gMidSurf: Hierarchical GPU-based Mid-surface Abstraction for Thin-walled CAD Models

Li Ye
Zhejiang University
Hangzhou 310007, China
li-ye@zju.edu.cn

Xinhang Zhou
Zhejiang University
Hangzhou 310007, China

Xingyu Yang
Zhejiang University
Hangzhou 310007, China

Peng Fan
Zhejiang University
Hangzhou 310007, China

Ruofeng Tong
Zhejiang University
Hangzhou 310007, China

Hailong Li
Poisson Software Co., Ltd.
Shenzhen 518129, China

Peng Du
Zhejiang University
Hangzhou 310007, China

Min Tang*
Zhejiang University
Hangzhou 310007, China
tang_m@zju.edu.cn

Abstract

Mid-surface abstraction is essential for finite element analysis of thin-walled CAD models, yet existing face pairing-based methods suffer from quadratic complexity and CPU-bound bottlenecks, limiting scalability for variable-thickness models. We present *gMidSurf*, a GPU-accelerated pipeline that transforms the two computational bottlenecks in mid-surface abstraction (face pairing and mid-point generation) into massively parallel operations. For face pairing, we introduce a hierarchical filtering strategy that progressively culls candidate pairs through three GPU-optimized gates: normal compatibility, simplified overlap criterion, and LBVH-based distance queries, reducing the search space by 10–$100\times$ while maintaining cache coherence. For mid-point generation, we employ parallel distance dilation followed by bracket-and-bisect refinement for precise equidistant point localization. This method handles variable-thickness models with complex surfaces through complete dilation, thereby avoiding gaps and truncations that occur in previous methods. Experimental results on real-world benchmarks demonstrate that *gMidSurf* achieves $4.2\times$–$18.5\times$ speedups in face pairing and $4.8\times$–$9.8\times$ in mid-point generation compared to CPU implementations, yielding $5\times$–$15\times$ acceleration on a commodity GPU (NVIDIA RTX 5090D) compared to state-of-the-art methods.

Keywords: ***Mid-surface Abstraction, Variable Thin-walled Models, GPU Acceleration***

1. Introduction

Mid-surface abstraction is a foundational preprocessing task in computer-aided engineering (CAE). Thin-walled components are ubiquitous across aerospace, automotive, and mechanical engineering domains. By simplifying three-dimensional solid models into two-dimensional mid-surface representations, engineers can reduce computational degrees of freedom by an order of magnitude while maintaining plate/shell analysis fidelity, thereby significantly improving finite element analysis efficiency [3, 1].

Existing mid-surface abstraction methods fall into three primary categories: Medial Axis Transform (MAT) [2], Chordal Axis Transform (CAT) [14], and Face Pairing (FP) methods. MAT generates skeletal structures by computing the locus of maximal inscribed spheres, but its topological complexity necessitates extensive manual pruning [17]. CAT converts thin-walled solids into single-layer tetrahedral meshes, but struggles to maintain geometric quality in complex junction regions [15, 13]. In contrast, face pairing methods identify opposing wall faces and construct intervening sheets, preserving B-Rep modeling intent, making them the factual standard in CAD/CAE systems [16].

However, current face pairing-based methods face severe challenges when processing complex variable-thickness models. First, traditional face pairing algorithms rely on global candidate screening and expensive closest-point queries, leading to dramatic performance degradation on large-scale, free-form, variable-thickness models [20, 21]. Second, existing methods lack robustness for variable-

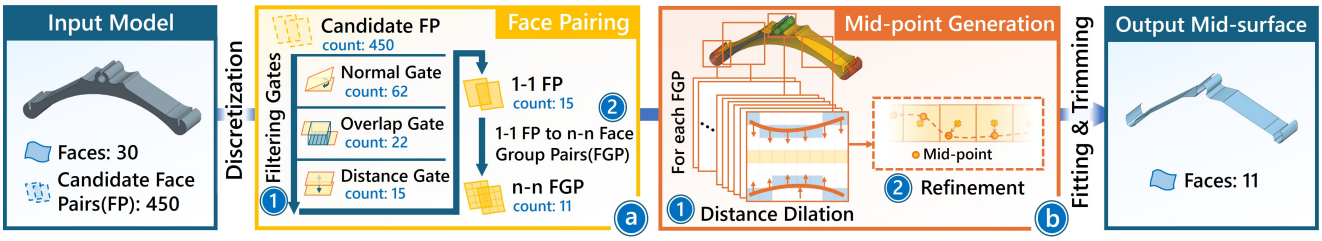

Figure 1. The algorithm overview of *gMidSurf* through a simple example. The input thin-walled model is first discretized, followed by (a) hierarchical GPU-based face pairing with three filtering gates (normal, overlap, and distance) to identify face group pairs, and (b) GPU-based mid-point generation via distance dilation and refinement. Finally, fitting and trimming operations produce the output mid-surface.

thickness geometry, where normal-projection heuristics frequently fail when the face angles vary significantly [25].

We introduce *gMidSurf*, a full-GPU pipeline that systematically addresses computational bottlenecks in both face pairing and mid-point generation stages (Fig. 1). For face pairing, we design a hierarchical filtering mechanism with three progressive gates (normal, overlap, distance, Fig. 1-a1) to reduce the candidate space by 10×–100×. For mid-point generation, we employ parallel distance dilation to generate initial mid-points (Fig. 1-b1), followed by bracket-and-bisect refinement with warp-level parallelism for precise equidistant point localization (Fig. 1-b2). The method supports variable-thickness models and handles complex *1-1* and *n-n* pairing scenarios (Fig. 1-a2). Our technical contributions are:

- **Full GPU-based face pairing with hierarchical culling.** We cast normal, overlap, and distance tests as GPU-friendly gates backed by LBVH traversal, pruning candidates by $10\times$–$100\times$ before fine checks and yielding near-linear scaling with face count.
- **Branchless mid-point solver for variable-thickness.** We introduce a bracket-and-bisect kernel that equalizes distances robustly across planes, quadrics, and free-form NURBS while minimizing warp divergence and maximizing throughput.

Beyond acceleration, our complete dilation-based mid-point generation inherently addresses the gap artifacts that arise in MidSurfer's directional sampling approach, producing more complete mid-surfaces without requiring manual intervention for face group ordering. To our knowledge, gMidSurf represents the first full GPU pipeline for face pairing-based mid-surface abstraction, addressing a long-standing efficiency bottleneck that has limited practical adoption in interactive CAD/CAE workflows.

