# HEXAHEDRAL MESH GENERATION OF LAYERED SOLIDS WITH SLOPED LATERALS

Yu-Yao Lin[1] Anil Sehgal[2]

*Technology of Computer Aided Design, Intel Corporation, Hillsboro, OR, U.S.A.*
[1]*yu-yao.lin@intel.com* [2]*anil.sehgal@intel.com*

## ABSTRACT

We present an algorithm for fully-automated all-hexahedral mesh generation for 3D models of Very Large Scale Integration (VLSI) geometries, based on layered 2D polygonal mask layouts as input. Specifically, we propose a mapping procedure in three stages, based on integrating topology of 2D polygon sets from different layers, which define the architecture of a 3D VLSI geometry model. Three successive steps ensure that all geometric levels share the same topological structure, enabling extruded 3D meshing. In the first stage, the intersections between each pair of polygon sets are collected from top to bottom. The polygon sets which are absent on lower geometric levels are mapped downward. In the second stage, the polygon sets which are absent on upper geometric levels are mapped upward. In the third stage, we integrate the topology of each geometric level and construct a mapping through the levels. Finally, we generate a quadrilateral mesh on the bottom level and extrude the mesh to a hexahedral mesh through the mapping generated by the third stage. In the examples, we present the full hexahedral meshing result of various 3D models representing various sections of typical VLSI geometries. Moreover, we applied a quadrilateral and a hexahedral mesh optimization to ensure the validity of the meshes.

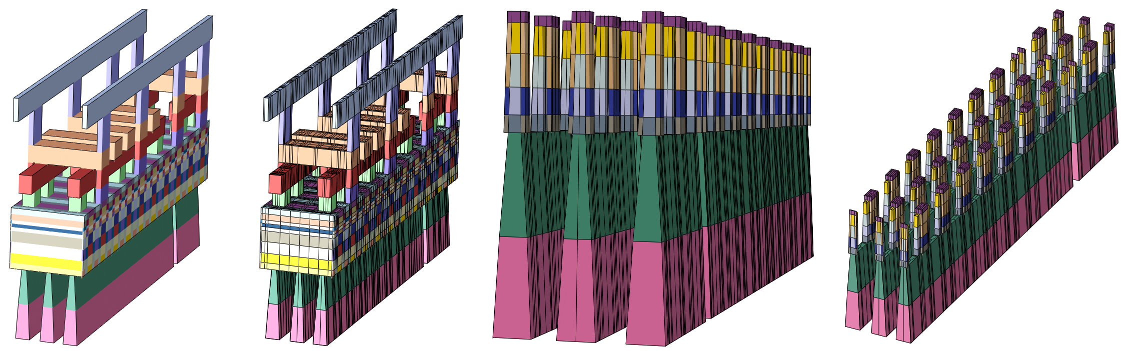
**Keywords: mesh generation, automatic hex-mesh generation, all-hexahedral meshing, mapped meshing, extrusion**

## 1. INTRODUCTION

Hexahedral meshes are widely used in finite element analysis such as thermal, mechanical, and a variety of other simulations. For certain simulations, hexahedral meshes are preferable due to the numerical accuracy and computational efficiency in comparison to tetrahedral meshes. Automated hex-meshing [1,2] has been widely studied for decades and there has been many existing tools to facilitate hex-meshing such as Abaqus [3], CUBIT [4], etc. Nevertheless, automated hexahedral mesh generation for a domain with multiple materials [5] is non-trivial in most cases, especially when a set of mesh quality constraints are taken into consideration.

A layered solid is a result of a variety of primitive manufacturing operations such as deposition, etching, polishing. Layered mesh generation is a typical meshing scheme for layer manufacturing processes [6, 7]. There has been some prior work of mesh generation for thin layered domains such as solid shells and thin section solids [8–11]. For instance, Quadros and Shimada [8] proposed an all-hexahedral meshing of thin section solids constructed by interpolating quadrilateral meshes from a chordal surface. Jaśkowiec *et al.* [10] developed a thin shell modeling by geometric transformation from a planar mesh to a curved laminated mesh. However, this type of layer meshings is either limited by a single geometric configuration or requires domain divisions. To handle the solids composed of complex geometries, the initial decomposition into simpler sub-domains has to be performed.

Model decomposition [12–14] is a common strategy for all-hexahedral meshing while it usually requires manual operations. In Intel, layered hexahedral meshes



(a) Simplified FinFET Transistor Geometry.

(b) Hex-mesh of the simplified FinFET Transistor Geometry.

(c) FinFET Transistor Mesh details.

(d) FinFET Transistor Mesh details.

**Figure 1**: A Simplified FinFET transistor and metal interconnect model composed of 16 layers for Thermal Modeling. The generic name "FinFET" is given as the source/drain region forms fin-shaped bodies on the silicon surface.

are applied for solving thermal stress and strain equations on VLSI models. To generate full hex-meshes satisfying our VLSI sloped geometries, we have tried manually decomposing solids and meshing by available meshing tools but failed to meet our needs. Typically, full hex-mesh with layer geometries can be generated with good quality without sloping material laterals and doing pure extrusion type of meshing. To do this, one can first collect geometries from all layers and imprint them to a 2D plane. Hex-meshing can therefore be achieved by extruding quadrilateral mesh with the constraint of the imprinted geometries. For example, MSC Apex [15] is a tool supporting quad-meshing on a basis with imprinted geometry and hex-meshing through extrusion. However, when sloped laterals are taken into account, projecting all geometries to a 2D basis is insufficient.

To deal with the sloped layer geometries, We have also tried producing sloped laterals by deforming material boundaries after straight extrusion, however, it only works for slight boundary deformation. When boundary geometry needs greater changes such as doubling the radius size of a circle, it rarely yields robust meshing results.

In the last few years, the creation of surface mappings [16], i.e. surface parameterizations, received a lot of attention due to their wide applicability in geometry processing. Inspired by surface mappings, we propose a novel approach that constructs a set of integrated geometry constraints through 2D surface mappings. Combine the typical mesh extrusion [17–21] with our meshing constraints, the full-hexahedral meshes with complex layered geometries are constructed. Our algorithm aims at automated all-hexahedral meshing for 3D layered solids of VLSI geometries that meets our requirement and solves the problems stated above. The produced meshes are without T-junctions and the need for decomposition for layered solids with sloped lateral material boundaries.

## 1.1 Layered Geometry in Semiconductor Industry

The algorithm proposed by this work focuses on VLSI models in the semiconductor industry. We present two examples in Figure 1 and 2 and experimental results for three types of model in Section 5. The structures of the models are following the definitions given in this subsection. The manufacturing process for semiconductors involves creating transistors, typically referred to as front-end processing. This is followed by creating layers of metal wires, surrounded by insulating material. Various layers of metals are connected in places by metal connectors called Via. The fin field-effect transistor or FinFET is a type of transistor commonly used in semiconductors, which is presented in Figure 1. It has been given the generic name "FinFET" because the source and the drain region form fin-shaped bodies on the silicon surface. To the integrated circuit (IC) packaging which refers to back-end processing, solder-bumps are fusible metal alloys used to connect with metal wires. Figure 4 shows that the solder-bumps are modeled by bottle shapes.

## 1.2 Contribution

The main contribution of this work is to propose a geometric and topological integration procedure, that allows us to automatically generate all-hexahedral meshes for layered solids with various regions, by extruding all quadrilateral meshes in $xy$-plane along $z$-axis. Figure 1 and 2 present the all-hexahedral meshes generated by our method. A layer of a layered solid is composed of 3D regions/materials which are represented by at least one set of polygons. The polygon sets representing a layer are topologically equivalent. If a layer is represented by two or more polygon sets, then we call it a *sloped layer* because there exists a 3D region with a sloped lateral surface defined by the polygon sets. To characterize the geometric and topological integration in Section 3, we detail the representation of layered solids in Section 2.

