







Parallel Constraint Graph Partitioning and Coloring for Realtime Soft-Body Cutting

Peng Yu¹ , Ruiqi Wang¹, Chunlei Li¹ , Yuxuan Li¹, Xiao Zhai², Yuanbo He³ , Hongyu Wu¹ , Aimin Hao¹ , Yang Gao^{†1} 

¹ State Key Laboratory of Virtual Reality Technology and Systems, Beihang University, China

²Weta FX, New Zealand

³Institute of Electrophysiology, Henan Academy of Innovations in Medical Science, China

Abstract

Real-time simulation of cutting is essential in fields requiring accurate interactions with digital assets, such as virtual manufacturing or surgical training. While Extended Position-Based Dynamics (XPBD) methods are valued for their numerical stability, their reliance on the Gauss-Seidel method leads to two critical limitations when facing high degrees of freedom: the residual stagnation that hinders convergence within limited temporal budget, and a fundamentally sequential nature that limits parallelization, thereby impeding real-time performance. Traditional parallelization approaches often rely on precomputed topological data that becomes outdated during mesh evolution, resulting in suboptimal performance in cutting applications. To address this limitation, this paper introduces a GPU-accelerated algorithm featuring an efficient constraint clustering pre-processing step to accelerate initial solver scheduling, combined with a novel graph coloring technique using GPU-optimized Shortcuts principles for parallel constraint resolution. Experiments show our combination of upfront clustering and dynamic graph re-coloring outperforms existing parallel XPBD implementations, empowering efficient solvers in virtual surgery, product design, and similar scenarios involving continuous geometry updates.

CCS Concepts

• **Computing methodologies** → *Computer Graphics*;

1. Introduction

Real-time cutting of soft bodies is a key requirement in virtual surgery [YZWP24, BTM17], robotics [HMN*21], and interactive applications. Existing methods rely on Finite Element Method (FEM) [JDP*25], Material Point Method (MPM) [HFG*18], or Position Based Dynamics (PBD) [YZWP24]. FEM provides high accuracy but incurs high computational cost, making real-time applications challenging, particularly in cutting scenarios. MPM supports discontinuities and dynamic boundaries but suffers from stability and performance issues at high resolution. PBD offers unconditional stability and real-time efficiency but reduced physical accuracy.

Extended Position-Based Dynamics (XPBD) [MMC16] has become a leading framework for real-time deformable object simulation, notably in virtual surgery and intelligent manufacturing. XPBD uses a Projected Gauss-Seidel (PGS) solver to enforce constraints efficiently. While PGS converges faster than Jacobi solvers, its limited parallelism due to data dependencies between adjacent constraints is a bottleneck. Graph coloring tech-

niques [PZY*20] improve parallelism by enabling concurrent processing of independent constraints. Nevertheless, high-DoF simulations face two challenges: iterative solvers suffer from "Residual Reduction Stall" [CHC*24, CHC*23], causing unrealistic stretching, and dynamic topological changes, such as cutting, invalidate static precomputed structures used in parallelization. Acceleration methods, including rigid-soft coupling [MAK24] and hybrid clustering-coloring [TTKA23], rely on precomputation and fail under topological changes. Fratarcangeli et al. [FTP16] proposed a GPU-based greedy coloring for Projective Dynamics [BML*14], enabling per-frame recoloring for dynamic topologies. However, coloring failures and excessive color usage reduce parallelization efficiency in practice.

This paper presents a GPU-accelerated algorithm that addresses these limitations for real-time cutting simulations. We introduce a novel graph coloring algorithm featuring a "Color Preemption" shortcut mechanism, which significantly enhances parallelism and accelerates convergence by adapting to dynamic graph topologies generated during cutting. To further improve performance, a neighbor-based constraint clustering approach is used as a pre-processing step for efficient graph initialization. The simulation's stability and visual fidelity are maintained through a robust tetrahedral subdivision and separation method, which incorporates trajectory

[†] gaoyangvr@buaa.edu.cn

correction to prevent degenerate sliver tetrahedra and uses a shared-vertex criterion for element separation. Our integrated approach of dynamic graph coloring, constraint clustering, and robust cutting demonstrates superior performance over existing parallel XPBD implementations, providing an efficient solver for softbody cutting applications.

2. Related Works

In recent years, the simulation of soft body cutting has drawn significant attention among graphics research, driven by its importance in applications like virtual reality, surgical training, industrial design, and digital manufacturing. The core challenge lies in achieving real-time performance without compromising physical accuracy, particularly when dealing with complex topological changes and heterogeneous material properties.

Various approaches have been proposed to address these challenges. Jia et al. [JPW*17] employed adaptive octree meshes with triangular internal structures to maintain stability and real-time performance during dynamic cutting, avoiding artifacts from degenerate elements. Kamarianakis et al. [KPA*22] combined linear blend skinning with particle-based soft body deformation to achieve real-time, geometry-driven cutting and continuous tearing. However, these methods incur significant computational overhead at high mesh resolutions, limiting real-time applicability.

PBD has gained popularity in interactive applications like video games and surgical simulation owing to its unconditional stability, excellent performance, and ease of implementation. Berndt et al. [BTM17] introduced a soft body cutting algorithm based on PBD, using Nvidia's Flex library [MMCK14]. To address visual artifacts arising from surface reconstruction, Yu et al. [YZWP24] further proposed the Aggregate Finding Connected Components (AFCC) algorithm as an enhancement. To reduce the sensitivity of PBD to time step size and iteration count, Macklin et al. [MMC16] proposed the XPBD method. Despite its well-recognized efficiency, Chen et al. [CHC*23] noted that XPBD can suffer from residual reduction stagnation when primal residual terms are ignored. Furthermore, XPBD typically uses local Gauss-Seidel (GS) iterations to enforce constraints.

Several strategies have been proposed to improve the performance of XPBD in high-resolution settings. Multi-resolution methods, such as HPBD's coarse-to-fine motion scheme [Mü08] and the Long Range method [MCMJ17], which exploits global constraint information to optimize fixed-endpoint rope solutions, offer performance gains in specific scenarios but can introduce visual artifacts. On the parallelization front, Mercier-Aubin et al. [MAK24] split objects into rigid and soft parts and used a hierarchical structure to accelerate constraint solving, though their implementation was CPU-based. Constraint graph parallelization has emerged as a critical direction for improving XPBD's efficiency. Ton-That et al. [TTKA23] performed constraint graph clustering and coloring during preprocessing, enabling parallel solving of same-colored clusters on the GPU, with sequential solving within each cluster. However, the limitation of this pre-computation strategy is evident in cutting scenarios: mesh topology changes invalidate pre-computed data, leading to a drastic drop in simulation performance.