2. Related Work

2.1. Face Pairing-based Mid-surface Abstraction

Mid-surface abstraction is commonly grouped into three methods: Medial Axis Transform (MAT), Chordal Axis Transform (CAT), and Face Pairing (FP). MAT extracts a skeletal locus of maximally inscribed spheres; while broadly useful for shape analysis, it typically produces spurious branches that demand extensive pruning and post-processing [2, 17]. CAT converts thin-walled solids into single-layer tetrahedral meshes and classifies elements to recover a sheet [15, 13]; however, complex ribs and free-form junctions stress classification robustness, and the geometric quality of the recovered surface is not guaranteed [9].

FP has become the dominant paradigm because it preserves B-Rep intent by explicitly identifying opposing wall faces and constructing the intervening sheet. Rezayat [16] first introduced Face Adjacency Graphs (FAG) with distance/normal criteria to detect pairs; subsequent work reduced clutter and improved topological consistency via boundary extension [11] and feature suppression [18]. Commercial solutions (e.g., Parasolid [19]) expose FP utilities that combine user-specified and automatically detected pairs; however, complex free-form regions still require substantial manual curation.

Across this literature, pipelines are largely serial, creating throughput bottlenecks. The face pairing stage must traverse large candidate sets and apply compound geometric tests; the mid-surface generation stage relies on intensive distance queries. These workloads exhibit poor cache coherence and limited vector efficiency on CPUs, resulting in multi-second (or longer) processing times even for models with only a few faces. To address this, *gMidSurf* introduces GPU parallel computing to these two core stages in mid-surface abstraction, achieving millisecond-level processing for complex variable-thickness models through hierarchical parallel face pairing and dilation-based mid-point extraction algorithms.

2.2. GPU Acceleration in Geometric Processing

Modern GPUs offer massive thread- and memory-level parallelism for the two core parts of mid-surface abstraction: (i) spatial search over large candidate sets and (ii) dense proximity/distance evaluation. On the search side, fast BVH/LBVH construction and traversal on GPUs have matured, enabling scalable broad- and mid-phase culling for large geometric workloads [10]. On the proximity

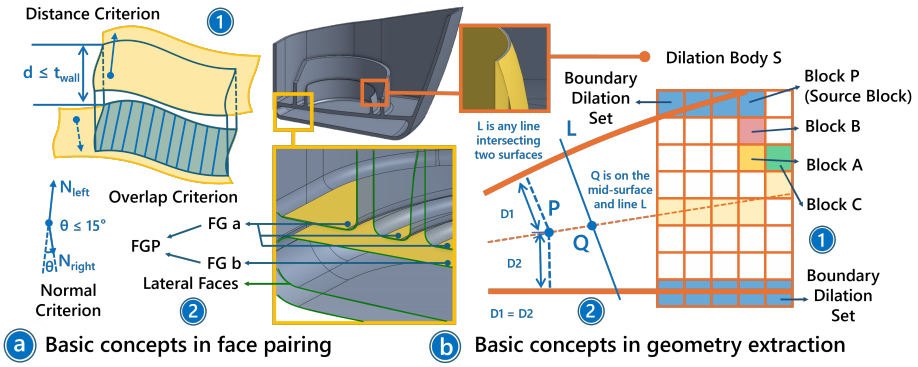

Figure 2. Basic concepts in face pairing and geometry extraction. a) Three pairing criteria and face group pair (FGP) definition. b) Dilation algorithm fundamentals and mid-surface definition.

side, recent *mesh–mesh* distance work shows that carefully organized BVH traversal, shared-memory buffering, and warp-synchronous voting/reduction can deliver order-of-magnitude gains over strong CPU baselines; notably, gDist reports 50×–200× improvements on general triangle meshes [5]. These results motivate our choices for GPU-resident face pairing and mid-point extraction.

Beyond proximity, GPU acceleration has been applied to skeleton/medial computations adjacent to mid-surface abstraction. Zhu et al. parallelize medial-axis estimation via dilation- and tracing-based schemes, reporting 8×–12× speedups [24, 23]; Jalba et al. demonstrate GPU-based surface and curve skeletonization of large 3D polygonal meshes, achieving two orders of magnitude speed-up [8]. While these efforts validate GPU-based geometric abstraction, they target subroutines (skeletonization) rather than the FP-based mid-surface pipeline.

Prior GPU work typically accelerates *one* component (e.g., BVH build/traversal, distance evaluation, or skeletonization) in isolation. In contrast, *gMidSurf* implements an *end-to-end* GPU strategy tailored to face pairing: (1) a hierarchical, divergence-aware culling scheme that formalizes normal, overlap, and distance tests atop LBVH traversal to shrink the candidate set by 10×–100× *before* fine checks; and (2) a branchless bracket-and-bisect kernel for distance-equalizing mid-point extraction that maintains warp coherence across planes, quadrics, and NURBS. This integration avoids host–device thrashing and converts the quadratic candidate explosion into a near-linear, throughput-friendly pass.

3. Preliminaries

Classical mid-surface pipelines comprise three main components: (i) face pairing, (ii) mid-surface geometry extraction, and (iii) surface fitting and trimming. Among these, (i) and (ii) dominate runtime [21] and form the efficiency bottlenecks we address.

3.1. Basic Concepts for Face Pairing

Definition 1. Three key criteria are widely used in the current face pairing method, as illustrated in Fig. 2-a1:

1. **Normal Criterion**: The outward normals N_{left} and N_{right} of the paired faces must be approximately antiparallel, satisfying:

$$|180^\circ - \angle(\mathbf{N}_{left}, \mathbf{N}_{right})| \leq \theta, \tag{1}$$

where θ denotes the angular tolerance threshold, typically set $\theta \leq 15^\circ$ in industrial applications.

2. **Overlap Criterion**: When projecting either face (the projection face) along its normal direction onto the other face (the target face), the projected area ratio should satisfy:

$$\text{Project}(f_{left}, f_{right}) \geq R \vee \text{Project}(f_{right}, f_{left}) \geq R, \tag{2}$$

where R defaults to 50% and can be specified.

3. **Distance Criterion**: the distance d between the two candidate faces should not exceed a threshold t_{wall}:

$$\text{Dist}(f_{left}, f_{right}) \leq t_{wall}, \tag{3}$$

where t_{wall} defaults to the maximum thickness of the thin-walled model and can be manually specified.