**Procedure 1** All-hexahedral mesh generation of layered solids with interior-sloped laterals.

**Input:** Layer stack $\mathcal{L}$
**Output:** All-hexahedral mesh $\mathcal{H}$
1: **function** INTEGRATEPOLYGONS($\mathcal{L}$)
2: $\quad\mathcal{P}$ = `IntegratePolygons`($\mathcal{L}$)
3: $\quad\Phi$ = `ConstructMap`($\mathcal{P}$)
4: **end function**
5: **function** QUADMESH($\mathcal{P}$)
6: $\quad\hat{P}_1$ = `GetTheBottomPolygonSet`($\mathcal{P}$)
7: $\quad\mathcal{Q}$ = `QuadMesh`($\hat{P}_1$)
8: **end function**
9: **function** EXTRUDETOHEX($\mathcal{Q}, \Phi$)
10: $\quad\mathcal{H} \leftarrow \emptyset$
11: $\quad\mathcal{Q}^b \leftarrow \mathcal{Q}$
12: $\quad l \leftarrow \mathcal{P}$.`Size`()
13: $\quad j \leftarrow 1$
14: $\quad$**while** $j < l$ **do**
15: $\qquad\mathcal{Q}^b$ = `OptimizeQuad`($\mathcal{Q}^b$)
16: $\qquad\mathcal{Q}^t$ = `ExtrudeQuad`($\mathcal{Q}^b, \Phi$)
17: $\qquad\mathcal{H}_j$ = `BuildHex`($\mathcal{Q}^t, \mathcal{Q}^b, \mathcal{H}$)
18: $\qquad\mathcal{H}_j$ = `OptimizeHex`($\mathcal{H}_j$)
19: $\qquad\mathcal{Q}^b \leftarrow \mathcal{Q}^t$
20: $\qquad j \leftarrow j + 1$
21: $\quad$**end while**
22: $\quad\mathcal{Q}^b$ = `OptimizeQuad`($\mathcal{Q}^b$)
23: **end function**

In Section 3, we introduce a mapping procedure that integrates the topology and geometry of 2D polygon sets and defines the geometric structure of VLSI models by multi-leveled polygon sets. Three successive stages ensure that all geometric levels share the same topological structure. In the first stage, the intersections between each pair of polygon sets are collected from top to bottom. The polygon sets which are absent on lower geometric levels are mapped downward (Section 3.1). In the second stage, the polygon sets which are absent on upper geometric levels are mapped upward (Section 3.2). In the third stage, we integrate the topology of each geometric level and construct a mapping through the levels (Section 3.3). In addition, polygon matching between topologically equivalent polygon sets is required to construct the mappings (Section 3.4). With the integrated constraints in the bottom level, we generate a quadrilateral mesh using the quadrilateral mesher Geompack [22] and extrude the mesh to a hexahedral mesh through the mapping generated in the third stage (Section 4). Finally, as the quadrilateral mesh might be distorted through the extruding process, quadrilateral and hexahedral mesh optimizations are working on the fly for mesh quality improvement. We implemented a quadrilateral optimizer and hexahedral optimizer by referring to the algorithm proposed by Escobar *et al.* [23], while this work focuses on the algorithm of automated hexahedral mesh generation of layered solids, we are not going to detail the optimizers.

Procedure 1 summarizes the main steps of the fully-automated all-hexahedral mesh generation. This work focuses on the process INTEGRATEPOLYGONS which integrates the geometries and topologies of a stack of layered geometries $\mathcal{L} = \{L_1, ..., L_n\}$ and generates a map $\Phi$ to enable the extrusion EXTRUDETOHEX from the quadrilateral mesh $\mathcal{Q}$ output by QUADMESH to a hexahedral mesh $\mathcal{H}$. The details of this process will be in Section 3.

**Algorithm 2** Geometric and topological integration of a stack of layers. $\mathcal{M}$ is a matrix storing polygon sets generated during the integration process.

**Input:** Stack of layers $\mathcal{L}$
**Output:** Stack of merged polygon sets $\mathcal{P}$
**function** INTEGRATEPOLYGONS($\mathcal{L}$)
$\quad\mathcal{L} \leftarrow$ `MapDownward`($\mathcal{L}$)
$\quad\mathcal{M} \leftarrow$ `MapUpward`($\mathcal{L}$)
$\quad l \leftarrow \mathcal{M}$.`NumberOfRows`()
$\quad\mathcal{P} \leftarrow$ `MergeRows`($\mathcal{M}, 1, l$)
**end function**

The algorithm `IntegratePolygons` in Line 2 is summarized in Algorithm 2 which performs the geometric and topological integration of $\mathcal{L}$ and generates a stack of polygon sets $\mathcal{P} = \{\hat{P}_1, ..., \hat{P}_l\}$ with the same topological structures, where $l$ is the number of the integrated polygon sets. In Line 3, `ConstructMap` constructs a map by generating a constrained Delaunay triangulation $\mathcal{T}_1$ to the bottom polygon set $\hat{P}_1$ and adopts the topological structure of $\mathcal{T}_1$ to generate a constrained triangulation to all the other polygon sets in $\mathcal{P}$. The triangulations are used to interpolate coordinates of nodes during mesh extrusion. Line 2 and 3 are the core concepts of this work and are elaborated in Section 3.

In Line 5, the process QUADMESH generates a quadrilateral mesh $\mathcal{Q}$ with the constraints of the bottom polygon set $\hat{P}_1$. Finally, the process EXTRUDETOHEX extrudes a hexahedral mesh $\mathcal{H}$ from $\mathcal{Q}$ with mesh optimizations `OptimizeQuad` and `OptimizeHex`. The algorithm `ExtrudeQuad` constructs a quadrilateral mesh $\mathcal{Q}^t$ by mapping the input mesh $\mathcal{Q}^b$ upward through the map $\Phi$. The algorithm `BuildHex` then constructs a layer of hexahedral mesh by connecting $\mathcal{Q}^b$ and $\mathcal{Q}^t$. At the end of the procedure, an all-hexahedral mesh $\mathcal{H} = \bigcup_{j=1}^{l-1} \mathcal{H}_j$ is generated.

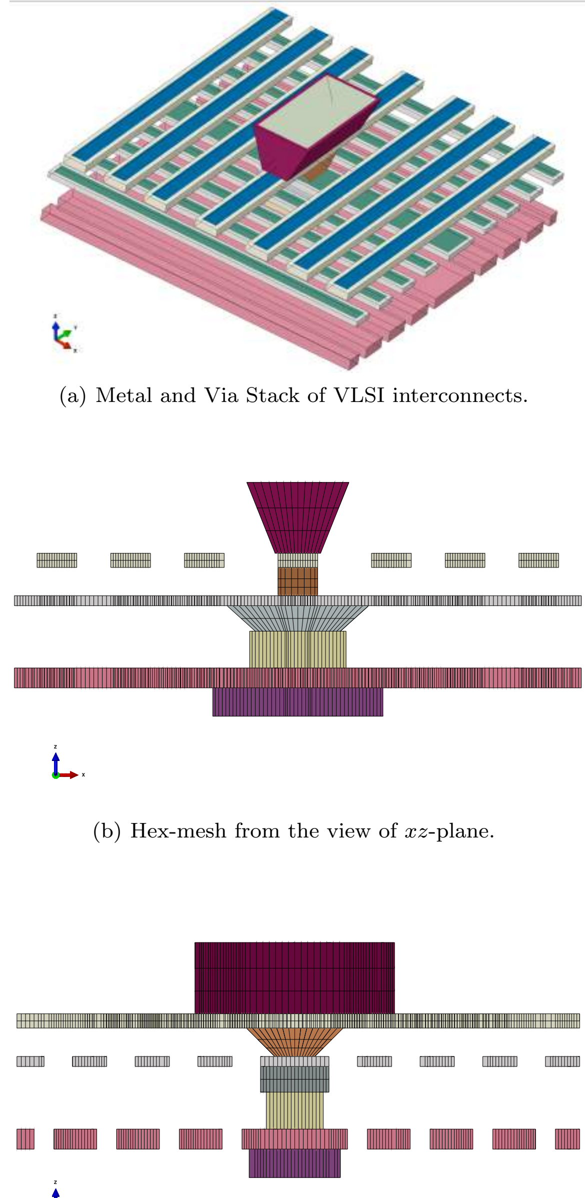
## 2. PRELIMINARIES

In this work, 3D VLSI models are represented by a stack of layers. Each layer is defined by a set of 2D polygonal mask layouts, i.e., a stack of 2D polygon sets. In this section, preliminary notions are deployed for studying the all-hexahedral meshing of the 3D layered solids.

To generate an all-hexahedral mesh for a layered solid $\Omega_{\mathcal{L}} \in R^3$, the structure of $\Omega_{\mathcal{L}}$ is represented by a stack of layers $\mathcal{L} = \{L_1, ..., L_n\}$. Each layer $L_i, i = 1, ..., n$ consists of a flat top surface $S_i^t$, flat bottom surface $S_i^b$, and lateral surface $S_i^l$. $S_i^t$ and $S_i^b$ are bounded by a 2D rectangle domain $\mathcal{D} \in R^2$ such that $S_i^t$, $S_i^b$ and $S_i^l$ form a hexahedron. The layer $L_i$ consists of multiple regions $\mathcal{A}_i = \{a_{i1}, ..., a_{im}\}$ with a thickness $h_i$. If a region $a_{ij}, j = 1, ..., m$ is with a vertical lateral surface, then it is defined by a 2D polygon $p_{ij}$ and the thickness $h_i$, such that the top and bottom surfaces $s_{ij}^t, s_{ij}^b$ of $a_{ij}$ are bounded by $p_{ij}$. On the other hand, if $a_{ij}$ is with a sloped lateral surface, then it requires a pair of topologically equivalent 2D polygons $p_{ij}^t$ and $p_{ij}^b$ to define its top and bottom surfaces. That is to say, a triple $\{p_{ij}^t, p_{ij}^b, h_i\}$ defines the region $a_{ij}$ of $L_i$.

A 2D polygon is bounded by a closed chain, or an outer loop, without self-intersections. It may have holes that are represented by inner loops. Each loop is formed by connecting nodes with edges on the $xy$-plane. The orientation of the outer loop is counter-clockwise, while the inner loops are clockwise. Some attributes may be assigned to a polygon like color, name, and material. In our setting, each hole has a corresponding polygon whose outer loop is identical to the hole and is counter-clockwise. The corresponding polygon represents either a region of a material or an empty space. Therefore, to the operations such as intersecting, merging, and mapping polygons in Section 3, only outer loops are considered as each inner loop has a corresponding outer loop to represent its geometry. While to the triangulation used for map generation and extrusion, inner loops are included to generate triangle mesh for each polygon and to construct independent regions for the final hexahedral mesh. All of the triangulations adopted in this work are generated using the constrained Delaunay triangulator [24].

