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

Li Ye[a], Xinhang Zhou[a], Xingyu Yang[a], Peng Fan[a], Ruofeng Tong[a], Hailong Li[b], Peng Du[a], Min Tang[a,*]

[a]*Zhejiang University, Hangzhou, 310007, China*
[b]*Shenzhen Poisson Software Co., Ltd., Shenzhen, 518129, China*

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× 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×–18.5× speedups in face pairing and 4.8×–9.8× in mid-point generation compared to CPU implementations, yielding 5×–15× acceleration on a commodity GPU (NVIDIA RTX 5090D) compared to state-of-the-art methods while maintaining geometric accuracy.

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

1. Introduction

Mid-surface abstraction is an essential 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 [1, 2]. As a geometric preprocessing step in the CAD-to-CAE pipeline, mid-surface abstraction provides idealized surface representations along with local thickness distributions. Subsequently, downstream analysts evaluate these representations for shell/plate element suitability based on structural criteria such as aspect ratio, thickness uniformity, and local feature complexity [3].

Existing mid-surface abstraction methods fall into three primary categories: Medial Axis Transform (MAT) [4], Chordal Axis Transform (CAT) [5], 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 [6]. CAT converts thin-walled solids into single-layer tetrahedral meshes, but struggles to maintain geometric quality in complex junction regions [7, 8]. 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 systems [9].

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 [10, 11]. Second, existing methods lack robustness for variable-thickness geometry, where normal-projection heuristics frequently fail when the face angles vary significantly [12].

The massive parallelism of modern GPUs presents opportunities to address these bottlenecks. GPUs have demonstrated remarkable advantages in spatial search and distance computation, including fast BVH/LBVH construction and traversal [13], and order-of-magnitude acceleration in distance calculations [14]. However, naive GPU ports underperform due to quadratic candidate explosion, branch divergence, and cache coherence issues.

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).

*Corresponding author. This work was funded in part by "Pioneer" and "Leading Goose" R&D Program of Zhejiang Province (No. 2025C01086).

Email addresses: li-ye@zju.edu.cn (Li Ye), 2xh@zju.edu.cn (Xinhang Zhou), xy_yang@zju.edu.cn (Xingyu Yang), fanpeng0103@zju.edu.cn (Peng Fan), trf@zju.edu.cn (Ruofeng Tong), lihailong@poissonsoft.com (Hailong Li), dp@zju.edu.cn (Peng Du), tang_m@zju.edu.cn (Min Tang)

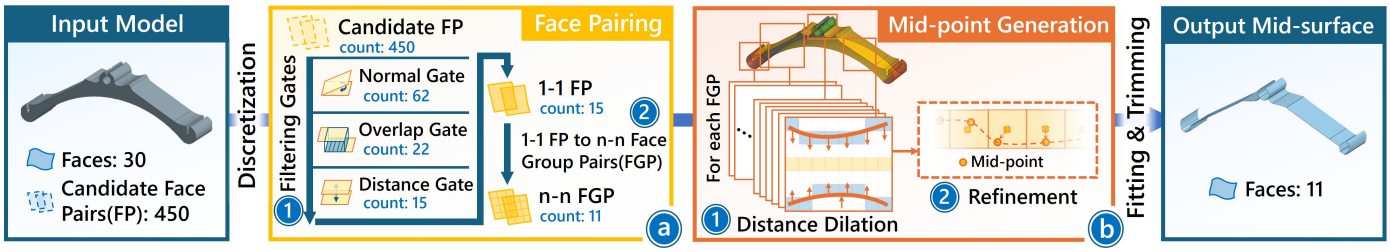

Figure 1: The algorithm overview of *gMidSurf* through a simple example. The input thin-walled model is first discretized, followed by (a) hierarchical GPU-based face pairing with three filtering gates (normal, overlap, and distance) to identify face group pairs, and (b) GPU-based mid-point generation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

On industrial benchmarks with variable-thickness scenarios, *gMidSurf* achieves 4.2×–18.5× speedups for face pairing and 4.8×–9.8× for geometry generation on a single NVIDIA RTX 5090D GPU, yielding 5×–15× end-to-end acceleration while maintaining geometric accuracy compared to existing methods. 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 × –100× 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.
- **End-to-end GPU design that preserves B-Rep intent.** *gMidSurf* keeps geometry resident on device through face pairing and mid-point generation, supports *1-1* and *n-n* face pairs, and interoperates with trimming on the host to balance robustness and speed.
- **Extensive evaluation on industrial-style models.** We report stage-wise and end-to-end speedups over a tuned multi-core CPU baseline and previous methods, and demonstrate more accurate results on variable-thickness parts and complex free-form models.

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 [4, 6]. CAT converts thin-walled solids into single-layer tetrahedral meshes and classifies elements to recover a sheet [7, 8]; however, complex ribs and free-form junctions stress classification robustness, and the geometric quality of the recovered surface is not guaranteed [15].

FP has become the dominant paradigm because it preserves B-Rep intent by explicitly identifying opposing wall faces and constructing the intervening sheet. Rezayat [9] first introduced Face Adjacency Graphs (FAG) with distance/normal criteria to detect pairs; subsequent work reduced clutter and improved topological consistency via boundary extension [16] and feature suppression [17]. Commercial solutions (e.g., Parasolid [18]) expose FP utilities that combine user-specified and automatically detected pairs; however, complex free-form regions still require substantial manual curation.

To tame model complexity, decomposition strategies segment the solid before pairing. Chong et al. [19] use concave-edge loops to isolate simpler subregions; Woo [10] proposes a divide-and-conquer hierarchy (but depends on Boolean operations that can be brittle on free-form surfaces); and Kulkarni et al. [20] leverage virtual decomposition with dedicated nodes for mid-surface generation and interaction parsing, improving topological correctness on sheet-metal parts when rich feature data are available. These approaches shrink the search space but still offer limited support for variable-thickness free-form faces. Recent work by Ye et al. [11] (MidSurfer) advances robustness by replacing normal-projection with distance-driven mid-point localization and by handling transitions (fillets/chamfers) more reliably, yet its efficiency remains bounded by CPU-side processing on large assemblies.

In a parallel line of research, Boneš et al. [21] present a parameter-free method for extracting mid-surfaces from segmented volumetric data via ridge-field transformation and slice-wise polyline tracing. While both their method and ours operate on discretized representations internally, the two approaches target fundamentally different input domains: Boneš et al. process binary voxel segmentations from microscopy imaging, whereas *gMidSurf* operates on B-Rep CAD models.