Definition 2. A face group (FG) is a set of topologically continuous faces sharing identical underlying geometry. We use face groups as fundamental units for face pairing.

Definition 3. Each face in the thin-walled model is classified into two distinct categories: face (group) pairs (FP / FGP) and lateral faces (Fig. 2-a2).

1. Face (group) pair: refers to a pair of faces or FGs in a thin-walled model that satisfies the aforementioned three criteria.

2. Lateral face: an elongated intermediate surface connecting the two constituent faces of a face group pair in thin-walled structures.

Remark on parameter settings. The default parameter values used in our pipeline follow established conventions in prior face pairing methods [20, 21, 16]. These values have been validated across diverse industrial applications and represent widely accepted defaults in practice.

3.2. Basic Concepts for Geometry Extraction

The following concepts are adapted from prior work on dilation-based model reduction methods, which inspired our method [6, 24].

Definition 4. A dilation block is a partition from a uniform grid subdivision perpendicular to the coordinate axes. We construct a dilation body from an FGP. Its *boundary set* comprises dilation blocks containing all FGP faces. Fig. 2-b1 illustrates a 2D example where two curves form a dilation body after subdivision, with blue blocks defining the boundary set.

Definition 5. The distance from dilation block A to dilation body S is the Euclidean distance from A's center to the nearest boundary block center in S. This nearest block is A's source block, and its FG is A's source FG. The local/global *Euclidean Distance Transformation (EDT)* comprises local/global distances of all blocks in the body [22]. In Fig. 2-b1, block P is the source block for yellow block A, with its FG as the touch FG.

Definition 6. Distance dilation from block A to its neighbor C (6-neighborhood in 3D) updates C's distance using A's information [22]. If A's source block offers a shorter path to C, it becomes C's new source block. Dilation repeats until all blocks have distances. Fig. 2-b1 shows a single dilation from boundary block P to block C (sequence: $P \rightarrow B \rightarrow A \rightarrow C$). Dilation from all boundary blocks in body S yields *complete dilation*, where all blocks acquire their distances.

The following concepts derive from prior work on precise mid-surface point extraction [21], which our approach parallelizes to post-process initial solutions after *complete dilation* (Fig. 2-b2).

Definition 7. Given two face groups FG_ℓ and FG_r forming a candidate FGP, a point P lies on the mid-surface $\mathcal{M}$ if

$$\forall P \in \mathcal{M},\ \mathrm{Dist}(P, FG_\ell) = \mathrm{Dist}(P, FG_r), \qquad (4)$$

where $\mathrm{Dist}(P, FG)$ is the minimum Euclidean distance from P to the faces in FG.

Definition 8. For any line intersecting two surfaces from distinct FG, there exists at least one mid-surface point between them. Given surfaces S_{left} and S_{right} intersected by line L at points A and B respectively, consider the continuous distance difference function:

$$f(Q) = \mathrm{Dist}(Q, S_{left}) - \mathrm{Dist}(Q, S_{right}), \qquad (5)$$

where $Q \in L$. Since $f(A) < 0$ and $f(B) > 0$ (as $A \in S_{left}$ and $B \in S_{right}$), the Intermediate Value Theorem guarantees the existence of at least one point Q_0 where $f(Q_0) = 0$, establishing the mid-surface point.

4. Hierarchical GPU-based Face Pairing

Conventional face pairing methods [20, 21, 23] require enumerating all pairwise combinations of faces in the model, evaluating three criteria for each candidate face pair, and testing feasibility against all criteria. This approach exhibits a significant drawback: even if a pair fails to meet one criterion, the computation of the remaining criteria must still be completed, resulting in substantial wasted computation. To address this, we introduce a three-stage *filtering gates* (Fig. 3) that progressively screen large-scale initial candidate face pairs, thereby avoiding redundant computations associated with global pairwise enumeration and significantly reducing ineffective computation.

4.1. Normal Gate

The first filtering gate is designed to reduce the immense computational cost associated with processing all candidate face pairs in the model. The face normal, being the most computationally inexpensive among the three criteria, is used as the initial filtering condition without introducing significant overhead. This gate primarily requires the normal of a candidate face pair to be approximately antiparallel. In our implementation, a GPU thread block is assigned to process each face, where threads within the block cooperate to process all triangles of a specific mesh, achieving mesh-level parallel computation. After computing the normals for all faces, pairs of faces are combined as candidate face pairs, and those failing to meet the *normal criterion* (Eq. 1) are filtered out. This gate rapidly eliminates the least likely candidates. Thereby, it significantly enhances overall computational efficiency and supplies a more promising set of candidates for subsequent filtering gates.

4.2. Overlap Gate

The *overlap criterion* (Eq. 2) constitutes the most stringent and computationally complex stage within the three-stage filtering pipeline. This gate involves two primary computational tasks: the construction of oriented bounding boxes (OBBs) [7] and the calculation of the overlap ratios based on a *overlap criterion*. First, a GPU thread block is assigned to each face to compute its OBB. Each OBB is constructed only once per face and can be reused across candidate face pairs. Subsequently, a GPU thread

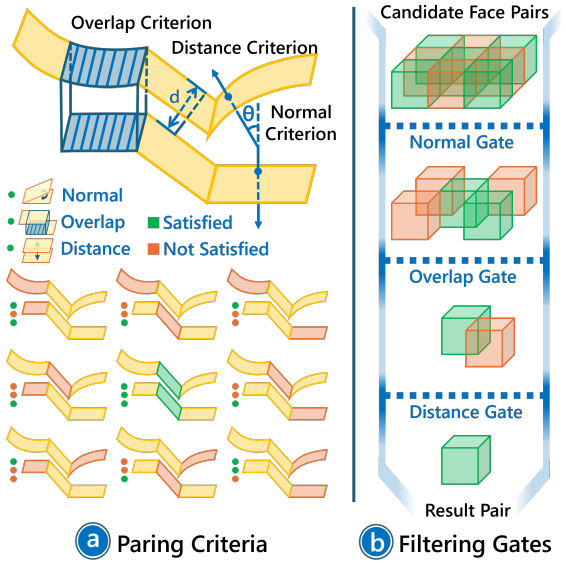

Figure 3. A simple example demonstrating the *filtering gates* mechanism through cyclic pairing of three faces. a) Status of different face pair combinations passing the criteria. b) The filtering process through the gate mechanism.

block is assigned to each pair to calculate the overlap ratio. For a given candidate face pair of *FaceA* and *FaceB* , *FaceA* is processed by projecting the centroids of all its triangle meshes along the face normal direction obtained by the first normal gate. The hit rate R_a is then determined by counting the proportion of these projections that intersect the principal direction plane of OBB of *FaceB*. The same operation is performed on *FaceB* to obtain its corresponding hit rate R_b. The values R_a and R_b collectively serve as the basis for determining whether the candidate pairs pass this filtering gate.

