



# Determining Optimal Print Orientation Using GPU-Accelerated Convex Hull Analysis

Charles Wade

chwa4670@colorado.edu  
University of Colorado Boulder  
Boulder, Colorado, USA

Michael Borish

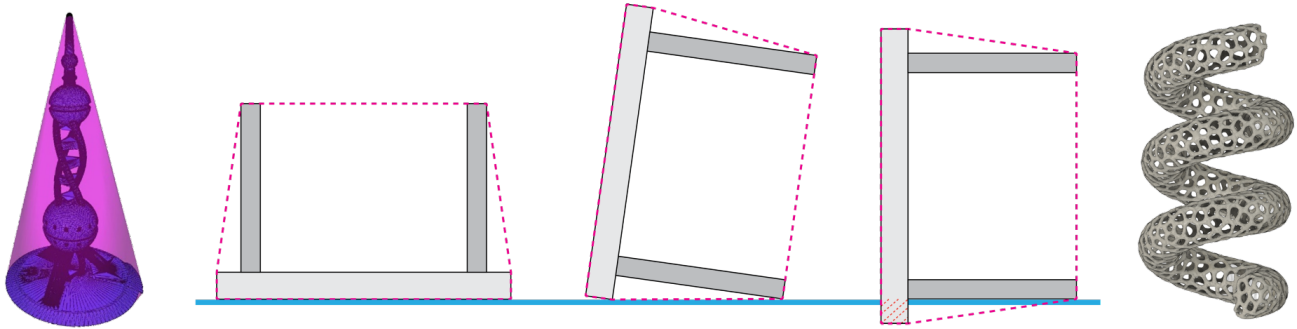
borishmc@ornl.gov  
Oak Ridge National Laboratory  
Oak Ridge, Tennessee, USA

Breanne Crockett

breanne.crockett@colorado.edu  
University of Colorado Boulder  
Boulder, Colorado, USA

Robert MacCurdy

maccurdy@colorado.edu  
University of Colorado Boulder  
Boulder, Colorado, USA



## ABSTRACT

In fused filament fabrication (FFF), the orientation of a part within the printer volume can dramatically affect print quality and probability of success. An object's orientation determines how much support structure will be required and the strength of adhesion between the deposited material and the build surface. Selecting a part's orientation is a non-trivial problem that users of FFF slicing software face routinely. Numerous part orientations need to be considered to find the best according to the results of the slicing process. This paper presents a method to automatically determine an optimal printing orientation for FFF that maximizes build-surface adhesion while minimizing the need for support structure. The algorithm considers the slicing angle and a configurable angle for overhang that requires supporting structure. By employing GPU acceleration and convex hull analysis to limit candidate orientations, the algorithm can run in real time as a preprocessing aid to users slicing parts.

## CCS CONCEPTS

• Computing methodologies → Mesh geometry models; • Applied computing → Computer-aided manufacturing; Computer-aided design.

## KEYWORDS

additive manufacturing, optimization, computer-aided design, GPU acceleration, toolpath-planning

## ACM Reference Format:

Charles Wade, Breanne Crockett, Michael Borish, and Robert MacCurdy. 2023. Determining Optimal Print Orientation Using GPU-Accelerated Convex Hull Analysis. In *Symposium on Computational Fabrication (SCF '23)*, October 08–10, 2023, New York City, NY, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3623263.3623360>

## 1 INTRODUCTION

Fused filament fabrication (FFF) is a type of layer-based additive manufacturing that uses extrusion to deposit material. The common workflow with FFF involves the design of a 3D model in CAD software, G-code generation, and construction by a machine. Various factors at each of these phases can affect the quality of the constructed part. When designing a part, the designer must be keenly aware of the constraints and limitations of the FFF machine that will construct their object. Although some factors such as density and layer-height can be configured in the slicing phase, most parameters are dictated by the machine capabilities. Additionally, most CAD and slicing software do not provide feedback on the relative printability of a part for a specific machine. The



This work is licensed under a [Creative Commons Attribution International 4.0 License](https://creativecommons.org/licenses/by/4.0/).

SCF '23, October 08–10, 2023, New York City, NY, USA  
© 2023 Copyright held by the owner/author(s).  
ACM ISBN 979-8-4007-0319-5/23/10.  
<https://doi.org/10.1145/3623263.3623360>

lack of communication between workflow phases of FFF results in uninformed designs that are not efficiently printed. Furthermore, many of the design constraints placed by FFF machines could be optimized using computational methods.

A primary example of FFF design constraints that could be optimized through computational methods is the selection of build orientation. When preparing a part, the designer needs to give special care to the stacking orientation of the printed layers. The layer direction influences mechanical characteristics, build time, surface finish, and material usage. The build orientation must be carefully selected to balance these factors, while also ensuring a high probability of a successful print. In the traditional FFF workflow, a user must either design a part with a specific build orientation in mind or determine the optimal one during the slicing process. Designing parts with a specific build orientation is a technical process that requires advanced knowledge of the machine and fabrication practice being used. Similarly, determining the optimal orientation of an arbitrary model at slice-time requires an experienced practitioner. A sub-optimal build orientation may result in poor adhesion to the build surface and wasted support material. Additionally, the near infinite number of build orientations adds complexity to the selection process. To address this problem, we present a computational tool to select build orientations in FFF.

In this paper, we propose a GPU-accelerated method for determining the optimal build orientation using convex hull analysis. Our algorithm focuses on build adhesion and accurate support calculation and introduces a heuristic to optimize multiple build parameters. We begin with a discussion of our selection criteria. Next, we outline our convex hull analysis and how it reduces the solution space. Then, we outline the specific GPU-based implementation and our heuristic for choosing the best orientation. Examples are presented in the case studies section to demonstrate the capabilities of the algorithm. Finally, we discuss these results and implications of the proposed method.