Fratarcangeli et al. [FTP16] proposed a GPU-based graph coloring method for soft body modeling, allowing per-frame re-coloring to better handle topological changes. Nevertheless, there is still room for improvement in the speed and scalability of current GPU-based solvers.

3. Background

We represent soft bodies using N particles with positions $\mathbf{x} \in \mathbb{R}^{3 \times N}$ and velocities $\mathbf{v} \in \mathbb{R}^{3 \times N}$. Applying implicit backward Euler integration to Newton's second law yields the following nonlinear governing equation:

$$\frac{\mathbf{M}}{\Delta t^2}(\mathbf{x}^{n+1} - \mathbf{x}) = -\nabla U^\top(\mathbf{x}), \quad (1)$$

where \mathbf{M} is the diagonal mass matrix, Δt is the timestep size, n is the timestep counter, and $U(\mathbf{x})$ represents the elastic energy potential. The unconstrained positions are given by $\tilde{\mathbf{x}} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+1}$. Under the XPBD framework, the potential energy is formulated as a quadratic function:

$$U(\mathbf{x}) = \frac{1}{2} C(\mathbf{x})^\top \alpha^{-1} C(\mathbf{x}), \quad (2)$$

where $C(\mathbf{x}) \in \mathbb{R}^m$ represents the constraint function, and α is a diagonal compliance matrix.

Introducing the Lagrange multiplier $\boldsymbol{\lambda} = -\tilde{\alpha}^{-1} C(\mathbf{x})$, with $\tilde{\alpha} = \alpha/\Delta t^2$, enables Equation 1 to be reformulated as a nonlinear system, which can then be linearized into:

$$\begin{bmatrix} \mathbf{M} & -\nabla C^\top \\ \nabla C & \tilde{\alpha} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ -C(\mathbf{x}) - \tilde{\alpha} \boldsymbol{\lambda} \end{bmatrix}. \quad (3)$$

Applying the Schur complement with respect to \mathbf{M} leads to a reduced linear system

$$(\nabla C(\mathbf{x}) \mathbf{M}^{-1} \nabla C(\mathbf{x})^\top + \tilde{\alpha}) \Delta \boldsymbol{\lambda} = -C(\mathbf{x}) - \tilde{\alpha} \boldsymbol{\lambda}. \quad (4)$$

Equation 4 can be solved using a direct solver [GHF*07]. However, the cubic time complexity in factorization and the substantial memory demands limit its applicability in high-resolution scenarios. Therefore, XPBD typically employs a GS solver, which sequentially solves each constraint:

$$\Delta \boldsymbol{\lambda}_j = \frac{-C_j - \tilde{\alpha}_j \boldsymbol{\lambda}_j}{\nabla C_j \mathbf{M}^{-1} \nabla C_j^\top + \tilde{\alpha}_j}. \quad (5)$$

The position of the i -th node affected by the constraint function C_j is then updated according to

$$\Delta \mathbf{x}_i = -m_i^{-1} \Delta \boldsymbol{\lambda}_j \nabla C_{ji}. \quad (6)$$

In this paper, we model soft body deformation using Stable Neo-Hookean elasticity [MM21, SGK18].

4. Method

To achieve real-time performance for cutting simulations within the XPBD framework, this paper introduces optimizations for both parallel constraint solving and convergence acceleration. Section 4.1 presents our novel ShortCut Graph Coloring (SCGC) algorithm; extension of the ECL-GC method [APB20], we propose a

new shortcut to accelerate GPU-based parallel graph coloring, enabling concurrent constraint solving. Section 4.2 describes our constraint clustering method, which is designed to accelerate the system's convergence. Finally, Section 4.3 explains our robust geometric handling of the cutting process.

4.1. Shortcut Graph Coloring Algorithm (SCGC)

In XPBD, constraints are typically resolved sequentially in Gauss–Seidel fashion; solving them in parallel without accounting for dependencies can cause conflicting updates on shared particles, reducing accuracy, violating race conditions, or producing non-deterministic results. Yet, many constraints are independent and can be processed in parallel without loss of accuracy or determinism. An effective optimization is to group such constraints, which can be achieved via graph coloring: the constraint adjacency is first represented as a graph, then colored so that adjacent constraints receive different colors. Constraints with the same color are independent and can be solved in parallel, while color groups are processed sequentially.

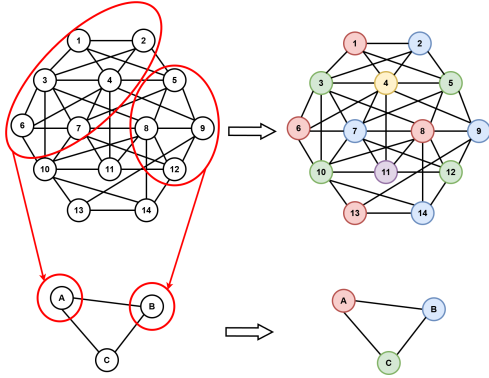


Figure 1: From left to right, graph coloring is applied to partition the constraint graph into independent sets. Each color corresponds to a group of constraints that can be solved in parallel without data conflicts. From top to bottom, graph clustering reduces the number of colors.

To parallelize the constraint solver, we build upon the Jones-Plassmann method [JP93], a classic algorithm for parallel graph coloring. To manage parallel execution and prevent race conditions, a global, unique priority is assigned to every vertex. This priority is determined primarily by the vertex's degree (higher degree means higher priority). While the algorithm executes in parallel across all uncolored vertices in each iteration, this priority scheme defines a logical order: a vertex is only eligible for coloring after all of its higher-priority neighbors have been colored. The algorithm adheres to a strict greedy coloring policy. When a vertex becomes eligible for coloring, it inspects the colors already assigned to its neighbors and chooses the smallest non-negative integer (i.e., the "best" or highest-priority color) that is not currently in use by any neighbor. ECL-GC [APB20], a state-of-the-art implementation, introduced two powerful shortcuts to accelerate convergence by breaking dependencies early. Our core contribution is

the design and integration of a third, novel shortcut, termed Color Preemption, which results in our ShortCut Graph Coloring (SCGC) algorithm. The SCGC algorithm makes a vertex eligible for coloring if it satisfies the standard dependency-free condition or any of our three complementary shortcut criteria (summarized in Algorithm 1):

- **Shortcut 1 (Best-Color Availability):** A vertex can be colored if its best available color is not being considered by any of its uncolored higher-priority (degree) neighbors. (Line 6-7)
- **Shortcut 2 (Color Set Disjointness):** A dependency on a higher-priority neighbor can be ignored if their respective sets of possible colors are completely disjoint. (Line 8-10)
- **Shortcut 3 (Color Preemption - Our Contribution):** A vertex can be colored if its best available color is of a higher priority (i.e., a smaller integer) than the best available colors of all its uncolored higher-priority neighbors. (Line 11-12)

By uniquely leveraging the greedy nature of the coloring choice, our new shortcut identifies parallelization opportunities not captured by the original two shortcuts. The effectiveness of SCGC is demonstrated in the challenging domain of dynamic soft-body simulation, where frequent topological changes necessitate highly efficient recoloring. Experiments show that our three-shortcut approach reduces the average coloring time by 10% to 20% (Table 2) compared with the original two-shortcut ECL-GC.