4.3. Distance Gate

This final gate is designed to eliminate candidate face pairs whose distance exceeds the model's wall thickness. To reduce computational cost, this gate employs a conservative distance approximation in place of exact calculation.

We assign a GPU thread block to each candidate pair. A LBVH is constructed using spatial ordering via Morton codes [12] and is maintained in shared memory to minimize data transfer overhead. For a candidate pair, Face A and Face B, the following operations are performed in parallel: calculate the minimum distance from each mesh centroid in Face A to Face B, recording the maximum value D_a among these minimum distances; perform the same operation on Face B to obtain D_b. The value $(D_a + D_b)/2$ is used as the conservative distance estimate between the two faces. After acquiring the criteria for different combinations, the combinations satisfying both criteria $Overlap \geq 50\%$ and $|180° - \angle(Normal)| \leq 30°$ are filtered (such combinations exhibit higher probabilities of forming a face pair in thin-walled regions). The distance of these filtered combinations is then partitioned into intervals to construct a frequency distribution histogram. Finally, the maximum distance within the peak frequency interval is defined as the t_{wall} of the model. Candidate pairs are then filtered against this t_{wall}, outputting the final *1-1* face pair.

4.4. Parallel Bipartite Graph Optimization from *1-1* FP to *n-n* FGP

After acquiring all *1-1* face pairs from the hierarchical filtering gates, we construct FGPs through GPU-accelerated graph processing. Each face is represented as a graph node, forming an undirected graph structure through edges connecting matched face pairs, which requires maximum connected component extraction and bipartite graph partitioning. Each face is initialized as its own parent node in a union-find data structure. Connected components are merged over multiple parallel processing iterations using atomic operations. For each maximal connected component, perform graph coloring: starting from an arbitrary vertex, if a node is colored red, its adjacent nodes are colored blue, and vice versa. The component is processed using breadth-first traversal with color propagation. Upon completion of the traversal, red nodes constitute the left face group and blue nodes form the right face group. This process eliminates redundant face pair couplings, providing concise FGPs for subsequent mid-point generation.

5. GPU-based Mid-point Generation

5.1. Initial Mid-Point Generation

To efficiently generate initial mid-points on GPU, we employ a parallel distance dilation algorithm (Fig. 4-a). Through hierarchical GPU kernel design with improved Double Queues based Distance Dilation (DQDD) algorithm [22], this method achieves efficient propagation of distance information from boundary sets to interior regions. Unlike traditional CPU serial implementations, our parallel approach fully exploits GPU's memory hierarchy and massive parallelism capabilities.

Our GPU implementation comprises four main phases:

(1) OBB-based Adaptive Grid Initialization

We compute the Oriented Bounding Box (OBB) [7] instead of an Axis-Aligned Bounding Box (AABB) because OBB provides tighter bounds for skewed FGPs, reducing invalid dilation blocks and boundary set counts. Since we employ OBB-based filtering during face pairing, we use the GPU-based Eberly interpolation method [4] to efficiently merge OBBs (Fig. 4-a1), avoiding recalculation by manipulating previously computed OBBs. Each FGP is assigned to a separate GPU thread for parallel creation of OBBs. After creating the OBB, we perform grid subdivision. The block size is set to $1/10$ of the FGP's wall thickness (from the dis-

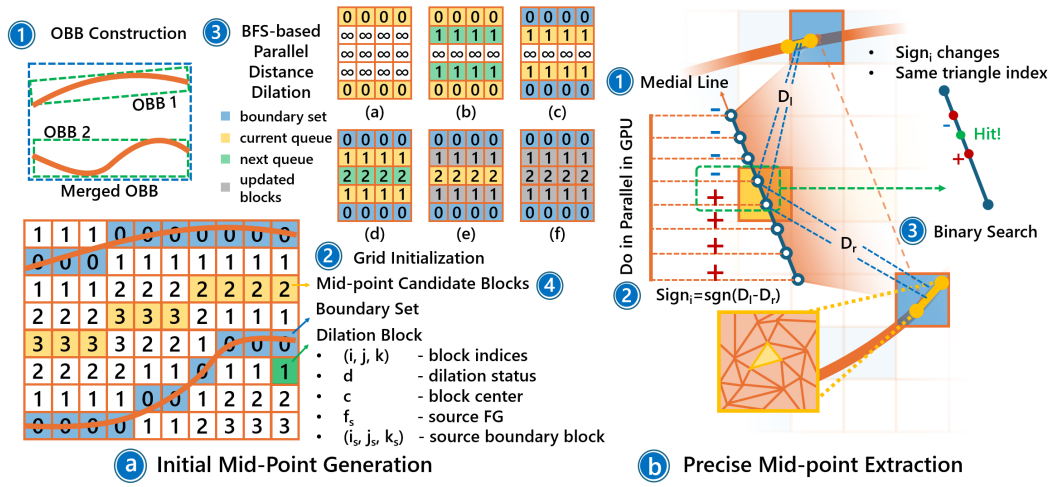

Figure 4. The two-step mid-point generation method. a) Initial mid-point generation via distance dilation using the DQDD algorithm, with color-coded queues showing the propagation process and identification of candidate blocks. b) Precise mid-point extraction through medial line construction, sign change detection, and binary search localization.

tance gate in face pairing), ensuring sufficient resolution for thin-wall features. The grid count in each dimension is:

$$n_i = \max\left(\left\lceil \frac{L_i}{h} \right\rceil + 1, n_{min}\right), \quad i \in \{x, y, z\} \tag{6}$$

where L_i is the OBB extent along the i-th axis, and n_{min} is the minimum grid count (default 10). The "+1" ensures complete boundary inclusion.