## 2 RELATED WORK

Numerous methods have been proposed to solve the optimal build orientation problem [Alexander et al. 1998; Canellidis et al. 2009; Pandey et al. 2004; Strano et al. 2013; Taufik and Jain 2013]. Although specific implementations vary, virtually all existing works follow a similar format: target a process type, determine what build characteristics matter for that process, and employ an optimization strategy. Early attempts by Frank et al. to solve the build orientation problem revolve around expert tools used to select a build orientation [Frank and Fadel 1995]. Thompson et al. improves on these expert-methods by using quantitative measurements of 3D model orientations [Thompson and Crawford 1995]. However, this method still requires relative weighting by a user to pick build factors when optimizing.

Perhaps the most popular factor to optimize for when determining optimal build orientation is support structures. Shen et al. propose a novel method to optimally orient parts on a dynamic build platform [Shen et al. 2020]. Their method minimizes support structures using multi-objective particle swarm optimization. When optimizing for traditional static build plates, there is significant variation in how existing solutions calculate the required supports.

Simple methods compute the surface area of overhanging facets to estimate the required supports [Schrantz 2016]. Morgan et al. use a more accurate approach that computes the 3D support volume by constructing prisms between the build surface and overhanging facets [Morgan et al. 2016]. Ezar et al. compute 3D support volume quickly by leveraging the rasterization capabilities of a GPU [Ezair et al. 2015]. Similarly, Paul and Anand use a voxel-based approach to compute support volume quickly and accurately [Paul and Anand 2015]. The quality of the selected orientation is correlated to the accuracy of the support volume calculation. However, the higher accuracy calculations are more computationally complex and time consuming. The trade off between accuracy and run time must be considered when developing an optimization method for selecting build orientation.

Another common optimization criterion is the aliasing that occurs when discretizing objects into layers. The stair-stepping effect caused by slicing into layers can degrade surface finish and tolerances. Jibin proposes an algorithm to optimize build orientation that attempts to minimize aliasing, support volume, and build time [Jibin 2005]. Thrimurthulu et al. outline a method to minimize surface roughness of layered parts based on build orientation [Thrimurthulu et al. 2004]. Their work also focuses on the implications of build orientation on the surface finish of parts constructed with adaptive layer-height slicing.

Mechanical strength is also of concern when picking an object orientation. Depending on compression and shear forces the object will experience in use, certain orientations may exhibit better strength characteristics. Quintana et al. analyze the effect of build orientation on the strength of SLA printed parts and found that part orientation was correlated with the direction of layer-to-layer interfaces [Quintana et al. 2010]. This suggests that parts are best orientated to compress their layers, while minimizing shear and tension. Although mechanical strength can be critical for certain applications, it also requires expert knowledge on how the part will be loaded. We choose not to focus on mechanical build orientation optimization due to the complexity involved with computing external interfaces and unknown material properties.

Most solutions in literature analyze multiple factors to select an optimal build orientation, resulting in numerous optimization schemes. Early attempts such as Gupta et al. choose an optimal orientation by minimizing a weighted cost function [Gupta et al. 1999]. Some modern solutions instead rotate in small increments about X and Y to exhaust the candidate orientation space. Phatak and Pande encode X and Y rotation as chromosomes for a genetic algorithm that optimizes build orientation according to a fitness function [Phatak and Pande 2012]. Canellidis et al. employ an identical chromosomal encoding to Phatak, however, choose to focus on a wider variety of build parameters [Canellidis et al. 2009]. Choosing candidate orientations by rotating in small values about X and Y results in an infinite solution space that can be expensive to search. Although these genetic algorithms can provide reasonable solutions using this encoding, their computational complexity is too high to provide real time feedback for users. While genetic algorithms often result in reasonable solutions, they provide no guarantees on optimality, especially with the infinite search space of X and Y rotation.

Perhaps the most similar work to ours is that of Zwier and Wits [Zwier and Wits 2016]. Their solution employs the convex-hull principal to reduce the number of candidate orientations by placing coincidental hull and mesh faces on the build surface. They also focus on minimizing support structures by selecting the orientation with the smallest ratio of overhang area to build surface area. Zwier's and Wits' approach provides a solution much faster than genetic algorithms but still requires several minutes to arrive at an optimal orientation. In contrast, our proposed method considers all faces on the convex hull, not just the faces that are coincident with the mesh, as potential orientations. Additionally, we implement a more accurate 3D volume calculation for minimizing support structure, compared to Zwier and Wits's 2D area estimation. Additionally, the proposed algorithm aims to maximize surface adhesion by placing greater emphasis on the cross-sectional build surface area of the first layer. Finally, our proposed work is GPU-accelerated so it can find the optimal orientation quicker than many of the aforementioned methods and can be employed in real time during the design and slicing processes.

### 3 METHODS

#### 3.1 Selection Criteria

Our proposed algorithm is interested in the role of build orientation in the FFF process. In particular, the method was developed to orient large objects with complex geometry on the Big Area Additive Manufacturing (BAAM) system [Love and Duty 2015]. The large volume of this system places special emphasis on two objectives: minimization of support volume and maximization of build-surface adhesion [Roschli et al. 2019]. On smaller desktop FFF machines, supports are often used to print objects with significant overhang. These structures can be printed out of the same build material, or a separate water-soluble material; however, supports must always be removed after printing. Although support structures enable greater design flexibility, support structures should be avoided when possible to reduce wasted material usage. Similarly, certain soft desktop materials such as TPU do not perform well with traditional support structure calculations. On BAAM, support structures are infeasible. The large printed objects would require heavy machinery to remove supports and result in a poor surface finish. Likewise, soluble supports would require an enormous water-bath to fully submerge BAAM parts. Consequently, our optimization criteria includes the minimization of support volume.

