j+1), points[j], (100000*tag + j+1))          // Adding the nodes

Pins at the
corners
                     hinge1 = copy of points[0] translated 1mm in direction of points[1]
                     Add to N: 'node' + (100000*tag + 5001) + x,y,z coordinates of hinge1               // id format: SBP50
                     Add to nodesInfo: ((100000*tag + 5001), hinge1, (100000*tag + 5001))

                     hinge2 = copy of points[1] translated 1mm in direction of points[0]
                     Add to N: 'node' + (100000*tag + 5002) + x,y,z coordinates of hinge2               // id format: SBP50
                     Add to nodesInfo: ((100000*tag + 5002), hinge2, (100000*tag + 5002))

                     hinge3 = copy of points[1] translated 1mm in direction of points[2]
                     Add to N: 'node' + (100000*tag + 5003) + x,y,z coordinates of hinge3               // id format: SBP50
                     Add to nodesInfo: ((100000*tag + 5003), hinge3, (100000*tag + 5003))

                     hinge4 = copy of points[2] translated 1mm in direction of points[1]
                     Add to N: 'node' + (100000*tag + 5004) + x,y,z coordinates of hinge4               // id format: SBP50
                     Add to nodesInfo: ((100000*tag + 5004), hinge4, (100000*tag + 5004))

                     hinge5 = copy of points[2] translated 1mm in direction of points[3]
                     Add to N: 'node' + (100000*tag + 5005) + x,y,z coordinates of hinge5               // id format: SBP50
                     Add to nodesInfo: ((100000*tag + 5005), hinge5, (100000*tag + 5005))

                     hinge6 = copy of points[3] translated 1mm in direction of points[2]
                     Add to N: 'node' + (100000*tag + 5006) + x,y,z coordinates of hinge6               // id format: SBP50
                     Add to nodesInfo: ((100000*tag + 5006), hinge6, (100000*tag + 5006))
```

```
        hinge7 = copy of points[3] translated 1mm in direction of points[0]
        Add to N: 'node' + (100000*tag + 5007) + x,y,z coordinates of hinge7          // id format: SBP50
        Add to nodesInfo: ((100000*tag + 5007), hinge7, (100000*tag + 5007))

        hinge8 = copy of points[0] translated 1mm in direction of points[3]
        Add to N: 'node' + (100000*tag + 5008) + x,y,z coordinates of hinge8          // id format: SBP50
        Add to nodesInfo: ((100000*tag + 5008), hinge8, (100000*tag + 5008))
Geom.
Transf.
        vector1 = vector from points[0] to points[1]
        vector2 = vector from points[0] to points[3]
        normal = cross product between vector 1 and vector 2
        Add to G: 'geomTransf Linear' + tag + x,y,z coordinates of normal
Fiber Parallel
beams
        if P[i] == 1 or P[i] == 2
                                                                                      // assign material pr
            mat = '$A_P12_FPa $E_P12_ FPa $G_P12_ FPa $J_P12_ FPa $Iy_P12_ FPa $Iz_P12_ FPa' + tag   parallel beams of pl
                                                                                      (OpenSees format)
        if P[i] == 3
                                                                                      // assign material pr
            mat = '$A_P3_FPa $E_P3_FPa $G_P3_FPa $J_P3_FPa $Iy_P3_FPa $Iz_P3_FPa' + tag   parallel beams of pl
                                                                                      format)
        if P[i] == 4
                                                                                      // assign material pr
            mat = '$A_P4_FPa $E_P4_FPa $G_P4_FPa $J_P4_FPa $Iy_P4_FPa $Iz_P4_FPa' + tag   parallel beams of pl
                                                                                      format)

        Create an array of 4 hatches parallel to the longitudinal direction (between points[0] and points[1])   // Fig. X

                                                                                      // Assigning indices
        for  j = 0 to 2                                                               beams
            FPa_tag = 10000*tag + 1000 + j+1
            Add to N: 'node' + (FPa_tag + '1') +  x,y,z coordinates of FPa[j] start point   // id format: SBPLn
            Add to nodesInfo: (FPa _tag + '1'), FPa start point, (FPa _tag + '1')
            Add to N: 'node' + (FPa_tag + '2') +  x,y,z coordinates of FPa[j] end point     // id format: SBPLn
            Add to nodesInfo: (FPa _tag + '2'), FPa end point, (FPa _tag + '2')
            Add to E: 'element elasticBeamColumn' + FPa _tag + (FPa _tag +'1') + (FPa _tag + '2') + mat   // id format: SBPLn