(2) Parallel Boundary Set Setup

Boundary Set initialization serves as the starting point for distance dilation (Fig. 4-a2). We employ a parallel sampling strategy to mark boundary blocks: 1) Sample surface meshes with adaptive density; 2) Transform samples to local coordinates in parallel; 3) Compute grid indices (i, j, k); and 4) Atomically set grid states and source information. Atomic operations via `atomicCAS` ensure correctness when multiple threads access the same blocks. Only when the state is -1 (uninitialized) is it set to 0 (boundary), avoiding race conditions.

(3) BFS-based Parallel Distance Dilation

We employ the DQDD algorithm [22] for parallelized breadth-first search (BFS). The algorithm propagates from the boundary set using a double-queue scheme across current and next queues. Fig. 4-a3 illustrates this: yellow blocks denote the current queue, green blocks the next queue, and gray blocks are being updated, respectively. Interior blocks are initialized with state -1 (unknown). The initial current queue is the boundary set, while the next queue is null.

1. For each block A in the current queue, we propagate to its 6 neighbors. If neighbor B is unknown or has a smaller updated distance, we update its state and enqueue it:

$$d(B) \leftarrow \min\left(d(B),\, d(A) + \|A - B\|_2\right), \tag{7}$$

where $d(\cdot)$ is the dilation state of a block and $\|A-B\|_2$ the distance between blocks A and B. Specifically, as the block size is uniform, we have set it to 1 by default.

2. When the current queue is exhausted, it swaps with the next queue, which becomes the new current queue. This repeats until no blocks remain, yielding complete dilation states. With the DQDD strategy, the boundary set expands into the interior, ensuring consistent and complete dilation.

(4) Mid-point Candidate Identification

After the complete dilation, mid-point candidate blocks are identified based on the following condition, as illustrated in Fig. 4-a4:

- **Different Source Faces**: There exists at least one neighbor in the 6-neighborhood whose source face is different from the source face of the current block.
- **No Existing Mid-point Candidates**: There should be no existing mid-point candidates in the 6-neighborhood of the current block.
- **Dilation State Bounds**: The dilation state must lie within half the max/min wall thickness range (from the distance gate), defining the likely range for mid-point candidates.

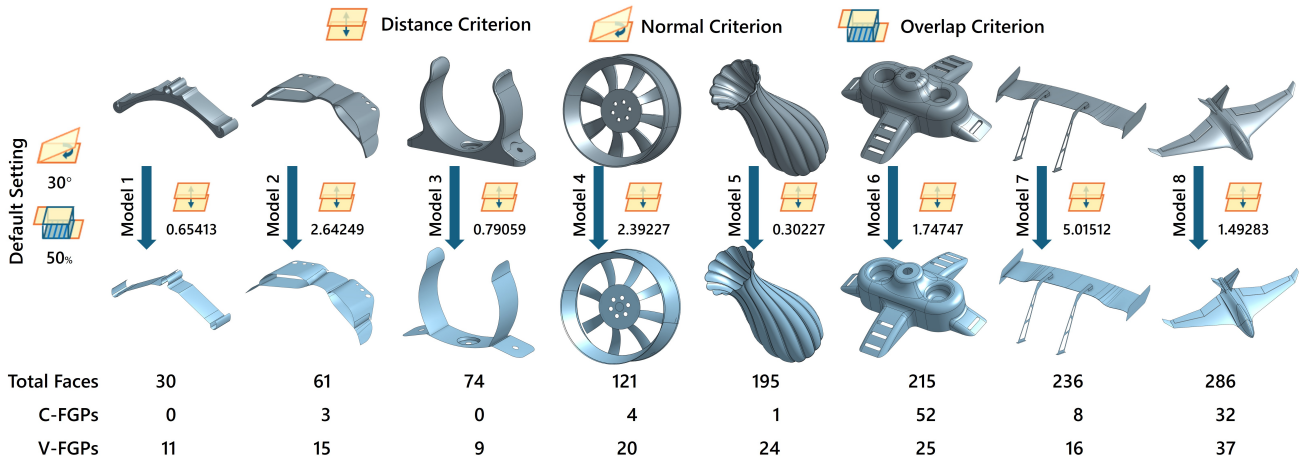

Figure 5. Mid-surface results for each benchmark. Top: original models. Bottom: extracted mid-surfaces. Each model includes the total face count, the number of constant/variable-thickness FGPs (C/V-FGPs), default pairing criteria settings, and the wall thickness selected by the distance gate.

5.2. Precise Mid-point Extraction

After obtaining candidate blocks from initial dilation, we design a GPU-based parallel algorithm to locate the final mid-point within each block (Fig. 4-b). Notably, the detailed mid-point extraction method has been published elsewhere and is out of the scope of this paper [21]. However, we implemented it in parallel, with the algorithm fully leveraging the GPU capabilities while maintaining mathematical rigor.

GPU Thread Allocation Strategy. We assign one GPU thread block per candidate block, each containing 1024 threads. Considering the need to simultaneously perform distance queries on both FG_{left} and FG_{right}, we organize the threads into 32 warps, with each warp serving as the basic computational unit for LBVH nearest distance queries. This allocation strategy enables each block to process distance calculations for 16 sampling points in parallel for both sides.

(1) Medial Line Construction. For each candidate block, we construct the medial line through its geometric center, directed from the nearest boundary block center of FG_{left} to that of FG_{right} (Fig. 4-b1). This guarantees intersection with both FGs, ensuring at least one equidistant mid-point exists (Definition 8 in basic concepts).

(2) Parallel Distance Queries.

The parallel operations within each GPU thread block are divided into two phases: First, we uniformly sample 16 points $\{P_1, P_2, \ldots, P_{16}\}$ along the medial line. Then, utilizing the LBVH query algorithm previously implemented in the distance gate, each warp computes the shortest distances d^i_{left} or d^i_{right} from sampling points to its FG. The sign of the distance difference is determined by:

$$\text{Sign}_i = \text{sgn}(d^i_{\text{left}} - d^i_{\text{right}}). \tag{8}$$

(3) Precise Mid-point Localization.

We employ the sign change detection strategy (Fig. 4-b2) consistent with MidSurfer [21]: we check in parallel for sign differences between adjacent sampling point pairs (P_i, P_{i+1}) while verifying their nearest triangle indices to both FGs are consistent. When adjacent points have the same nearest triangle indices, all points on the segment have the same nearest triangles, ensuring numerical stability for binary search (Fig. 4-b3).