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 [13]. On the proximity side, recent *mesh–mesh*

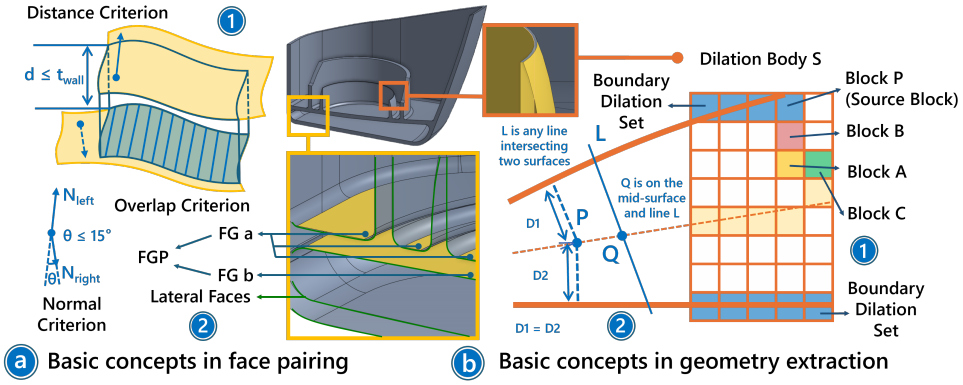

Figure 2: Basic concepts in face pairing and geometry extraction. a) Three pairing criteria and face group pair (FGP) definition. b) Dilation algorithm fundamentals and mid-surface definition.

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 [14]. 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 [22, 23]; Jalba et al. demonstrate GPU-based surface and curve skeletonization of large 3D polygonal meshes, achieving two orders of magnitude speed-up [24]. While these efforts validate GPU-based geometric abstraction, they target subroutines (skeletonization) or volumetric data 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 and Method Overview

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 [11] and form the efficiency bottlenecks we address. We first introduce basic concepts (adapting several from variable-thickness settings [22, 11, 10]), then outline our workflow.

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:
$$\left|180^\circ - \angle(\mathbf{N}_{left}, \mathbf{N}_{right})\right| \leq \theta, \quad (1)$$
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:
$$\text{Project}(f_{left}, f_{right}) \geq R \vee \text{Project}(f_{right}, f_{left}) \geq R, \quad (2)$$
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}:
$$\text{Dist}(f_{left}, f_{right}) \leq t_{wall}, \quad (3)$$
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.

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 [25, 22].

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 [26]. 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 [26]. 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 [11], 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

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

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:

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

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.

3.3. Method Overview

To explain the proposed GPU-parallel mid-surface generation method, the entire processing pipeline is briefly described as follows, as illustrated in Fig. 3:

(1) Discretization: The input CAD model is discretized into triangular meshes for GPU processing. Each face is adaptively sampled to ensure geometric fidelity [27]. This step is performed once and kept on GPU to avoid costly data transfers.

(2) Hierarchical GPU-based Face Pairing: This stage identifies all valid FGPs that constitute the mid-surface. It consists of two sub-steps:

- *Three-criteria-based Filtering Gates:* We employ a hierarchical culling strategy to efficiently prune invalid face pairs using three progressively stricter gates. Specifically:
  - The normal gate leverages GPU's vectorization capabilities to evaluate normal compatibility in parallel.
  - The overlap gate utilizes simplified projection algorithms and block-parallel strategies for efficient projection overlap computation.
  - The distance gate employs extended LBVH distance gates and hierarchical pruning strategies to substantially reduce candidate FGPs.

  Each criterion gate is carefully designed to fully exploit GPU's SIMD architecture, minimizing branch divergence through warp-level cooperative processing.

- *Parallel Bipartite Graph Optimization for* 1-1 *FP to* n-n *FGP:* Faces are abstracted as graph nodes, with matched face pairs connected. GPU-parallel connected component algorithms identify maximal subgraphs, each representing an FGP. Parallel graph algorithms eliminate redundant pairs, resulting in concise FGPs for mid-surface generation.

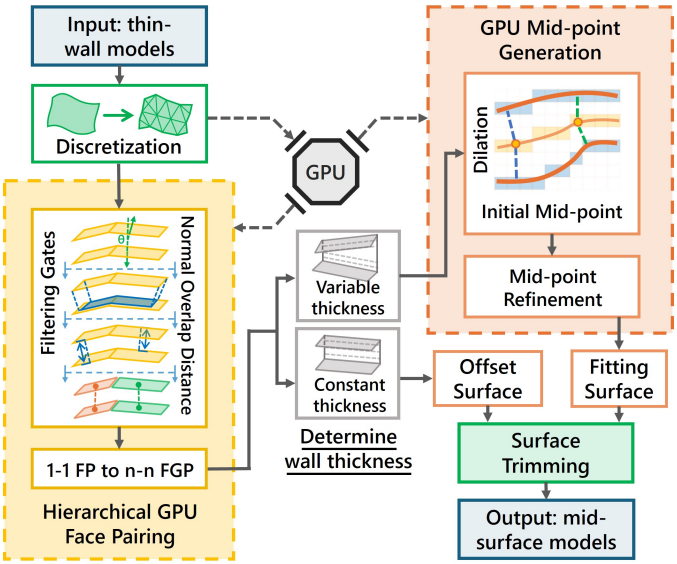

Figure 3: The algorithm pipeline of *gMidSurf* consists of two key stages, including hierarchical GPU-based face pairing and dilation-based precise mid-point generation to rapidly abstract the correct mid-surface.

(3) GPU-based Mid-point Generation: This stage utilize a bracket-and-bisect method to obtain accurate mid-points through two steps:

- *Initial Mid-point Generation - Parallel Distance Dilation:* Using an improved distance dilation algorithm, we implement parallel distance transformation on GPU. Through GPU's memory hierarchy and atomic operations, we compute Euclidean distances from dilation blocks to boundary blocks, generating initial mid-point candidates.

- *Precise Mid-point Extraction - Parallel Mid-point Refinement:* We locate equidistant mid-points through GPU-parallel LBVH search. The final mid-point is determined through medial line construction, sign change detection, and binary search localization.

(4) Surface Fitting and Trimming: Finally, all computed mid-points are fitted using NURBS surfaces, and necessary trimming operations are performed based on model topology to generate the final mid-surface. The trimming process ensures consistency between mid-surface boundaries and the original model [11].