Algorithm 1: Parallel Short Cut Graph Coloring Algorithm

Input: Graph $G = (V, E)$
Output: Color set Q

```

1 while uncolored nodes exist do
2   foreach node  $v \in V$  in parallel do
3      $N \leftarrow$  number of higher-priority neighbors of  $v$ ;
4     if all  $u \in N$  are colored then
5        $Q[v] \leftarrow$  best color of  $v$ ;
6       // shortcut1:
7       if all  $u \in N$  cannot use best color of  $v$  then
8          $Q[v] \leftarrow$  best color of  $v$ ;
9       // shortcut2:
10      foreach  $u \in N$  do
11        if available colors of  $u$  and  $v$  are disjoint then
12          remove edge between  $u$  and  $v$ ;
13      // shortcut3:
14      if all candidate colors of  $u$  ( $u \in N$ ) are worse than
15        best color of  $v$  then
16         $Q[v] \leftarrow$  best color of  $v$ ;
17 return  $Q$ ;
```

As illustrated in Figure 1, after applying SCGC coloring, adjacent graph nodes are assigned different colors. The color and cluster information associated with each constraint are then passed to the XPBD simulation module to enable parallel constraint solving.

4.2. Constraint Graph Clustering

Since constraints of different colors are solved sequentially, the number of colors directly determines the degree of parallelism in XPBD. For high-resolution models, direct graph coloring often produces too many colors, and standard coloring algorithms are slow on large graphs, hindering real-time cutting simulation.

To mitigate this, Ton-That et al. [TTKA23] introduced a clustering-based preprocessing strategy. Adjacent nodes are grouped into clusters ("super nodes") and edges between clusters are preserved if any original nodes are connected. This reduces graph size, accelerates coloring, and typically lowers the total number of colors, thereby improving XPBD parallelism and overall performance (Figure 1). However, the Greedy K-Neighbour clustering algorithm [TTKA23] requires frequent use of breadth-first search (BFS). This results in relatively slow execution, limiting its efficiency, especially for large-scale or time-critical simulations.

We proposed a serial neighbor clustering algorithm with a size limit Ks . As described in Algorithm 2, clustering is performed iteratively, with each node forming a group alongside up to $Ks - 1$ of its neighbors, excluding itself. This process continues until every node belongs to a cluster. We assign one CUDA kernel per cluster, and constraints within a cluster are solved serially. Despite its simplicity, this approach proves effective for large-scale graphs, balancing clustering efficiency with computational overhead.

Algorithm 2: Serial Neighbor Clustering Algorithm

Input: Undirectional graph $G = (V, E)$, cluster size limit Ks

Output: Cluster set C

```

1 Mark all nodes in  $V$  as unclustered;
2  $C \leftarrow \emptyset$ ;
3 foreach node  $v \in V$  do
4   if  $v$  is unclustered then
5     Initialize new cluster  $c$ ;
6     Add  $v$  to  $c$  and mark  $v$  as clustered;
7      $N \leftarrow$  list of unclustered neighbors of  $v$ ;
8     Select up to  $(Ks - 1)$  nodes from  $N$ ;
9     foreach node  $u$  in selected neighbors do
10      Add  $u$  to  $c$  and mark  $u$  as clustered;
11   Add cluster  $c$  to  $C$ ;
12 return  $C$ ;
```

4.3. Soft Body Cutting

To accurately simulate soft body cutting, we use a tetrahedral subdivision approach. When a tetrahedron intersects a cutting plane, it is subdivided into smaller tetrahedra along the plane's surface, as illustrated in Figure 2. Two intersection scenarios can arise: three-point and four-point intersections. A three-point intersection subdivides the tetrahedron into four smaller tetrahedra, as the top figure shows, the tetrahedron is subdivided as : $AB_1C_1D_1$, BB_1C_1C , $B_1C_1D_1C$, and B_1CD_1D . While a four-point intersection results in six smaller tetrahedra : $AE_1F_1H_1$, ADF_1H_1 , $DF_1G_1H_1$, $BE_1F_1H_1$, BCF_1H_1 , and $CF_1G_1H_1$. The original tetrahedron is removed from its cluster and no longer participates in subsequent computations,

while the newly generated tetrahedra form a new cluster. This operation modifies the graph topology, producing an updated input graph for coloring. To remain efficient, only the local clusters around the cut are updated, avoiding a full reclustering. This subdivision preserves mesh structure and topological consistency while ensuring the efficiency of the cutting operation.

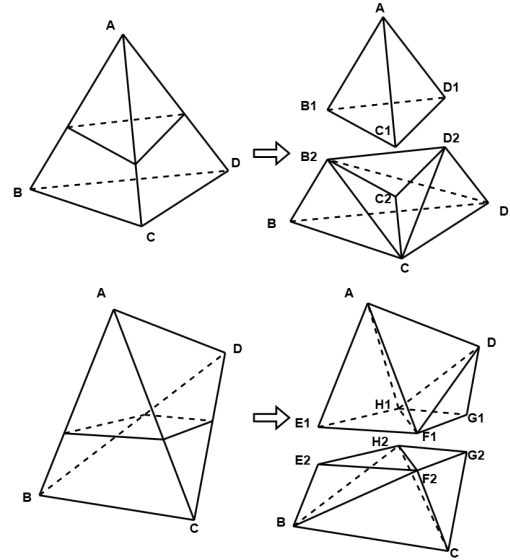


Figure 2: Tetrahedral subdivision after cutting: (Top) Tetrahedron $ABCD$ intersected at three particles, (Bottom) Tetrahedron $ABCD$ intersected at four particles

A common challenge arises when an intersection point is located extremely close to a tetrahedron vertex, which can create tetrahedra with impractically small volumes. While often visually imperceptible, these tiny elements significantly increase the computational burden during subsequent simulation steps. To prevent such issues, we apply a trajectory correction to the cutting path, snapping intersection points within a threshold distance to their nearest vertex. This adjustment ensures larger, more uniform tetrahedron volumes, which improves simulation robustness and performance with minimal impact on the visual fidelity of the cutting simulation. As shown in Figure 3, the cutting plane is very close to one of the model's vertices. Without correction, this would result in the generation of many extremely small tetrahedra. To avoid this, the cutting plane is adjusted to snap directly to the nearby vertex.