6. Implementation and Results

The proposed algorithm was implemented as a prototype system in C/C++, utilizing Parasolid V35 [19] and CUDA toolkit version 12.6 on a Windows 11 (64-bit) platform. We employed an NVIDIA GeForce RTX 5090D as our primary test GPU (with 21760 CUDA cores at 2.41GHz and 32GB memory). All computations on the GPU are conducted using single-precision floating-point arithmetic. The testing environment involves a standard PC running Windows 11 Pro 64-bit, equipped with an Intel i7-14700K CPU operating at 3.4GHz (28 cores) and 32GB of RAM.

Extensive testing used diverse solid models from the GrabCAD online library[1], with 8 iconic models chosen as benchmark models (M1-M8) to demonstrate specific algorithmic capabilities. Our benchmark selection follows similar criteria to MidSurfer [21]. The selected models exhibit progressively increasing face counts (30–286), growing model complexity, and varying numbers of constant/variable-thickness FGPs, ensuring comprehensive coverage of typical industrial scenarios and enabling direct comparison with prior methods. Fig. 5 shows the selected models and their mid-surfaces generated by *gMidSurf*.

- **Model 1**: An instrument microphone clip, consisting

[1] https://grabcad.com/library

Table 1. Comparative study: Performance comparison of different methods across different stages (Time in seconds). Shows speedup ratios for face pairing, mid-point generation, and total end-to-end performance. All CPU-based methods are implemented without multi-threading acceleration.

			Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8
Face Pairing	Woo et al. [20]	Time	2.991	30.440	46.130	76.410	153.571	210.831	296.814	397.251
		Speedups	28.22	> 50.0	> 50.0	> 50.0	> 50.0	> 50.0	> 50.0	> 50.0
	MidSurfer [21]	Time	0.650	1.984	3.731	8.871	16.617	29.144	38.511	49.361
		Speedups	6.13	13.14	27.03	49.84	> 50.0	> 50.0	> 50.0	> 50.0
	gMidSurf (*cpu*)	Time	0.565	1.214	0.586	1.822	2.487	2.283	3.028	5.874
		Speedups	5.33	9.79	4.24	10.23	11.43	12.41	17.11	18.47
	gMidSurf	Time	**0.106**	**0.124**	**0.138**	**0.178**	**0.218**	**0.184**	**0.177**	**0.318**
Mid-Point Generation	Zhu et al. [25]	Time	4.553	6.718	10.468	12.507	13.397	7.319	5.289	16.161
		Speedups	26.01	18.01	23.42	16.63	15.48	8.94	9.85	17.38
	Parasolid [19]	Time	0.952	1.155	2.433	2.613	2.132	2.781	1.940	4.399
		Speedups	5.44	3.10	5.44	3.47	2.46	3.40	3.61	4.73
	MidSurfer [21]	Time	**0.133**	0.407	**0.236**	0.839	1.301	**0.787**	0.652	1.422
		Speedups	0.76	1.09	0.53	1.12	1.50	0.96	1.21	1.53
	gMidSurf (*cpu*)	Time	1.436	1.813	3.755	6.887	8.512	7.797	2.602	8.342
		Speedups	8.21	4.86	8.40	9.16	9.84	9.52	4.85	8.96
	gMidSurf	Time	0.175	**0.373**	0.447	**0.752**	**0.865**	0.819	**0.537**	**0.930**
Total	MidSurfer [21]	Time	1.233	3.001	4.217	10.790	19.138	31.577	40.123	52.673
		Speedups	1.48	2.29	4.07	5.37	8.38	11.90	23.97	15.87
	gMidSurf (*cpu*)	Time	2.751	3.885	6.191	9.789	10.919	11.730	7.090	15.106
		Speedups	3.31	2.97	5.98	4.87	4.78	4.42	4.24	4.55
	gMidSurf	Time	**0.831**	**1.307**	**1.035**	**2.010**	**2.283**	**2.653**	**1.674**	**3.318**

of 30 faces and no constant-thickness FGP (C-FGPs), used to validate the algorithm's basic capabilities.

- **Model 2**: A motorcycle winglet with 61 faces, including 15 variable-thickness FGPs (V-FGPs) and 3 C-FGPs. This model presents a scenario where both C-FGPs and V-FGPs exist.
- **Model 3**: A pipe bracket with 74 faces, where most V-FGPs exhibit significant curvature variations, is used to evaluate the algorithm's performance in geometric correctness.
- **Model 4**: A car wheel rim from the automotive industry, featuring 121 faces, including 20 V-FGPs and 19 *n-n* face pairs. Its complex topology and diverse variable-thickness scenarios pose significant challenges.
- **Model 5**: A table flower pot, containing most V-FGPs with 195 faces. It demonstrates the algorithm's capability to handle models with great curvature variations.
- **Model 6**: A multi-port distributor from the mechanical design domain, with 215 faces and 52 C-FGPs, demonstrating scenarios where constant-thickness structures predominate.
- **Model 7**: A rear spoiler contains 236 faces, with all FGPs exhibiting highly similar geometric primitives and gentle curvature variations.
- **Model 8**: An aircraft model from a real-world aerospace scenario with 286 faces and highly complex topological and geometric structures, used to verify algorithm robustness under the most challenging scenes.

6.1. Performance

In terms of efficiency, we conducted multi-stage comparative studies evaluating our method against existing mid-surface abstraction algorithms, including Zhu et al. [25], Woo et al. [20], MidSurfer [21] and the commercial solution (Parasolid [19]), with the main results presented in Table 1,

Face Pairing. Face pairing, as the most time-consuming stage in mid-surface abstraction, exhibits quadratic computational complexity with face count. Traditional approaches require pairwise combinations of all faces, sequentially computing three criteria to evaluate face pairs, as seen in MidSurfer. Methods like Woo et al. exhibit exponential

