machines

Article

# High-Accuracy and Efficient Simulation of Numerical Control Machining Using Tri-Level Grid and Envelope Theory

**Zhengwen Nie * and Yanzheng Zhao**

School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China; yzh-zhao@sjtu.edu.cn
* Correspondence: zhengwenme@sjtu.edu.cn

**Abstract:** Virtual simulation of high-resolution multi-axis machining processes nowadays plays an important role in the production of complex parts in various industries. In order to improve the surface quality and productivity, process parameters, such as spindle speed, feedrate, and depth of cut, need to be optimized by using an accurate process model of milling, which requires both the fast virtual prototyping of machined part geometry for tool path verification and accurate determination of cutter–workpiece engagement for cutting force predictions. Under these circumstances, this paper presents an effective volumetric method that can accurately provide the required geometric information with high and stable computational efficiency under the condition of high grid resolution. The proposed method is built on a tri-level grid, which applies two levels of adaptive refinement in space decomposition to abolish the adverse effect of a large fine-level branching factor on its efficiency. Since hierarchical space decomposition is used, this multi-level representation enables the batch processing of affected voxels and minimal intersection calculations, achieving fast and accurate modeling results. To calculate the instantaneous engagement region, the immersion angles are obtained by fusing the intersection points between the bottom-level voxel edges and the cutter surface, which are then trimmed by feasible contact arcs determined using envelope theory. In a series of test cases, the proposed method shows higher efficiency than the tri-dexel model and stronger applicability in high-precision machining than the two-level grid.

**Keywords:** accuracy; computer-aided manufacturing; efficiency; grazing point; virtual machining

check for updates

Academic Editor: Gianni Campatelli

Received: 23 December 2024
Revised: 12 January 2025
Accepted: 16 January 2025
Published: 18 January 2025

**Citation:** Nie, Z.; Zhao, Y. High-Accuracy and Efficient Simulation of Numerical Control Machining Using Tri-Level Grid and Envelope Theory. *Machines* **2025**, *13*, 69. https://doi.org/10.3390/machines13010069

## 1. Introduction

Multi-axis milling is a versatile machining technology used for producing delicate parts with freeform surfaces [1,2]. To verify and optimize the milling process before the parts are manufactured on real machine tools, virtual machining (VM) technology is proposed to verify NC codes, reduce cycle times and surface errors, and predict instantaneous cutting states. However, these goals cannot be achieved without accurate geometrical and physical modeling of the milling process. The former aims to compute the cutter-swept volume (CSV), machined part geometry, and cutter–workpiece engagement (CWE), while the latter conducts dynamics analysis using the obtained information.

Cutter–workpiece engagement refers to the instantaneous contact region between the cutting tool and the in-process workpiece (IPW). Based on this geometry, the in-cut edge segments can be determined for cutting force predictions, which subsequently initiate the physical modeling process. In freeform surface machining, CWE varies along a tool path, and in general, unless very simple workpiece geometry is machined, it is challenging to find an exact solution, which has attracted extensive investigation [3–8].

During the milling process, whether the cutting forces are computed or not, it is necessary to geometrically represent the machined part according to the cutting tools and their trajectories, that is, according to tool paths. This calculation process is referred to as a workpiece update in this research, and the obtained geometric information is very useful in general NC milling simulations. First, this information indicates whether the desired part can be manufactured by designed tool paths within the specified tolerance. Second, unexpected collisions between the milling tool and in-process workpieces, such as local gouging, rear gouging, and global interference, can also be detected [9–11]. Such collisions will lead to severely compromised surface qualities or even safety accidents.

The basic idea for NC simulations is to remove every cutter-swept volume from the blank until the final machined part is obtained [3,4,12,13]. Mathematically, this volume is defined as the totality of all 3D points that belong to the trace of a cutter moving along an arbitrary path. It can be modeled as the volume enclosed by an ingress surface at the initial location, an egress surface at the final location, and an envelope surface in between, which is the collection of the grazing points on each tool instance. According to the envelope theory, the product of the velocity and the surface normal vector on a grazing point is always equal to zero. Another way is to remove the workpiece material inside the tool instance at each cutter location (CL) point sampled along a tool path, which has also been widely used. Meanwhile, the instantaneous engagement region between the cutter and in-process workpiece is extracted and represented as a CWE map, where immersion angles are computed by a further discretization of the contact area.

