www.cad-journal.net
Computer-Aided Design AND APPLICATIONS

Voxel-Based Collision Avoidance for 5-Axis Additive Manufacturing

Alexandre Desgrees du Lou[1] , Yeganeh Bahoo[2] , Robert Hedrick[3] , Austin Vuong[2]

[1]IMT Mines Albi, alexandre.desgrees_du_lou@mines-albi.fr
[2]Toronto Metropolitan University, {bahoo,austin.vuong}@torontomu.ca
[3]CAMufacturing Solutions Inc., bob.hedrick@camufacturing.com

Corresponding author: Alexandre Desgrees du Lou, alexandre.desgrees_du_lou@mines-albi.fr

Abstract. This paper introduces two innovative algorithms designed to convert an existing toolpath into a collision-free trajectory, addressing one of the critical challenges in 5-axis Additive Manufacturing (AM) the risk of collisions during the complex printing process. The first algorithm adapts a voxel-based strategy, drawing on the methodology of Nishat et al. to facilitate enhanced collision detection and avoidance. The second algorithm combines the Gilbert-Johnson-Keerthi (GJK) method, the Expanding Polytope Algorithm (EPA), a bounding box strategy, and a decomposition into convex elements. A unique aspect of this approach is the optimization of the deposition head's orientation through a weighted normalization vector and the application of Lami's theorem for memory retention of previous collision extraction vectors. Additionally, we explore the dynamic updating of the voxelized model during printing and the implementation of Bezier curves to smooth the orientation transitions of the deposition head. These contributions not only demonstrate a significant improvement in managing collision risks in 5-axis AM but also highlight the potential for more reliable and efficient manufacturing processes. Experimental results underscore the effectiveness of our algorithms, showcasing their capability to ensure collision-free toolpaths.

Keywords: Voxelization, Collision Detection and Avoidance, GJK, EPA, Optimization, Additive Manufacturing
DOI: https://doi.org/10.14733/cadaps.2025.845-866

1 INTRODUCTION

Additive Manufacturing (AM) stands as a revolutionary advancement in manufacturing technologies. With this advancement arises a fresh challenge: the potential for collisions during the print process. This research narrows its focus on this collision management challenge, aiming to harness the power of voxel-based solutions to bolster the efficiency and reliability of this nascent technology [5].
AM has become instrumental in modern production, known for its flexibility in creating complex, customized

designs directly from digital models. While traditional AM methods construct objects layer-by-layer, thereby limiting the geometrical freedom and often requiring additional support structures, 5-axis AM transcends these limitations. By introducing two additional rotational axes to the printing head movement, 5-axis AM allows for more intricate product designs, reduces the need for support structures, and can potentially improve the mechanical properties of printed objects by minimizing the anisotropic behavior often seen in traditional AM materials. However, these advancements are not without new challenges.
The increased complexity of 5-axis movements means there is a higher risk of collision between the printing apparatus and the object being printed. This risk is particularly pronounced when producing complex or nested geometries, requiring advanced strategies for collision detection and avoidance.
Our approach to manage collisions unfolds in two primary stages: the first involves scrutinizing for possible collisions between the deposition head and the 3D printed model. If a collision is detected, the second stage kicks in, where the goal is to ascertain the best collision-free deposition head orientation scenario. As the AM process adds material to the work volume, modelling the material deposited by the toolpath is essential.

1.1 Our Contribution

In this paper, we present two algorithms for modifying an existing toolpath into a collision-free toolpath.

The toolpath is determined by a sequence of coordinates in $\mathbb{R}^3$, symbolized by the set P. Each point along this path has a distinct deposition head orientation, represented by a normalized vector in the same space. The objective is to adjust these vectors as needed to prevent any collisions, thereby ensuring a collision-free toolpath.

1. The first algorithm Sec. 3 reutilizes the methodology of Nishat et al. [16], as discussed in Section 2, applied to a voxel-based.
2. The second algorithm Sec. 4 combines several techniques: the Gilbert-Johnson-Keerthi (GJK) [9], Expanding Polytope Algorithm (EPA) [3], bounding box strategy [10], and decomposition into convex elements. The major contribution lies in the use of a normalized vector on weights to obtain the best extraction direction, as well as the use of Lami's theorem to design a vector that retains memory of previous collision extraction vectors as shown in Equation 2 and Figure 11.

The last contribution is in updating the voxel printed model during the printing process and smoothing the orientation of the vector through the use of Bezier curves Sec. 5.

The paper is structured as follows. In the subsequent subsection 1.2, we describe our input data and the testing environment. Section 2 discusses "related work" on five-axis printing, particularly their approaches to collision detection and avoidance. We delve into the advantages of voxel-based collision detection and printed model updating over time and highlight the significance of smoothing the deposition head orientation. In Section 3, we describe our initial approach, extending the results of Nishat et al. [16] to a voxel framework. Section 4 presents our new collision detection and avoidance algorithm that combines a bounding box strategy [10], which promises quicker pinpointing of the collision zone, with the Gilbert-Johnson-Keerthi (GJK) approach [9], renowned for its efficiency in testing collisions for convex entities. The subsequent phase of this algorithm focuses on identifying the most secure and efficient method to generate a collision-free deposition head axis vector. Here, the Expanding Polytope Algorithm (EPA) [3], a derivative of GJK, is employed to provide a solution for the closest collision-free deposition head axis vector to the original one. Finally, in Section 5, we describe the method used for printed model updating and smoothing the deposition head orientation.

1.2 Input data details and the testing environment

The initial framework for our study was defined by several key components. The deposition head and the CAD model are defined by their respective STL files, called the deposition head and mold respectively throughout this paper. Along with these, we considered constraints such as the height h,the contact tip to workpiece distance between the deposition head and path point as measured along the deposition head axis, the maximum angular variations of θ between consecutive points on the toolpath, and an overall maximum angular variation of Θ for the deposition head.

Our testing system specifications are as follows: **Chip:** Apple M1, **Memory:** 8GB, and **Storage:** 256GB Flash Storage. Figure 1a displays the 3D printed model "Recreated Mold", and Figure 1b illustrates the 3D printed model with red points corresponding to the AM toolpath.

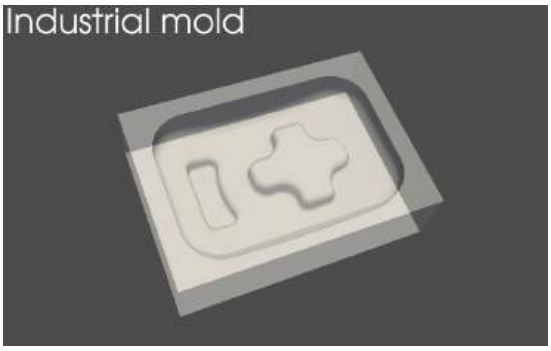
The second toolpath was provided by the industrial partner: CAMufacturing Solutions Inc. (CAMu), and the toolpath is carried out in the industrial mold called CAMu Mold, which can be seen in Figure 1c. The toolpath (dark blue lines) is a .NC file, which provides the coordinates for each printing point, along with the direction of the deposition head, as depicted in Figure 1e. The light blue line represents the direction of the deposition head for specific points, see Figure 1f. For each point of the toolpath, the deposit head is positioned at a point we will call C, which is offset $6\ mm$ along the blue deposition head axis vector. In the event of a collision, the rotation will be performed from point C. The deposition head is represented as an STL file, which was provided by CAMu for the purpose of our study, as depicted in Figure 1d.

The dimensions of the CAMu Mold are $280\ mm$ in x, $200\ mm$ in y, and $80\ mm$ in z dimensions. The Deposition Head measures $58\ mm$ in both x and y dimensions, and $160\ mm$ in z. Finally, the Recreated Mold has dimensions of $143\ mm$ in x, $128\ mm$ in y, and $74\ mm$ in z.