We found no previous work that optimizes for build surface adhesion, but it is an important determinant of success in a 3D print [Devicharan and Garg 2019]. There are several considerations that can lead to better build surface adhesion. First, the build surface should be maintained evenly at the glass-transition temperature of the material being printed [Rahman et al. 2016; Södergård and Stolt 2002; Wang and Gardner 2018]. This ensures the first layer of the print remains pliable under the shifting thermal and mechanical stresses applied during the printing process. When printing is completed, that part should be cooled evenly to reduce warping. The second consideration for surface adhesion is the material compatibility of the build substrate and build material(s). Various commercial products exist to improve the chemical bonding of the part to the build surface [BuildTak [n. d.]; Gloop! [n. d.]]. Although

these solutions can improve the likelihood a part remains bonded with the build surface, they add additional consumable cost to the printing workflow.

A simpler and cheaper alternative to additives is optimizing the part's geometry to maximize the area in contact with the build surface. Depending on the geometry of the part this could be as simple as orienting the object in such a way to place large faces against the build surface. Many slicing softwares also allow for rafts and brims to be added below or around the first layer geometry to give the part a wider foot-print [Simplify3D 2019]. Similar to supports, these slicer-based additions result in wasted material that must be removed after printing and as such are not feasible on the BAAM system. An increased first-layer area reduces part warping, improves the probability of success, and can be optimized for without an additional cost.

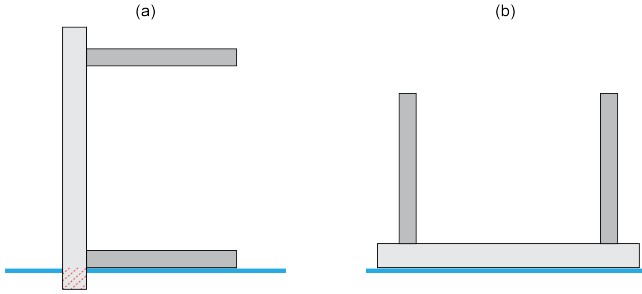
Other factors such as the build orientation's role in anisotropic strength and aliasing were considered. However, we choose not to evaluate anisotropic strength because of the variance in loading criteria necessary to develop a general-purpose tool for all geometries. Similarly, we choose to disregard aliasing as an optimization criterion because of the subjective nature of evaluating surface finish quality and the focus on BAAM.

#### 3.2 Determining Candidate Orientations / Limiting Problem Space

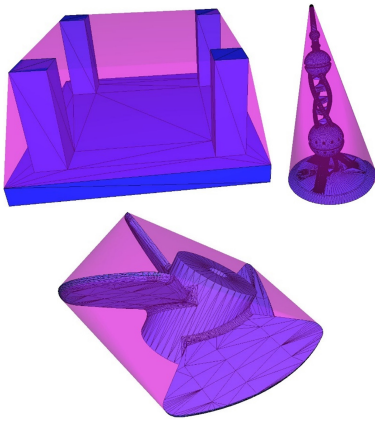
Given an input triangulated mesh, selecting candidate orientations is a non-trivial problem with a seemingly infinite solution space. If the area of the first layer is considered as an objective to maximize, one approach would be to identify candidate orientations by grouping adjacent triangular facets with the same unit normal vector. When the object is oriented such that a grouping of facets is placed against the build surface, i.e. the unit vector is orthogonal to the build surface, a candidate orientation is created. The area of faces in each group is then summed and then maximized. Although this method would reduce the number of candidate orientations to no more than the number of facets in a mesh, it does not account for orientations that would result in an intersection with the build surface. This can be illustrated in two-dimensions where a table is oriented to a face on a leg in figure 1 (a), resulting in a collision with the build surface. In figure 1 (b), the same table can also be oriented so that the tabletop is on the build surface to avoid a collision. This example illustrates that only a subset of external facets are valid orientations.

To identify the subset of external facets that are valid orientations and do not result in collision with the build plate, the convex hull can be used. Intuitively, the convex hull is a closed minimal bounding volume of a mesh. We use the Quickhull algorithm proposed by Barber et Al. and provided by the Computational Geometry Algorithms Library [Barber et al. 1996; Hert and Schirra 2023] to compute the convex hull. Figure 2 shows the convex hull for multiple 3D meshes. We refer to the set of faces that are both the convex hull and the mesh as set A. Set A represents the faces on the input mesh that are printable orientations. The subset of external facets,  $A$ , on mesh  $X$  can be expressed as:

$$A = F_X \cap F_\beta \quad (1)$$



**Figure 1: A 2D table model shown in gray oriented two ways on a build surface. Red section in (a) highlights collision with the build surface.**

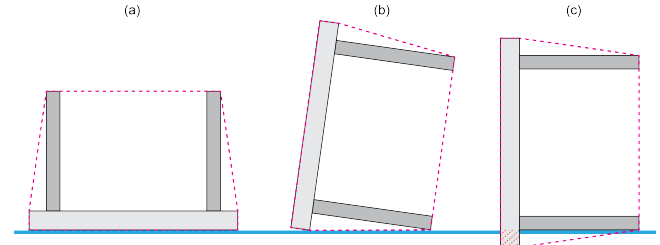


**Figure 2: 3D models shown in blue, and their corresponding convex hulls shown in pink**

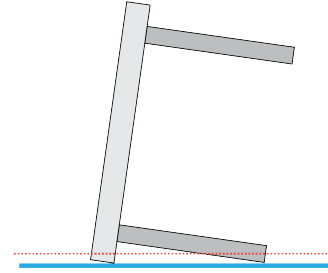
where  $F_X$  is the set of facets in mesh  $X$  and  $F_\beta$  is the set of facets in the convex hull of  $X$ .