Figure 3 shows a 3-layered solid. The top layer $L_3$ consists of metal wires represented by the red rectangles as shown in Figure 3(a); the middle layer $L_2$ and bottom layer $L_1$ both contain a Via of which the top and bottom shapes are represented by a pair of rectangles. See Figure 3(c) and 3(e) for the regions of Via.

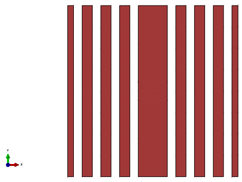


(a) Metal and Via Stack of VLSI interconnects.

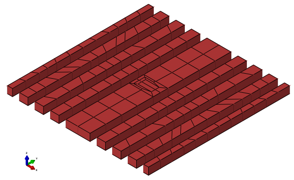
(b) Hex-mesh from the view of $xz$-plane.

(c) Hex-mesh from the view of $zy$-plane.

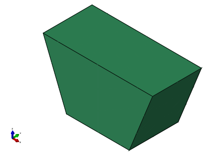
**Figure 2**: A VLSI Metal and Via Model composed of 8 layers.



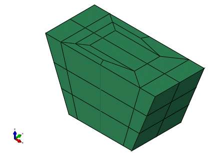
(a) Top view of contour of the metal wires layer.



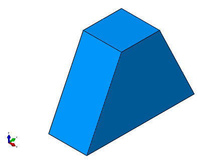
(b) Hex-meshed metal wires.



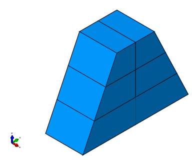
(c) 3D contour of the Via in the middle layer.



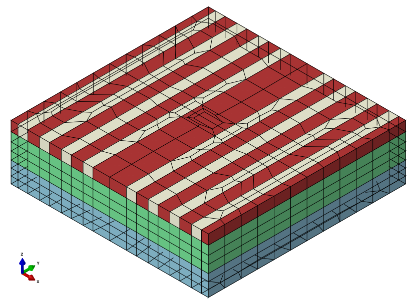
(d) Hex-meshed Via of the middle layer.



(e) 3D contour of the Via in the bottom layer.

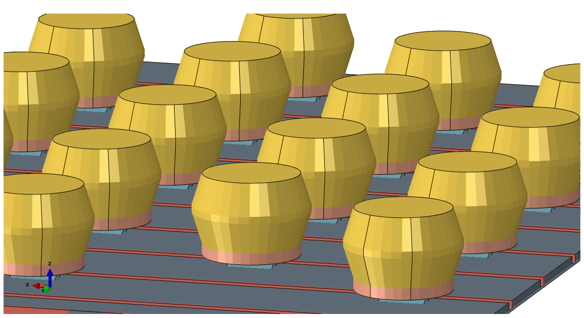


(f) Hex-meshed Via of the bottom layer.

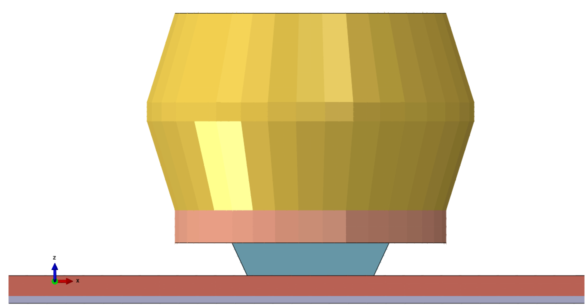


(g) All-hexahedral mesh of the 3-layered solid.

**Figure 3**: Hexahedral mesh of a model with three layers. The geometries of the layers are defined in a rectangle domain. The quad mesh is generated within the rectangular boundary and the hex-mesh is extruded along $+z$-axis. Aside from the metal wires and Vias, the other meshed volumes represent empty spaces whose materials are defined by air.



(a) The lateral surface of each of the Solder Bump regions (yellow) is approximated by four 2D circles.



(b) A Solder Bump region in yellow. A series of 2D circles are interpolated between each pair of adjacent input circles.

**Figure 4**: A layered solid of Solder Bump regions.

More generally, if the region $a_{ij}$ is with a curved lateral surface, we can approximate the surface by more than two topologically equivalent polygons. See Figure 4(a) and 4(b), the lateral surface of each of the bottle-shaped regions is approximated by four 2D circles. Notice that a series of horizontal surfaces are interpolated between each pair of adjacent input surfaces.

To well-define a layer $L_i$, we summarize the classification of layers as follows:

- **Straight layer:** If all regions of $L_i$ are with vertical lateral surfaces, then $S_i^t$ and $S_i^b$ are both defined by a set of polygons $P_i = \{p_{i1}, ..., p_{im}\}$.
- **Sloped layer-1:** If there exists a region of $L_i$ with a sloped lateral surface, then $S_i^t$ and $S_i^b$ are defined by $P_i^t = \{p_{i1}^t, ..., p_{im}^t\}$ and $P_i^b = \{p_{i1}^b, ..., p_{im}^b\}$ respectively. Each pair of polygons $p_{ik}^t$ and $p_{ik}^b$ are topologically equivalent, where $k = 1, ..., m$.
- **Sloped layer-2:** If there exists a region of $L_i$ whose curved lateral surface requires $w + 2$ polygons, $w > 0$, to approximate, then $L_i$ is defined by a series of topologically equivalent polygon sets $\{P_i^b, P_i^1, ..., P_i^w, P_i^t\}$. The number of the input polygon sets is $W_i = w + 2$. Figure 4 shows an example in which $w = 2$ for the top layer with solder bumps.

## 3. GEOMETRIC AND TOPOLOGICAL INTEGRATION

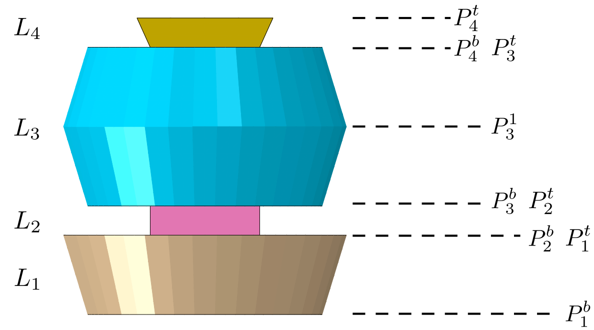
Our algorithm is inspired by the well-known geometric processing technique, surface mapping. In this section, we elaborate on how meshing constraints are integrated into a set of topologically equivalent polygon sets through mappings. A series of 2D polygon sets are given as the geometric representations of 3D layers. By integrating the topology and geometries of the polygon sets, a map that achieves the automated full hex-meshing is constructed ultimately. The geometric and topological integration includes the following steps:

(i) Intersecting and downward mapping polygon sets of layers from top to bottom;

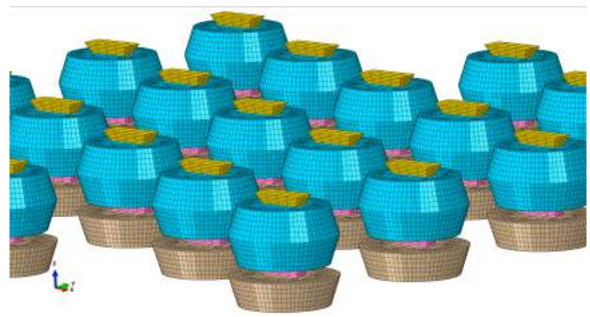
(ii) Merging and upward mapping polygon sets of layers from bottom to top;

(iii) Generating a map for mesh extrusion.

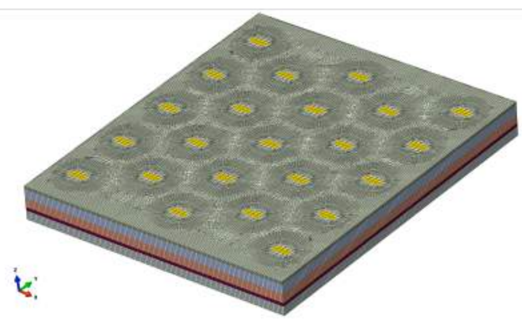
We first adopt a 4-layered solid as shown in Figure 5 to demonstrate the integration process, then we use




(a) $L_1$ and $L_4$ are defined by a pair of topologically equivalent polygon sets $\{P_1^b, P_1^t\}$ and $\{P_4^b, P_4^t\}$ respectively; $L_3$ is defined by $\{P_3^b, P_3^1, P_3^t\}$ where $P_3^1$ is an intermediate polygon set; $L_2$ is a straight layer defined by one polygon set, i.e., $P_2^b \equiv P_2^t$. Note that $\{P_3^b, P_2^b, P_1^t\}$ are in the same level as arranged in Table 1 as $L_2$ is a straight layer.



(b) Layered geometries.



(c) All-hexahedral mesh of the 4-layered solid. All of the geometries are enclosed by a 2D rectangle domain.