In modern manufacturing, an efficient geometric modeling method will accelerate the optimization process of various machining parameters significantly. Meanwhile, with the rapid development of automotive, aerospace, and biomedical industries, the demand for high-quality machine parts is increasing sharply. However, few of the existing virtual machining technologies can trade off high efficiency against high quality satisfactorily, leading to the necessity of the present study.

The rest of this paper is structured as follows: Section 2 summarizes the geometric modeling methods utilized in virtual machining and points out their benefits and drawbacks to help identify the preferred modeling frame. Section 3 illustrates the tri-level representation of the workpiece, and Section 4 expounds on the implicit function of a generalized cutter. The simulation procedures of the proposed method are then elaborated in Section 5. Section 6 provides a quantitative comparison to demonstrate the computational improvement the proposed method brings over the tri-dexel model and two-level grid, followed by conclusions and future work in Section 7.

## 2. Existing Methods

A variety of geometric modeling methods have been proposed and applied in NC milling simulations, aiming to achieve both computational accuracy and efficiency in modeling the machining geometry while being flexible enough to handle different types of cutters and tool movements. These existing methods can be roughly classified into three groups: solid modeling, vector modeling, and voxel modeling. Solid modeling is able to provide the highest level of precision by utilizing the exact representation of the cutter and the workpiece geometry, while the other two groups are discrete modeling approaches, which strive to achieve higher computational speed at the cost of accuracy.

In solid modeling [14–16], the workpiece material removal process is simulated by making use of the geometric and topological algorithms built into the solid modeler kernel. Specifically, constructive solid geometry (CSG) and boundary representation (B-rep) are the most widely used data structures in the field of solid modeling. CSG builds a solid object by combining primitive objects with Boolean operators, while B-rep describes an object with algebraic and parametric surfaces in the forms of Bezier, B-spline, and NURBS (Non-Uniform Rational B-Splines) patches. Compared to CSG, B-rep offers a richer set of operations in terms of modifying workpiece surfaces, hence gaining more attention in modern NC milling simulations. Because the geometry of both the in-process workpiece and cutter-swept volume keeps changing, large amounts of intersections between surfaces need to be solved numerically in order to determine the intersection curves for building new patches. A typical cutter location data file usually contains a huge number of cutter movements, so solving these complex intersections has become one of the most time-consuming processes in virtual machining. Meanwhile, the computation load needed to maintain and process the tree of numerous tool swept volumes in order to generate the machined part geometry grows significantly as the program runs. In addition, the high complexity of the algorithms required to manipulate intersections in multi-axis NC simulation is more likely to result in computational breakdowns as a result of round-off errors.

The second modeling method utilizes a collection of spatial geometric elements to define the cutter and workpiece for simplified Boolean operations. The typical example of such a discrete modeling method is called Z-map representation [17–20], which represents the workpiece by a 2D array of vectors with one end on a fixed coordinate plane and the other end on the top surface of the workpiece. To simulate the cutting operations, these vectors are updated as the cutter sweeps through different regions of the grid to capture their new heights, i.e., the in-process workpiece boundary. During the milling process, the vectors can be updated easily if the cutter geometry is fairly simple. For a more complicated cutter like a bull-end mill, these vectors are usually updated by solving high-order nonlinear equations. In order to clip Z-map vectors with less computation time, Chung et al. [17] represented the cutter-swept surface generated by translational motion by an implicit function of $z = f(x, y)$ in three-axis milling and determined quadruple roots using an analytic formula; however, this approach may introduce intolerable numerical errors. Tunç et al. [18] conducted cutting force and stability simulations through the extended Z-mapping approach, where multiple dexels are connected at one vector location. Li et al. [19] developed an improved Z-map algorithm that combines servo rectangular encirclement and the angle summation method to simulate the part surface topography, while Xiao et al. [20] performed the same task based on the sequential quadratic programming algorithm. Eventually, to resolve the issue of poor sensitivity along the vector direction of any single set of parallel vectors, the most advanced tri-dexel representation, also called triple-ray representation, was proposed, which is essentially three orthogonal dexel models combined together, and it has been used by various computer-aided manufacturing (CAM) systems [6,15,21].