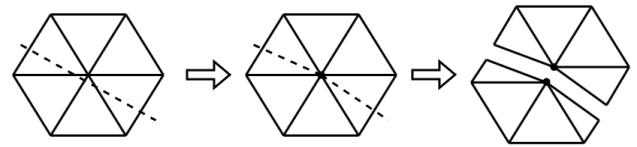


Figure 3: Cutting plane correction

Following the tetrahedron subdivision, the next crucial step is determining which tetrahedra should be separated to represent the

cut path and which remain connected. A natural approach is to classify tetrahedra based on their relative positions with respect to the cutting plane, assuming tetrahedra on the same side are connected. However, due to the previously mentioned trajectory correction and the inherent non-planarity of complex cutting paths, such treatment can lead to situations where the “same side of the plane” criterion becomes unreliable because the cutting surface might no longer be a continuous, well-defined plane.

Instead, we use a more robust criterion based on shared vertices. As illustrated in Figure 5, tetrahedra remain connected if they share at least one original vertex in addition to sharing the intersection point created by the cut; otherwise, they belong to different sides and should be separated. To simplify the demonstration, we illustrate the cutting process using two-dimensional triangles. After cutting triangles ABC and ACD , the mesh is subdivided into six smaller triangles: AE_1F_1 , AF_1G_1 , BCE_2 , CE_2F_2 , CF_2G_2 , and CDG_2 . Among these, triangles AE_1F_1 and AF_1G_1 share vertex A , and consequently also share point F_1 . Similarly, triangles BCE_2 and CE_2F_2 share point E_2 ; CE_2F_2 and CF_2G_2 share point F_2 ; and CF_2G_2 and CDG_2 share point G_2 . These shared particles define the connectivity between adjacent triangles, effectively separating the original two triangles where appropriate while maintaining a consistent and connected mesh structure across the cut. This strategy effectively prevents spurious separation during simulation.

5. Results

To evaluate our methods, we implemented our cutting simulation using CUDA. All experiments were conducted on a PC equipped with an Intel i9 10900X CPU, an NVIDIA GeForce RTX 3090 GPU, and 128 GB of host memory.

5.1. Performance and Quality Evaluation

To evaluate the performance of our GPU-parallel cutting simulation algorithm, we conducted deformation and cutting experiments on three representative soft body models: liver, bunny, and armadillo, under multiple resolutions (liver: 4,705 / 11,367 / 88,958 tets; armadillo: 54,985 / 121,190 / 226,840 tets; sphere: 6,851 / 27,792 / 118,202 tets). The performance was compared against several baseline graph coloring algorithms, including the GPU-based Vivace method [FTP16, CLYY24], the GPU-based data-driven WLC method [CLF* 17], and the CPU-based Maximum Cardinality Search (MCS) [PP05, CLYY24] and Greedy coloring approaches used by Ton-That et al. [TTKA23]. Experiments were conducted under two configurations: with and without constraint graph clustering, which controls the granularity of parallelism (Figure 4). For robustness and fair comparison, we modified the Vivace method by setting its palette shrinkage factor to $s = 1$, preventing potential deadlocks in complex graphs and ensuring termination across all test cases.

Experimental results are shown in Figure 4 and Figure 11. Across all tested models and resolutions, our method consistently required the fewest colors and achieved the shortest frame times. Constraint graph clustering improved performance for all algorithms by reducing the number of colors, thereby enhancing parallelism and reducing constraint solving costs. As illustrated in Fig-

ure 11, the per-frame runtime of the liver cutting simulation (11,367 tets) is primarily dominated by cutting, coloring, and solving, highlighting the critical importance of efficient coloring for overall simulation speed.

Further analysis (Table 2 confirms that our proposed GPU-parallel graph coloring algorithm consistently outperformed all baselines including the ECL-GC method [APB20] and obtained the same number of colors in various scenarios. Our SCGC achieved the smallest number of colors and the fastest constraint solving times across all configurations, demonstrating its robustness and scalability. In contrast, the Vivace method [FTP16] tended to produce more colors due to its greedy strategy, which often leads to sub-optimal global solutions in complex graphs. Our SCGC algorithm avoids this issue by adopting a more global approach, incorporating shortest-path inspired logic to improve color assignment. Overall, these results validate the effectiveness of our method in maximizing parallelism and accelerating XPBD’s constraint solving, thereby enabling efficient real-time cutting simulations.

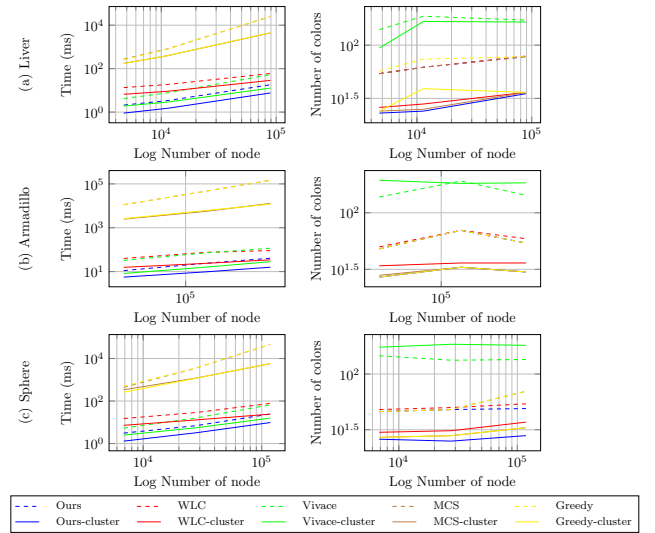


Figure 4: Comparison of different coloring algorithms w.r.t. time to complete (left column), and number of used colors (right column).

5.1.1. Evaluation of Graph Clustering

To assess the effectiveness of our clustering algorithm, we compared it with the Greedy K-Neighbor Clustering method [TTKA23]. As shown in Table 1 (11K Liver model), our approach delivers a thousand-fold speedup in clustering time while generating slightly fewer clusters. Importantly, this acceleration does not compromise quality: the number of colors required and the resulting simulation performance remain nearly identical, with our method even offering a slight performance advantage. Although clustering is only performed during initialization, this dramatic efficiency gain shortens setup time, improves user experience, and even suggests potential for asynchronous cluster updates during runtime operations such as cutting.

We further investigated the impact of cluster size on solver convergence and simulation performance. We tested cluster sizes of

Cluster size	Methods	Clusters	# Colors	Cluster Time(ms)	FPS
5	Greedy K-Neighbour	2245	26	79084	108
	Ours	1972	32	11	110
10	Greedy K-Neighbour	1098	19	79476	132
	Ours	1177	24	11	134
15	Greedy K-Neighbour	711	21	78392	109
	Ours	909	23	8	126
20	Greedy K-Neighbour	527	23	114426	74
	Ours	764	23	8	101
25	Greedy K-Neighbour	414	24	69107	67
	Ours	676	22	7	91

Table 1: Comparison of performance and color distribution across 11K Liver model for our clustering algorithm and Greedy K-Neighbour clustering algorithm [TTKA23].