**Figure 5**: Hexahedral mesh of a 4-layered solid. The bottom layer $L_1$ and the top layer $L_4$ both include regions with sloped laterals; $L_3$ is a sloped layer with curved lateral regions; $L_2$ is a straight layer.

| | $L_4$ | $L_3$ | $L_2$ | $L_1$ |
|---|---|---|---|---|
| $l_5$ | $P_{54} \leftarrow P_4^t$ | $P_{53}$ | $P_{52}$ | $P_{51}$ |
| $l_4$ | $P_{44} \leftarrow P_4^b$ | $P_{43} \leftarrow P_3^t$ | $P_{42}$ | $P_{41}$ |
| $l_3$ | $P_{34}$ | $P_{33} \leftarrow P_3^1$ | $P_{32}$ | $P_{31}$ |
| $l_2$ | $P_{24}$ | $P_{23} \leftarrow P_3^b$ | $P_{22} \leftarrow P_2^b \equiv P_2^t$ | $P_{21} \leftarrow P_1^t$ |
| $l_1$ | $P_{14}$ | $P_{13}$ | $P_{12}$ | $P_{11} \leftarrow P_1^b$ |

**Table 1**: The polygon sets in green are generated by downward mapping through $\phi_3$. The polygon sets in red are generated by downward mapping through $\phi_1$. The polygon sets in blue are generated by upward mapping through $\phi_{34}$. The polygon sets in orange are generated by upward mapping through $\phi_4^{-1}$. Finally, the polygon sets in each column belong to the same layer and share the same topological structure.

the simpler model of Figure 3 to better elaborate algorithm details.

To show that our approach supports curved lateral approximation, the 4-layered solid in Figure 5 is constructed with a sloped layer $L_3$ defined by 3 polygon sets. The input polygon sets of the 4-layered solid is described as follows: The bottom layer $L_1$ and the top layer $L_4$ both include regions with sloped laterals, and are defined by a pair of topologically equivalent polygon sets $\{P_1^b, P_1^t\}$ and $\{P_4^b, P_4^t\}$ respectively. $L_3$ is a sloped layer with curved laterals approximated by $\{P_3^b, P_3^1, P_3^t\}$ where $P_3^1$ is an intermediate polygon set. That is to say, the number of input polygon sets of $L_3$ is $W_3 = 3$. $L_2$ is a straight layer which requires only one polygon set as input, i.e., $P_2^b \equiv P_2^t$.

We arrange the polygon sets to Table 1 to illustrate how polygon intersection, mapping, and merging are performed. Since $L_2$ is a straight layer, $\{P_3^b, P_2^b, P_1^t\}$ in Figure 5(a) are arranged to the same level in Table 1. Initially, the polygon sets $P_{ji}$ in black are given by the input polygon sets of layer $L_i$ and recorded at level $l_j$. The number of levels/rows of polygon sets is calculated by $l = 1 + \Sigma_{i=1}^{n}(W_i - 1)$, where $n$ is the number of layers, $W_i$ is the number of input polygon sets of layer $L_i$. In our implementation, polygon sets are stored in a $l \times n$ matrix $\mathcal{M}$.

The overall process is summarized in the following list:

1. Downward mappings
   (a) Intersect $L_3$ with $L_4$ by generating intersection nodes on their top and bottom polygon sets $P_{43}$ and $P_{44}$.
   (b) Interpolate new nodes on $P_{33}$ and $P_{23}$ by referring to the intersection nodes on $P_{43}$. Interpolate new nodes on $P_{54}$ by referring to the intersection nodes on $P_{44}$.
   (c) Construct a multi-level map $\phi_3$ with the constraints of $P_{23}, P_{33}, P_{43}$.
   (d) $\phi_3$ maps $P_{44}$ to $P_{34}$, and $P_{34}$ to $P_{24}$. The bottom polygon set of $L_4$ is updated to $P_{24}$.
   (e) Intersect $L_2$ with $L_3, L_4$ by generating intersection nodes on $P_{22}$ and $P_{23}, P_{24}$.
   (f) Interpolate new nodes on $P_{33}$ and $P_{43}$ by referring to the intersection nodes on $P_{23}$. Interpolate new nodes on $P_{34}, P_{44}, P_{54}$ by referring to the intersection nodes on $P_{24}$.
   (g) Intersect $L_1$ with $L_2, L_3, L_4$ by generating intersection nodes on $P_{21}$ and $P_{22}, P_{23}, P_{24}$.
   (h) Interpolate new nodes on $P_{33}, P_{43}$ by referring to the intersection nodes on $P_{23}$. Interpolate new nodes on $P_{34}, P_{44}, P_{54}$ by referring to the intersection nodes on $P_{24}$. Interpolate new nodes on $P_{11}$ by referring to the intersection nodes on $P_{21}$.
   (i) Construct a map $\phi_1$ with the constraints of $P_{21}, P_{11}$.
   (j) $\phi_1$ maps $P_{24}$ to $P_{14}$, $P_{23}$ to $P_{13}$, and $P_{22}$ to $P_{12}$.
2. Upward mappings
   (a) Create merged polygon sets $\tilde{P}_2 = P_{23} \cup P_{24}$, $\tilde{P}_3 = P_{33} \cup P_{34}$, and $\tilde{P}_4 = P_{43} \cup P_{44}$.
   (b) Construct a multi-level map $\phi_{34}$ with the constraints of $\tilde{P}_2, \tilde{P}_3, \tilde{P}_4$.
   (c) $\phi_{34}^{-1}$ maps $P_{22}$ to $P_{32}$, $P_{32}$ to $P_{42}$, $P_{21}$ to $P_{31}$, and $P_{31}$ to $P_{41}$.
   (d) Construct a map $\phi_4$ with the constraints of $P_{44}$ and $P_{54}$.
   (e) $\phi_4^{-1}$ maps $P_{43}$ to $P_{53}$, $P_{42}$ to $P_{52}$, and $P_{41}$ to $P_{51}$.
3. Map for extrusion
   (a) Merge the polygon sets row by row to a set of topologically equivalent polygon sets $\mathcal{P} = \{\hat{P}_1, \hat{P}_2, \hat{P}_3, \hat{P}_4, \hat{P}_5\}$.
   (b) Construct a multi-level map $\Phi$ with the constraints of $\mathcal{P}$.

1a, 1e, and 1g perform intersections of polygon sets between layers in order to generate intersection nodes which are imprinted on all the polygon sets at the same row. 1b, 1f, and 1h apply node interpolations to the column of polygon sets of layers either upward or downward. Node interpolations guarantee the topological equivalence of polygon sets of each layer. 1c, 1i, 2b, and 2d construct maps for mapping polygon sets. 1d, 1j, 2c, and 2e generate new polygon sets by mapping the top or bottom polygon set of layers. In 2a, $P_{23}, P_{33}, P_{43}$ are topologically equivalent and $P_{24}, P_{34}, P_{44}$ are also topologically equivalent. Moreover, the intersection nodes and interpolated nodes of the polygons guarantee the topological equivalence after polygon merging. In the last step, 3a merges all polygons at each row such that the merged polygon sets share the same topological structure. 3b constructs a map which preserves material boundaries of layers and enables mesh extrusion.

Besides the above example, we conclude the operations required by the integration process into a series of algorithms. The algorithms are accompanied by more details and illustrations through the example of Figure 3. In the example, a 3-layered solid composed of a straight layer with metal wires and two sloped layers with Via is considered. Figure 6 to 9 demonstrate the integration process of the 3-layered solid and the algorithms are presented through Section 3.1 to 3.4.

**Algorithm 3** Downward mappings

**Input:** Layer stack $\mathcal{L}$
**Output:** Layer stack $\mathcal{L}$
1: **function** MAPDOWNWARD($\mathcal{L}$)
2: $q_{intersect} \leftarrow \emptyset$
3: $q_{map} \leftarrow \emptyset$
4: $q_{intersect}$.`push`($L_n$)
5: **for** $i = n - 1, ...1$ **do**
6: **while** !$q_{intersect}$.`empty`() **do**
7: $L \leftarrow q_{interect}$.`front`()
8: $(L, L_i) \leftarrow$ `GetControlNodes`$(L, L_i)$
9: $q_{map}$.`push`($L$)
10: $q_{intersect}$.`pop`()
11: **end while**
12: **if** $L_i$ is a sloped layer **then**
13: $\mathcal{P} \leftarrow$ `GetPolygonSets`($L_i$)
14: $\phi_i =$ `ConstructMap`($\mathcal{P}$)
15: **for** $L \in q_{map}$ **do**
16: $q_{map}$.`pop`()
17: $P \leftarrow$ `GetTheBottomPolygonSet`($L$)
18: $P \leftarrow$ `MapDownward`$(P, \phi_i)$
19: $q_{intersect}$.`push`($L$)
20: **end for**
21: **else**
22: $q_{intersect} \leftarrow q_{map}$
23: $q_{map} \leftarrow \emptyset$
24: **end if**
25: $q_{intersect}$.`push`($L_i$)
26: **end for**
27: **end function**

## 3.1 Downward Mappings

Algorithm 3 performs downward mappings by maintaining two queues $q_{intersect}$ and $q_{map}$. Initially, the top layer $L_n$ is pushed into the queue $q_{intersect}$. The other layers are iterated from $j = n - 1$ to 1 such that the function `GetControlNodes` in Line 8 performs polygon intersection between the bottom polygon set of $L$ from queue $q_{intersect}$ and the top polygon set of $L_i$. By referring to the generated intersection nodes, `GetControlNodes` interpolates new nodes to all the other polygon sets of $L$ and $L_i$. In Line 9, $L$ is pushed into $q_{map}$ to wait for being mapped downward. At the end of intersections, if $L_i$ is a sloped layers, then a map $\phi_i$ is constructed by `ConstructMap` given in Algorithm 7. The bottom polygon sets of layers in $q_{map}$ are mapped downward to generate new bottom polygon sets in Line 18. The mapped layers are pushed into $q_{intersect}$ for intersections in the next iteration. If $L_i$ is a straight layer, then we move all the layers from $q_{map}$ to $q_{intersect}$. $L_i$ is then pushed into $q_{intersect}$ and the next iteration continues.