Voxel modeling or space partitioning is the third geometric modeling method employed by virtual machining [22–27]. This volumetric modeling method has an evident advantage over vector modeling since the fundamental operation involved in workpiece updates is the simple deactivation of any cubic element falling into the cutter-swept volume. Such an operation has a very low computational cost compared to the operation involved in updating other classes of models; hence, it is more efficient. A basic uniform grid, which means the entire volume is uniformly subdivided, is the most fundamental data structure for workpiece representation [22–24]. However, both memory consumption and computation time will increase dramatically when a high grid resolution is required to provide accurate enough modeling results. In order to mitigate such overhead, more advanced voxel grids are proposed for a milling simulation, which can localize and reduce the finest update calculations effectively. Yau et al. [25] utilized the octree to simulate the cutting process with a generalized automatically programmed tool (APT), while Joy et al. [26,27] computed the machined part geometry using the two-level grid. In general, voxel modeling, due to its low computational complexity and hierarchical space decomposition, is regarded as the most promising approach, as it can provide the best combination of accuracy, efficiency, and flexibility among all geometric modeling methods.

Even though the two-level grid has been widely used by many researchers, the limitation of such a fixed hierarchical structure is still obvious. Since the available grid resolution is measured by the product of the branching factors at both levels, a higher resolution suggests that either of the two branching factors should be increased. A large top-level branching factor increases the number of tiles sharply and hence the memory usage, while a large bottom-level branching factor indicates a large ratio of inactive-to-active voxels and redundant storage. More importantly, a large bottom-level subdivision factor will introduce a large quantity of repetitive fine-level traversal operations since numerous coarse voxels need to be accessed for an update multiple times due to the small sampling interval, leading to a longer computation time. Such a limitation reduces its adaptivity in geometry for extreme-scale volume modeling. Moreover, since a voxel model is a fully discrete representation of a solid object, the precision of the voxel-based CWE geometry is not satisfactory [22–24]. Thus, a voxel-based CWE determination method that can achieve the subvoxel resolution and precision also needs to be developed.

## 3. Representation of Workpiece

### *3.1. Basic Concepts*

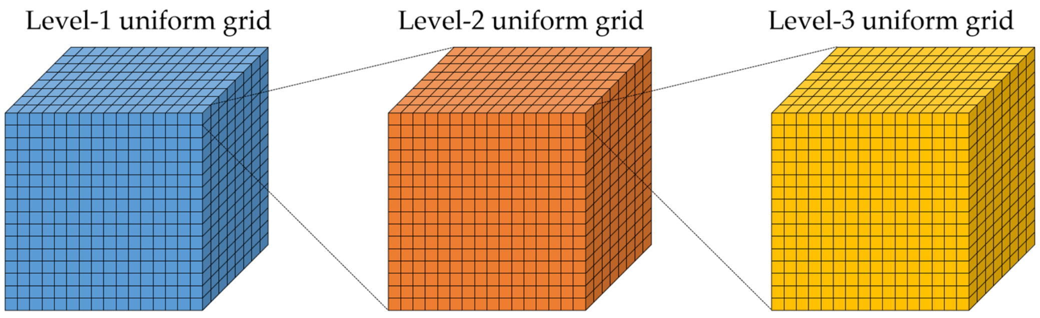
Voxel modeling, as a typical spatial occupancy enumeration type of a solid representation scheme, is mainly utilized in fields in which spatial data can be easily obtained from 3D scans of a surface or a volume, such as volumetric imaging in medicine and the representation of terrain in simulations. A voxel, essentially a small axis-aligned cube, is a 3D counterpart to a pixel, and a collection of such elementary cubes constitutes a voxel model of a solid object. Each voxel in the spatial grid is represented by a set of 3D indexes, indicating its positions along three orthogonal directions. Voxel modeling can also be understood as activating specific voxels that belong to the interior of the object. During this voxelization process, active voxels can include internal voxels (entirely inside the object) and surface voxels (partially inside the object), while external voxels (entirely outside the object) should stay inactive. In voxel modeling, the edge length of each voxel element or the finest edge length in a hierarchical data structure decides the available grid resolution of the volume. A higher resolution for a given space usually indicates the voxel model can describe the object with higher precision, but more time is required to manipulate the model.