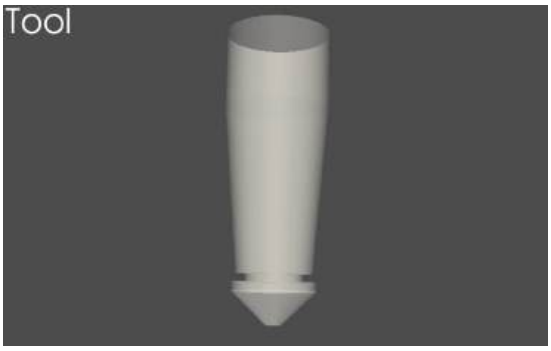
(a) Simple mold cavity STL model

(b) AM toolpath

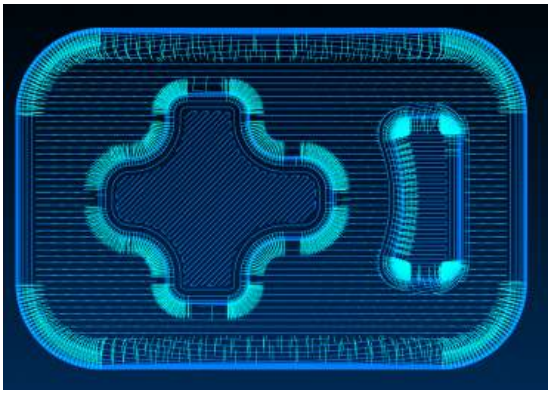
Figure 1: Visualization of Mold Design and Toolpath.



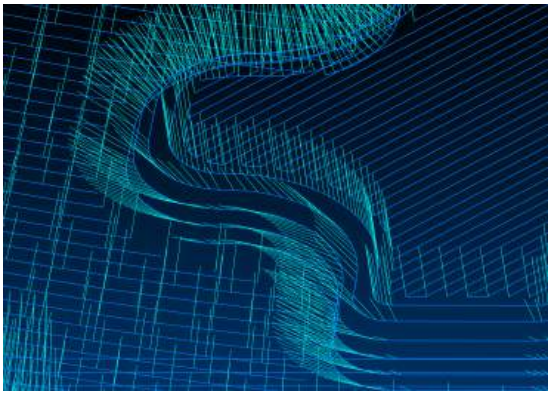
(c) STL mold cavity with convex and concave sections



(d) STL of deposition head from industrial partner



(e) AM toolpath



(f) Deposition head axis (light blue lines)

Figure 1: Visualization of Mold Design and Toolpath.

2 RELATED WORK

Collision detection and avoidance in multi-axis additive manufacturing (AM) have been explored through several innovative approaches. Nishat et al. demonstrate an algorithm for generating collision-free toolpaths in multi-axis AM, prioritizing efficient orientation and trajectory planning to mitigate collision risks [16]. Fang et al. examine layer-by-layer collision-free printing by optimizing setup orientations, aligning with Nishat et al.'s strategy for selecting optimal tool vectors [7]. Plakhotnik et al. focus on collision avoidance by dynamically updating tool vectors during the AM process, introducing penalties for deviations from ideal tool orientations [18]. Additionally, Jiang et al. delve into collision avoidance through effective scheduling in path planning, especially when employing multiple deposition heads in AM, showcasing a strategic approach to manage complex geometries and workspace constraints [13]. These studies collectively address the intricate challenges of geometric complexity and workspace management in additive manufacturing, offering comprehensive strategies for improving the efficiency and reliability of AM processes.

Voxelization stands out in 3D modeling for its distinct advantages over mesh-based representations, especially relevant in the fields of additive manufacturing and 5-axis machining. The method's core advantage stems from its discretization of space into voxels, significantly simplifying collision detection to basic boolean operations [12, 8]. This simplification offers a marked departure from the complex, computationally intensive mesh-based collision detection methods, such as those utilized by Nishat et al., including Möller's triangle-

triangle intersection algorithm [15] and its refinements by Devillers and Guigue [6]. While these mesh-based approaches are known for their precision, they require substantial computational efforts to perform detailed geometric analyses. The voxel-based approach effectively circumvents these complexities by reducing the detection of collisions to the assessment of voxel occupancy states, a process inherently suited for efficient parallel processing. This characteristic of voxel models aligns perfectly with the capabilities of modern Graphics Processing Units (GPUs), which excel at handling large volumes of parallelizable tasks. Consequently, voxelization not only streamlines the collision detection process but also significantly benefits from the rapid advancements in GPU technologies, facilitating real-time collision detection and response mechanisms that are critical for dynamic manufacturing environments. A notable illustration of this efficiency is provided by Hermann et al. [11], who developed a framework for collision detection optimized for GPU architectures. This approach not only enhances the efficiency of collision detection but also allows for seamless integration of real-time sensor data and increased responsiveness in trajectory planning and execution, underscoring the importance of fully exploiting GPU capabilities for parallel processing of collision data. Importantly in additive manufacturing progresses the printed model evolves, inherently increasing the probability of collisions. Voxel models adeptly address this challenge by enabling rapid updates to the in-progress printed model. By adjusting voxel densities to reflect newly formed parts of the object, voxelization ensures that collision detection remains accurate and responsive throughout the printing process. This adaptability is particularly valuable in dynamic manufacturing environments where modifications to the print model can occur in real-time.

In our exploration of advanced manufacturing techniques, we spotlight the approach by Nishat et al. [16] due to its innovative contributions to collision detection and avoidance in additive manufacturing. Their method stands as a pivotal reference for its strategic blend of geometric modeling and algorithmic efficiency. Our focus on this work stems from its potential as a foundation for further advancements. By dissecting and understanding its intricacies, we set the stage for introducing significant enhancements that aim to overcome its limitations, specifically around computational efficiency and the adaptability of collision detection algorithms. In the approach of Nishat et al., the object surfaces are modelled using a triangular mesh. In this scenario, a collision is equivalent to a triangle intersection between two meshed objects. A simplified convex hull of the deposition head is used for collision detection with 3D printed model, which increases the algorithm's performance by reducing the number of intersection tests to be performed. Their process for calculating a collision-free trajectory consists of three steps:

1 - Generation of a variety of deposition head vectors to encompass all deposition head tilting possibilities represented by the set S in spherical coordinates :

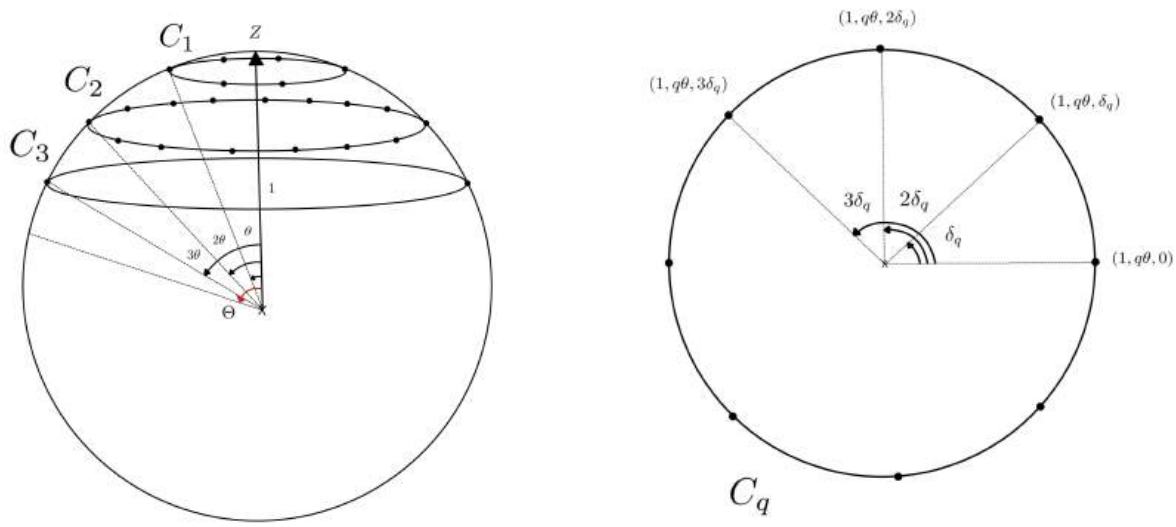
$$S = \left\{ (1, q\theta, h\delta_q) \,\middle|\, \forall q \in [\![0, \left\lfloor \frac{\Theta}{\theta} \right\rfloor]\!],\ \delta_q = \arccos\left(\frac{\cos(\theta) - \cos^2(q\theta)}{\sin^2(q\theta)} \right),\ \forall h \in [\![0, \left\lfloor \frac{2\pi}{\delta_q} \right\rfloor]\!] \right\}$$

As can be seen in Figure 2, set S is actually the disjoint union of the sets C_q for q in $[\![0, \lfloor \Theta/\theta \rfloor]\!]$, where C_q is the set of $(1, q\theta, h\delta_q)$ for h in $[\![0, \lfloor 2\pi/\delta_q \rfloor]\!]$. Thus each deposition head vector is defined as the difference between a point in set S and the origin, which is represented in the Figure 2 by a cross "x". For a better understanding of the origin of δ_q, please refer [16] Section 3.1.