Figure 6 shows how downward mappings are applied to the 3-layered solid of Figure 3. In consists of the following steps:

(D1) The bottom polygon set $P_{33}$ of $L_3$ intersects with the top polygon set $P_{32}$ $L_2$;

(D2) Interpolated nodes are generated on the bottom polygon set $P_{22}$ of $L_2$ by referring to the intersection nodes on $P_{32}$;

(D3) Construct a map $\phi_2$ with the constraints of $P_{22}, P_{32}$;

(D4) $P_{23}$ is generated by mapping $P_{33}$ through $\phi_2$. The bottom polygon set of $L_3$ is therefore become $P_{23}$;

(D5) $P_{23}, P_{22}$ intersect with the top polygon set $P_{21}$ of $L_1$;

(D6) Interpolated nodes are generated on the bottom polygon set $P_{11}$ of $L_1$ by referring to the intersection nodes on $P_{21}$;

(D7) Construct a map $\phi_1$ with the constraints of $P_{21}, P_{11}$;

(D8) $P_{13}, P_{12}$ are generated by mapping $P_{23}, P_{22}$ through $\phi_1$. The bottom polygon set of $L_3$ and $L_2$ are thereafter become $P_{13}$ and $P_{12}$.

## 3.2 Upward Mappings

Algorithm 4 shows the upward mapping of polygon sets. To facilitate the upward mapping, polygon sets output by Algorithm 3 are arranged into a matrix $\mathcal{M}$ by `ArrangePolygonSetsToMatrix`. Algorithm 5 gives the details of the arrangement where `InitializeAMatrix` initializes an empty $l \times n$ matrix $\mathcal{M}$ and `FillMatrix` stores polygon sets of layers into $\mathcal{M}$. That is, the polygon set of layer $L_i$ in level $j$ is filled into $\mathcal{M}$ as $P_{ji}$. Note that the polygon sets of top layer $L_n$ has all been generated through downward mapping. Therefore, the number of levels/rows $l$ equals to the number of polygon sets of $L_n$.

Line 9 to 11 in Algorithm 4 scans the rows of matrix $\mathcal{M}$ upward to find the first row $r_b$ where the number of polygon sets in row $r_b$ is greater than the number of polygon sets in row $r_b + 1$. Moreover, due to the downward mappings, $\mathcal{M}.\texttt{RowSize}(r_b + 1) <= \mathcal{M}.\texttt{RowSize}(r_b)$ is always true. Line 14 to 16 finds the second row $r_t$ where $\mathcal{M}.\texttt{RowSize}(r_t + 1) < \mathcal{M}.\texttt{RowSize}(r_t)$. From row $r_b$ to row $r_t$, `MergeRows` in Line 17 merges $\mathcal{M}.\texttt{RowSize}(r_t)$ polygon sets as detailed by Algorithm 6. With the constraints of the resulting polygon sets, `ConstructMap` constructs a map $\phi$ by triangulating each of the sets with the same triangular structure. Line 19 to 23 performs upward mappings through $\phi^{-1}$ to fill matrix $\mathcal{M}$ upward. For general cases, the upward filling process repeats until $\mathcal{M}$ is fully filled.

Figure 7 performs the upward mappings of the 3-layered solid as follows:

(U1) Merge the polygon sets $\{P_{23}, P_{22}\}$ of row 2 and $\{P_{33}, P_{32}\}$ of row 3, and construct a map;

(U2) Map $P_{21}$ upward through the map.

The red frames show the merged polygon sets which play the constraints of map construction. The top polygon set $P_{21}$ of $L_1$ is mapped upward through the map.

## 3.3 Map for Extrusion

After filling matrix $\mathcal{M}$, Algorithm 6 merges polygon sets $\{P_{j1}, ..., P_{jn}\}$ at each row $j$ to a polygon set $\hat{P}_j$ to realize mesh extrusion with the constraints of the merged polygon sets. The polygon constraints in each row well preserve the shapes of regions through extrusion. Figure 8 shows that the polygon sets of each row are merged such that the resulting polygon sets are topologically equivalent and are used to construct a multi-level map for extrusion.

To perform the extrusion, a multi-level map is constructed by following Algorithm 7:

1) Matching polygons across the polygon sets $\hat{P}_j$ for $j = 1, ..., l$;

2) Generating a Delaunay triangulation $\Delta_1$ with the constraints of the merged polygon set $\hat{P}_1$ in the bottom level;

3) Generating a triangulation for each of the other merged polygon sets using the same structure as $\Delta_1$.

Figure 9 shows the triangulations for the map of the 3-layered solid.

**Algorithm 4** Upward mappings

**Input:** Layer stack $\mathcal{L}$
**Output:** Matrix of polygon sets $\mathcal{M}$

1: **function** MapUpward($\mathcal{L}$)
2: $\quad \mathcal{M} \leftarrow \texttt{ArrangePolygonSetsToMatrix}(\mathcal{L})$
3: $\quad l \leftarrow \mathcal{M}.\texttt{NumberOfRows}()$
4: $\quad n \leftarrow \mathcal{M}.\texttt{NumberOfColumns}()$
5: $\quad$ **if** $l <= 1$ **then**
6: $\quad\quad$ return
7: $\quad$ **end if**
8: $\quad r_b, r_t \leftarrow 1$
9: $\quad$ **while** $r_b < l$ **and** $\mathcal{M}.\texttt{RowSize}(r_b + 1) == \mathcal{M}.\texttt{RowSize}(r_b)$ **do**
10: $\quad\quad r_b \leftarrow r_b + 1$
11: $\quad$ **end while**
12: $\quad$ **while** $r_t < l$ **do**
13: $\quad\quad r_t \leftarrow r_b + 1$
14: $\quad\quad$ **while** $r_t < l$ **and** $\mathcal{M}.\texttt{RowSize}(r_t + 1) == \mathcal{M}.\texttt{RowSize}(r_t)$ **do**
15: $\quad\quad\quad r_t \leftarrow r_t + 1$
16: $\quad\quad$ **end while**
17: $\quad\quad \mathcal{P} \leftarrow \texttt{MergeRows}(\mathcal{M}, r_b, r_t)$
18: $\quad\quad \phi = \texttt{ConstructMap}(\mathcal{P})$
19: $\quad\quad$ **for** $i = \mathcal{M}.\texttt{RowSize}(r_t) + 1, ..., n$ **do**
20: $\quad\quad\quad$ **for** $j = r_b, ..., r_t$ **do**
21: $\quad\quad\quad\quad P_{(j+1)i} = \texttt{MapUpward}(P_{ji}, \phi)$
22: $\quad\quad\quad$ **end for**
23: $\quad\quad$ **end for**
24: $\quad\quad r_b \leftarrow r_t$
25: $\quad$ **end while**
26: **end function**

**Algorithm 5** Arrange polygon sets to a matrix.

**Input:** Layer stack $\mathcal{L}$
**Output:** Matrix of polygon sets $\mathcal{M}$

1: **function** ArrangePolygonSetsToMatrix($\mathcal{L}$)
2: $\quad l = \texttt{GetNumberOfPolygonSets}(L_n)$
3: $\quad n = \texttt{GetNumberOfLayers}(\mathcal{L})$
4: $\quad \mathcal{M} \leftarrow \texttt{InitializeAMatrix}(l, n)$
5: $\quad$ **for** $i = n, ..., 1$ **do**
6: $\quad\quad s = \texttt{GetNumberOfPolygonSets}(L_i)$
7: $\quad\quad$ **for** $j = 1, ..., s$ **do**
8: $\quad\quad\quad \texttt{FillMatrix}(\mathcal{M}, L_i, i, j)$
9: $\quad\quad$ **end for**
10: $\quad$ **end for**
11: **end function**

**Algorithm 6** Merge polygon sets row by row.

**Input:** Polygon matrix $\mathcal{M}$, row indices $r_b, r_t$
**Output:** Polygon sets $\{\tilde{P}_{r_b}, ..., \tilde{P}_{r_t}\}$