Fiber
perpendicular
beams
        if P[i] == 1 or P[i] == 2
                                                                                      // assign material pr
            mat = '$A_P12_FPe $E_P12_ FPe $G_P12_ FPe $J_P12_ FPe $Iy_P12_ FPe $Iz_P12_ FPe' + tag   perpendicular beam
        if P[i] == 3
                                                                                      // assign material pr
            mat = '$A_P3_ FPe $E_P3_ FPe $G_P3_ FPe $J_P3_ FPe $Iy_P3_ FPe $Iz_P3_ FPe' + tag   perpendicular beam
        if P[i] == 4
                                                                                      // assign material pr
            mat = '$A_P4_ FPe $E_P4_ FPe $G_P4_ FPe $J_P4_ FPe $Iy_P4_ FPe $Iz_P4_ FPe' + tag       perpendic

        Create an array of 6 hatches perpendicular to the longitudinal direction (between points[0] and points[2])   // Fig. X

                                                                                      // Assigning indices
        for  j = 0 to 4                                                               perpendicular beam
            FPe_tag = 10000*tag + 2000 + j+1
            Add to N: 'node' + (FPe _tag + '1') +  x,y,z coordinates of FPe [j] start point   // id format: SBPTrs
            Add to nodesInfo: (FPe _tag + '1'), FPe start point, (FPe _tag + '1')
            Add to N: 'node' + (FPe _tag + '2') +  x,y,z coordinates of FPe [j] end point     // id format: SBPTrs
            Add to nodesInfo: (FPe _tag + '2'), FPe end point, (FPe _tag + '2')
            Add to E: 'element elasticBeamColumn' + FPe _tag + (FPe _tag +'1') + (FPe _tag + '2') + mat   // id format: SBPTrs

Joints
                                                                                      // Assigning indices
      for i = 0 to i = length of J -1                                                 according to the Op

        J_num = i+1
        J_tag = Js[i] + Je[i] + J_num
        Add to E: 'element twoNodeLink' + J_num +  (J_num+'19') + (J_num+'29') +      // id format: $S_iB_jP_kS$
        ' -mat $Stiffness_Tension $Stiffness_IP $Stiffness_OOP $Stiffness_about_Tension $Stiffness_about_OOP
      $Stiffness_about_IP -dir 1 2 3 4 5 6 -orient    1     0.0    0.0'
        Add to N: 'node' + (J_num+'1') +  x,y,z coordinates of J[i] start point       // id format: $S_iB_jP_kS$
        Add to nodesInfo: (J_tag + '1'), J[i] start point, (J_num+ '1')
        Add to N: 'node' + (J_num+'2') +  x,y,z coordinates of J[i] end point         // id format: $S_iB_jP_kS$
        Add to nodesInfo: (J_tag + '2'), J[i] end point, (J_num+ '2')
```

Boundary elements

```
for i = 0 to i = length of C -1
    Initialize PointsOnPline = []                                              // list for sorting all points on
    tag = 1000*S[i] + 10*B[i] + P[i]
    normal = cross product between vector from points[0] to points[1] and vector from points[0] to points[3]
        mat = ' $A_cnt $E_cnt $G_cnt $J_cnt $Iy_cnt $Iz_cnt' + tag             // assign material properties perpendicular beams
    for j = 0 to j = length of nodesInfo -1                                    // convert points to curve par
        if length of nodesInfo[j][0] < 10                                      // tags of all corners and inne
            if the four first digits of nodesInfo[j][0] correspond to the tag of the cell      // select only points lying on t
                Add to PointsOnPline: nodesInfo[j][2], curve parameter at nodesInfo[j][1]
        else                                                                   // tags of all joints
            if the four first digits of nodesInfo[j][0] correspond to the tag of the cell and the last one is 1    // select only points lying on t
                Add to PointsOnPline: nodesInfo[j][2], curve parameter at nodesInfo[j][1]
            if the four next digits of nodesInfo[j][0] correspond to the tag of the cell and the last one is 2     // select only points lying on t
                Add to PointsOnPline: nodesInfo[j][2], curve parameter at nodesInfo[j][1]
    Sort PointsOnPline by curve parameters
    for j = 0 to j = length of PointsOnPline -1
        cont_tag = 10000*tag + 4000 + j + 1                                    // identifier for the boundary
        string = 'element elasticBeamColumn' + cont_tag + PointsOnPline [j][0] + PointsOnPline [j+1][0] + mat     // id format: SBBPCnt
        if (the element connects corner nodes to corner pins)                  // modify string of hinge elem
            Replace 'elasticBeamColumn' by 'twoNodeLink' in string
        Add string to E
```

Export