Facets that are in the set  $B = F_\beta - F_X$  are facets of the convex hull but not facets of the mesh  $X$ , so it may appear that these are not valid orientations. This incorrect assumption is corroborated when the surface adhesion is calculated as the sum of the area of the mesh's facets in contact with the build surface. In Figure 3(b), the table is oriented to place a facet in set  $B$  from the convex hull on the build plate. This facet results in only two points from the mesh in contact with the bed, resulting in zero build surface area using the facets area calculation. Figure 4 highlights the inadequacy of this calculation by depicting the height of the first layer slice and the true build surface area. When the surface area is calculated as the sum of polygons resulting from first-layer cross-sectioning, it can be observed that orientations from set  $B$  result in positive, non-zero areas and are therefore valid. While the cross-sectional area computation is more complex, the different results can be significant as illustrated in Figure 5. If only facets on the mesh are considered, then the object would be oriented on its long edge. Because the orange cross-sectional plane intersects an area that is larger than any one of the triangular faces shared by the mesh and the convex hull, the optimal orientation of this shape is upside down.

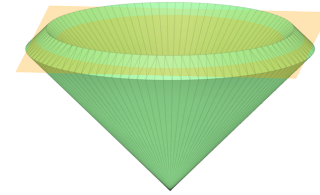
Leveraging the convex hull to determine candidate orientations bounds the solution space to be no larger than the set of all facets in a mesh. The number of potential orientations can be further reduced by combining facets in  $F_\beta$  that have parallel unit normal vectors, as they would result in the same part orientation. Even with large meshes that are comprised entirely of external facets, the number of possible candidate orientations is small when compared with methods that rotate in small increments about  $X$  and  $Y$  axes.



**Figure 3: 2D table example in various orientation show with the convex hull in pink. (a) and (b) show orientations on the convex hull. (c) shows an orientation on an internal facet**



**Figure 4: 2D table with a cross-sectioning plane shown as the red dotted line**



**Figure 5: A curved object highlighting the benefit of using cross-sectional area**

### 3.3 Parallelism and GPU-Acceleration

Although convex hull analysis limits the number of possible candidate orientations, determining corresponding first-layer area and required support volume are expensive operations. With the goal of making the proposed method available as a pre-slicing aid to users, multi-threading and GPU-acceleration were used to calculate support volume and surface area in parallel.



The overall approach is given in algorithm 1. First, the convex hull is computed on the CPU. An initial set of candidate orientations is created by identifying unique faces on the hull. Then, each orientation is assigned a CPU thread for support and area calculation. This phase is completed in parallel since orientations do not affect one another. After all CPU threads have completed, the GPU is used quickly to sort the orientations in ascending first-layer area. With this list, a heuristic selects the optimal orientation for the user.

---

**Algorithm 1** Process overview of the proposed algorithm

---

**Require:** A closed manifold triangle mesh

```

1: hull_faces ← Compute convex hull and return triangular
   faces
2: candidate_orientations ← merge all hull_faces that are
   coplanar
3: Copy mesh to GPU
4: for all candidate_orientations do {in parallel on GPU}
5:   candidate_area ← area(candidate_orientation)
6:   candidate_support_vol ← supportVolume(candidate_orientation)
7: end for
8: gpuSynchronize()
9: Sort candidate_orientations by first layer area on GPU using
   a reduction
10: optimal ← Heuristic(candidate_orientations)
11: return optimal

```

---

**3.3.1 GPU-Accelerated First Layer Area Calculation.** Algorithm 2 describes the first layer area calculation subprocess. This method was implemented using the general-purpose compute abilities provided by NVIDIA CUDA. It starts by creating a plane from a candidate orientation using any point on the triangle and its normal vector. This plane is then translated along the inverse normal vector by a layer height. The layer height is an input configured by the user to match the printing process. Each triangle in the mesh is then assigned to a free GPU thread to compute the intersection between it and the translated plane. If the plane and triangle intersect, the resulting points are saved to global memory. After all GPU triangles have been computed, the intersections are copied from the GPU memory to the host CPU memory. The segments are stitched into a polygon and their areas computed. Finally, the areas of all polygons are summing to determine the total first layer foot-print. This result is saved to the candidate orientation for multi-objective analysis later.

**3.3.2 GPU-Accelerated Support Volume Calculation.** Algorithm 3 outlines how support volume is calculated on the GPU. The process begins by copying the mesh from CPU memory to the GPU. This is done a single time before individual candidate orientations are studied to avoid excessive transfer operations. Next, triangles in the mesh are assigned to free GPU threads. Then, the angle between the triangle's normal vector and the build surface's normal vector,  $\mu$ , is calculated.  $\mu$  is then compared with the user-specified critical support angle  $\theta$ . The critical support angle is determined based on printer and material capabilities and is often around 45°. If  $\mu > \theta$  then the triangle must be supported.