1: **function** MergeRows($\mathcal{M}, r_b, r_t$)
2: $\quad n = \mathcal{M}.\texttt{NumberOfColumns}()$
3: $\quad n' = n - \mathcal{M}.\texttt{RowSize}(r_t) + 1$
4: $\quad$ **for** $j = r_b, ..., r_t$ **do**
5: $\quad\quad \tilde{P}_j \leftarrow \texttt{MergePolygonSets}(P_{jn}, ..., P_{jn'})$
6: $\quad$ **end for**
7: **end function**

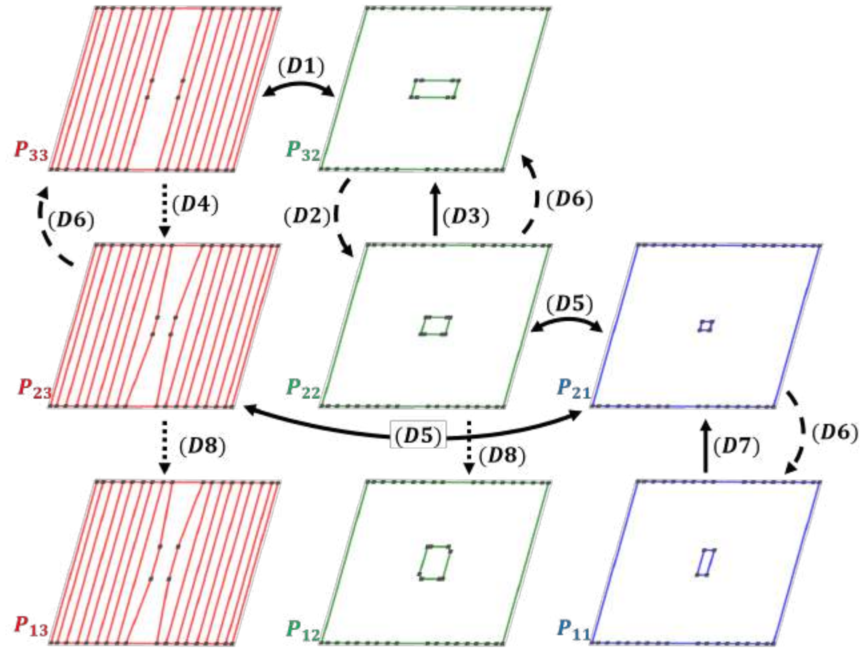



**Figure 6**: Polygon sets generation by downward mappings. $(D1)$ Intersect $P_{33}$ and $P_{32}$; $(D2)$ Interpolate nodes on $P_{22}$ according to the nodes on $P_{32}$ after $(D1)$ is done; $(D3)$ Construct a map $\phi_2$ with the constraints of $P_{22}, P_{32}$ and the nodes generated in step $(D1)$ and $(D2)$; $(D4)$ Map down $P_{33}$ through $\phi_2$ to $P_{23}$; $(D5)$ Intersect $P_{21}$ with each polygon set at the same level; $(D6)$ Interpolate nodes on $P_{11}$ according to the nodes on $P_{21}$ after $(D5)$ is done; $(D7)$ Construct a map $\phi_1$ with the constraints of $P_{21}, P_{11}$ and the nodes generated in step $(D5)$ and $(D6)$; $(D8)$ Map polygon sets at the second level downward through $\phi_1$ such that $P_{13}$ and $P_{12}$ are generated.

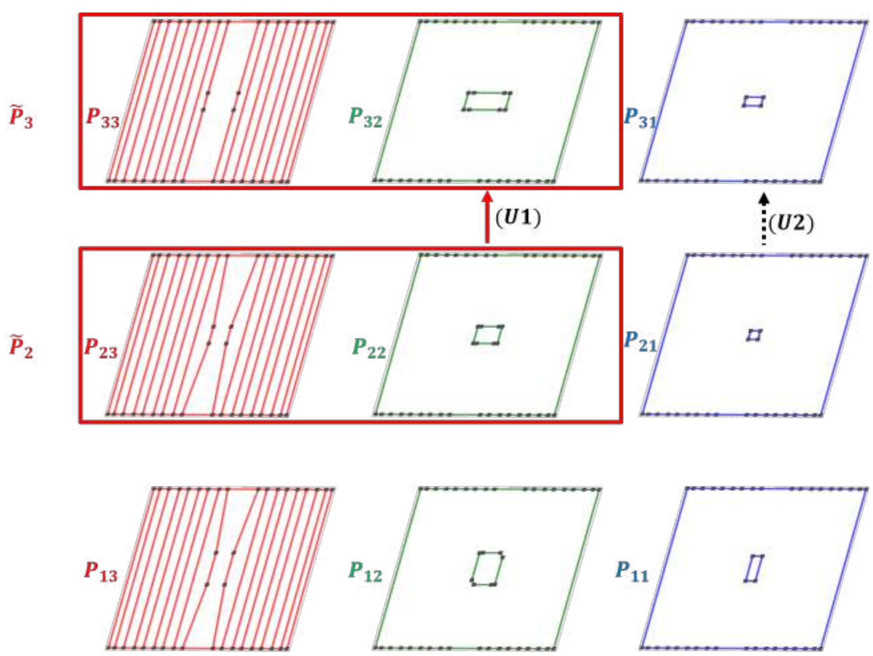



**Figure 7**: Polygon sets generation by upward mappings. $(U1)$ Merge the polygon sets $\{P_{23}, P_{22}\}$ of row 2 and $\{P_{33}, P_{32}\}$ of row 3, and construct a map; $(U2)$ Map $P_{21}$ upward through the map.

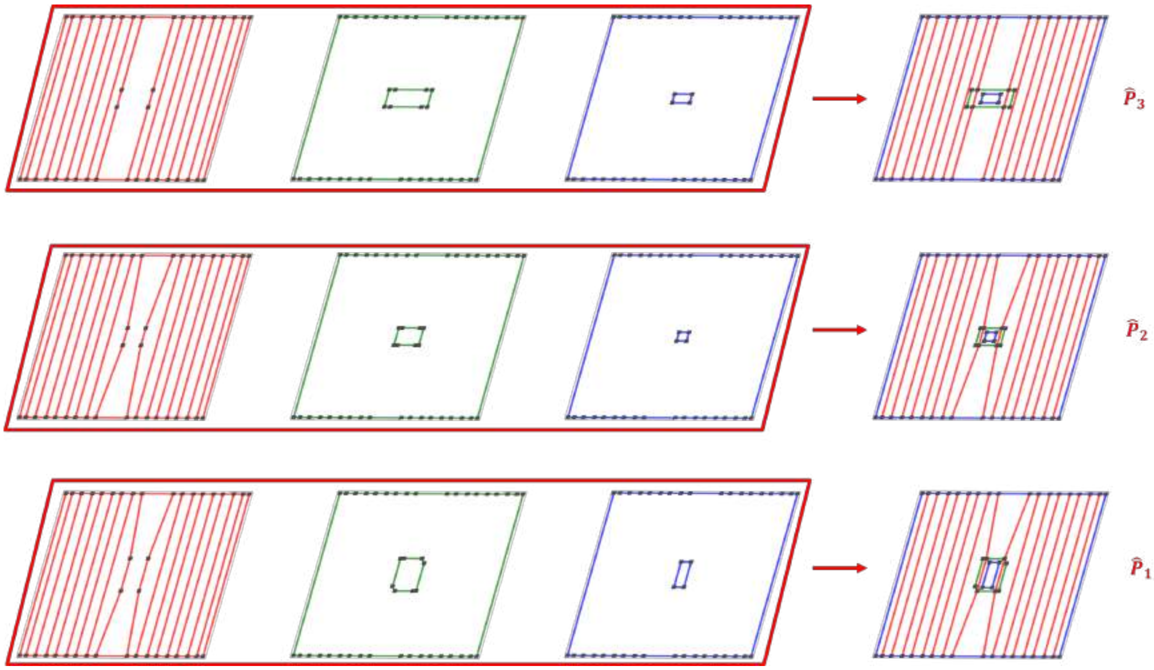



**Figure 8**: Merge all polygon sets at the same row into $\mathcal{P} = \{\hat{P}_1, \hat{P}_2, \hat{P}_3\}$; Under the constraints of $\hat{P}_1$, a Delaunay triangulation is generated. As the merged polygon sets are topologically equivalent, a topologically equivalent triangulation of the other rows are generated accordingly. The triangulations form a multiple-layered map $\Phi$.

**Algorithm 7** Construct a map.

**Input:** Polygon set $\mathcal{P}$
**Output:** Map $\Phi$

```
function ConstructMap(P)
    P ← MatchPolygons(P)
    P̃_1 = GetTheBottomPolygonSet(P)
    Δ_1 = ConstrainedDelaunayTriangulate(P̃_1)
    for j = 2, ..., l do
        Δ_j = ConstrainedTriangulate(P̃_{j-1}, P̃_j)
    end for
    Φ = {Δ_1 ∪ P̃_1, ..., Δ_l ∪ P̃_l}
end function
```

## 3.4 Polygon Matching

The input polygonal layouts of downward mapping consist of topologically equivalent polygon pairs so polygon matching is not required. However, for upward mapping and the map construction for extrusion which contain the steps of polygon merging, the topological structure of merged polygon sets changed after polygon merging, therefore, polygon matching is necessary.