2 - For each point in set P, they conduct a collision test for all vectors in set S, leading to the construction of a graph. In this graph, points from P denote levels and the collision-avoiding vectors from S serve as vertices. All vertices at one level are linked to those on the next, with weights assigned based on the angular variation of the deposition head. This variation is measured between the vector associated with a vertex at the current level and that at the next one.

3 - Calculation a collision-free path using Dijkstra's algorithm on the configuration graph.

Therefore, the performance of their algorithm depends on several key factors: the efficiency of the chosen collision detection algorithm, the size of the configuration graph, which is influenced by the number of points



(a) Partial visualization of set S

(b) Subset C_q of set S

Figure 2: Spatial representations of set S and its subset C_q

on the toolpath and the number of configurations (deposition head vectors) considered for each point. This dependence highlights the inherent challenges in managing a large number of configurations and toolpath points, which can significantly affect the computation time and complexity of the algorithm, thus representing significant drawbacks of this method.

This study demonstrates that transitioning from mesh-based object representations to voxel-based ones marks a notable improvement, facilitating the parallelization of collision tests and thereby reducing the computational complexity of the process.

Although this method provides accurate collision detection, testing all possible orientations of the print head remains computationally expensive. The Gilbert, Johnson, and Keerthi (GJK) algorithm [9] accelerates collision detection for convex shapes. This limitation is partially overcome by techniques such as convex decomposition `CoACD` [22], though this can significantly increase computation time depending on the desired accuracy of decomposition. Another limitation lies in the fact that GJK provides a translational solution to collisions where a rotational solution is necessary. A personal contribution is then made to address this issue.

Plakhotnik et al. highlight the importance of smoothing the deposition head orientation to enhance the quality and efficiency of production [18]. Smoothing helps to minimize equipment wear and reduce irregularities on produced surfaces, thus ensuring a high-quality finish. However, the implementation of smoothing techniques, such as Bezier curves, must be carefully balanced to maintain production efficiency while avoiding collisions.

Utilizing the advancements in path planning, Tharwat et al. demonstrate an intelligent path planning model that leverages Bezier curves optimized by a Chaotic Particle Swarm Optimization algorithm [21]. This model effectively navigates a mobile robot through unknown environments by dynamically adjusting paths to avoid obstacles. The utilization of Bezier curves in this context underscores their potential for generating smooth, continuous trajectories, critical for both robotic navigation and advanced manufacturing processes. This approach, analogous to what shown in [1], showcases the versatility of Bezier curves in optimizing trajectories for complex tasks, including additive manufacturing, where precise control over deposition head movement and collision avoidance is paramount.

3 VOXEL-BASED ADAPTATION FOR THE METHOD OF NISHAT ET AL.

In our approach, we substitute triangular meshes with voxels. However, the overarching objective remains to find a collision-free toolpath. Following the three steps of the Nishat et al. approach, we generate the set S in the same manner. The primary difference is that when we modify the orientation of the deposition head, it is composed of voxels rather than meshes. For the second step, the collision testing process is different. It no longer involves a collision process between triangles from different meshes but instead, it involves the creation of a voxel grid, where each voxel has a density value of 1 if it is part of the mold, and 0 otherwise. The idea is to test, for each position in P and for each orientation in S of the deposition head, whether any of the deposition head's voxels have a value of 1. To do this, we tested two Python libraries:
`Open3D` [23] and `PyVista` [20], both offering a method for voxelizing an `STL` object. `PyVista`'s voxelization encompasses the entire volume of the object (31.23 seconds for the deposition head for a voxel size of 0.5 millimeter), which requires substantially more time than `Open3D`'s surface-based voxelization (0.12 seconds for the deposition head for a voxel size of 0.5 millimeter). However, a key advantage of `PyVista` is its ability to verify object closure, thus ensuring a watertight object. Conversely, with `Open3D`, not the entire surface is voxelized, potentially leading to false positives during collision tests. For the upcoming tasks, we have chosen to use the `Open3D` library due to its superior performance in terms of execution time for obtaining the voxelized surface of objects, which is notably faster compared to `PyVista`.
The voxel models for the deposition head and molds are structured as follows: each model is a set of points (x, y, z), where each voxel is identified by one vertex, as explained in Figure 3.

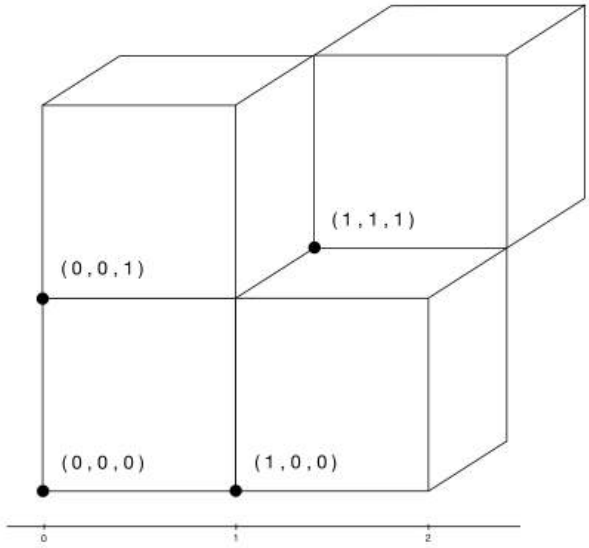
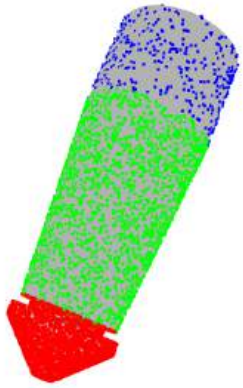


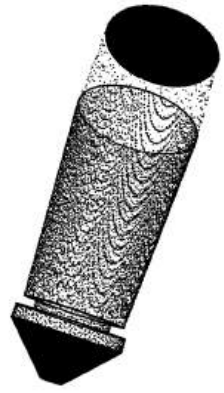
Figure 3: Voxel representation

For the sake of readability, the coordinates will be dilated by a factor corresponding to the voxel size of 0.5 mm.
We assume that the deposition head's lower part is at a higher risk of collisions compared to its upper part. To address this, we enhance voxel density on the surface of areas prone to risks. Specifically, we divide the deposition head into three sections: the lower section receives a substantially increased voxel density, while the middle and upper sections retain a lower density. This strategy effectively reduces the total number of voxels

for analysis, cutting down the execution time of the program. However, it also slightly increases the chance of false positives. Finally, step three remains the same as that of Nishat et al., utilizing Dijkstra's Algorithm to find the shortest path that adheres to the machine's printing constraints.



(a) Deposition head derived from risk-zone voxelization, distributed as 58%, 35%, and 7% of 13,820 voxels. Execution Time 0.024s.



(b) Voxel model of 35,905 voxels obtained using the `Open3D` voxelization function. Execution Time 0.12s.

Figure 4: Comparative visualization of tools post-voxelization for voxel size of 0.5 mm.