### *3.2. Tri-Level Grid*

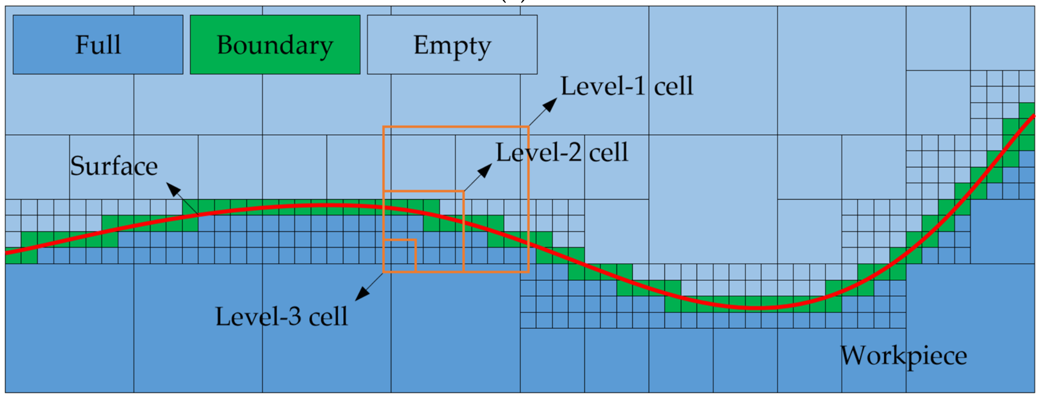
Voxel modeling, with its enumerative nature through spatial indexing, is very suitable for simulating the material removal process in virtual machining as it simply deactivates all voxels inside the cutter-swept volume. It is also sufficiently robust to handle model updates from complex tool motions since it does not have to maintain topological information explicitly. However, for the simplest uniform grid, where the voxel space is uniformly subdivided, the total number of involved voxels is in the order of $\mathbf{O}(N^3)$, where $N$ is the available grid resolution. As a result, its memory usage and simulation time will rise drastically when a high modeling accuracy is required, and a uniform grid is thus rarely adopted by practical simulations.

In order to mitigate the high memory overhead of a dense uniform grid, an effective approach is to exploit adaptive refinement in spatial decomposition. The resulting hierarchical data structure is a two-level grid, also referred to as a tiled grid, where the volume is first decomposed into uniformly sized cells, also called blocks, bricks, or tiles, and each cell is then partitioned into a finer subgrid if it intersects the surface of the solid object. Note that to quickly reach a target subgrid for a workpiece update, every tile that has been partitioned should be recorded by a key-value container, such as a hash table and map, where its 3D index serves as the key. To simulate the cutting process, an activated voxel at the coarse level that has been swept by the cutting tool entirely needs to be deleted and removed from the container, while an active coarse voxel should be added into the container and partitioned with all finer voxels that fall into the cutter's interior removed if it is partially cut by the tool for the first time.

The two-level grid manages to reduce both memory usage and computation time significantly; however, the aforementioned shortcoming restricts its further application in high-precision machining. To increase the adaptivity of the two-level grid, a tri-level grid, as shown in Figure 1, is employed, where each bottom-level voxel continues to be subdivided if it intersects with the object surface. Then, the available grid resolution is measured by the product of the branching factors at all three levels. All surface voxels at three levels constitute a surface voxel model of the object, and a solid voxel model, which is used to represent the workpiece model in this study, can be obtained if all internal voxels are also activated. By adding another level of adaptive refinement, the finest voxels that need to be updated can be approached very efficiently.