The well-known Hungarian matching algorithm [25] is utilized to match polygons and nodes in pairs. In the classic Hungarian algorithm, there are $k$ agents and $k$ tasks. Any agent can be assigned to perform any task, incurring some costs that may vary depending on the agent-task assignment. It is required to perform all tasks by assigning exactly one agent to each task and exactly one task to each agent in such a way that the total cost of the assignment is minimized. The run-time complexity is $O(k^3)$ and the optimality is guaranteed. In our polygon matching, $k$ is the number of polygons.

In our setting, the input is two sets of polygons $P_1$ and $P_2$ where the numbers of polygons are equal, i.e., $k = |P_1| = |P_2|$. The key of applying Hungarian algorithm is how to define the cost of matching a pair of polygon. The cost can be defined by the difference of the attributes between two polygons $p, q$ such as area $\delta A(p,q)$, 2D centroid $\delta C(p,q)$, number of nodes $\delta n(p,q)$, number of holes $\delta h(p,q)$, etc. The cost function can be defined as

$$\begin{aligned} c(p,q) = \delta A(p,q) * w_0 + \delta C(p,q) * w_1 \\ + \delta n(p,q) * w_2 + \delta h(p,q) * w_3 \end{aligned}$$

where $w_0, w_1, w_2, w_3$ are weights associated to each of the terms.

Polygon matching and node matching are not only imperative to constructing mappings but also to maintaining topology consistency. After constructing the final map for extrusion, some polygon manipulations are applicable. Edge refinement may be required to better approximate curved geometries, edge simplification and polygon removal may be applied to simplify

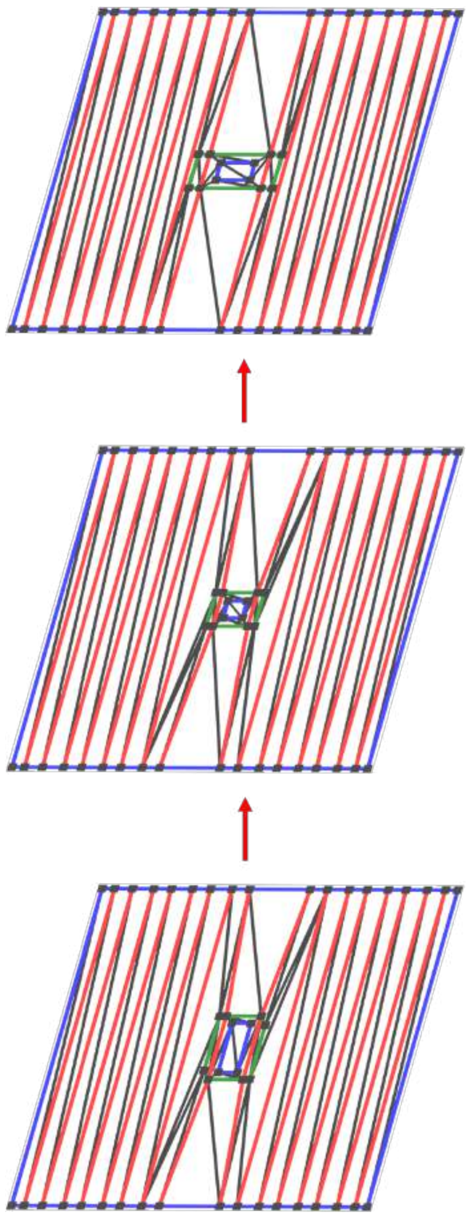


**Figure 9**: Generate a Delaunay triangulation under the constraint of the bottom polygon set. The number of polygons in the three sets is equal and the polygons are matched across the sets. That is, each polygon in a set is paired with one polygon in each of the other sets. Therefore, by referring to the triangulation at the bottom level, a triangulation with the same structure is constructed to each of the upper polygon sets.

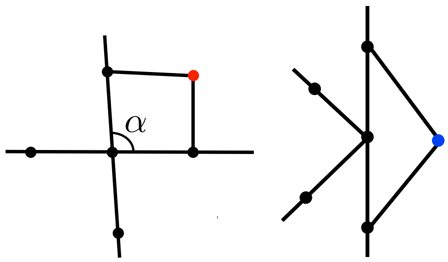
geometries or to reduce mesh size. When edge refinement (simplification) is applied to the polygon set at some level, the edges at other levels matched to the refined edge can be located and refined accordingly. If a polygon is eliminated at some level, the matched polygons at other levels are also eliminated. The polygon structures are locally reconstructed so that the topology across all the levels is still consistent.

## 4. MESH EXTRUSION AND OPTIMIZATION

At the bottom row of the right-most column in Figure 8, the polygon set imprints the mapped geometry from all layers. We utilize Geompack [22] quadmesher to generate quad-mesh. The quad-mesh is extruded along the $z$-axis to an all-hexahedral mesh with interior-sloped laterals through the generated map $\Phi$.

During the process of extrusion, the extruded quadrilateral meshes might be distorted. Explicit tangling where elements overlap with each other should not happen through our approach, as quad-nodes are generated by interpolating a new position in a triangle. However, implicit tangling might happen as the interpolation does not guarantee quad-element convexity. Therefore, quadrilateral and hexahedral mesh optimizers are automatically applied. We implemented quadrilateral and hexahedral optimizers by referring to the tet-mesh optimization proposed by Escobar *et al.* [23]. As the objective of this work is to discusses the algorithm of automated all-hexahedral mesh generation of layered solids, details of the optimizations are skipped. Figure 11 shows a comparison of the quadrilateral mesh before and after optimization. Notice that the nodes on the material boundaries (between the grey and yellow regions) are fixed so the optimization process does not distort the material shapes. In fact, for nodes on horizontal/vertical boundary, only horizontal/vertical movement is allowed in our optimizations.

There exists a tricky case that a valid corner might become invalid after extrusion. Figure 10 gives an example where $\alpha$ is a corner of polygon constraints of quadrilateral meshing. The red node in Figure 10(a) is extruded to the blue node in Figure 10(b). The angle of $\alpha$ is distorted to $\pi$ after extrusion. In this case, the straightened edges are on the boundary of distinguished materials on the extruded layer. The invalid corner is unable to be fixed through optimizations. To tackle the situation, if an angle $\theta_1$ of a corner $\alpha_1$ in the merged polygon set of bottom level $\hat{P}_1$ has a matched corner $\alpha_j$ with an angle $\theta_j \geq \pi$ in merged polygon set $\hat{P}_j$, where $1 \leq j \leq l$, then $\alpha_1$ and all of its matched corners are bisected. That is, the polygon in which this type of corner locates is decomposed into two polygons. The other corner to connect with for



(a) Before extrusion. (b) After extrusion.

**Figure 10**: $\alpha$ is a corner of polygon constraints of quadrilateral meshing. The red node on the left is extruded to the blue node on the right. The angle of $\alpha$ is distorted to $180^\circ$ after extrusion. In this case, the straightened edges are on the boundary of distinguished materials on the extruded layer. The invalid corner is unable to be fixed through optimizations.

decomposition is determined by minimizing the sum of the differences between each of the four bisected angles and their average.

## 5. EXPERIMENTAL RESULTS

We present the hexahedral meshing results of our algorithm by four tested layered models. The *metal-Via* model contains 3 sloped Via layers and 7 metal wire layers. The *FinFET-1* model contains 13 sloped layers and the *FinFET-2* model contains 10 sloped layers. Figure 1 shows a simplified *FinFET* model and Figure 4 shows the *solder-bump* model that contains 2 sloped layers. A snapshot of the *metal-Via* model is illustrated in Figure 11. Figure 2 shows a simplified *metal-Via* with only 8 layers while the *metal-Via* in Table 2 is a full *metal-Via* model composed of 40 layers. In our experience, increasing the number of layers increases the value of $\theta_{max}$ and decreases the value of $J_{min}$ as polygon intersection happens more often and polygons are more likely to deform.

In Table 2, the total number of hex-elements, layers, and regions of the models are shown. $t_1$ is the over-all meshing time, $t_2$ is the time of executing the geometric and topological integration `IntegratePolygons` of Algorithm 2. $\theta_{min}$ and $\theta_{max}$ are the minimal and maximal angles of quad-elements. $J_{min}$ and $J_{max}$ represent the minimal and maximal scaled Jacobians of hex-elements.