3.1 Rotation Mechanism

After defining S, the set that represents the various orientation vectors of the deposition head to encompass all possible tilting angles, as illustrated in Figure 2. It is necessary to determine each deposition head for every deposition head vector. For efficient computation of the deposition head's orientation, we employ Rodrigues' rotation formula [19], as shown in Equation 1. Rodrigues' formula provides a powerful method for calculating the rotation matrix $R_{Y'}(\theta)$ for a given axis of rotation $\mathbf{Y}'$ and angle θ. In the formula, I represents the identity matrix, indicating that when the rotation angle θ is zero, the rotation matrix reduces to the identity matrix, implying no rotation. The matrix Q is the skew-symmetric matrix derived from the rotation axis, and θ represents the rotation angle. The terms involving $\sin(\theta)$ and $(1 - \cos(\theta))$ incorporate the sinusoidal variation with the angle of rotation, reflecting the fundamental characteristics of rotational movement in three-dimensional space. The formula efficiently encapsulates the rotation of any vector in three-dimensional space around a specified axis $\mathbf{Y}'$ by an angle θ, making it particularly suitable for calculating the new orientations of the deposition head in additive manufacturing processes.

$$R_{Y'}(q\theta) = I + \sin(q\theta)Q + (1 - \cos(q\theta))Q^2 \tag{1}$$

The process begins with the voxelized deposition head oriented along the z-axis ($\mathbf{Z}$). Let $\mathbf{s} = (1, q\theta, h\delta_q) \in S$, the deposition head vectors. To obtain the voxelized deposition head oriented along $\mathbf{s}$, we need to find the vector $\mathbf{Y}'$ around which it will rotate by $q * \theta$, as shown in Figure. 5:

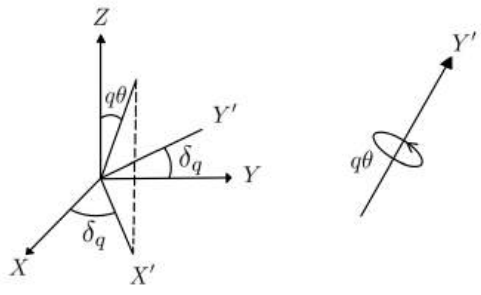


Figure 5: Y' rotation when $h = 1$

1 - To calculate $\mathbf{X}'$ for any h in $[\![0, \left\lfloor \frac{2\pi}{\delta_q} \right\rfloor]\!]$, we perform a rotation $h\delta_q$ of x-axis ($\mathbf{X}$) around $\mathbf{Z}$. This is achieved using the rotation matrix about $\mathbf{Z}$:

$$R_z(h\delta_q) = \begin{pmatrix} \cos(h\delta_q) & -\sin(h\delta_q) & 0 \\ \sin(h\delta_q) & \cos(h\delta_q) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Then, we multiply $R_z(h\delta_q)$ by $\mathbf{X}$:

$$\mathbf{X}' = R_z(h\delta_q)\mathbf{X}$$

2 - After calculating $\mathbf{X}'$, we determine $\mathbf{Y}'$ using the cross product of vectors $\mathbf{Z}$ and $\mathbf{X}'$. This can be written as:

$$\mathbf{Y}' = \mathbf{Z} \times \mathbf{X}'$$

3 - Finally, we can determine the rotation matrix around the axis $\mathbf{Y}'$ using Rodrigues' rotation formula:

$$R_{Y'}(q\theta) = I + \sin(q\theta)Q + (1 - \cos(q\theta))Q^2$$

where Q is the skew-symmetric representation of $\mathbf{Y}'$ and I the identity matrix.

$$Q = \begin{pmatrix} 0 & -Y'_z & Y'_y \\ Y'_z & 0 & -Y'_x \\ -Y'_y & Y'_x & 0 \end{pmatrix}$$

Each voxel v of the deposition head will now have new coordinates v' such that $v' = R_{Y'}(q\theta)v$.

The process leading to a new voxel model with non-integer coordinates. This inherent characteristic increases the complexity of subsequent collision tests. One idea is to round off the voxel values, this will lead to a modification of the object's shape. Although this deviation might appear minimal, it would lead to a maximum positional error of $\sqrt{3} \times$ voxel size. In our case, with a voxel size of 0.5 millimeter, the error would be $\frac{\sqrt{3}}{2}$ millimeter, which is relatively small. Another idea would be to rotate the triangular mesh first and then perform voxelization, thus ensuring that the obtained coordinates are integers while preserving the object's shape. However, doing this for every element in S would take a significant amount of time depending on the voxelization speed.

3.2 Results

By employing parallelization for optimization, we achieved results with a maximum angular variation of $\Theta = 70^\circ$ and a variation of $\theta = 10^\circ$ for both our recreated custom mold and the CAMu industrial's mold, using the deposition head depicted in Figure 4a. The time to create the set S of deposition head orientations was 0.028 seconds. The results are shown in Table 1. In this table, the numbers 5200 and 4744 represent the count of points (coordinates in x, y, z) on the toolpath for the Recreated Mold and CAMu Mol models, respectively. Before completing the printing process, the deposition head must pass through these points.

Mold	Execution Time	Total Time
Recreated Mold (5,200 points)	88.3 s	88.6 s
CAMu Mold (4,744 points)	83.3 s	83.6 s

Table 1: Results of optimization for two different molds without parallelization

The total time is the summation of the voxelization time for both the deposition head and the mold, the time required to obtain the set S, and the time to determine the collision-free path.

As we can observe in Table 2, a comparison has been made between the execution times for achieving a collision-free toolpath. This comparison specifically relates to our voxel-based approach under conditions that were closely equivalent to those presented by Nishat et al. in Table 1 of their paper. The scenario considered involves a toolpath comprising 273 points, akin to the bathtub example, here referred to as the recreated mold. The sole difference between our setups lies in the hardware used. For context, the specifications of the testing system used by Nishat et al. are as follows:

- **Chip**: Intel(R) Core(TM) i9-10885H CPU @ 2.40GHz
- **Memory**: 32GB 2933MHz DDR4
- **Storage**: 953.86GB (Model: SKHynix_HFS001TD9TNI-L2B0B)

In contrast, my system is less powerful. This discrepancy in hardware underscores the notable improvement facilitated by the use of a voxel-based approach in our study. Furthermore, while the solution from Nishat et al. was implemented in C++, our implementation was carried out in Python using PyTorch for parallelization. This direct comparison not only demonstrates the efficiency of voxel-based methods in computational performance for generating collision-free toolpaths but also highlights the potential of leveraging Python with parallel computing frameworks to achieve competitive results.

Method	Execution Time	Parallelization
Nishat et al. (C++ with Moller's Algorithm)	66 s	Yes
Commercial third-party library (Sequential)	4.4 s	No
Our Approach with Recreated Mold (Python 3.9 on CPU)	6 s	No
Our Approach with Recreated Mold (Python 3.10 on GPU T4x2)	1.2 s	Yes

Table 2: Comparative Analysis of Execution Times for Computing Collision-Free Toolpaths

4 GJK WITH BOUNDING BOX AND EPA ALGORITHM

4.1 GJK Algorithm

The Gilbert-Johnson-Keerthi (GJK) algorithm is a sophisticated method used for detecting the intersection of two convex sets in three-dimensional space. It capitalizes on the concept of the Minkowski difference between two sets, $A = \{a \,|\, a \in \mathbb{R}^3\}$ and $B = \{b \,|\, b \in \mathbb{R}^3\}$. The critical insight hinges on the idea that if the Minkowski difference $A \ominus B = \{a - b \,|\, a \in A, b \in B\}$ includes the origin, then the sets intersect.
Rather than fully constructing the Minkowski difference, which is computationally demanding, the GJK algorithm smartly approximates the intersection testing. It leverages specific support functions ψ, to work efficiently with subsets of points located on the surfaces of A and B respectively. ψ is defined as:

$$\psi_A : \mathbb{R}^3 \to \mathbb{R}, \quad \mathbf{d} \mapsto \max_{\mathbf{a} \in A}(\mathbf{a} \cdot \mathbf{d})$$

As shown in Figure 6, ψ applied to the set A (ψ_A) takes a vector $\mathbf{d}$ from $\mathbb{R}^3$ and returns a point from set A that is farthest in the direction of $\mathbf{d}$.

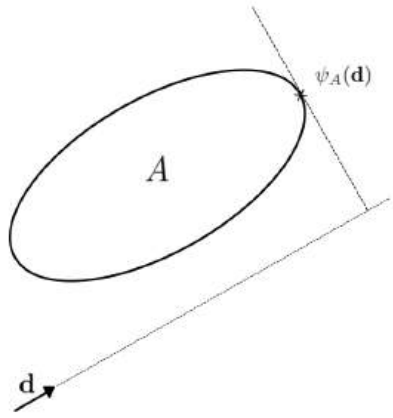


Figure 6: 2D representation of the ψ function

Indeed, applying this function to the set $A \ominus B$ yields the following equation:

$$\psi_{A\ominus B}(\mathbf{d}) = \psi_A(\mathbf{d}) - \psi_B(-\mathbf{d})$$

These functions (ψ_A,ψ_B) are instrumental in deducing the necessary conditions for the presence of the origin within the Minkowski difference, symbolizing a collision or intersection. Here, a collision is defined as the overlap of two objects.
Its primary constraint is its applicability only to convex objects. However, the Python library `CoACD` [22] offers a solution by decomposing a non-convex object into a collection of convex sub-objects. Decomposition can take some time, depending on the complexity of the object.