5, 10, 15, 20, and 25, measuring residual reduction over 500 iterations of the first frame. Figure 7 shows that introducing clustering (vs. baseline 0, i.e., no clustering) consistently accelerates GS solver convergence, with cluster sizes 5 and 20 achieving the fastest residual drop. However, Table 3 reveals a trade-off: while small clusters significantly improve performance, overly large clusters reduce simulation speed due to increased sequential workload within each parallel task, offsetting the gains from reduced coloring. Overall, a cluster size of 5 offers the best balance between fast convergence and high simulation throughput.

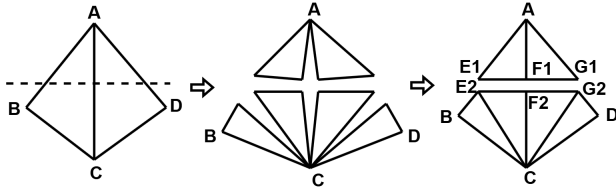


Figure 5: Separation and connection of tetrahedra

5.1.2. Assessing Color Distribution Uniformity

The quality of a graph coloring algorithm is typically assessed along two key dimensions:

- **Number of Colors:** The number of colors directly determines the number of sequential steps required for constraint solving. A smaller number of colors implies that constraints can be partitioned into fewer independent groups, leading to greater parallelism, fewer GPU kernel launches, and, consequently, improved overall simulation performance.
- **Uniformity of Color Class Sizes:** Beyond the number of colors, the uniformity of the node (constraint) distribution across the color classes is also crucial. A significant skew in color class sizes, where some classes contain a large number of constraints while others have very few, leads to inefficient hardware utilization. This imbalanced workload distribution results in idle threads and reduced effective GPU throughput, especially for the smaller color classes.

We evaluated load balancing on the liver (11,367 and 88,958

tets), bunny (19,378 tets), and armadillo (54,985 tets) models, comparing our method with Vivace [FTP16], WLC [CLF*17], and greedy coloring [PP05, TTKA23], using standard deviation, range, and chi-squared statistic (χ^2). As shown in Table 2, Vivace achieved the smallest standard deviation and range, but required more colors and yielded lower performance. Our method and ECL-GC obtained the lowest χ^2 , reflecting the most uniform color distribution, along with fewer colors and faster solving time. With our new shortcut, our method further outperformed ECL-GC in both solving time and frame rate. Overall, our algorithm combines high parallelism with balanced workload distribution, maximizing GPU efficiency.

5.2. Convergence

Our experiments show that the proposed dynamic constraint graph clustering and coloring methods greatly accelerate GS solver convergence. As illustrated in Figure 7, on the Liver model (11,367 tets, $E = 1.0e7$, $\mu = 0.49$, $timestep = 0.016$, $damping = 0.05$), the clustered and colored GS approach consistently converges faster than Jacobi and Chebyshev solvers. Table 3 further confirms that GS requires far fewer iterations and less runtime to reduce the residual to 1.5. Among tested cluster sizes, the best convergence is achieved with a cluster size of 10.

We evaluate the convergence behavior of our proposed clustered GS method by analyzing the power density spectrum of the residual, as shown in Figure 10. This figure contrasts our method (using cluster sizes 0 to 20) with two baseline methods, Jacobi and Chebyshev, across three snapshots of the iterative process: 1, 15, and 500 iterations. A key observation can be made from the middle column, which captures the state after 15 iterations. Here, it is evident that a larger cluster size in the GS framework significantly enhances the reduction of both high- and low-frequency residuals. The baseline methods, however, perform poorly in this regard, failing to effectively dampen the high-frequency components. This disparity diminishes by the 500th iteration, as seen in the rightmost column, where all methods achieve a similar, low-level residual. This confirms that while all tested solvers are capable of reaching the solution, our clustered GS method provides a marked improvement in convergence speed by more rapidly eliminating challenging high-frequency errors.

Model	Methods	# Tets	# Clusters	# Colors	Color Distribution			Color(ms)	Cut(ms)	Solve(ms)	FPS
					STD	Range	$\bar{\chi}^2$				
Liver 11K	Vivace [FTP16]	11376	2071	295	2.31	13	31.93	3	10	63	13
	WLC [CLF*17]			50	38.02	161	42.13	9		5.0	40
	ECL-GC [APB20]			50	30.24	91	26.65	1.5		4.3	63
	Ours			50	30.24	91	26.65	1.2		4.3	65
Liver 88K	Vivace [FTP16]	88957	11391	313	5.83	35	8.03	10	20	70	10
	WLC [CLF*17]			51	219.01	917	49.04	16		14	20
	ECL-GC [APB20]			48	194.69	538	32.31	4		11	28
	Ours			48	194.69	538	32.31	3.5		9.8	30
Bunny	Vivace [FTP16]	19378	2988	279	2.92	14	20.67	6	11	54	14
	WLC [CLF*17]			49	59.09	263	46.00	14		6.3	32
	ECL-GC [APB20]			48	47.72	149	28.24	2		4.8	56
	Ours			48	47.72	149	28.24	2		4.5	57
Armadillo	Vivace [FTP16]	54984	6141	313	4.31	26	10.15	8	15	77	10
	WLC [CLF*17]			51	156.72	729	42.13	14		6.7	28
	ECL-GC [APB20]			51	132.84	418	41.42	3		4.2	45
	Ours			51	132.84	418	41.42	2.6		3.6	47

Table 2: Comparison of performance and color distribution across various models for different graph coloring algorithms, following constraint graph clustering.