(**a**)



(**b**)

**Figure 1.** Tri-level grid representation and illustration: (**a**) layout of the tri-level grid; (**b**) 2D illustration of the tri-level grid.

## *3.3. Surface Reconstruction*

In order to visualize the in-process workpiece or machined part, a boundary representation, such as a triangle mesh, can be built by implementing the marching cubes algorithm [28] based on the voxel model. Triangle mesh has been widely used by many CAM systems due to its strong capability to represent free-form surfaces with simple triangles. Furthermore, by changing the triangle size according to the local curvature of the surface, the storage of the mesh model can be optimized for multi-axis milling simulations. The marching cubes algorithm is an effective approach for visualizing a conceptual surface called an isosurface by extracting its polygonal mesh from a 3D scalar field. An isosurface is formed from a set of 3D points satisfying the equation $f(x, y, z) = c$, where $c$ is a user-specified iso-value and remains constant for any point located on the isosurface. Therefore, an isosurface can be viewed as a surface within a cube where each point has the same parametric value. This algorithm works by iterating ("marching") over a uniform grid of cubes superimposed over a region of the function, taking eight cube vertices at a time, and then determining the polygons needed to represent the part of the isosurface passing through this cube. The obtained polygons are then fused into the final mesh model. For the first step, an index to a precalculated array of 256 possible polygon configurations ($2^8 = 256$) is generated for each cube being considered, and each of the eight scalar values corresponds to one bit in an 8-bit integer. A scalar value higher than the user-specified iso-value sets the bit to one, while a lower scalar value sets the bit to zero. Then, the final integer would be the actual index to the polygon indices array, which guarantees any neighboring polygons can be connected appropriately. Finally, each vertex of the extracted polygons, which should be placed in the appropriate position along the voxel edge, is determined by linearly interpolating the two scalar values that are connected by that edge. In this research, the cutting tool is represented by an implicit function, which, however, is not for the computation of the scalar value of each vertex. The use of this function aims to identify every intersecting voxel as well as all voxel edges crossed by the cutter surface, as shown in Figure 2. The edge intersection point is then determined by calculating the real intersection between the voxel edge and the cutter surface.

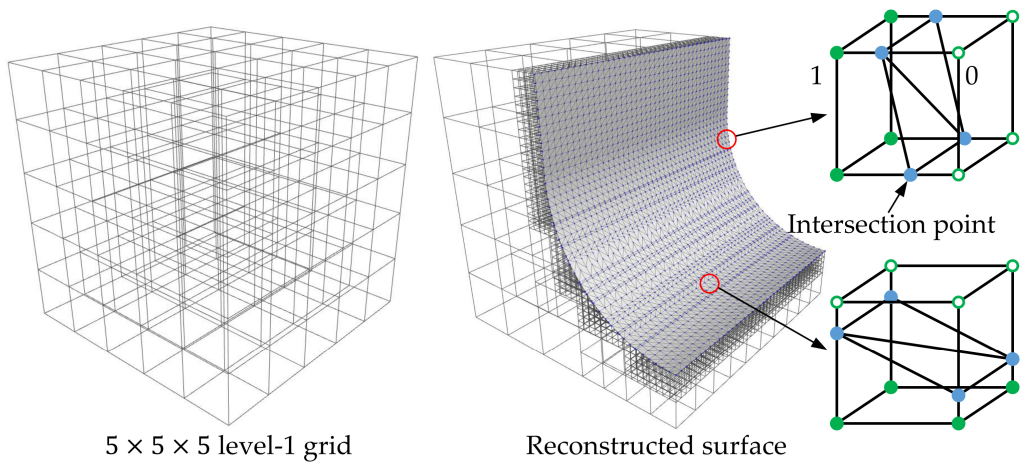


**Figure 2.** Surface reconstruction by a tri-level grid.

During the reconstruction process, the polygons inside each cube can be built appropriately according to the lookup table of the marching cubes algorithm as long as all vertex states and edge intersection points are correctly computed. However, a simple data regularization procedure is still required to eliminate noises, inconsistent data, or missing data caused by numerical errors. Specifically, if the distance between a newly computed intersection point and an active voxel vertex is smaller than a specified value, this intersection point is deemed on the vertex whether it is inside or outside the cube. The coordinates of the intersection point are then replaced by those of the vertex, whose state is set to false in the meantime.