(a) Convex Decomposition of simple mold by `CoACD`

(b) Convex Hull of deposition head

(c) Mesh from deposition head's Convex Hull

Figure 7: Visualization of Convex Decomposition and Hull Generation

As shown in Figure. 7 the deposition head's form closely mirrors its convex hull, ensuring this approximation doesn't lead to complications. The collision assessment involves applying GJK between the deposition head and a convex sub-object of the mold. This process is repeated until all sub-objects are examined. An optimization strategy involves focusing on sub-objects within the mold that are in close proximity to the deposition head. This is achieved by employing a bounding box [10] around the deposition head to narrow down the area of interest to only those sub-objects surrounding the object.
Figure 8 illustrates an application of this optimization strategy in the context of a collision state between voxelized convex sub-objects of the simple mold and the voxelized mesh from the deposition head's Convex Hull. It can also be applied directly to their meshes, which eliminates the need for the voxelization step.

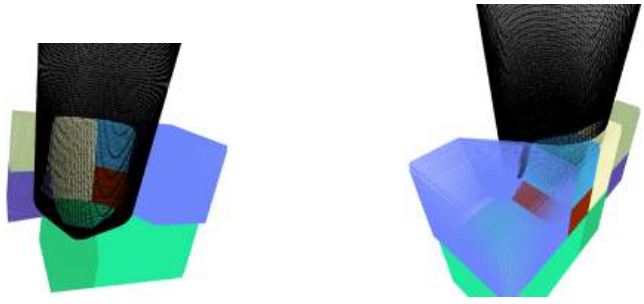
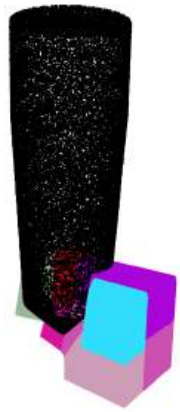
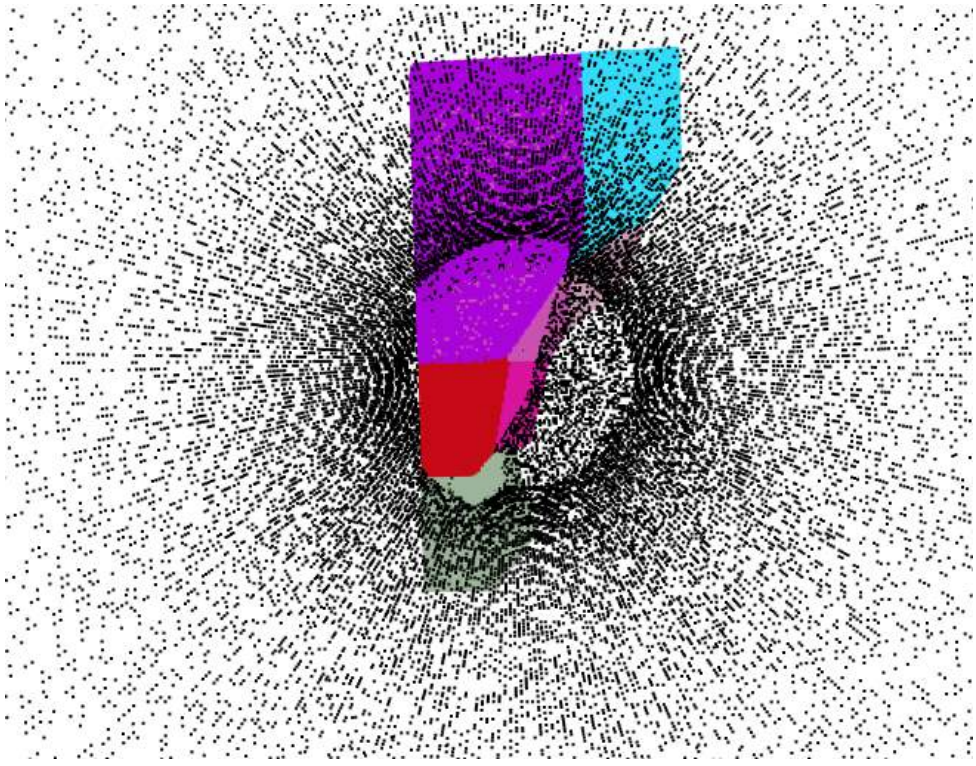


Figure 8: Visualization of a collision state between the simple mold and the convex hull of the deposition head

Collision with the green convex sub-objects



Collision with the purple convex sub-objects



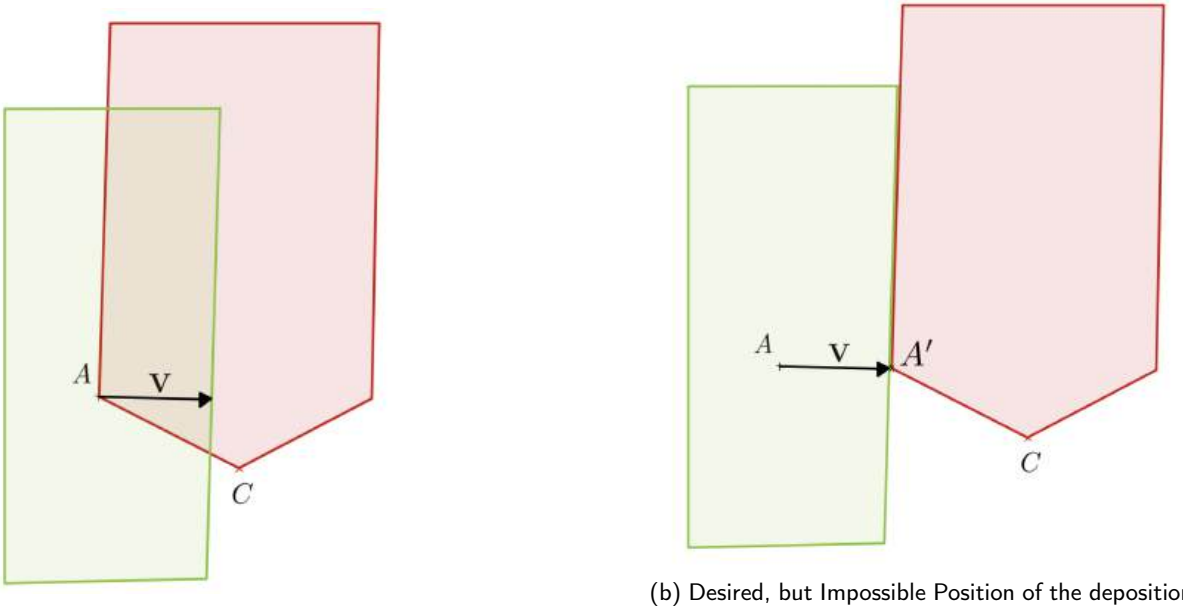
View from inside the Deposition Head

Figure 8: Visualization of a collision state between the simple mold and the convex hull of the deposition head

Figure 8 displays two collision states for two different points along the toolpath. The key observation here is that the collision state is actually an overlap between the Deposition Head and convex sub-objects.

4.2 EPA Algorithm

The Expanding Polytope Algorithm (EPA) serves as a natural progression from the GJK algorithm. Its primary purpose is to leverage the simplex, a byproduct of GJK, to identify the minimum displacement vector essential for collision resolution. This vector, which we will refer to as the extraction vector, represents the most efficient path to exit a collision state. It signifies the shortest distance between the origin and the surface of $A \ominus B$. Indeed, if we subtract the extraction vector from the set $A \ominus B$, then the origin no longer belongs to the set, thus eliminating the collision state.
This algorithm is highly efficient in terms of time complexity. However, it only works on convex objects. This problem is resolved by the `CoACD` library, which decomposes our object into a sum of n convex sub-objects. Nevertheless, this introduces a new problem: multiple sub-objects collide with the deposition head. Therefore, we have a collision extraction vector $\mathbf{V}_i$ available for each colliding sub-object i. To determine the correct extraction vector to prevent further collisions, we compute a weighted average of the magnitudes of the vectors. We compute the weights w_i for n vectors as $w_i = \frac{\|\mathbf{V}_i\|}{\sum_{k=1}^{n} \|\mathbf{V}_k\|}$ and then calculate the correct extraction vector $\mathbf{V}$ as $\mathbf{V} = \sum_{k=1}^{n} w_k \mathbf{V}_k$.
This method provides an extraction vector, $\mathbf{V}$, which suggests translating the deposition head to exit the collision state. However, any translation is infeasible during 3D printing. Thus, a rotation centered at the tip of the deposition head, in the direction of $\mathbf{V}$ is required. We can therefore iteratively vary the unit orientation $\mathbf{N}$ of the deposition head in the direction $\mathbf{V}$.