```
Export N, E, G, D as text in .tcl format
```

References

[1] The Institute for Computational Design and Construction (ICD), https://icd.uni-stuttgart.de/, n.d.
[2] Institute for Building Structures and Structural Design (ITKE), https://www.itke.uni-stuttgart.de/, n.d.
[3] Li J, Knippers J. Segmental timber plate shell for the landesgartenschau exhibition hall in schwäbisch gmünd—the application of finger joints in plate structures. Int. J. Sp. Struct. 2015;30:123–40. https://doi.org/10.1260/0266-3511.30.2.123.
[4] O.D. Krieg, T. Schwinn, A. Menges, J.-M. Li, J. Knippers, A. Schmitt, V. Schwieger, Biomimetic lightweight timber plate shells: computational integration of robotic fabrication, architectural geometry and structural design, in: Adv. Archit. Geom. 2014, Springer International Publishing, 2015, pp. 109–125. https://doi.org/10.1007/978-3-319-11418-7_8.
[5] Magna R, Gabler M, Reichert S, Schwinn T, Waimer F, Menges A, et al. From nature to fabrication: biomimetic design principles for the production of complex spatial structures. Int. J. Sp. Struct. 2013;28:27–39. https://doi.org/10.1260/0266-3511.28.1.27.
[6] Robert McNeel & Associates, Rhinoceros 3D, 2018. https://www.rhino3d.com/.
[7] D. Rutten, Robert McNeel and associates, Grasshopper 3D, 2019. https://www.grasshopper3d.com/.
[8] SOFiSTiK AG, SOFiSTiK, https://www.sofistik.com/.
[9] ANSYS®, 2020. https://www.ansys.com/.
[10] Laboratory for Timber Constructions (IBOIS), https://www.epfl.ch/labs/ibois/
[11] Rezaei Rad A, Weinand Y, Burton H. Experimental push-out investigation on the in-plane force-deformation behavior of integrally-attached timber Through-Tenon joints. Constr. Build. Mater. 2019;215:925–40. https://doi.org/10.1016/j.conbuildmat.2019.04.156.
[12] Rezaei Rad A, Burton H, Weinand Y. Performance assessment of through-tenon timber joints under tension loads. Constr. Build. Mater. 2019;207:706–21. https://doi.org/10.1016/j.conbuildmat.2019.02.112.
[13] M. Deuss, A.H. Deleuran, S. Bouaziz, B. Deng, D. Piker, M. Pauly, ShapeOp—A Robust and Extensible Geometric Modelling Paradigm, in: M. Thomsen, M. Tamke, C. Gengnagel, B. Faircloth, F. Scheurer (Eds.), Model. Behav., Springer International Publishing, Cham, 2015, pp. 505–515. https://doi.org/10.1007/978-3-319-24208-8_42.
[14] C. Robeller, M. Konakovic, M. Dedijer, M. Pauly, A Double-layered Timber Plate Shell - Computational Methods for Assembly , Prefabrication and Structural Design, in: Adv. Archit. Geom. 2016, vdf Hochschulverlag AG, Zürich, Switzerland, 2016, pp. 104–122. https://doi.org/10.3218/3778-4.
[15] Nguyen AC, Vestartas P, Weinand Y. Design framework for the structural analysis of free-form timber plate structures using wood-wood connections. Autom. Constr. 2019;107:102948. https://doi.org/10.1016/j.autcon.2019.102948.
[16] Gregory BL, Shephard MS. The generation of airframe finite element models using an expert system. Eng. Comput. 1987;2:65–77. https://doi.org/10.1007/BF01213975.
[17] K. Suresh, Automating the CAD/CAE dimensional reduction process, in: E. G., S. V. (Eds.), Eighth ACM Symp. Solid Model. Appl., Department of Mechanical Engineering, University of Wisconsin – Madison, 1513 University Avenue, Madison, WI 608-262-3594, United States, 2003, pp. 76–85. https://doi.org/10.1145/781606.781621.
[18] Lee K. Principles of CAD/CAM/CAE Systems. Addison-Wesley; 1999.
[19] M. Hirz, W. Dietrich, A. Gfrerrer, J. Lang, Integrated computer-aided design in automotive development: Development processes, geometric fundamentals, methods of CAD, knowledge-based engineering data management, 1st ed., Springer-Verlag Berlin Heidelberg, 2013. https://doi.org/10.1007/978-3-642-11940-8.