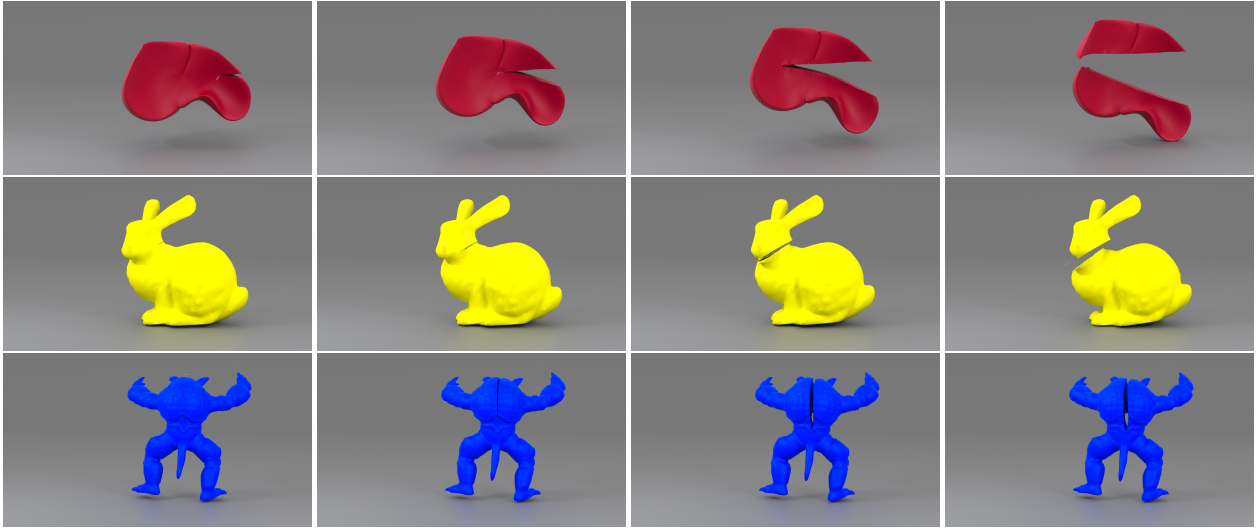


Figure 6: Illustrations of the real-time cutting process for liver, bunny, and armadillo models (from top to bottom).

5.3. Dynamic Cutting Scenarios

To evaluate the robustness and real-time performance of our graph coloring algorithm under dynamic cutting, we conducted continuous cutting experiments on three representative soft body models: Liver, Armadillo, and Bunny. During each experiment, we recorded the evolution of the number of colors and per-frame computation time for the Vivace method [FTP16], the WLC method [CLF*17], and our approach (Figure 8).

As shown in Figure 8 (left), the number of colors produced by all methods increases with the growing complexity of the constraint graph induced by progressive cutting. However, both our method and WLC show slower growth compared to Vivace, demonstrating

better efficiency in preserving parallelism under dynamic topology changes. In particular, during the initial cutting phase, our method consistently achieves about 40% fewer colors than Vivace, establishing a stronger baseline for subsequent parallel solving.

Figure 8 (right) further shows that our method maintains a highly stable and consistently low per-frame computation time throughout the entire cutting process, outperforming WLC in execution speed. In contrast, Vivace suffers from pronounced degradation, with per-frame time rising significantly as cutting proceeds, likely due to the additional overhead introduced by its greedy strategy under frequent topological changes.

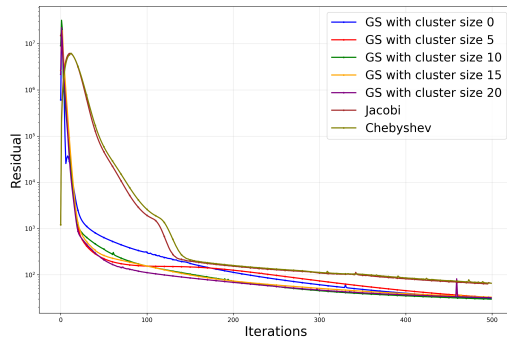


Figure 7: Residual Convergence of Jacobi, Chebyshev and GS Methods with Varying Cluster Sizes.

Methods	# Max Cluster Num	# Iterations	Times(ms)
GS	1	20128	23062
	5	17383	20538
	10	17157	19710
	15	16861	22751
	20	16288	25212
	25	17051	27940
Jacobi	1	378702	28897
Chebyshev	1	516600	49582

Table 3: Performance metrics across different methods. The relaxation factor of GS is 0.8, while Jacobi and Chebyshev are both 0.1. Number of iterations required by different methods to reduce the residual to 1.5.

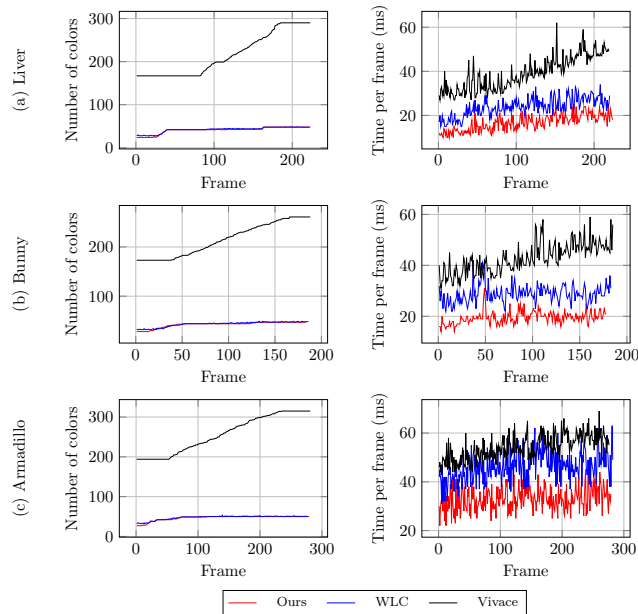


Figure 8: Dynamic changes in color counts (Left) and cutting simulation performance (Right) for different graph coloring algorithms during topological modifications in cutting scenarios (Figure 6). For both the left (color counts) and right (performance) columns, the rows (from top to bottom) respectively display data covering 220, 180, and 280 frames from the onset of cutting.

6. Conclusion and Future Works

In this paper, we presented a GPU-accelerated framework that robustly handles real-time cutting simulations within the XPBD method. We addressed the core challenge of frequent topological changes by proposing a novel SCGC algorithm. Our key innovation, a "Color Preemption" shortcut, enhances state-of-the-art graph coloring techniques to create a highly adaptive parallel solver that thrives in dynamic scenarios where precomputation is infeasible. To support this fast solver, we introduced a constraint clustering preprocess for efficient graph initialization and a robust tetrahedral cutting method to ensure simulation stability. Our integrated system demonstrably outperforms existing parallel XPBD implementations, delivering stable, real-time frame rates during complex interactive cutting operations.

While our framework significantly advances real-time cutting performance, current limitations offer avenues for future research. The current implementation primarily focuses on the cutting operation itself and does not inherently address complex, general collision scenarios. Furthermore, our constraint clustering is currently a static preprocessing step. For highly extensive or complex cutting scenarios where topological changes become extreme, the initial clusters may become suboptimal as the model is increasingly fragmented. Developing a lightweight, asynchronous mechanism to dynamically update the clusters during simulation could further enhance performance and robustness. Future explorations on these topics will ensure robust real-time performance across an even wider range of dynamic and complex interaction scenarios.