(a) Origin of the extraction vector $\mathbf{V}$

(b) Desired, but Impossible Position of the deposition head

Figure 9: Visualization of the output of the EPA Method for Collision Avoidance

As visualized in Figure 9, EPA provides an extraction vector $\mathbf{V}$, which suggests translating deposition head represented by the red object to exit the collision state. However, any translation is infeasible during 3D printing. Thus, a rotation centered at point C (the tip) in the direction of $\mathbf{V}$ is required.
This method involves iteratively adjusting the deposition head vector $\mathbf{N}$ towards $\mathbf{V}$. In 3D, as shown in Fig. 10 the angle difference $|\beta - \phi|$ must be less than θ. The new deposition head vector $\mathbf{N}'$ is determined by rotating

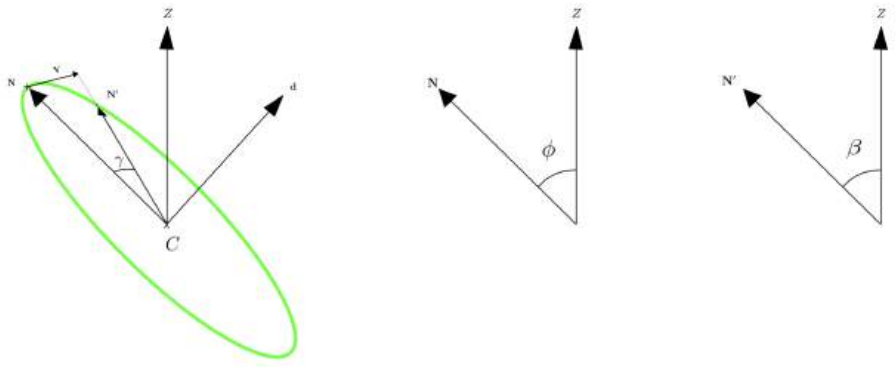


Figure 10: Variation of the deposition head vector in 3D

$\mathbf{N}$ by an angle γ around vector $\mathbf{d} = \mathbf{N} \times \mathbf{V}$. Subsequently, to ensure the condition is met, we only need to verify that:

$$|\arccos(\mathbf{Z} \cdot \mathbf{N}) - \arccos(\mathbf{Z} \cdot \mathbf{N}')| < \theta$$

which is:

$$|\arccos(\mathbf{Z} \cdot \mathbf{N}) - \arccos(\mathbf{Z} \cdot \mathbf{R}_{\mathbf{d}}(\gamma)\mathbf{N})| < \theta$$

4.3 Algorithm

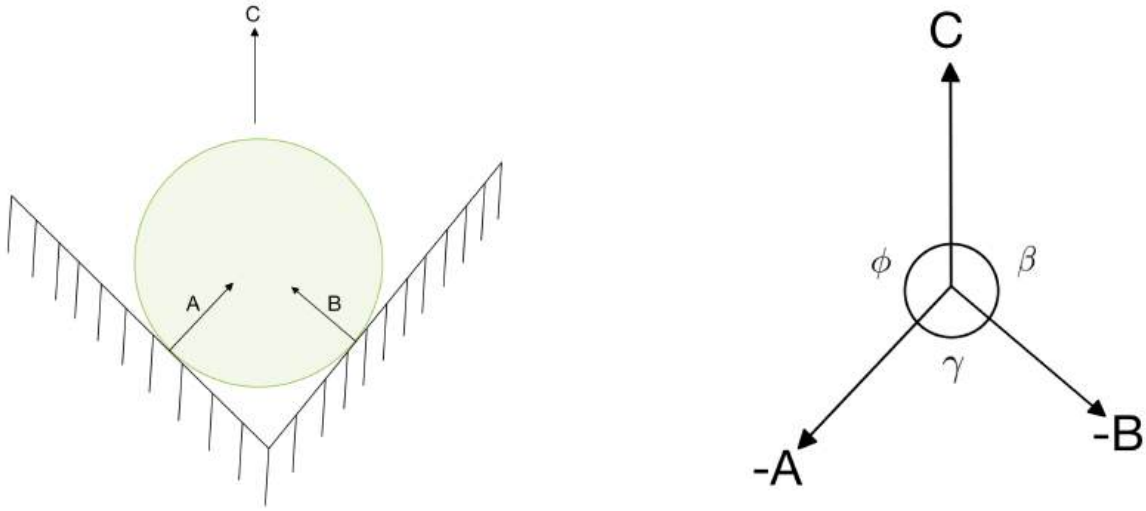
The operation of the algorithm is as follows:

1. **Convex decomposition:** The program uses CoACD to decompose the object into convex subparts.
2. **STL to voxel transformation:** The STL model of the deposition head is converted into a voxel model with a distribution of voxels categorized into three zones: low, middle, and high.
3. There are two methods to change the orientation of deposition head positioning:
   - The initial orientation of the deposition head is aligned with the z-axis, and it will be modified from this starting position.
   - The second method continuously adjusts the deposition head's orientation throughout the path. For each point, the deposition head's orientation is determined based on the result of the procedure that exits the collision state from the previous point.
4. **Mask retrieval:** Using the deposition head's bounding box, only the 3D printed model region in collision with the deposition head is selected. The goal then is to execute the GJK algorithm solely on the convex subparts within this zone.
5. **Deposition head zone selection:** Identify the region of the deposition head in contact with the colliding 3D printed model zone using the bounding box of this zone. This selected region is then used in the orientation adjustment method to conduct collision tests of equivalent density.
6. **GJK execution:** Execute the GJK algorithm on these zones. Subsequently, the extraction vector is determined using the EPA following the GJK.
7. Two iterative methods exist for adjusting the orientation:
   - The first method involves obtaining the extraction vector initially and iterating the deposition head's rotations in this direction. However, this can lead to new collisions.

- The other option is to recalculate a new extraction vector after each rotation. The new extraction direction is computed using Lami's theorem [14] to retain information about both the previous and new vectors.

Lami's theorem provides a relationship among three vectors to maintain an object in equilibrium. This can be visualized as a sphere wedged in a corner; See Fig. 11. The relationship provided by Lami's theorem is expressed as follows:

$$\frac{\|\mathbf{A}\|}{\sin(\beta)} = \frac{\|\mathbf{B}\|}{\sin(\phi)} = \frac{\|\mathbf{C}\|}{\sin(\gamma)} \tag{2}$$



(a) Sphere in equilibrium on a non-convex support

(b) Link between vectors

Figure 11: Lami's theorem application

In our scenario, let's assume that A represents the previous extraction vector and B the new one. Then, we can derive the new extraction vector C, which is a combination of vectors A and B in order to retain the information.

4.4 Results

Aspect	Recreated Mold	CAMu Mold
Decomposition into convex sub-elements	12.7 s	13.2 s
Execution Time (after summing all program actions)	71.2 s	95.4 s

Table 3: Decomposition and Execution Times for the Molds without parallelization

4.5 Results

Aspect	Recreated Mold
Decomposition into convex sub-elements	12.7 s
Execution Time (after summing all program actions)	71.2 s

Table 4: Decomposition and Execution Times for the Mold without parallelization

As demonstrated in Table 4, the total execution time for our second method with GJK is comparable to that of our voxel-based approach, as shown in Table 1. However, as the complexity of the object's geometry increases, such as with the CAMufacturing Solutions Inc. Mold, the execution time also rises. Despite this, the results of these two approaches remain superior to those of Nishat et al. [16], even without considering parallelization. Consequently, our results represent an improvement, indicating not only efficiency gains but also potentially broader applicability in real-world scenarios where execution speed is crucial.