## 4. Representation of Generalized Cutter

A generalized cutting tool can be represented by an implicit surface function, which is used to determine the geometric relationship between an arbitrary 3D point and the cutter surface by simply putting its coordinates into the function. A negative value means the point is inside the cutter, a positive value means the point is outside the cutter, and zero means the point is on the cutter surface. The use of an implicit function is geometrically precise, and its programming is also very easy and straightforward.

According to the definition of an automatically programmed tool, the generalized cutter geometry, as shown in Figure 3, can be described fully using the following geometric parameters [25]: $d$ is the cutter diameter, twice the radial distance between the tool axis and the intersection between lower and upper line segments; $r$ is the radius of the corner torus; $e$ is the radial distance between the center of the corner circle and the tool axis; $f$ is the height of the center of the corner circle measured from the tool tip; $\alpha$ is the angle between the lower line segment and the cutter bottom line, $0° \leq \alpha < 90°$; $\beta$ is the angle between the cutter side and the tool axis, $-90° < \beta < 90°$; $h$ is the overall cutting edge length.

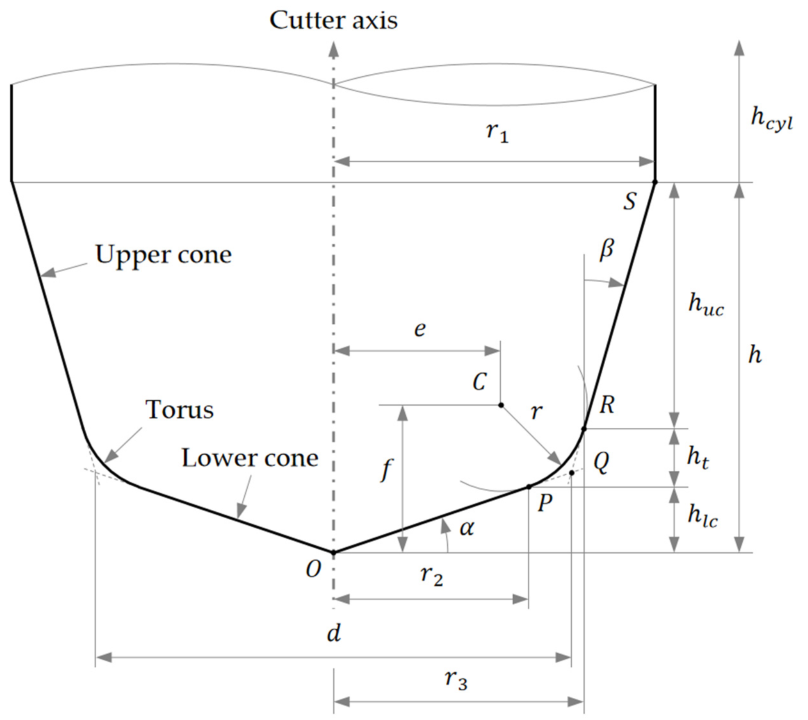


**Figure 3.** Geometric definition of generalized cutting tool.

Based on these user-specified independent parameters, several dependent parameters also need to be determined for the construction of the implicit function:

$$r_1 = d/2 + (h - (d\tan\alpha)/2)\tan\beta \tag{1}$$

$$r_2 = \left(u + \sqrt{u^2 - 4\cos^2\alpha(e^2 + f^2 - r^2)}\right)/2 \quad \text{with} \quad u = 2\cos^2\alpha(e + f\tan\alpha) \tag{2}$$