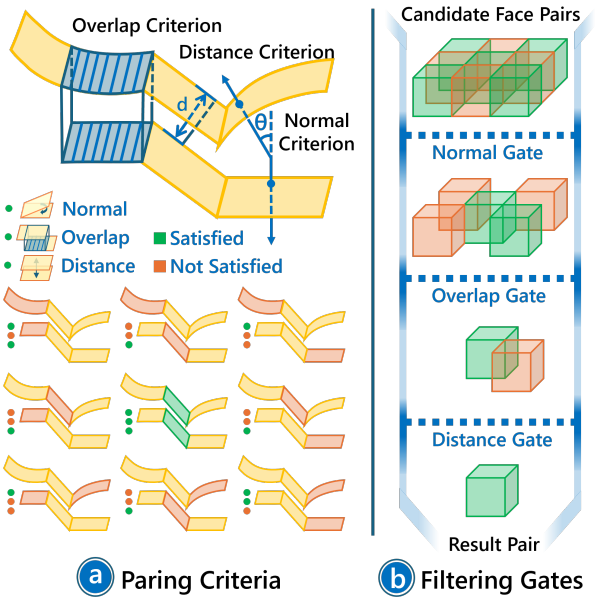

Figure 4: A simple example demonstrating the *filtering gates* mechanism through cyclic pairing of three faces. a) Status of different face pair combinations passing the criteria. b) The filtering process through the gate mechanism.

4. Hierarchical GPU-based Face Pairing

Conventional face pairing methods [10, 11, 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. 4) 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

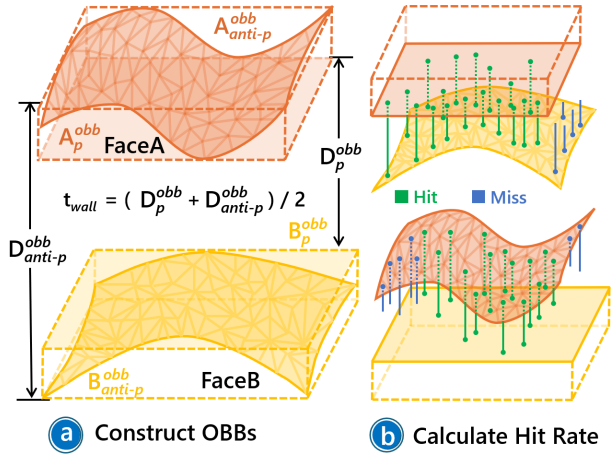

Figure 5: Illustration of the *simplified overlap criterion*. a) Construction of OBBs with distance measurements to principal planes for wall thickness estimation. b) Hit rate calculation through centroid projection onto opposing face OBB planes.

tasks: the construction of oriented bounding boxes (OBBs) [28] and the calculation of the overlap ratios based on a *simplified 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 block is assigned to each pair to calculate the overlap ratio. For a given candidate face pair of *FaceA* and *FaceB* illustrated by Fig. 5-b, *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. This *simplified overlap criterion* offers the following two key advantages:

Efficient overlap evaluation. Our *Simplified Overlap Criterion* reduces the point-to-mesh projection problem to a point-to-plane projection calculation. This converts costly ray-face intersections into GPU-friendly ray-plane intersections, significantly reducing computational complexity while preserving accuracy.

Efficient subsequent distance calculation. This design enables the simultaneous acquisition of the minimum distances between the two opposing boundary planes of the OBB, as shown in Fig. 5-a: D^{obb}_{p} to the principal direction plane and D^{obb}_{anti-p} to the anti-principal direction plane. When the condition $|D^{obb}_{p} - D^{obb}_{anti-p}| \leq tol$ is satisfied, where tol is a predefined tolerance. $(D^{obb}_{p} + D^{obb}_{anti-p})/2$ can be directly utilized as the estimated face distance. Simultaneously, D^{obb}_{anti-p} serves as an additional culling method for subsequent Linear bounding volume hierarchy (LBVH) [13] queries in distance calculation, providing pre-filtering optimization for the distance 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.

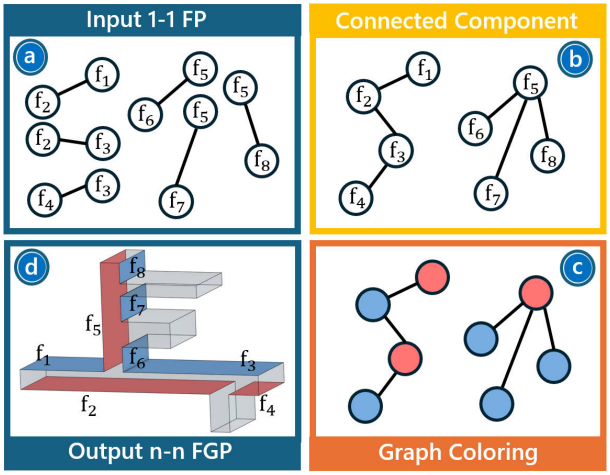

Figure 6: Process of the *1-1 FP to n-n FGP*. a) Input 1-1 FP. b) Construction of the maximum connected component. c) For each maximal connected component, perform graph coloring. d) Output n-n FGP.

We assign a GPU thread block to each candidate pair. A LBVH is constructed using spatial ordering via Morton codes [29] 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. Benefiting from the *Simplified Overlap Criterion*, we can perform additional pruning on the LBVH to accelerate distance queries, while also providing an alternative approach for determining the distance value. The final t_{wall} (Eq. 3) is determined by the following equation:

$$t_{wall} = \begin{cases} (D_a + D_b)/2, & |D_p^{obb} - D_{anti-p}^{obb}| \geq tol \\ (D_p^{obb} + D_{anti-p}^{obb})/2, & |D_p^{obb} - D_{anti-p}^{obb}| < tol \end{cases} \tag{6}$$

where *tol* is a predefined tolerance.

After acquiring the criteria for different combinations, the combinations satisfying both criteria $Overlap \geq 50\%$ and $|180^\circ - \angle(Normal)| \leq 30^\circ$ 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 (Fig. 6-a), 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 (Fig. 6-b) are merged over multiple parallel processing iterations using atomic operations. For each maximal connected component, perform graph coloring (Fig. 6-c): 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, as shown in Fig. 6-d. 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. 7-a). Through hierarchical GPU kernel design with improved Double Queues based Distance Dilation (DQDD) algorithm [26], 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) [28] 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 [30] to efficiently merge OBBs (Fig. 7-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 distance gate in face pairing), ensuring sufficient resolution for thin-wall features. The grid count in each dimension is:

$$n_i = \max\left(\left\lceil \frac{L_i}{h} \right\rceil + 1, n_{min}\right), \quad i \in \{x, y, z\} \tag{7}$$

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. The final block size is:

$$h_{actual} = \min\left\{\frac{L_x}{n_x}, \frac{L_y}{n_y}, \frac{L_z}{n_z}\right\}, \tag{8}$$

and grid counts are recalculated using Equation 7 to form the final dilation body S. This adaptive strategy ensures consistent precision and efficiency of the dilation algorithm across models of different scales.