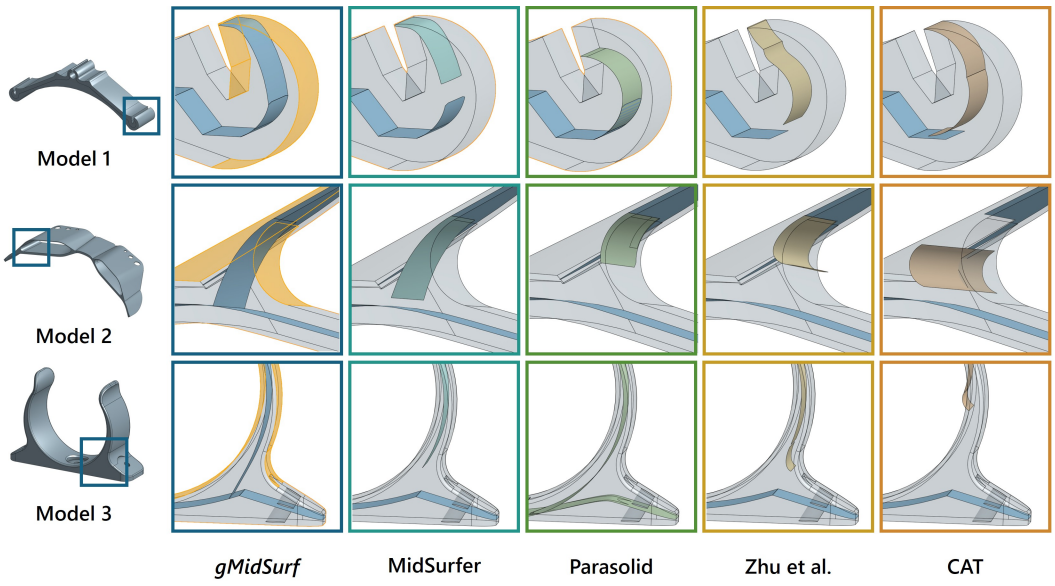

Figure 6. Comparison of geometric correctness across different methods on representative models. Highlighted regions indicate the FGP involved in mid-surface abstraction, with the color-coded results indicating the different methods.

complexity due to virtual decomposition based on edge convexity/concavity, resulting in numerous faces during pairing. In contrast, *gMidSurf* demonstrates near-linear time growth in both CPU and GPU implementations through hierarchical filtering gates: while combinations requiring the simple normal gate grow quadratically, the computationally intensive overlap and distance gates are applied only to linear-sized candidate sets.

For small-scale models, such as Models 2 and 3, GPU kernel launch overhead becomes non-negligible, resulting in similar GPU execution times. Model 5 exhibits peak times on both the CPU and GPU due to the highly varying face curvature. Adaptive discretization increases the sampling density in high-curvature regions, resulting in Model 5 having the highest triangular face count and significantly increasing the load in overlap and distance gates.

Overall, *gMidSurf* achieves over $50\times$ speedup versus MidSurfer when faces exceed 100. For CPU-GPU comparison, kernel launch overhead becomes negligible beyond 100 faces, with speedup ratios reaching $10.23\times$–$18.47\times$ and growing approximately linearly with face count.

Mid-point Generation. The results reveal that *gMidSurf* $_{(cpu)}$ is significantly less efficient than MidSurfer. The reason is *gMidSurf* $_{(cpu)}$ employs an algorithm with higher time complexity to extract more complete mid-points within an FGP. Notably, *gMidSurf* outperforms MidSurfer on most models. However, due to GPU-CPU architectural differences, for models with small face counts, *gMidSurf* shows comparable or slightly inferior performance to MidSurfer (e.g., Models 1 and 3).

From the computational trend of *gMidSurf*, we can observe that the processing time generally exhibits a positive correlation with FGP count. However, performance declines when handling models with great curvature transitions. For example, although Model 5 has fewer input FGPs, its execution time on the GPU is higher than Model 6.

While *gMidSurf* shows performance variability on small-scale inputs or complex geometric features, it demonstrates decisive advantages over traditional methods (Zhu et al.) and commercial solutions (Parasolid). Due to complex geometric computations and operations, *gMidSurf* achieves a speedup of $8\times$–$26\times$ compared to Zhu et al.'s method. Through industrial engineering implementation, *gMidSurf* also gains $3\times$–$6\times$ efficiency improvement over Parasolid.

Total End-to-End Performance. Overall time costs reveal that the face pairing stage is the performance bottleneck in MidSurfer. In *gMidSurf*, however, thanks to effective optimization through the filtering gates, the bottleneck has shifted to subsequent intrinsic overhead. As the task scale increases, *gMidSurf*'s speedups compared to MidSurfer show an upward trend, while maintaining a relatively stable acceleration range compared to *gMidSurf* $_{(cpu)}$. When the number of model faces exceeds 100, *gMidSurf* achieves a speedup ratio of $5\times$–$15\times$ over MidSurfer, while maintaining a stable $3\times$–$6\times$ speedup ratio compared to *gMidSurf* $_{(cpu)}$.

6.2. Correctness

Fig. 6 reports the correctness of different methods. Benefiting from *complete dilation*, *gMidSurf* achieves satisfactory results on all models. We specifically select representative models and compare our outcomes with MidSurfer [21], Parasolid [19], Zhu et al. [25], and CAT [15]

methods. The highlighted portions in the figure show the FGPs involved in generation, with mid-surface results from different methods marked by different colors.

It can be observed that both the CAT and Zhu et al. methods fail to generate satisfactory results, their mid-surfaces are either non-smooth or extend beyond model boundaries. Further examining the connectivity of the generated mid-surfaces, CAT performs the worst. This primarily stems from CAT's reliance on mesh-based results, its generated mid-points are often inaccurate with significant positional oscillations. Zhu et al. perform better than CAT, as its mid-points obtained through projection and intersection approximate the exact solution to some extent. However, this method creates large gaps because it uses the normal direction as the projection direction during sampling. When the angle between two faces exceeds 15, it fails to find opposing projection points, introducing substantial mid-point errors. Since most variable-thickness models contain FGPs at varying angles, this leads to significant cumulative errors.

MidSurfer, as the most recent mid-surface abstraction work, demonstrates strong support for variable-thickness models. It maintains good geometric accuracy while preserving correct topological structure. However, it occasionally suffers from gap issues due to its directional sampling strategy. Specifically, MidSurfer constructs sampling points on the designated "left" face group (FG_{left}) and computes closest points on the "right" face group (FG_{right}). When FG_{left} has smaller surface area or excessive curvature compared to FG_{right}, the sampling points may only cover a local region, causing all projections to cluster in a limited area of FG_{right}. This results in mid-surface truncation at boundaries. While this problem can be resolved by manually swapping the left and right face groups, such intervention is unacceptable for fully automated algorithms. In contrast, *gMidSurf*'s dilation-based approach propagates distance information symmetrically from both face groups simultaneously, eliminating any dependency on face group ordering and ensuring complete mid-surface coverage regardless of geometric configurations.