7. Acknowledgement

This work was supported in part by the National Key R&D Program of China (No.2023YFC3604500), the Postdoctoral Fellowship Program of CPSF under grant number GZC-20233375, the Guangxi Science and Technology Major Program (No. GuiKeAA24206017), the Open Project Program of State Key Laboratory of Virtual Reality Technology and Systems, Beihang University (No.VRLAB2025C16), and the Beijing Hospitals Authority Clinical Medicine Development of special funding support, code: YGLX202525.

References

- [APB20] ALABANDI G., POWERS E., BURTSCHER M.: Increasing the parallelism of graph coloring via shortcutting. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2020), pp. 262–275. 2, 3, 5, 7
- [BML*14] BOUAZIZ S., MARTIN S., LIU T., KAVAN L., PAULY M.: Projective dynamics: fusing constraint projections for fast simulation. *ACM Trans. Graph.* 33, 4 (July 2014). URL: <https://doi.org/10.1145/2601097.2601116>, doi:10.1145/2601097.2601116. 1
- [BTM17] BERNDT I., TORCHELSEN R., MACIEL A.: Efficient surgical cutting with position-based dynamics. *IEEE Computer Graphics and Applications* 37, 3 (2017), 24–31. doi:10.1109/MCG.2017.45. 1, 2
- [CHC*23] CHEN Y., HAN Y., CHEN J., MA S., FEDKIW R., TERAN J.: Primal extended position based dynamics for hyperelasticity. In *Proceedings of the 16th ACM SIGGRAPH Conference on Motion, Interaction and Games* (New York, NY, USA, 2023), MIG '23, Association for Computing Machinery. URL: <https://doi.org/10.1145/3623264.3624437>, doi:10.1145/3623264.3624437. 1, 2
- [CHC*24] CHEN Y., HAN Y., CHEN J., ZHANG Z., MCADAMS A., TERAN J.: Position-based nonlinear gauss-seidel for quasistatic hyperelasticity. *ACM Trans. Graph.* 43, 4 (July 2024). URL: <https://doi.org/10.1145/3658154>, doi:10.1145/3658154. 1
- [CLF*17] CHEN X., LI P., FANG J., TANG T., WANG Z., YANG C.: Efficient and high-quality sparse graph coloring on gpus. *Concurrency and Computation: Practice and Experience* 29, 10 (2017), e4064. e4064 cpe.4064. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4064>, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4064, doi:https://doi.org/10.1002/cpe.4064. 5, 6, 7, 10
- [CLYY24] CHEN A. H., LIU Z., YANG Y., YUKSEL C.: Vertex block descent. *ACM Trans. Graph.* 43, 4 (July 2024). URL: <https://doi.org/10.1145/3658179>, doi:10.1145/3658179. 5
- [FTP16] FRATARCANGELI M., TIBALDO V., PELLACINI F.: Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph.* 35, 6 (Dec. 2016). URL: <https://doi.org/10.1145/2980179.2982437>, doi:10.1145/2980179.2982437. 1, 2, 5, 6, 7, 10
- [GHF*07] GOLDENTHAL R., HARMON D., FATTAL R., BERCOVIER M., GRINSPUN E.: Efficient simulation of inextensible cloth. *ACM Trans. Graph.* 26, 3 (July 2007), 49–es. URL: <https://doi.org/10.1145/1276377.1276438>, doi:10.1145/1276377.1276438. 2
- [HFG*18] HU Y., FANG Y., GE Z., QU Z., ZHU Y., PRADHANA A., JIANG C.: A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Trans. Graph.* 37, 4 (July 2018). URL: <https://doi.org/10.1145/3197517.3201293>, doi:10.1145/3197517.3201293. 1
- [HMN*21] HEIDEN E., MACKLIN M., NARANG Y., FOX D., GARG A., RAMOS F.: Disect: A differentiable simulation engine for autonomous robotic cutting. *arXiv preprint arXiv:2105.12244* (2021). 1
- [JDP*25] JIA S., DONG Q., PAN Z., YU X., XIU W., ZHANG J.: Accelerate cutting tasks in real-time interactive cutting simulation of deformable objects. *IEEE Computer Graphics and Applications* (2025), 1–14. doi:10.1109/MCG.2025.3538985. 1
- [JP93] JONES M. T., PLASSMANN P. E.: A parallel graph coloring heuristic. *SIAM Journal on Scientific Computing* 14, 3 (1993), 654–669. URL: <https://doi.org/10.1137/0914041>, arXiv:https://doi.org/10.1137/0914041, doi:10.1137/0914041. 3
- [JPW*17] JIA S.-Y., PAN Z.-K., WANG G.-D., ZHANG W.-Z., YU X.-K.: Stable real-time surgical cutting simulation of deformable objects embedded with arbitrary triangular meshes. *Journal of Computer Science and Technology* 32 (2017), 1198–1213. 2
- [KPA*22] KAMARIANAKIS M., PROTOPSALTIS A., ANGELIS D., TAMIOLAKIS M., PAPAGIANNAKIS G.: Progressive tearing and cutting of soft-bodies in high-performance virtual reality. *arXiv preprint arXiv:2209.08531* (2022). 2
- [MAK24] MERCIER-AUBIN A., KRY P. G.: A multi-layer solver for xpb. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Goslar, DEU, 2024), SCA '24, Eurographics Association, p. 1–11. URL: <https://doi.org/10.1111/cg.15186>, doi:10.1111/cg.15186. 1, 2
- [MCMJ17] MÜLLER M., CHENTANEZ N., MACKLIN M., JESCHKE S.: Long range constraints for rigid body simulations. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (New York, NY, USA, 2017), SCA '17, Association for Computing Machinery. URL: <https://doi.org/10.1145/3099564.3099574>, doi:10.1145/3099564.3099574. 2
- [MM21] MACKLIN M., MÜLLER M.: A constraint-based formulation of stable neo-hookean materials. MIG '21, Association for Computing Machinery. URL: <https://doi.org/10.1145/3487983.3488289>, doi:10.1145/3487983.3488289. 2
- [MMC16] MACKLIN M., MÜLLER M., CHENTANEZ N.: Xpb: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games* (New York, NY, USA, 2016), MIG '16, Association for Computing Machinery, p. 49–54. URL: <https://doi.org/10.1145/2994258.2994272>, doi:10.1145/2994258.2994272. 1, 2
- [MMCK14] MACKLIN M., MÜLLER M., CHENTANEZ N., KIM T.-Y.: Unified particle physics for real-time applications. *ACM Trans. Graph.* 33, 4 (July 2014). URL: <https://doi.org/10.1145/2601097.2601152>, doi:10.1145/2601097.2601152. 2
- [Mü08] MÜLLER M.: Hierarchical position based dynamics. vol. 8, pp. 1–10. doi:10.2312/PE/vriphys/vriphys08/001–010. 2
- [PP05] PEREIRA F. M. Q. A., PALSBERG J.: Register allocation via coloring of chordal graphs. In *Proceedings of the Third Asian Conference on Programming Languages and Systems* (Berlin, Heidelberg, 2005), APLAS'05, Springer-Verlag, p. 315–329. URL: https://doi.org/10.1007/11575467_21, doi:10.1007/11575467_21. 5, 6, 10
- [PZY*20] PAN J., ZHANG L., YU P., SHEN Y., WANG H., HAO H., QIN H.: Real-time vr simulation of laparoscopic cholecystectomy based on parallel position-based dynamics in gpu. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)* (2020), pp. 548–556. doi:10.1109/VR46266.2020.00076. 1
- [SGK18] SMITH B., GOES F. D., KIM T.: Stable neo-hookean flesh simulation. *ACM Trans. Graph.* 37, 2 (Mar. 2018). URL: <https://doi.org/10.1145/3180491>, doi:10.1145/3180491. 2
- [TTKA23] TON-THAT Q.-M., KRY P. G., ANDREWS S.: Parallel block neo-hookean xpb using graph clustering. *Computers & Graphics* 110 (2023), 1–10. URL: <https://www.sciencedirect.com/science/article/pii/S009784932200187X>, doi:https://doi.org/10.1016/j.cag.2022.10.009. 1, 2, 4, 5, 6, 10
- [YZWP24] YU P., ZHAO Z., WANG R., PAN J.: Real-time soft body dissection simulation with parallelized graph-based shape matching on gpu. *Computer Methods and Programs in Biomedicine* 250 (2024), 108171. URL: <https://www.sciencedirect.com/science/article/pii/S0169260724001676>, doi:https://doi.org/10.1016/j.cmpb.2024.108171. 1, 2