5 UPDATING THE PRINTED MODEL AND REDUCING JITTERING

In our 3D printing model, we simulate material extrusion through voxelized spheres, updating their values as the deposition head moves for real-time updates. To ensure smoother transitions and reduce jitter, we've implemented an angular adjustment function using Bezier curves. This smooths out angular changes and segments the toolpath for efficient computation, resulting in continuous, stable motion.

5.1 Updating the Printed Model

In the live updating of the printed model, a sphere of a certain radius is created to represent the material extruded from the deposition head, which is then voxelized as shown in Fig. 12 . As the deposition head moves linearly from point A to point B, the voxels under the deposition head, corresponding to the sphere's position, are densified, changing their value from 0 to 1. This occurs each time the deposition head travels a distance along the linear path. Consequently, the printed model is continuously updated during the printing process. The deposition head maintains a constant distance h perpendicular to the printing point. The sphere is deposited at the printing point, following the tangent direction. Additionally, there's no issue of overlap between two spheres if the distance d separating them is less than their diameter, as it's merely a matter of voxel density changing from 0 to 1.

(a) Bead (b) Voxelized Bead

Figure 12: Comparison of Bead Geometry Before and After Voxelization Under the Deposition Head

5.2 Reducing Jittering

In our function, which adjusts the inclination of the deposition head, the output is a vector of dimensions $N \times 2$. Here, N denotes the number of points along the path. Each path point is associated with two coordinates: the angle θ, resulting from the dot product between the deposition head vector and z-axis; and the rotation angle δ around z-axis, starting from x-axis.
To smooth the angular variation, this vector is first transformed to represent the angular difference between successive points, yielding a new array of size $(N-1) \times 2$. This array captures the deposition head's angular variation for N points.
The angular variation is considered in the counterclockwise direction. If the angular difference $|\Delta|$ between two consecutive points is larger than $|2\pi - \Delta|$, this angular variation is chosen to ensure the shortest path to the new position.
To mitigate jittering, the Bezier Function [2] is employed, mathematically represented for K control points as $B(t) = \sum_{i=0}^{K-1} P_i B_{i,K-1}(t)$, where $B_{i,K-1}(t)$ are the Bernstein polynomials over the interval $[0, 1]$.
To define the control points P, we conceptualize them as vectors in $\mathbb{R}^3$. The first coordinate represents the position of the control point to preserve the ordering of points along the toolpath. The second coordinate y corresponds to the angular variation θ, and the third coordinate z) corresponds to the angular variation δ.
If there are N points on the toolpath, there will be $N-1$ control points, each derived from the angular variations between consecutive points. We compute the Bezier curves using segments of 10 control points each. After computing, these segments are reconnected.
The continuity at the connection points of these curves is ensured to be at least C^1 continuity, meaning the position and the first derivative (tangent) are continuous across segments. This level of continuity is achieved by aligning the tangents at the end of one segment with the tangents at the beginning of the next, ensuring smooth transitions without abrupt changes in the toolpath.

6 VOXEL OCCUPANCY ANALISYS FOR COLLISION ACCURACY

The accuracy of our method depends on the extent to which voxels occupy the surface of the voxelized object. Larger voxel sizes result in more unoccupied volume, which alters the representation of the object. To measure the model's precision, we focus on the 8 vertices of each voxel. These vertices, representing the voxel's boundaries, provide critical data points for measuring discrepancies between the voxelized model and the actual object.
Using the Signed Distance Function (SDF) [17] method, we project the 8 vertices of each voxel onto the surface of the STL object to obtain their distances from this surface. These distances are negative if the points are inside the object and positive if they are outside. We retain voxels with at least one vertex outside the surface and measure the average discrepancies for these vertices. Then, we calculate the mean of these averages, reflecting the voxel model's precision (i.e., the extent to which surface voxels are occupied).

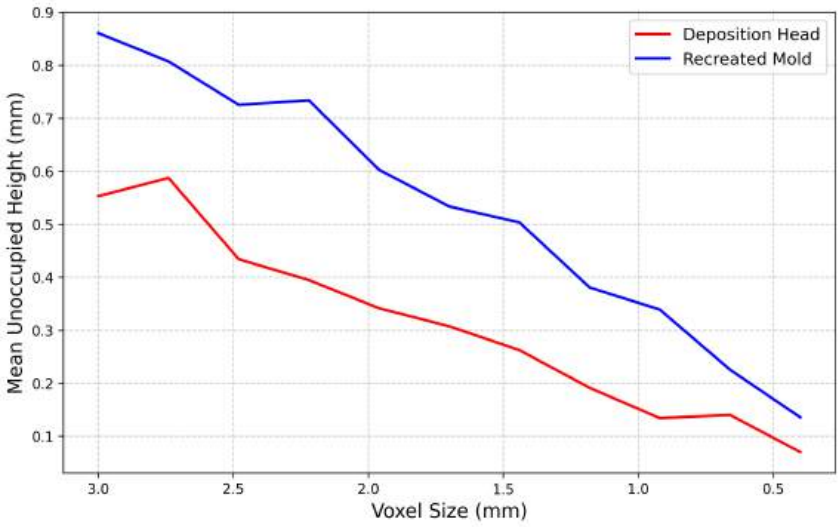


Figure 13: Relationship between Voxel Size and Mean Unoccupied Voxel Volume Height at STL Surface

This method allows us to quantify the average height of the unoccupied volume within the surface voxels, which is directly related to the accuracy of voxel-based collision tests. Since collision detection is a density overlap test (True: presence, False: absence), this detection is accurate because there is always some unoccupied volume height for each of the two voxelized models. Additionally, there are no false positives, only true negatives, as the voxel approximation is always constructed to encompass the STL model, thus including the STL volume within the voxel space.

By optimizing voxel size, we can improve collision detection precision while maintaining acceptable computational efficiency, which is crucial for industrial applications where both speed and accuracy are important. Figure 13 results from the voxelization process of the Recreated Mold at various voxel sizes ranging from $3\ mm$ to $0.4\ mm$. It illustrates that as the voxel size decreases, the average height of the unoccupied volume also diminishes. Consequently, the likelihood of encountering a false negative collision event decreases. However, such fine resolution requires significant memory capacity to store the increased number of voxels.

7 MEMORY REQUIREMENTS OF VOXEL-BASED MODELS

This section evaluates and compares the memory requirements of voxel-based models across various voxel sizes to the memory used for the original STL models. Each mesh, loaded from an STL file, is voxelized at resolutions ranging from 3 mm to 0.1 mm. Following voxelization, a three-dimensional tensor representing the bounding box of the STL model is created. This tensor consists of boolean elements, where a value of 1 (or True) indicates a voxel at the surface of the object, and 0 otherwise. However, the storage of boolean values in Python, typically encoded over several bits, requires significant memory, especially for storing zeros. To conserve memory, the tensor can be transformed into a sparse format [4]. The concept of sparsity, which refers to the proportion of zero elements in a dataset, is employed to enhance storage efficiency. The key idea is to store only the pertinent values, in this case, the surface voxels that are True. Thus, only the indices and values of these nonzero elements are stored, significantly reducing memory requirements.

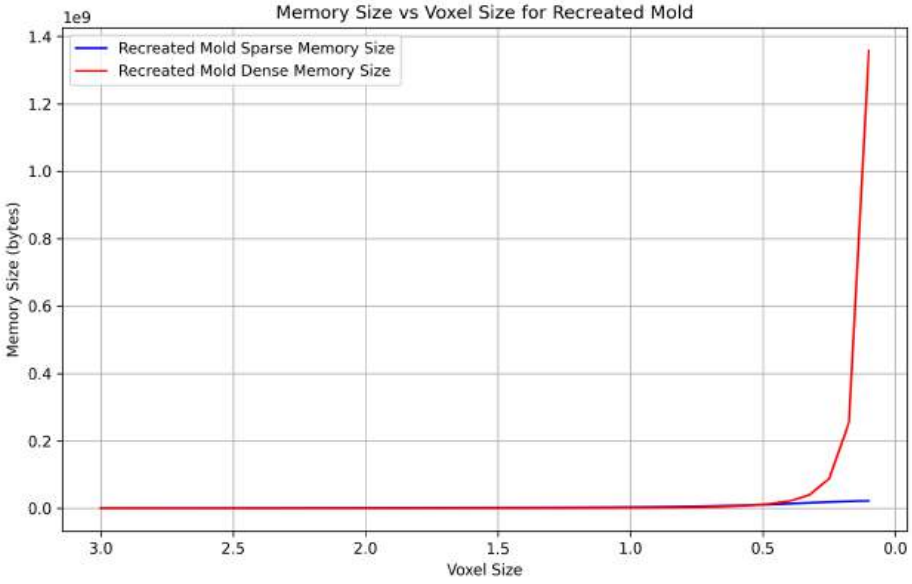


Figure 14: Comparison of memory usage for boolean tensor and sparse storage.