---

**Algorithm 2** Algorithm to compute the first-layer area on a GPU

---

**Require:** A candidate orientation as point & normal, a layer height, and a closed manifold triangle mesh

```

1: plane ← Create an infinite plane that contains point oriented
   along the normal vector
2: Shift plane along it's normal vector by one layer height
3: for all triangles in mesh do {in parallel on GPU}
4:   if plane intersects the triangle then
5:     Save intersection line on GPU indexed by thread ID
6:   end if
7: end for
8: gpuSynchronize()
9: Copy intersections from GPU to CPU memory
10: Stitch intersection lines into polygon(s)
11: total_area ← 0
12: for all polygons do
13:   total_area += area(polygon)
14: end for
15: return total_area

```

---

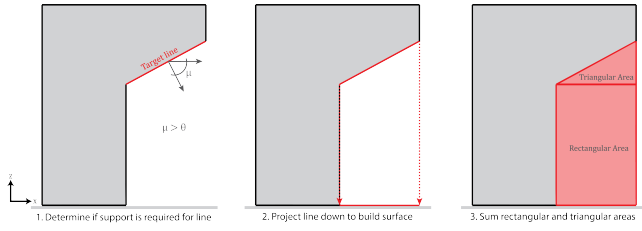
To compute support volume, a fast approximation is employed. Figure 6 shows our method in the simplified 2D case. The method works by projecting the triangle that needs to be supported down onto the 2D build surface. Lines are drawn between each original point and their corresponding projected point. A triangular prism is then formed using the lines and the original and projected triangles. Since the original triangle might not be parallel with the projected triangle, an extra step is required to compute the volume. The prism is split into two volumes along a plane located at the point in the original triangle that is closest to the build surface and parallel to the build surface. The lower volume forms a proper triangular prism whose volume can be computed with equation 2. The upper volume forms a non-regular triangular pyramid whose volume can be computed with equation 3. The two volumes are summed to yield the total required support volume for this triangle. After all triangles have been analyzed, a reduction is performed to sum the volume of all triangles in parallel. The result is the total volume of supports that will be required to print the object with respect to a particular build orientation.

$$V_{\text{prism}} = \frac{1}{2} \cdot \text{Area}_{\text{base}} \cdot \text{Height} \quad (2)$$

$$V_{\text{pyramid}} = \frac{1}{3} \cdot \text{Area}_{\text{base}} \cdot \text{Height} \quad (3)$$

### 3.4 Heuristic for an Optimum

The computation phase results in a list of candidate orientations, each with a support volume and build surface area. Ideally, the optimal orientation will both maximize surface area and minimize support volume. In practice, we found that many simple to medium complexity models did have an orientation that met both criteria. However, our proposed algorithm would be incomplete without a method for selecting an orientation when it is impossible to achieve the best of both constraints. These Pareto optimal situations force the designer to pick what matters more to them: support volume or surface area in contact with the build surface. A weighted cost



**Figure 6: Method to compute support volume shown in simplified 2D example. This same method can be expanded into 3-dimensions and instead calculate the volume using a triangular prism instead of a polygon**

---

**Algorithm 3** Algorithm to compute support volume of a mesh on a GPU

---

**Require:** A manifold triangle mesh oriented by a candidate orientation

```

1: Copy mesh to GPU
2: for all triangles in mesh do {in parallel on GPU}
3:   Project triangle downward onto 2D bed plane
4:   support_prism ← form a triangular prism from the original
     and project triangles
5:   support_required_for_triangle ← volume(support_prism)
6:   Save support_required_for_triangle on GPU using
     thread ID as index
7: end for
8: gpuSynchronize()
9: total_support_volume ← Sum(support_required_for_triangles)
  # Parallel GPU Reduction
10: return total_support_volume

```

---

function was considered, but it was found that acceptable weighting factors depended heavily on the scale of the model, i.e. models for large scale or industrial machines needed different weights than models for desktop sized machines. Instead, we developed a heuristic rule to pick a single orientation as the optimum. The heuristic is preconditioned on the fact that an optimum choice will be on the Pareto frontier. Additionally, we prioritize minimizing support volume over maximizing area in contact with the build surface. This reflects our focus on using this algorithm with large-volume industrial 3D-printers where supports are infeasible. It should be noted, priority can alternatively be given to maximizing surface area on the bed as well. The heuristic takes as input all candidate orientations with their corresponding computed support volume and first-layer surface area, and returns a single optimum orientation. The procedure is given in algorithm 4. Intuitively, the heuristic determines a trade-off is acceptable if the increase in surface area is greater than the (undesirable) increase in support volume. In practice, tuning the percent change threshold is critical for identifying the best part orientations. Given a wide variety of engineering grade models for large-scale 3D-printers, we found 5% gave the best results. However, specific geometries may require this threshold to be higher or lower.

---

**Algorithm 4** Heuristic to pick an optimum orientation in a Pareto optimal space

---

**Require:** A list of candidate orientations with their corresponding support volume and first-layer surface area

```

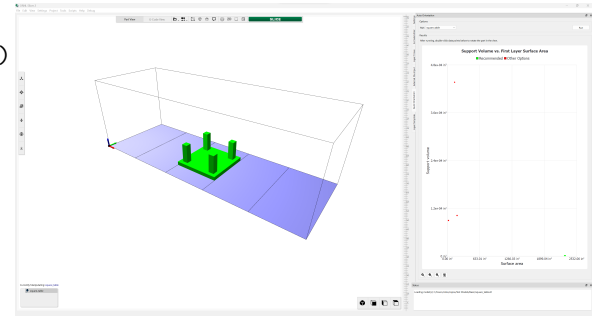
1: threshold ← 5 percent # This is tuned based on scale of the
   machine and geometric features
2: Eliminate all orientations not on the Pareto frontier
3: Sort remaining orientations from least to greatest support vol-
   ume
4: best_orientation ← orientation with least support volume
5: for all remaining orientations do
6:   if Moving from best_orientation to this orientation in-
     creases the surface area by at least the threshold and in-
     creases the support volume by no more than the threshold
     then
7:     best_orientation ← this orientation
8:   end if
9: end for
10: return best_orientation

```

---

### 3.5 Integration into Slicing Software

To make it easier for users to interpret the heuristic results of this algorithm, a simple UI featuring a color-coded dot plot was developed and integrated into ORNL Slicer 2.0 [Roschli et al. [n. d.]]. A simple example of this UI featuring the table from Section 4.1 is shown in Figure 7.