We also compare *gMidSurf* with the commercial solution (Parasolid). Admittedly, Parasolid's approach maintains excellent topological correctness and connectivity. Nevertheless, its generated mid-surfaces are often obtained directly through *offset* operations, which significantly suppress the original feature. Additionally, Parasolid's robustness is suboptimal. For instance, with the same face pair, swapping the left and right faces may cause extraction failures (generating error codes). *gMidSurf* addresses these issues through our dilation-based mid-point extraction algorithm. Through *complete dilation*, we comprehensively compute the local space occupied by each FGP, ensuring completeness of initial mid-points results and thus eliminating gap formation. Consequently, *gMidSurf*'s generated mid-surfaces exhibit superior correctness compared to previous methods and commercial software.

7. Conclusion

We present *gMidSurf*, an efficient mid-surface abstraction method for thin-walled CAD models. Through hierarchical face pairing and two-step mid-point refinement, we achieve full GPU-based acceleration of the two core stages in mid-surface abstraction. Our method employs filtering gates and a simplified overlap criterion to rapidly prune candidate face pairs, and extracts mid-points efficiently via complete dilation and parallel precise mid-point refinement. Extensive testing on 8 variable-thickness benchmarks demonstrates that *gMidSurf* achieves $3\times$–$15\times$ end-to-end acceleration on a commodity GPU (RTX 5090D) without compromising geometric accuracy. To our knowledge, no publicly available GPU algorithm has previously provided satisfactory performance for this specific task. Our method makes a valuable contribution by addressing this challenge.

References

[1] C. G. Armstrong. Modelling requirements for finite-element analysis. *Computer-aided design*, 26(7):573–578, 1994. 1

[2] H. Blum. A transformation for extracting new descriptors of shape. *Models for the perception of speech and visual form*, pages 362–380, 1967. 1, 2

[3] P. Dabke, V. Prabhakar, and S. Sheppard. Using features to support finite element idealizations. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 13805, pages 183–193. American Society of Mechanical Engineers, 1994. 1

[4] D. Eberly. *3D game engine design: a practical approach to real-time computer graphics*. CRC Press, 2006. 5

[5] P. Fan, W. Wang, R. Tong, H. Li, and M. Tang. gDist: Efficient distance computation between 3d meshes on GPU. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11. ACM, 2024. 3

[6] W. Gao, S. Gao, Y. Liu, J. Bai, and B. Hu. Multiresolutional similarity assessment and retrieval of solid models based on DBMS. *Computer-Aided Design*, 38(9):985–1001, 2006. 4

[7] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, 1996. 4, 5

[8] A. C. Jalba, J. Kustra, and A. C. Telea. Surface and curve skeletonization of large 3D models on the GPU. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1495–1508, 2013. 3

[9] K.-Y. Kwon, B.-C. Lee, and S.-W. Chae. Medial surface generation using chordal axis transformation in shell structures. *Computers & Structures*, 84(26-27):1673–1683, 2006. 2

[10] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH construction on GPUs. In *Computer Graphics Forum*, volume 28, pages 375–384. Wiley Online Library, 2009. 2

[11] H. Lee, Y.-Y. Nam, and S.-W. Park. Graph-based midsurface extraction for finite element analysis. In *11th International Conference on Computer Supported Cooperative Work in Design*, pages 1055–1058. IEEE, 2007. 2

[12] G. M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company, 1966. 5

[13] D. C. Nolan, C. M. Tierney, C. G. Armstrong, T. T. Robinson, and J. E. Makem. Automatic dimensional reduction and meshing of stiffened thin-wall structures. *Engineering with Computers*, 30:689–701, 2014. 1, 2

[14] L. Prasad. Morphological analysis of shapes. *CNLS newsletter*, 139(1):1997–07, 1997. 1

[15] W. R. Quadros and K. Shimada. Hex-layer: Layered all-hex mesh generation on thin section solids via chordal surface transformation. In *Proceedings of 11th International Meshing Roundtable*, pages 169–180, 2002. 1, 2, 9

[16] M. Rezayat. Midsurface abstraction from 3d solid models: general theory and applications. *Computer-Aided Design*, 28(11):905–915, 1996. 1, 2, 4

[17] D. J. Sheehy, C. G. Armstrong, and D. J. Robinson. Computing the medial surface of a solid from a domain Delaunay triangulation. In *Proceedings of the third ACM symposium on Solid modeling and applications*, pages 201–212, 1995. 1, 2

[18] D.-P. Sheen, T.-g. Son, D.-K. Myung, C. Ryu, S. H. Lee, K. Lee, and T. Yeo. Transformation of a thin-walled solid model into a surface model via solid deflation. *Computer-Aided Design*, 42(8):720–730, 2010. 2

[19] Siemens Digital Industries Software. Parasolid: Geometric modeling kernel. https://www.plm.automation.siemens.com/global/en/products/plm-components/parasolid.html, 2022. 2, 7, 8, 9

[20] Y. Woo. Abstraction of mid-surfaces from solid models of thin-walled parts: A divide-and-conquer approach. *Computer-Aided Design*, 47:1–11, 2014. 1, 4, 8

[21] L. Ye, X. Zhou, P. Fan, R. Tong, H. Li, P. Du, and M. Tang. MidSurfer: Efficient mid-surface abstraction from variable thin-walled models. *Computer-Aided Design*, 190:103965, 2026. 1, 3, 4, 7, 8, 9

[22] H. Zhu, Y. Liu, J. Bai, and X. Ye. Constructive generation of the medial axis for solid models. *Computer-Aided Design*, 62:98–111, 2015. 4, 5, 6

[23] H. Zhu, Y. Liu, H. Wang, and J. Zhao. Efficient construction of the medial axis for a CAD model using parallel computing. *Engineering with Computers*, 34(2):413–429, 2018. 3, 4

[24] H. Zhu, Y. Liu, J. Zhao, and H. Wang. Calculating the medial axis of a CAD model by multi-CPU based parallel computation. *Advances in Engineering Software*, 85:96–107, 2015. 3, 4

[25] H. Zhu, Y. Shao, Y. Liu, and C. Li. Mid-surface abstraction for complex thin-wall models based on virtual decomposition. *International Journal of Computer Integrated Manufacturing*, 29(8):821–838, 2016. 2, 8, 9