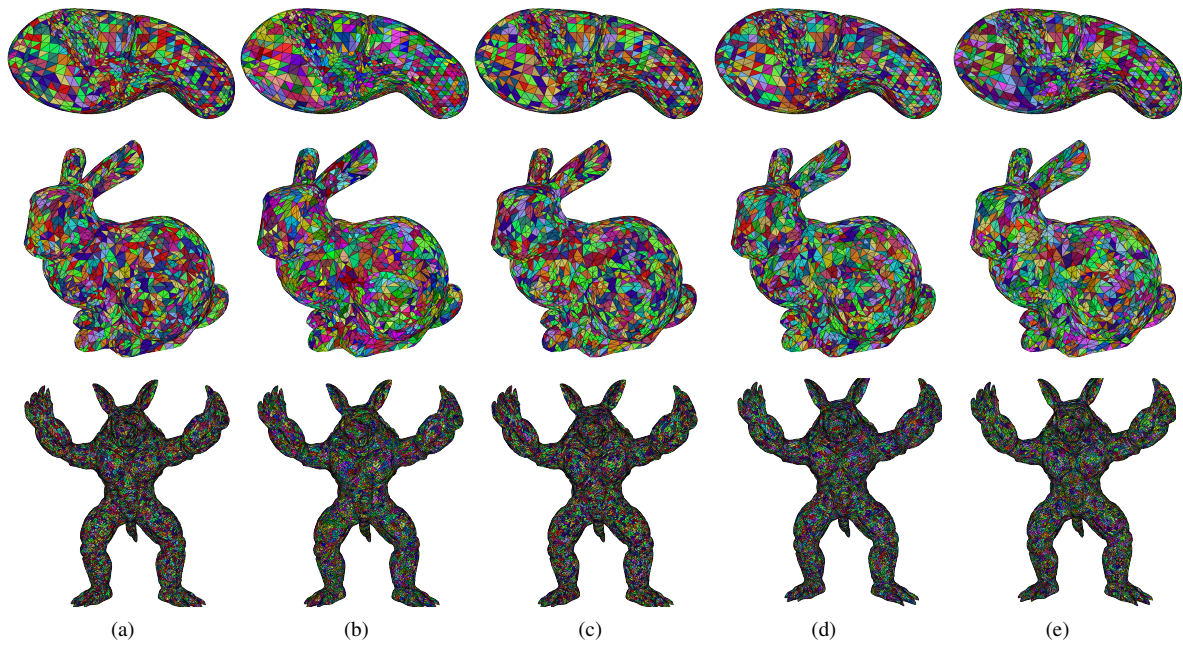
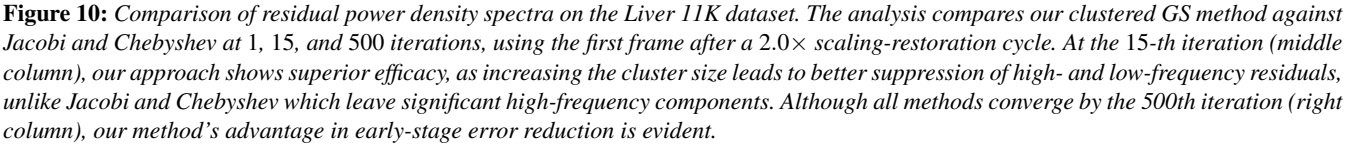


Figure 9: Comparison of constraint graph coloring across different models. Rows (top to bottom) display the coloring results for the Liver (11367 tets, 2823 nodes, 1177 clusters), Bunny (19379 tets, 4670 nodes, 2077 clusters), and Armadillo (121190 tets, 27951 nodes, 12807 clusters) models, respectively. Each column presents a different method with its corresponding color counts (Liver/Bunny/Armadillo): (a) Our method: 24/28/33 colors. (b) Vivace [FTP16]: 167/173/182 colors. (c) WLC [CLF* 17]: 29/34/36 colors. (d) MCS [PP05]: 25/29/33 colors. (e) Greedy [TTKA23]: 39/20/47 colors.



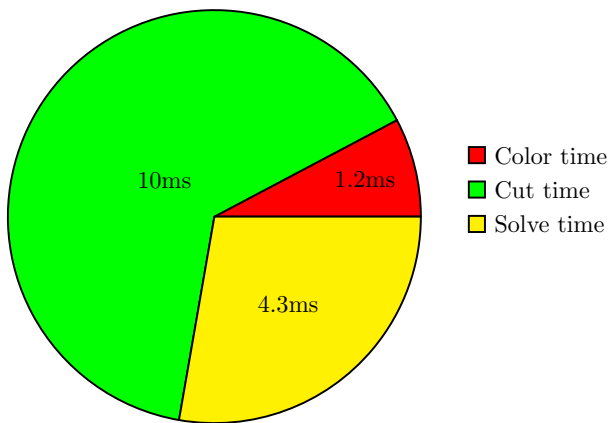


Figure 11: Time distribution within a single frame during the cutting process in liver model (top row in Figure 6)