To improve mesh quality under the constraints of hundreds and thousands of polygonal boundaries, in some cases tuning parameters of optimization such as the number of iterations, maximum angle, and step length of line search is required. Table 2 shows the measures of the most complex models we have tackled.

| Name | metal-Via | FinFET-1 | FinFET-2 | solder-bump |
|---|---|---|---|---|
| #hexes | 2002336 | 196911 | 237546 | 1056263 |
| #layers | 40 | 32 | 24 | 5 |
| #regions | 112 | 210 | 157 | 14 |
| $t_1$ (sec.) | 631.92 | 103.67 | 65.75 | 95.76 |
| $t_2$ (sec.) | 58.93 | 88.88 | 46.39 | 18.87 |
| $\theta_{min}$ | 3.69 | 2.72 | 1.03 | 5.39 |
| $\theta_{max}$ | 175.47 | 177.58 | 178.91 | 175.55 |
| $J_{min}$ | 0.0024 | 0.014 | 9.31e-05 | 0.039 |
| $J_{max}$ | 1 | 1 | 1 | 1 |

**Table 2**: Results of four layered models with multiple regions.

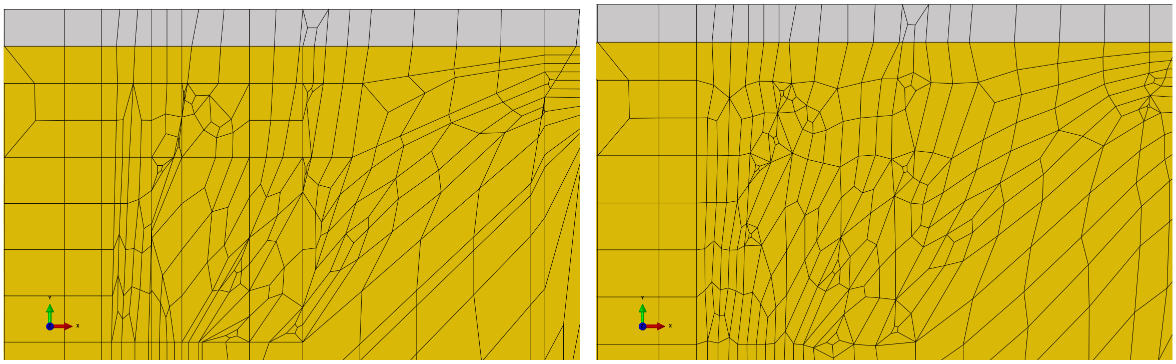
Although the angles and Jacobians are not optimized, these meshes are routinely applied for solving thermal stress and strain equations using both internal solvers and commercial solvers such as Abaqus [3] without issues. We have also carried out benchmarks comparing numerical solvers on the same mesh to ensure that results are not skewed due to numerical issues.

## 6. CONCLUSION AND FUTURE WORK

In this paper, we have introduced an automated all-hexahedral mesh generation for 3D models of VLSI geometries. The models are layered solids with interior sloped-lateral surfaces. The core idea is to integrate topology and geometry from the top layer to the bottom layer of a given solid. The integration builds a stack of polygonal constraints such that the polygon sets in all levels of the stack are topologically equivalent. A final mapping is constructed through a stack of polygon sets and a quadrilateral mesh is generated according to the polygon constraints at the bottom level. By extruding the quadrilateral mesh through the final map, an all-hexahedral mesh is generated.

There is no guarantee of the quality of meshes by the quality of the triangulations of a map. However, the mapping strategy guarantees that there are no shape distortions of regions in layered solids. Moreover, quad-mesh and hex-mesh optimization are applied to improve mesh quality. Another important property of our mapping method is that mesh adaptation does not incur any shape distortions. That is, the edges of polygons can be refined to generate finer meshes. If an edge of a polygon is refined, then the corresponding edges of the polygons in all the other rows must be refined accordingly. It means that the polygon sets are still topologically equivalent, and the refined edges remain straight through the mapped extrusion. Polygon degeneration is also handled because polygon matching helps to maintain topological equivalence.

Interesting future work includes constraint triangulation with nodes in polygon interiors for map construction, modeling 3D region boundaries by general-curved surfaces, and developing robust quad/hex-



(a) Before quad-optimization.

(b) After quad-optimization.

**Figure 11**: The snapshots show a comparison of mesh quality before and after quad-mesh optimization. The grey and yellow regions represent metal wires in some layer of the metal-Via model from the experimental results.

optimization algorithms to comprehensively improve the quality of our layered meshes.

## References


[1] Kremer M., Bommes D., Lim I., Kobbelt L. "Advanced automatic hexahedral mesh generation from surface quad meshes." *Proceedings of the 22nd International Meshing Roundtable*, pp. 147–164. Springer, 2014

[2] Yu W., Zhang K., Li X. "Recent algorithms on automatic hexahedral mesh generation." *2015 10th International Conference on Computer Science & Education (ICCSE)*, pp. 697–702. IEEE, 2015

[3] Dassault Systemes. "ABAQUS 2017." URL `http://www.3ds.com/products-services/simulia/products/abaqus`

[4] Blacker T.D., Owen S.J., Staten M.L., Quadros W.R., Hanks B., Clark B.W., Meyers R.J., Ernst C., Merkley K., Morris R., McBride C., Stimpson C.J., Plooster M., Showman S. "CUBIT geometry and mesh generation toolkit 15.1 user documentation." 2 2016. URL `https://www.osti.gov/biblio/1430472`

[5] Zhang Y., Hughes T.J., Bajaj C.L. "Automatic 3d mesh generation for a domain with multiple materials." *Proceedings of the 16th international meshing roundtable*, pp. 367–386. Springer, 2008

[6] Roberts I.A., Wang C., Esterlein R., Stanford M., Mynors D. "A three-dimensional finite element analysis of the temperature field during laser melting of metal powders in additive layer manufacturing." *International Journal of Machine Tools and Manufacture*, vol. 49, no. 12-13, 916–923, 2009

[7] Luo Z., Zhao Y. "A survey of finite element analysis of temperature and thermal stress fields in powder bed fusion additive manufacturing." *Additive Manufacturing*, vol. 21, 318–332, 2018

[8] Quadros W.R., Shimada K. "Hex-Layer: Layered All-Hex Mesh Generation on Thin Section Solids via Chordal Surface Transformation." *IMR*, pp. 169–180. Citeseer, 2002

[9] Kulikov G., Plotnikova S. "Exact geometry SaS solid-shell element for 3D stress analysis of FGM piezoelectric structures." *Curved and Layered Structures*, vol. 5, no. 1, 116–135, 2018

[10] Jaśkowiec J., Stankiewicz A., Pluciński P. "Three-dimensional numerical modelling of multi-layered shell structures using two-dimensional plane mesh." *Advances in Engineering Software*, vol. 149, 102840, 2020

[11] Reiter S., Logashenko D., Vogel A., Wittum G. "Mesh generation for thin layered domains and its application to parallel multigrid simulation of groundwater flow." *Computing and Visualization in Science*, vol. 23, no. 1, 1–8, 2020

[12] Tautges T.J. "The generation of hexahedral meshes for assembly geometry: survey and progress." *International Journal for Numerical Methods in Engineering*, vol. 50, no. 12, 2617–2642, 2001

[13] Lu Y., Gadh R., Tautges T.J. "Feature based hex meshing methodology: feature recognition and volume decomposition." *Computer-Aided Design*, vol. 33, no. 3, 221–232, 2001

[14] Su Y., Lee K., Kumar A.S. “Automatic hexahedral mesh generation for multi-domain composite models using a hybrid projective grid-based method.” *Computer-Aided Design*, vol. 36, no. 3, 203–215, 2004

[15] Pura J. “An Introduction to Hex Meshing for Finite Element Analysis (FEA) with MSC Apex.” URL `https://simulatemore.mscsoftware.com/an-introduction-to-hex-meshing-for-finite-element-analysis-fea-with-msc-apex/`

[16] Floater M.S., Hormann K. “Surface parameterization: a tutorial and survey.” *Advances in multiresolution for geometric modelling*, pp. 157–186, 2005

[17] White D.R., Mingwu L., Benzley S.E., Sjaardema G.D. “Automated hexahedral mesh generation by virtual decomposition.” *Proceedings of the 4th International Meshing Roundtable, Sandia National Laboratories, Albuquerque, USA*, pp. 165–176. 1995

[18] Shih B.Y., Sakurai H. “Automated hexahedral mesh generation by swept volume decomposition and recomposition.” *5th International Meshing Roundtable*, vol. 280. Citeseer, 1996

[19] Owen S.J. “A survey of unstructured mesh generation technology.” *IMR*, vol. 239, 267, 1998

[20] Knupp P.M. “Applications of mesh smoothing: copy, morph, and sweep on unstructured quadrilateral meshes.” *International Journal for Numerical Methods in Engineering*, vol. 45, no. 1, 37–45, 1999

[21] White D.R., Tautges T.J. “Automatic scheme selection for toolkit hex meshing.” *International Journal for Numerical Methods in Engineering*, vol. 49, no. 1-2, 127–144, 2000

[22] Joe B. “GEOMPACK—a software package for the generation of meshes using geometric algorithms.” *Advances in Engineering Software and Workstations*, vol. 13, no. 5-6, 325–331, 1991

[23] Escobar J.M., Rodrıguez E., Montenegro R., Montero G., González-Yuste J.M. “Simultaneous untangling and smoothing of tetrahedral meshes.” *Computer Methods in Applied Mechanics and Engineering*, vol. 192, no. 25, 2775–2787, 2003

[24] Shewchuk J.R. “A two-dimensional quality mesh generator and Delaunay triangulator.” *Computer Science Division University of California at Berkeley, Berkeley, California*, pp. 94720–1776, 2008. URL `http://www.cs.cmu.edu/quake/triangle.html`

[25] Kuhn H.W. “The Hungarian method for the assignment problem.” *Naval research logistics quarterly*, vol. 2, no. 1-2, 83–97, 1955