Remark. A naïve implementation allocates and initializes a separate grid for each FGP, which incurs repeated GPU memory allocation and per-block state initialization. Profiling reveals that this independent allocation dominated over 90% of the mid-point generation time. To eliminate this bottleneck, we pre-allocate a *unified grid pool* sized to accommodate the largest FGP bounding box across all FGPs. When processing each FGP, the grid pool is reused by performing a lightweight state reset via `cudaMemset`, clearing only the dilation states and source information while retaining the allocated memory. This avoids redundant `cudaMalloc`/`cudaFree` cycles and

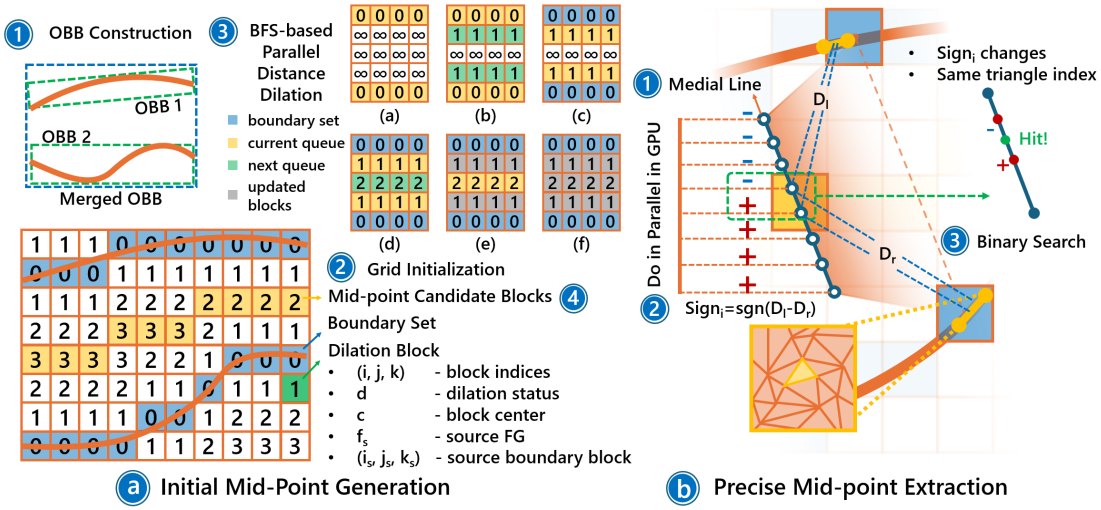

Figure 7: The two-step mid-point generation method. a) Initial mid-point generation via distance dilation using the DQDD algorithm, with color-coded queues showing the propagation process and identification of candidate blocks. b) Precise mid-point extraction through medial line construction, sign change detection, and binary search localization.

reduces initialization overhead to approximately 40% of the mid-point generation time.

(2) Parallel Boundary Set Setup

Boundary Set initialization serves as the starting point for distance dilation (Fig. 7-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.

Notably, to efficiently manage the dilation block on the GPU, we design a compact data structure. This structure contains all necessary information for the distance dilation:

$$\text{DilationBlock} = \{(i, j, k), d, \mathbf{c}, f_s, (i_s, j_s, k_s)\}, \tag{9}$$

where (i, j, k) denotes the dilation block indices, d is the dilation state, and $\mathbf{c}$ is the block center. The source face f_s identifies the boundary FG from which the dilation originates. The source indices (i_s, j_s, k_s) record the boundary block that provides the shortest distance for tracking mid-points.

(3) BFS-based Parallel Distance Dilation

We employ the DQDD algorithm [26] for parallelized breadth-first search (BFS). The algorithm propagates from the boundary set using a double-queue scheme across current and next queues. Fig. 7-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:

$$d(B) \leftarrow \min\left(d(B),\, d(A) + \|A - B\|_2\right), \tag{10}$$

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. 7-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.

These conditions ensure that only blocks located between distinct FG and meeting the dilation state criteria are identified as mid-point candidates, effectively filtering out potential candidate blocks.

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. 7-b). Notably, the detailed mid-point extraction method has been published elsewhere and is out of the scope of this paper [11]. 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

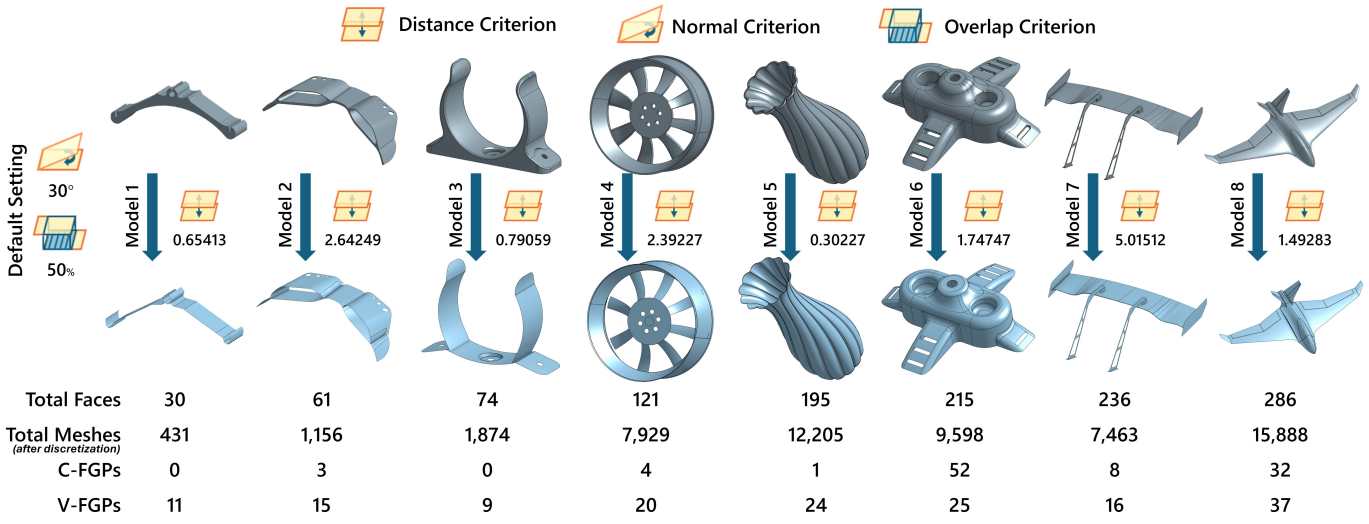

Figure 8: Mid-surface results for each benchmark. Top: original models. Bottom: extracted mid-surfaces. Each model includes the total face count, the total number of meshes after discretization, the number of constant- and variable-thickness FGPs (C/V-FGPs), the default pairing criteria settings, and the wall thickness selected by the distance gate.