**Figure 7: The proposed method implemented as an interactive tool in the ORNL Slicer 2.0 program.**

As shown in this image, the dot plot showcases the optimum point in green with other possible orientations shown in red. Users can double click on any of the points in the plot and the slicing software will align the part with the orientation represented by the dot. The green dot corresponds to the chosen points highlighted in previous examples and exists on the Pareto Frontier. For this example, the green dot corresponds to the orientation of the table being flipped upside down. As mentioned, in this case, there is no tradeoff for surface or support volume, and so, this orientation is clearly superior. However, this is not always true as demonstrated by other previous examples. In those cases, a chosen point is still highlighted, but a user is free to select a neighboring orientation if they feel it better serves the construction probability. The code for the proposed method, along with instructions to compile it, are

available as part of the ORNL Slicer 2.0 open source repository [Rochli et al. [n. d.]].

## 4 CASE STUDIES

We conduct a case study on three illustrative examples – a table, a propeller, and intersecting rings. These models were chosen to demonstrate the core of the proposed algorithm, highlight the tradeoffs in selecting an optimum, and the benefits of limiting candidate orientations. For each example, we include a render of the input orientation with the best and worst orientations as found by our algorithm. In each render, the build plate is depicted as a purple plane. Additionally, we provide the algorithms run time for each example as tested on an Intel i7-12850HX CPU and NVIDIA RTX 2060 GPU. Finally, we compare all candidate orientations for each model with a graph where each axis denotes an optimization criterion and each point a single candidate orientation. Desirable orientations have high surface area and low support volumes, and are located in the lower right of the graph.

### 4.1 Example 1: Table

Figure 8 depicts a model of a table that was oriented using the proposed algorithm. The model is constructed from four legs and a flat top. The best orientation as determined by the algorithm places the table upside-down. Figure 9 shows the heuristic simply picks the minimum support volume and maximum area without any tradeoffs required. Consequently, the algorithm determined the worst orientation would be to stand the table on its legs. This results in very little surface area and the largest support volume. Although this model contains 76 total triangles, there are only 10 unique candidate orientations. The algorithm runs in approximately 0.007 seconds to find the optimal orientation.

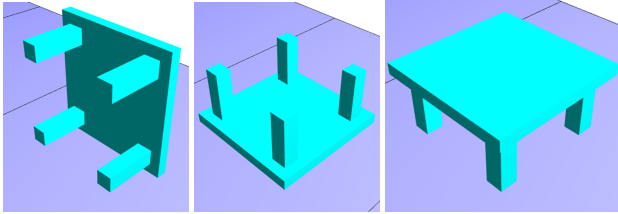


Figure 8: (a) table model initial orientation, (b) best orientation, (c) worst orientation

### 4.2 Example 2: Propeller

Figure 10 shows a model of a propeller oriented with the algorithm. The algorithm determined the best orientation would be to place the flat circular section on the build surface. Figure 10 highlights a tradeoff made by the heuristic. Although orientations with better surface area exist, they would require considerably more support material. This model was oriented in 0.75 seconds by comparing 1,162 candidate orientations.

### 4.3 Example 3: Rings

Figure 12 demonstrates orienting a large model with the proposed algorithm. The input model is a set of intersecting rings that is

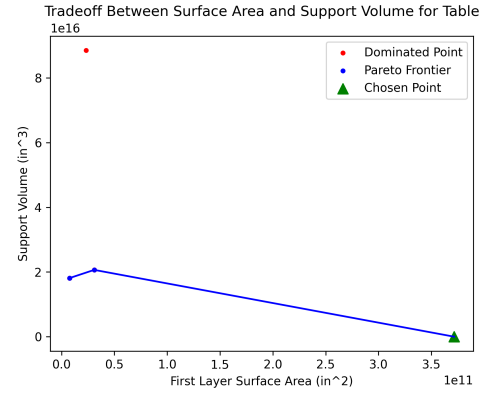


Figure 9: support volume versus first layer surface area for the table model

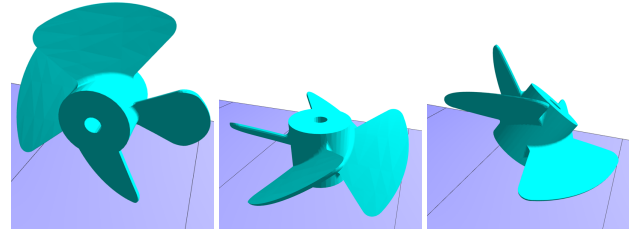


Figure 10: (a) propeller model initial orientation, (b) best orientation, (c) worst orientation

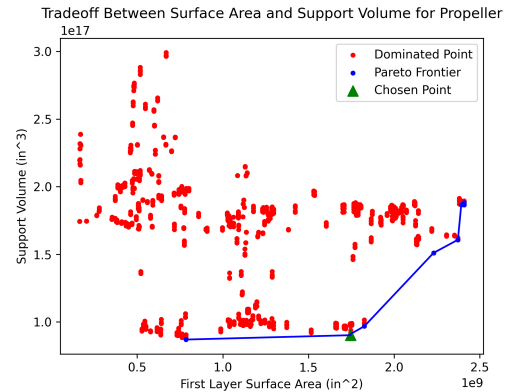


Figure 11: support volume versus first layer surface area for the prop model

constructed from 64,896 triangles. Using convex hull analysis, 8,846 candidate orientations were identified. From figure 13, eight global optimum can be observed. These eight unique orientations that result in the same surface area and support volume are the result of symmetry in the input model. The example took 11.9 seconds to determine the optimal orientation.