Figure 14 illustrates the comparative storage requirements for Recreated Mold using boolean tensors and sparse storage formats. The boolean tensor storage, shown with a red line, exhibits explosive growth, whereas the sparse storage, represented by a blue line, increases linearly up to a certain voxel size. However, below a voxel size of 0.5 mm, the difference in storage between the two methods is negligible. Therefore, for models requiring a large number of voxels or smaller voxel sizes, it is necessary to switch to a sparse storage structure. In our specific case, having chosen a voxel size of 0.5 mm, we will continue to use the boolean tensor format. The table 5 is provided to compare the memory requirements for the STL file, boolean tensor, and sparse format across our three models for a voxel size of 0.5 mm.

Model	STL Memory (GB)	Boolean Tensor (GB)	Sparse Storage (GB)
CAMu Mold	0.00253	0.036	0.012
Deposition Head	0.00008	0.004	0.003
Recreated Mold	0.00006	0.011	0.008

Table 5: Memory requirements for different models in various formats.

Despite using boolean values and sparse storage, this table highlights the substantial increase in memory requirements for voxel models compared to the original STL models.

This analysis demonstrates that despite the increased execution speed associated with using voxels instead of the STL model, which enables efficient parallelization, a balance must be struck between the precision of the collision detection test and memory usage. The precision of the collision test increases as the voxel size decreases; however, smaller voxel sizes also lead to higher storage requirements. An optimal compromise for

the models we have used is achieved with a voxel size of 0.5 mm. This size is reasonable both in terms of memory storage and collision test precision. It is important to note again that true negatives do not occur due to the adopted voxelization approach. However, this could potentially lead to false collision detections.

8 CONCLUSION AND FUTURE WORK

In this research, we have successfully developed and demonstrated two algorithms that enhance the process of detecting and avoiding collisions within toolpaths for additive manufacturing (AM). These methods not only accelerate the detection process but also ensure the generation of viable, collision-free alternatives, marking a substantial step forward in the operational efficiency of AM technologies. While our current methodologies show promising results, their adaptation to the fast-paced demands of industrial AM processes necessitates further optimization to achieve the desired speeds. Parallelization emerges as a promising avenue to address this challenge, potentially enabling real-time collision avoidance capabilities even in the most complex manufacturing scenarios.

Our findings suggest that a strategic focus on concentrating voxel tests within areas of high collision risk, thereby reducing the number of voxels requiring examination, could decrease execution times. This approach implies a more intelligent allocation of computational resources, prioritizing areas where the likelihood of collision is greatest and thus enhancing the overall efficiency of the collision detection and avoidance process.

Looking forward, our study opens several pathways for future research. These include the refinement of voxel-based methods to further minimize computational demands without compromising accuracy, the exploration of advanced parallelization techniques to support real-time processing requirements, and the development of adaptive algorithms capable of dynamically adjusting to varying levels of complexity within the AM process. Additionally, the integration of machine learning algorithms could offer predictive insights into collision risk areas, automating the optimization process for toolpath generation in unprecedented ways.

REFERENCES

[1] Aldair, A.A.; Rashid, M.T.; Rashid, A.T.: Navigation of mobile robot with polygon obstacles avoidance based on quadratic bezier curves. Iranian Journal of Science and Technology, Transactions of Electrical Engineering, 43(4), 757–771, 2019.

[2] Bézier, P.: Numerical control-mathematics and applications. Translated by AR Forrest, 1972.

[3] Cameron, S.: Enhancing gjk: Computing minimum and penetration distances between convex polyhedra. In Proceedings of international conference on robotics and automation, vol. 4, 3112–3117. IEEE, 1997. http://doi.org/10.1109/ROBOT.1997.606761.

[4] Candès, E.J.; Wakin, M.B.: An introduction to compressive sampling. IEEE signal processing magazine, 25(2), 21–30, 2008.

[5] Cohen-Or, D.; Kaufman, A.: Fundamentals of surface voxelization. Graphical models and image processing, 57(6), 453–461, 1995. http://doi.org/10.1006/gmip.1995.1039.

[6] Devillers, O.; Guigue, P.: Faster triangle-triangle intersection tests. Ph.D. thesis, INRIA, 2002.

[7] Fang, G.; Zhang, T.; Zhong, S.; Chen, X.; Zhong, Z.; Wang, C.C.: Reinforced fdm: Multi-axis filament alignment with controlled anisotropic strength. ACM Transactions on Graphics (TOG), 39(6), 1–15, 2020.

[8] Gibson, S.F.F.: Beyond volume rendering: visualization, haptic exploration, and physical modeling of voxel-based objects. In Visualization in Scientific Computing95: Proceedings of the Eurographics Workshop in Chia, Italy, May 3–5, 1995, 10–24. Springer, 1995.

[9] Gilbert, E.G.; Johnson, D.W.; Keerthi, S.S.: A fast procedure for computing the distance between complex

objects in three-dimensional space. IEEE Journal on Robotics and Automation, 4(2), 193–203, 1988. http://doi.org/10.1109/56.2083.

[10] Gottschalk, S.A.: Collision queries using oriented bounding boxes. The University of North Carolina at Chapel Hill, 2000.

[11] Hermann, A.; Drews, F.; Bauer, J.; Klemm, S.; Roennau, A.; Dillmann, R.: Unified gpu voxel collision detection for mobile manipulation planning. In 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, 4154–4160. IEEE, 2014.

[12] Jang, D.; Kim, K.; Jung, J.: Voxel-based virtual multi-axis machining. The International Journal of Advanced Manufacturing Technology, 16, 709–713, 2000.

[13] Jiang, Z.; Wang, H.; Sun, Y.: Improved co-scheduling of multi-layer printing path scanning for collaborative additive manufacturing. IISE Transactions, 53(9), 960–973, 2021.

[14] Kumar, D., A.; Kushreshtha: Engineering Mechanics: Statics and Dynamics. Jaypee University of Information Technology, Solan, HP, 2015.

[15] Möller, T.: A fast triangle-triangle intersection test. Journal of graphics tools, 2(2), 25–30, 1997.

[16] Nishat, R.I.; Bahoo, Y.; Georgiou, K.; Hedrick, R.; Jill, R.: Collision-free multi-axis tool-path for additive manufacturing. Computer-Aided Design and applications, 2023.

[17] Osher, S.; Fedkiw, R.; Piechor, K.: Level set methods and dynamic implicit surfaces. Appl. Mech. Rev., 57(3), B15–B15, 2004.

[18] Plakhotnik, D.; Glasmacher, L.; Vaneker, T.; Smetanin, Y.; Stautner, M.; Murtezaoglu, Y.; van Houten, F.: Cam planning for multi-axis laser additive manufacturing considering collisions. CIRP Annals, 68(1), 447–450, 2019.

[19] Rodrigues, O.: Des lois géométriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire. Journal de mathématiques pures et appliquées, 5, 380–440, 1840.

[20] Sullivan, C.; Kaszynski, A.: Pyvista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (vtk). Journal of Open Source Software, 4(37), 1450, 2019. http://doi.org/10.21105/joss.01450.

[21] Tharwat, A.; Elhoseny, M.; Hassanien, A.E.; Gabel, T.; Kumar, A.: Intelligent bézier curve-based path planning model using chaotic particle swarm optimization algorithm. Cluster Computing, 22, 4745–4766, 2019.

[22] Wei, X.; Liu, M.; Ling, Z.; Su, H.: Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. ACM Transactions on Graphics (TOG), 41(4), 1–18, 2022. http://doi.org/10.1145/3528223.3530103.

[23] Zhou, Q.Y.; Park, J.; Koltun, V.: Open3d: A modern library for 3d data processing. arXiv preprint arXiv:1801.09847, 2018.