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. 7-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:

$$\text{Sign}_i = \text{sgn}(d^i_{\text{left}} - d^i_{\text{right}}). \tag{11}$$

(3) Precise Mid-point Localization.

We employ the sign change detection strategy (Fig. 7-b2) consistent with MidSurfer [11]: 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. 7-b3).

6. Implementation and Results

The proposed algorithm was implemented as a prototype system in C/C++, utilizing Parasolid V35 [18] 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. Fig. 8 shows the selected models and their mid-surfaces generated by *gMidSurf*.

The benchmark models are selected to cover a diverse range of geometric and topological configurations relevant to mid-surface extraction, including variable and constant thickness, high curvature, n-n pairings, and complex free-form surfaces. These models are intended to systematically evaluate algorithmic capabilities—geometric correctness, GPU acceleration effectiveness, and scalability with model complexity—rather than to represent complete FEA-ready assemblies with specified loading conditions and boundary constraints. In industrial practice, the extracted mid-surfaces serve as geometric input to the analyst, who then determines the appropriate idealization strategy (pure shell, mixed-dimensional, or local solid refinement) based on structural engineering judgment and domain-specific requirements.

- **Model 1**: An instrument microphone clip, consisting 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 (>15° angular variation between paired faces), used to evaluate geometric correctness under challenging dilation conditions.
- **Model 4**: A car wheel rim from the automotive industry, featuring 121 faces, including 20 V-FGPs and 19 n-n face pairs. Its multi-ring topology, local blends, and

[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. [10]	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 [11]	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. [12]	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 [18]	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 [11]	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 [11]	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**

diverse variable-thickness scenarios pose significant challenges for face pairing.

- **Model 5**: A table flower pot with 195 faces, containing the most V-FGPs among all benchmarks. 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—the highest C-FGP count among all benchmarks—demonstrating scenarios where constant-thickness structures predominate.
- **Model 7**: A rear spoiler with 236 faces, with all FGPs exhibiting highly similar geometric primitives and gentle curvature variations, testing the filtering gates under low-discrimination conditions.
- **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 configurations.

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. [12], Woo et al. [10], MidSurfer [11] and the commercial solution (Parasolid [18]), 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 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 (also shown in Table 2).

Overall, *gMidSurf* achieves over 50× 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×–18.47× 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. No-

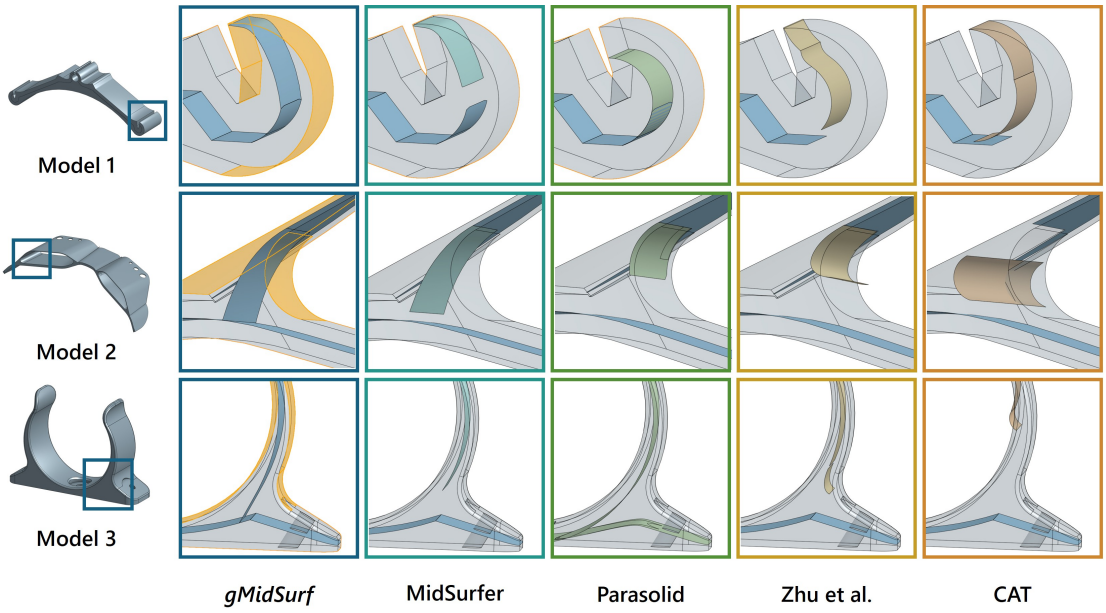

Figure 9: Comparison of geometric correctness across different methods on representative models. Highlighted regions indicate the FGP involved in mid-surface abstraction, with the color-coded results indicating the different methods.

tably, *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 (detailed in Section 6.4-*Time Ratios*).

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×–26× compared to Zhu et al.'s method. Through industrial engineering implementation, *gMidSurf* also gains 3×–6× 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 (detailed in Section 6.4-*Time Ratios*). 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×–15× over MidSurfer, while maintaining a stable 3×–6× speedup ratio compared to *gMidSurf* (*cpu*).

6.2. Correctness

Fig. 9 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 [11], Parasolid [18], Zhu et al. [12], and CAT [7] 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. This primarily occurs because its initial mid-points are obtained by computing the closest points from one side to another. When one side has excessive curvature, all sampled points on the other side find their closest points in a local region of the opposite side, causing the mid-surface to be truncated at boundaries. While this problem can be resolved by swapping the face groups, such manual intervention is unacceptable for automated algorithms.

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

Table 2: Ablation study: Impact of key features in the face pairing stage (Filtering Gates, Simplified Overlap Criterion (SOC)).

	gMidSurf	Filtering Gates		SOC	
	Time	Time	Speedups	Time	Speedups
M1	**0.106**	0.168	1.58	0.113	1.07
M2	**0.124**	0.353	2.85	0.149	1.20
M3	**0.138**	0.553	4.01	0.170	1.23
M4	**0.178**	0.950	5.34	0.217	1.22
M5	**0.218**	2.950	13.53	0.515	2.36
M6	**0.184**	2.617	14.22	0.519	2.82
M7	**0.177**	4.697	**26.52**	0.760	**4.29**
M8	**0.348**	6.738	19.36	1.368	3.93

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.