[20] Robert McNeel & Associates, RhinoCommon – https://developer.rhino3d.com/guides/rhinocommon/, 2020.
[21] Rezaei Rad A, Burton H, Weinand Y. Out-of-plane (flatwise) behavior of through-tenon connections using the integral mechanical attachment technique. Constr. Build. Mater. 2020;262. https://doi.org/10.1016/j.conbuildmat.2020.120001.
[22] Rezaei Rad A, Burton HV, Weinand Y. Macroscopic model for spatial timber plate structures with integral mechanical attachments. J. Struct. Eng. (U.S.) 2020;146. https://doi.org/10.1061/(ASCE)ST.1943-541X.0002726.
[23] Robert McNeel & Associates, Rhino.Python – https://developer.rhino3d.com/guides/rhinopython/, 2020.
[24] S. Mazzoni, F. McKenna, M. Scott, G. Fenves, OpenSees [Computer Software]: The open system for earthquake engineering simulation, 2013. http://opensees.berkeley.edu/.
[25] Nolan DC, Tierney CM, Armstrong CG, Robinson TT. Defining simulation intent. Comput. Des. 2015;59:50–63. https://doi.org/10.1016/j.cad.2014.08.030.
[26] Sun L, Tierney CM, Armstrong CG, Robinson TT. Decomposing complex thin-walled CAD models for hexahedral-dominant meshing. Comput. Des. 2018;103:118–31. https://doi.org/10.1016/j.cad.2017.11.004.
[27] Rémondini L, Léon J, Trompette P. Towards an integrated architecture for the structural analysis of mechanical structures. Comput. Struct. 1998;67:299–307. https://doi.org/10.1016/S0045-7949(97)00139-9.
[28] Piacentino G. Grasshopper data trees and. Python 2019. , https://developer.rhino3d.com/guides/rhinopython/grasshopper-datatrees-and-python/.
[29] Fugier D, RhinoScriptSyntax, https://developer.rhino3d.com/guides/rhinopython/python-rhinoscriptsyntax-introduction/.
[30] Gamerro J, Bocquet JF, Weinand Y. A calculation method for interconnected timber elements using wood-wood connections. Buildings 2020;10:61. https://doi.org/10.3390/buildings10030061.
[31] Gamerro J, Bocquet JF, Weinand Y. Experimental investigations on the load-carrying capacity of digitally produced wood-wood connections. Eng. Struct. 2020;213:110576. https://doi.org/10.1016/j.engstruct.2020.110576.
[32] Finland VTT Expert Services Ltd, Kerto-VTT-C-184-03-Certificate:2016 Kerto-S and Kerto-Q Structural Laminated Veneer Lumber, Certificate NO.184/03, 2004.

[33] CEN (European Committee for Standardization), EN 12369-1: Wood-based Panels – Characteristic Value for Structural Design – OSB, particleboard and fibreboards, Brussels, Belgium, 2001.
[34] ANNEN SA, http://www.annen.lu/, 2020.
[35] Nguyen AC, Weinand Y. Displacement study of a large-scale freeform timber plate structure using a total station and a terrestrial laser scanner. Sensors 2020;20:413. https://doi.org/10.3390/s20020413.
[36] FARO Technologies Inc., FARO®, 2020. https://www.faro.com/.
[37] Blaß HJ, Streib J. Ingenious Hardwood: BauBuche Beech Laminated Veneer Lumber Design Assistance for Drafting and Calculation in Accordance with Eurocode 5. Creuzburg, Germany: Pollmeier Massivholz GmbH & Co.KG; 2017. https://doi.org/https://www.pollmeier.com/en/downloads/design-manual.html#gref.
[38] Marletta M, Pantò B. A new discrete element model for the evaluation of the seismic behaviour of unreinforced masonry buildings. Eng. Struct. 2012;40. https://doi.org/10.1016/j.engstruct.2012.02.039.
[39] Pantò B, Cannizzaro F, Caddemi S, Caliò I. 3D macro-element modelling approach for seismic assessment of historical masonry churches. Adv. Eng. Softw. 2016;97:40–59. https://doi.org/10.1016/j.advengsoft.2016.02.009.