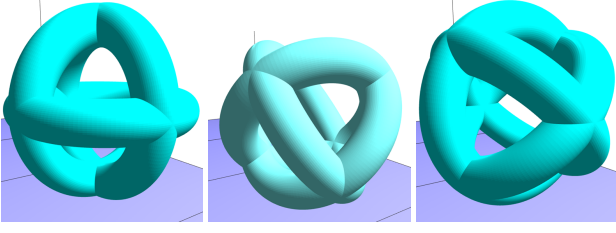


Figure 12: (a) rings model initial orientation, (b) one of the best orientations, (c) worst orientation

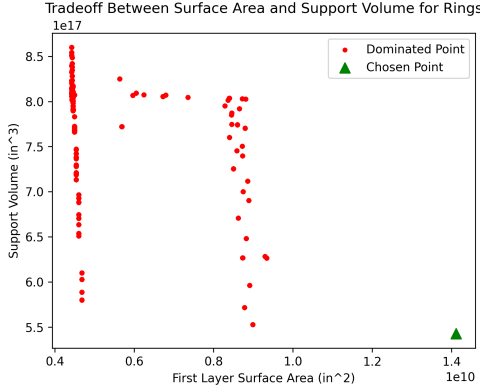


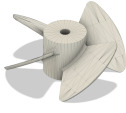



Figure 13: support volume versus first layer surface area for the rings model. Note that multiple symmetric orientations dominate all other options as shown by the green triangle.

## 5 RUNTIME RESULTS

The proposed method was tested on several models to demonstrate the runtime complexity. The first two objects are standard engineering models and represent typical triangular face counts for desktop 3D-printers. The two other example objects are organic lattice structures that contain hundreds of thousands of faces. These objects represent the high triangle count meshes commonly used on large-scale 3D systems such as BAAM. Table 1 shows the results of the algorithm on models by increasing complexity. Figure 14 shows that when compared with other part orientation algorithms Morgan et al. and Zwier & Wits, the proposed method offers a much faster approach. To be included as an interactive component of the 3D printing workflow, an orientation algorithm must run relatively quickly. While previous methods were infeasible for giving designers real-time feedback, our GPU-accelerated method aims to provide results in approximately one minute or less, even on very large input meshes. Previous methods were infeasible for giving designers feedback about their models in real time. This coupled with the direct integration into slicing software offers designers an insightful method to evaluate their part's optimal build orientation. In the worst case, this algorithm behaves similar to Zwier's and Wits' method, running in  $O(n^2)$ , where  $N$  is the number of faces in the mesh. This worst-case occurs when the number of faces in the convex hull is equal to the number of faces in the original mesh

(such as a sphere), thereby offering no complexity reduction. However, the proposed GPU accelerated algorithm demonstrated a large reduction in computation time for models with many faces when compared with other convex hull-based methods such as Zwier and Wits.

Table 1: Comparison of the runtime of the algorithm using increasingly more complex models.

Design	Triangular Faces	Computation Time (s)
	7,286	0.75
	64,896	11.9
	299,256	22.6
	675,528	66.7

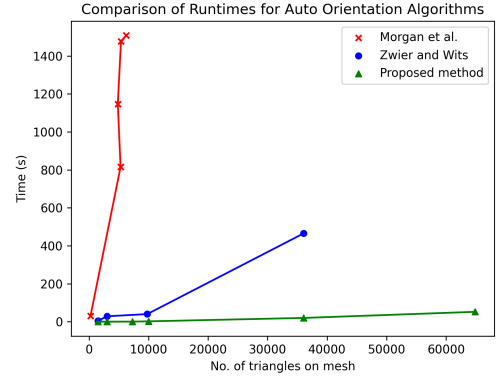


Figure 14: runtimes of other auto-orientation algorithms compared with the proposed method.

## 6 CONCLUSION AND FURTHER WORK

We have presented an efficient algorithm for determining the optimal build orientation of FFF parts. This algorithm can provide users with real time feedback about the tradeoffs between support material usage versus adhesion on the build surface. This better informs designers about the characteristics of their parts and potentially reduces the number of iterations required to achieve an optimal print. More investigation into directly integrating build orientation optimization tools into CAD software is needed to fully



realize the benefits of design feedback. The GPU accelerated nature of the algorithm also allows for analysis to be conducted on complex models. This is particularly valuable for large and industrial scale additive manufacturing where models can reach hundreds of thousands of triangular faces.

The speed of this algorithm makes it viable for integration into larger optimization processes. Further, the methods for computing support volume and surface area in this algorithm were implemented to support arbitrary slicing planes. Future investigation into using the proposed algorithm with angled slicing practices could yield a method to determine the optimal slicing angle and the corresponding orientation. These two factors in tandem could yield a great reduction in support requirements.

Adding more parameters into the cost metric could also yield better results. Factors such as aliasing, topological optimization, and anisotropy could be studied to influence the selection of an optimal orientation. Likewise, an improved support volume calculation method could be employed to better approximate internal structures. Ezar et al. provides a GPU accelerated support volume calculation method that is well suited for the proposed algorithm. Finally, a study comparing the different optimization factors that influence various printing techniques would be critical to understanding what build orientation parameters should be considered. This could yield a solution that can recommend certain printing technologies to designers based on the characteristics of their parts.

## ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Energy Efficiency and Renewable Energy, Office of Advanced Manufacturing, under contract number DE-AC05-00OR22725. This work is described in US patent application, Serial Number 16/842,274 (filed on April 7, 2020.)