6.3. Ablation Studies

Table 2 presents an ablation study to highlight the contribution of each component in *Hierarchical GPU-based Face Pairing*. The table lists three sets of experimental data: the *gMidSurf* column shows the running time of the face pairing, the *Filtering Gates* column shows the running time and speedup ratio when the hierarchical filtering gates are disabled, as well as the *Simplified Overlap Criterion (SOC)* column.

Filtering Gates. The results demonstrate that the effectiveness of the hierarchical filtering mechanism exhibits significant nonlinear growth with model complexity. For smaller-scale models (M1–M4), disabling the filtering gates results in relatively modest performance degradation, with speedups ranging from 1.58×–5.34×. However, as the number of faces increases, this effect becomes dramatically pronounced. Starting from M5, where the candidate face pair count grows at $O(n^2)$ rate, the advantage of hierarchical filtering gates escalates sharply, achieving speedups of 13.53×–26.52× across M5–M8, with M7 reaching a peak. This clearly demonstrates that hierarchical filtering gates effectively avoid expensive geometric computations on numerous invalid FGPs through progressive culling, showcasing excellent scalability on large-scale models.

Simplified Overlap Criterion. The results reveal that this feature serves as a fast computation method specifically designed for variable-thickness models with gentle curvature variations. This criterion simplifies point-to-mesh projection by converting it to point-to-plane projection through OBB principal direction planes, dramatically reducing overlap gate overhead. More importantly, the minimum distance information obtained during the projection process provides effective pruning cues for the distance gate, enabling earlier termination of

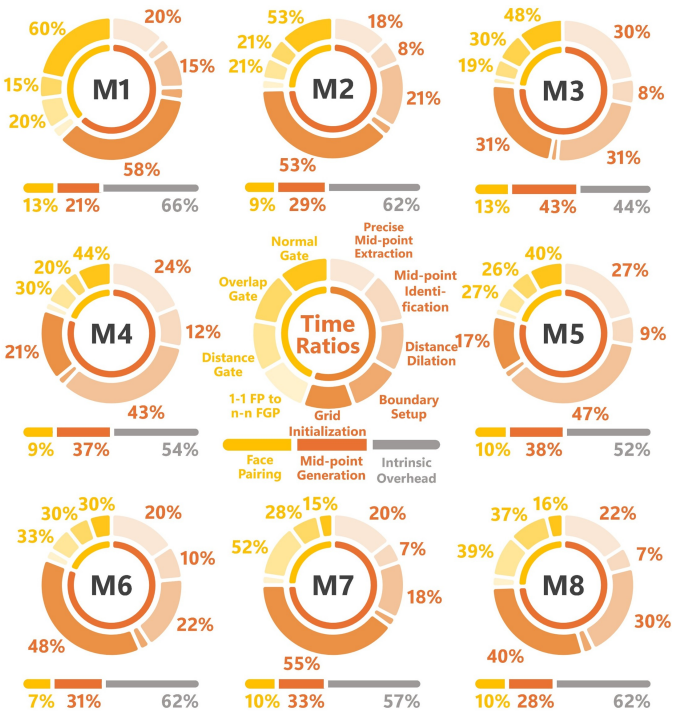

Figure 10: Time distribution across different stages for all benchmarks. Each model displays the percentage breakdown for face pairing (yellow), mid-point generation (orange), and intrinsic time overhead (gray). The pie charts show sub-stage details, and the bars below indicate overall proportions.

invalid branches during subsequent LBVH traversal. This characteristic is particularly effective for variable-thickness models with smooth geometric features, where a large number of FGPs can directly utilize the distance estimates from the overlap gate, avoiding the full distance computation across all candidate face pairs. Experimental results show that this optimization achieves speedups of 1.07×–4.29× across all benchmarks. This design is especially well-suited for standardized parts commonly encountered in real-world industrial scenarios, which typically exhibit gradual transitions in wall thickness.

6.4. Running Time Ratios

To gain deeper insights into the performance of *gMidSurf*, we conducted a detailed profiling of time consumption across all stages. Fig. 10 illustrates the time distribution for various benchmarks. In contrast to traditional methods, the face pairing stage accounts for only 7%–13% of the total time, making it the least time-consuming component and validating the efficiency of our hierarchical filtering gates. The mid-point generation stage occupies 21%–43%, although it's relatively time-intensive, but remains acceptable. Besides, it is necessary to acknowledge that the intrinsic time overhead (offset surface, NURBS fitting, and trimming operation) consumes an average of 40%–60% of the total time, making it the most time-consuming component in the current pipeline; these overheads cannot be eliminated within the current algorithmic framework. Nevertheless, as demonstrated in Table 1, benefiting from the innovative algorithms in the above two stages, the end-to-end total time still achieves a comprehensive speedup of 3×–15× compared to prior methods.

Face Pairing. Within the face pairing stage, the time consumption of the three filtering gates exhibits a clear hierarchical pattern. As the model face count increases, more candidate

face pairs pass through the relatively simple normal gate into the computation-intensive overlap and distance gates, leading to a gradual increase in their time proportions. Moreover, when face groups share similar geometric primitives, the filtering capability of early-stage gates weakens significantly. This phenomenon is particularly pronounced in Model 7, which contains numerous nearly parallel faces. As a result, the model renders the normal gate and overlap gate almost incapable of effective filtering, ultimately causing the distance gate to consume 52% of the total pairing time.

Mid-point Generation. In the mid-point extraction stage, grid initialization and distance dilation occupy a large proportion. As described in Section 5.1, our unified grid pool strategy reduces the grid initialization overhead from over 90% (in the naïve per-FGP allocation) to approximately 40% of the mid-point generation time. On the other hand, since we adopt 1/10 of the wall thickness as the block size to ensure sufficient resolution, FGPs with significant curvature variations generate finer grids, causing a larger dilation body, and the computation time grows cubically. This is fully manifested in Models 3, 4, and 5, where the V-FGPs of these models exhibit significant geometric curvature variations, with distance dilation time proportions reaching 31%, 43%, and 47%, respectively, notably higher than the 20%–30% observed in other models.

7. Discussion and Future Work

Our approach has several limitations. While *gMidSurf* achieves significant performance gains through hierarchical culling and massively parallel geometric kernels on GPU, several challenges remain when processing industrial-scale complex models.

7.1. Architectural constraints with industrial model complexity

Real-world industrial assemblies exhibit heterogeneous topological structures and complex geometric features that challenge GPU's uniform memory access patterns. Our LBVH traversals and distance queries suffer from thread divergence when processing free-form NURBS surfaces with extreme aspect ratios. When FGPs contain mixed planar, quadric, and high-order NURBS surfaces, varying geometric complexity creates load imbalance, degrading SM utilization from ideal 80% to 35-45%. Moreover, large-scale assemblies (>5000 faces) exceed GPU capacity: the distance matrix and LBVH structures require $O(n^2)$ storage, and even with our batching strategy, memory bandwidth becomes a bottleneck for aerospace structures with dense rib and fillet details. Recent work on adaptive space partitioning [22] and compressed BVH [31] has the potential to mitigate these issues. However, integrating such techniques into our unified GPU pipeline requires careful architectural redesign.

7.2. Trade-offs between precision and performance

A key implementation choice in *gMidSurf* is the use of single-precision floating-point (FP32) arithmetic. This decision is driven by the throughput advantage on commodity GPUs: the FP32 performance is approximately twice that of double-precision floating-point (FP64) on the NVIDIA RTX series. In the face pairing stage, adopting double-precision (FP64) would necessitate 64-bit mesh discretization, leading to a substantial increase in memory and reduced memory bandwidth efficiency. Since our three hierarchical gates utilize "approximate values" to screen geometric compatibility rather than performing exact boundary evaluations, we performed additional validation confirming that the precision difference between FP32 and FP64 in this stage is negligible and does not impact the final pairing results. In the mid-point generation stage, we conducted precision validation by comparing our FP32 results against FP64 reference solutions across all benchmarks. The maximum positional deviation of the generated mid-surface points was within 10^{-5}, which is well within the typical CAD modeling tolerance of 10^{-4} for general mechanical parts. For high-precision applications requiring even tighter tolerances (e.g., aerospace components at 10^{-6}), our bracket-and-bisect refinement kernel can be extended to FP64. We provide this as a configurable trade-off, allowing users to prioritize either extreme throughput or ultra-high precision depending on the specific industrial requirements.

7.3. Limitations in real-time interactive performance.

Although *gMidSurf* achieves 3×–6× speedups over CPU baselines, performance remains insufficient for interactive CAD workflows demanding sub-second response. For automotive sheet-metal parts with hundreds of FGPs, end-to-end processing requires seconds, failing to meet immediate feedback needs for parametric design iteration. However, it should be noted that mid-surface abstraction is typically a one-time preprocessing task within the broader CAE pipeline. Compared to the extensive time required for manual model editing and modification, a few seconds of automated generation represents a relatively small fraction of the total workflow. When design modifications trigger local topology changes, our current method requires complete recomputation.

To bridge the gap toward truly interactive performance, a promising strategy is to implement incremental updates that focus only on the modified regions of the model, rather than re-processing the entire assembly. Furthermore, potential improvements can be integrated across the different pipeline stages. For face pairing and mid-point generation, Panozzo et al. [32] demonstrated that cache-aware geometric queries can achieve 10× speedup, while GPU stream compaction techniques [33] offer 1.5×–3× performance gains by reducing memory transfer latency. The current pipeline possesses intrinsic overhead regarding the surface fitting and trimming operation. For surface fitting, leveraging high-performance GPU-accelerated libraries such as Gpufit [34] for large-scale nonlinear parameter estimation will be indispensable for expediting NURBS fitting and evaluation. For the trimming operation, while a full GPU migration remains non-trivial, offloading critical sub-operations such as surface intersection, projection, and mass-parallel ray-surface queries to CUDA-based kernels [35] offers a viable path to alleviate host-side overhead, leading toward a fully integrated and highly responsive mid-surface abstraction pipeline.

7.4. Considerations for downstream FEA applicability

While this paper focuses on the geometric aspects of mid-surface extraction, it is important to discuss the relationship between the extracted mid-surface and its suitability for downstream shell/plate finite element analysis. Classical plate and shell theories are derived under specific geometric assumptions:

the thickness-to-span ratio must be sufficiently small (typically <1/10 for Kirchhoff–Love thin-plate theory, <1/5 for Reissner–Mindlin theory), and the thickness should vary smoothly relative to the span [3, 36]. Regions violating these assumptions—such as areas with rapid thickness transitions, low aspect ratios, or complex multi-branch junctions—may yield inaccurate results under shell element discretization compared to full 3D solid models [37].

gMidSurf provides two pieces of information that directly support *a priori* assessment of shell model validity. First, the wall thickness computed for each FGP by the distance gate (Section 4.3) yields a continuous thickness distribution across the mid-surface, from which thickness gradients and aspect ratios can be evaluated automatically. Second, the classification into constant-thickness (C-FGP) and variable-thickness (V-FGP) face group pairs provides an immediate indicator of geometric regularity. Standard engineering criteria—such as thickness-to-span ratio thresholds and maximum allowable thickness gradient—can be applied as a lightweight post-processing step on these data to flag regions where shell idealization may be unreliable [38].

For regions identified as problematic, the established engineering practice is mixed-dimensional modeling, where shell elements are coupled with 3D solid elements at local regions of concern [39, 40]. In this workflow, the mid-surface extraction output serves as the starting point: shell-eligible regions retain the mid-surface representation, while flagged regions are reverted to their original solid geometry for 3D meshing. *A posteriori* error estimation techniques can further validate the shell model by comparing local stress fields against 3D reference solutions in regions of concern [41]. Thus, mid-surface extraction and structural validity assessment are complementary stages in a mature CAD/CAE pipeline, and the efficiency gains provided by *gMidSurf* directly benefit the overall workflow by enabling rapid iteration of the geometric preprocessing stage.

7.5. Future directions: Learning-augmented hybrid approaches

A promising research avenue involves leveraging geometric deep learning to augment our GPU pipeline. Since mid-surface abstraction has well-defined mathematical formulations (Eq. 4), we can construct large-scale ground-truth datasets for mid-surfaces. Recent advances in 3D shape understanding [42, 43] demonstrate that graph neural networks can effectively learn topological reasoning on B-Rep models. We envision a two-stage approach: (1) training a GNN to directly predict face pairings from the face adjacency graph, bypassing expensive distance/normal/overlap evaluations; (2) employing neural implicit fields [44, 45] to represent mid-surface geometry, enabling continuous distance queries without explicit mesh discretization. Combining learned components with our rigorous geometric kernels promises interactive-rate performance while preserving robustness guarantees.