## REFERENCES

- Paul Alexander, Seth Allen, and Debasish Dutta. 1998. Part orientation and build cost determination in layered manufacturing. *Computer-Aided Design* 30, 5 (1998), 343–356.
- C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. 1996. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)* 22, 4 (1996), 469–483.
- BuildTak. [n. d.]. 3D printing solutions designed to work and built to last. <https://www.buildtak.com/>
- V Canellidis, J Giannatsis, and V Dedoussis. 2009. Genetic-algorithm-based multi-objective optimization of the build orientation in stereolithography. *The International Journal of Advanced Manufacturing Technology* 45 (2009), 714–730.
- R Devicharan and Raghav Garg. 2019. Optimization of the print quality by controlling the process parameters on 3D printing machine. *3D Printing and Additive Manufacturing Technologies* (2019), 187–194.
- Ben Ezair, Fady Massarwi, and Gershon Elber. 2015. Orientation analysis of 3D objects toward minimal support volume in 3D-printing. *Computers & Graphics* 51 (2015), 117–124.
- Dietmar Frank and Georges Fadel. 1995. Expert system-based selection of the preferred direction of build for rapid prototyping processes. *Journal of Intelligent Manufacturing* 6 (1995), 339–345.
- Gloop! [n. d.]. 3D gloop! <https://www.3dgloop.com/>
- Satyandra K Gupta, Qi Tian, and Lee Weiss. 1999. Finding near-optimal build orientations for shape deposition manufacturing. In *Machining Impossible Shapes: IFIP TC5 WG5. 3 International Conference on Sculptured Surface Machining (SSM98) November 9–11, 1998 Chrysler Technology Center, Michigan, USA*. Springer, 208–216.
- Susan Hert and Stefan Schirra. 2023. 3D Convex Hulls. In *CGAL User and Reference Manual* (5.5.2 ed.). CGAL Editorial Board. <https://doc.cgal.org/5.5.2/Manual/packages.html#PkgConvexHull3>
- Zhao Jibin. 2005. Determination of optimal build orientation based on satisfactory degree theory for RPT. In *Ninth international conference on computer aided design and computer graphics (CAD-CG'05)*. IEEE, 6–pp.
- Lonnie J Love and Chad Duty. 2015. Cincinnati big area additive manufacturing (BAAM). *Oak Ridge, TN* (2015).
- HD Morgan, JA Cherry, S Jonnalagadda, D Ewing, and J Sienz. 2016. Part orientation optimisation for the additive layer manufacture of metal components. *The International Journal of Advanced Manufacturing Technology* 86 (2016), 1679–1687.
- Pulak M Pandey, K Thrimurthulu, and N Venkata Reddy\*. 2004. Optimal part deposition orientation in FDM by using a multicriteria genetic algorithm. *International Journal of Production Research* 42, 19 (2004), 4069–4089.
- Ratnadeep Paul and Sam Anand. 2015. Optimization of layered manufacturing process for reducing form errors with minimal support structures. *Journal of Manufacturing Systems* 36 (2015), 231–243.
- Amar M Phatak and SS Pande. 2012. Optimum part orientation in rapid prototyping using genetic algorithm. *Journal of manufacturing systems* 31, 4 (2012), 395–402.
- Rolando Quintana, Jae-Won Choi, Karina Puebla, and Ryan Wicker. 2010. Effects of build orientation on tensile strength for stereolithography-manufactured ASTM D-638 type I specimens. *The International Journal of Advanced Manufacturing Technology* 46 (2010), 201–215.
- Miftahur Rahman, NR Schott, and Lakshmi Kanta Sadhu. 2016. Glass transition of ABS in 3D printing. In *COMSOL Conference, Boston, MA*.
- Alex Roschli, Michael Borish, Abigail Barnes, Charles Wade, Breanne Crockett, Liam White, and Cameron Adkins. [n. d.]. ORNL Slicer 2 - Open Source Copyright. Computer Software. <https://github.com/ORNLSlicer/Slicer-2.Web>.
- Alex Roschli, KT Gaul, AM Boulger, BK Post, PC Chesser, LJ Love, F Blue, and M Borish. 2019. Designing for big area additive manufacturing. *Addit Manuf* 25: 275–285.
- Christoph Schranz. 2016. Tweaker-auto rotation module for FDM 3D printing. *Salzburg Research Salzburg, Austria* (2016).
- Hongyao Shen, Xiaoxiang Ye, Guanhua Xu, Linchu Zhang, Jun Qian, and Jianzhong Fu. 2020. 3D printing build orientation optimization for flexible support platform. *Rapid Prototyping Journal* 26, 1 (2020), 59–72.
- Simplify3D. 2019. Rafts, skirts and brims! <https://www.simplify3d.com/resources/articles/rafts-skirts-and-brims/>
- Anders Södergård and Mikael Stolt. 2002. Properties of lactic acid based polymers and their correlation with composition. *Progress in polymer science* 27, 6 (2002), 1123–1163.
- Giorgio Strano, L Hao, RM Everson, and KE Evans. 2013. A new approach to the design and optimisation of support structures in additive manufacturing. *The International Journal of Advanced Manufacturing Technology* 66 (2013), 1247–1254.
- Mohammad Taufik and Prashant K Jain. 2013. Role of build orientation in layered manufacturing: a review. *International Journal of Manufacturing Technology and Management* 27, 1-3 (2013), 47–73.
- David C Thompson and Richard H Crawford. 1995. Optimizing part quality with orientation. In *1995 International Solid Freeform Fabrication Symposium*.
- KPPM Thrimurthulu, Pulak M Pandey, and N Venkata Reddy. 2004. Optimum part deposition orientation in fused deposition modeling. *International Journal of Machine Tools and Manufacture* 44, 6 (2004), 585–594.
- Lu Wang and Douglas J Gardner. 2018. Contribution of printing parameters to the interfacial strength of polylactic acid (PLA) in material extrusion additive manufacturing. *Progress in Additive Manufacturing* 3 (2018), 165–171.
- Marijn P Zwier and Wessel W Wits. 2016. Design for additive manufacturing: Automated build orientation selection and optimization. *Procedia Cirp* 55 (2016), 128–133.