7.6. Future directions: FEA validation and mixed-dimensional integration

A complementary research direction involves systematic validation of mid-surface FE models against full 3D solid FE solutions. Specifically, by selecting representative thin-walled configurations with controlled thickness variations, aspect ratios, and junction complexities, one could quantify the rate of change of thickness at which shell element accuracy degrades and establish quantitative guidelines for *a priori* region classification. Such a study would require collaboration with structural analysis experts to define appropriate loading scenarios, element types, and error metrics. The thickness fields and FGP classifications produced by *gMidSurf* provide a natural starting point for this investigation. Additionally, integrating automatic shell-suitability checking and mixed-dimensional model generation as a downstream module would create a more complete CAD-to-CAE pipeline, addressing the gap between geometric abstraction and structural analysis readiness.

8. 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×–15× 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] P. Dabke, V. Prabhakar, S. Sheppard, Using features to support finite element idealizations, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. 13805, American Society of Mechanical Engineers, 1994, pp. 183–193.

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

[3] D. Chapelle, K.-J. Bathe, The Finite Element Analysis of Shells—Fundamentals, 2nd Edition, Springer, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-16408-8.

[4] H. Blum, A transformation for extracting new descriptors of shape, Models for the perception of speech and visual form (1967) 362–380.

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

[6] D. J. Sheehy, C. G. Armstrong, 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, 1995, pp. 201–212.

[7] W. R. Quadros, K. Shimada, Hex-layer: Layered all-hex mesh generation on thin section solids via chordal surface transformation, in: Proceedings of 11th International Meshing Roundtable, 2002, pp. 169–180.

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

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

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

[11] L. Ye, X. Zhou, P. Fan, R. Tong, H. Li, P. Du, M. Tang, MidSurfer: Efficient mid-surface abstraction from variable thin-walled models, Computer-Aided Design 190 (2026) 103965.

doi:https://doi.org/10.1016/j.cad.2025.103965.
URL https://www.sciencedirect.com/science/article/pii/S0010448525001265

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

[13] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, D. Manocha, Fast BVH construction on GPUs, in: Computer Graphics Forum, Vol. 28, Wiley Online Library, 2009, pp. 375–384.

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

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

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

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

[18] Siemens Digital Industries Software, Parasolid: Geometric modeling kernel, https://www.plm.automation.siemens.com/global/en/products/plm-components/parasolid.html (2022).

[19] C. Chong, A. S. Kumar, K. Lee, Automatic solid decomposition and reduction for non-manifold geometric model generation, Computer-Aided Design 36 (13) (2004) 1357–1369.

[20] Y. H. Kulkarni, A. Sahasrabudhe, M. Kale, Leveraging feature generalization and decomposition to compute a well-connected midsurface, Engineering with Computers 33 (1) (2017) 159–170.

[21] E. Boneš, D. Khan, C. Bohak, B. A. Barad, D. A. Grotjahn, I. Viola, T. Theußl, Midsurfer: A parameter-free approach for mid-surface extraction from segmented volumetric data, IEEE Computer Graphics and Applications (2025) 1–14doi:10.1109/MCG.2025.3624572.

[22] H. Zhu, Y. Liu, J. Zhao, H. Wang, Calculating the medial axis of a CAD model by multi-CPU based parallel computation, Advances in Engineering Software 85 (2015) 96–107.

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

[24] A. C. Jalba, J. Kustra, A. C. Telea, Surface and curve skeletonization of large 3D models on the GPU, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (6) (2013) 1495–1508. doi:10.1109/TPAMI.2012.212.

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

[26] H. Zhu, Y. Liu, J. Bai, X. Ye, Constructive generation of the medial axis for solid models, Computer-Aided Design 62 (2015) 98–111.

[27] Y. Miao, R. Pajarola, J. Feng, Curvature-aware adaptive re-sampling for point-sampled geometry, Computer-Aided Design 41 (6) (2009) 395–403.

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

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

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

[31] D. Meister, S. Ogaki, C. Benthin, M. J. Doyle, M. Guthe, J. Bittner, A survey on bounding volume hierarchies for ray tracing, Computer Graphics Forum 40 (2) (2021) 683–712.

[32] D. Panozzo, Y. Lipman, E. Puppo, D. Zorin, Fields on symmetric surfaces, ACM Transactions on Graphics 31 (4) (2013) 1–12.

[33] N. Evangelou, Y. Chrysanthou, M. Nikodem, Fast radius search exploiting ray-tracing frameworks, in: Journal of Computer Graphics Techniques, Vol. 10, 2021, pp. 25–48.

[34] A. Przybylski, B. Thiel, J. Keller-Findeisen, B. Stock, M. Bates, Gpufit: An open-source toolkit for gpu-accelerated curve fitting, Scientific reports 7 (1) (2017) 15722.

[35] R. Leung, Gpu implementation of a ray-surface intersection algorithm in cuda (compute unified device architecture), arXiv preprint arXiv:2209.02878.

[36] J. N. Goodier, On the problems of the beam and the plate in the theory of elasticity, Transactions of the Royal Society of Canada, Third Series 32 (1938) 65–88.

[37] O. C. Zienkiewicz, R. L. Taylor, The Finite Element Method for Solid and Structural Mechanics, 6th Edition, Elsevier Butterworth-Heinemann, Oxford, 2005.

[38] A. Thakur, A. G. Banerjee, S. K. Gupta, A survey of CAD model simplification techniques for physics-based simulation applications, Computer-Aided Design 41 (2) (2009) 65–80. doi:10.1016/j.cad.2008.11.009.

[39] K. W. Shim, D. J. Monaghan, C. G. Armstrong, Mixed dimensional coupling in finite element stress analysis, Engineering with Computers 18 (3) (2002) 241–252. doi:10.1007/s003660200021.

[40] T. Robinson, C. Armstrong, R. Fairey, Automated mixed dimensional modelling from 2d and 3d cad models, Finite Elements in Analysis and Design 47 (2) (2011) 151–165. doi:https://doi.org/10.1016/j.finel.2010.08.010.
URL https://www.sciencedirect.com/science/article/pii/S0168874X10001447

[41] M. Ainsworth, J. T. Oden, A posteriori error estimation in finite element analysis, Computer Methods in Applied Mechanics and Engineering 142 (1–2) (1997) 1–88. doi:10.1016/S0045-7825(96)01107-3.

[42] C. R. Qi, H. Su, K. Mo, L. J. Guibas, PointNet: Deep learning on point sets for 3D classification and segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 652–660.

[43] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, J. M. Solomon, Dynamic graph cnn for learning on point clouds, ACM Transactions on Graphics 38 (5) (2019) 1–12.

[44] J. J. Park, P. Florence, J. Straub, R. Newcombe, S. Lovegrove, DeepSDF: Learning continuous signed distance functions for shape representation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 165–174.

[45] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, A. Geiger, Occupancy networks: Learning 3d reconstruction in function space, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4460–4470.