$$r_3 = \begin{cases} d/2 & \beta = 0 \\ e + \frac{\left(v\sin(2\beta) + \sqrt{v^2\sin^2(2\beta) - 4(v^2 - r^2)\sin^2\beta}\right)}{2} & \beta \neq 0 \end{cases} \quad \text{with} \quad v = (r_1 - e)\cot\beta - h + f \tag{3}$$

$$h_{uc} = \begin{cases} h - f + \sqrt{r^2 - (r_3 - e)^2} & \beta = 0 \\ (r_1 - r_3)\cot\beta & \beta \neq 0 \end{cases} \tag{4}$$

$$h_{lc} = r_2 \tan\alpha \tag{5}$$

$$h_t = h - h_{uc} - h_{lc} \tag{6}$$

The implicit function of a universal cutter consists of four standard components with arbitrary axes. A lower cone, a corner torus, and an upper cone represent the cutting edge, and a cylinder represents the cutter shank. Then, the implicit functions of these four components can be expressed as follows:

$$F(x,\ y,\ z) = \left\{ \boldsymbol{u}\cdot\boldsymbol{u} - \boldsymbol{m}\cdot\boldsymbol{u} - \begin{array}{cc} D_{cyl} & \boldsymbol{m}\cdot\boldsymbol{u} \geq h \\ D_{uc} & h_{lc} + h_t < \boldsymbol{m}\cdot\boldsymbol{u} < h \\ D_t & h_{lc} < \boldsymbol{m}\cdot\boldsymbol{u} \leq h_{lc} + h_t \\ D_{lc} & 0 < \boldsymbol{m}\cdot\boldsymbol{u} \leq h_{lc} \\ D_{bottom} & \boldsymbol{m}\cdot\boldsymbol{u} = 0 \end{array} \right.$$

$$\text{with} \begin{cases} D_{cyl} = {r_1}^2 \\ D_{uc} = (r_1 - (h - \boldsymbol{m}\cdot\boldsymbol{u})\tan\beta)^2 \\ D_t = \left(e + \sqrt{r^2 - (f - \boldsymbol{m}\cdot\boldsymbol{u})^2}\right)^2 \\ D_{lc} = \begin{cases} {r_2}^2 & \alpha = 0 \\ (\cot\alpha\,\boldsymbol{m}\cdot\boldsymbol{u})^2 & \alpha \neq 0 \end{cases} \\ D_{bottom} = \begin{cases} 0 & \alpha \neq 0 \\ \left((\boldsymbol{u}\cdot\boldsymbol{u} - \boldsymbol{m}\cdot\boldsymbol{u}) + {r_2}^2 - \left|(\boldsymbol{u}\cdot\boldsymbol{u} - \boldsymbol{m}\cdot\boldsymbol{u}) - {r_2}^2\right|\right)/2 & \alpha = 0 \end{cases} \end{cases} \tag{7}$$

where $\boldsymbol{m}$ is the unit vector along the cutter axis, and $\boldsymbol{u}$ is the vector from the tool tip to the 3D point being considered.

The use of the implicit surface function aims to detect the contact conditions of a target voxel with the cutter surface for workpiece update by inputting the coordinates of all its vertices into the implicit function. A voxel with all vertices inside the tool is an internal voxel that should be removed, a voxel with all vertices outside the tool is an external voxel that should be preserved, and a voxel with vertices partially inside the tool is an intersecting voxel, which should be decomposed for further updates.

## 5. Multi-Axis Milling Simulation

The present work employs the method of tool path sampling that takes tool instances along a tool path at a regular interval in order to construct the machined part surface and compute the instantaneous contact region. This method can be applied to a generalized milling tool and any kind of tool path, and it has been widely used for the generation of the cutter-swept volume in many works.

The simulation process proposed by this research is outlined in Figure 4. At each sampled cutter location, the in-process workpiece voxel model is updated by continuously subdividing coarser voxels that intersect the tool instance and removing any cell falling into the interior of the cutter. For each bottom-level intersecting voxel, the edges straddling the cutter surface need to be identified in order to compute and update their intersection points. Meanwhile, the states of all affected vertices need to be updated in time.

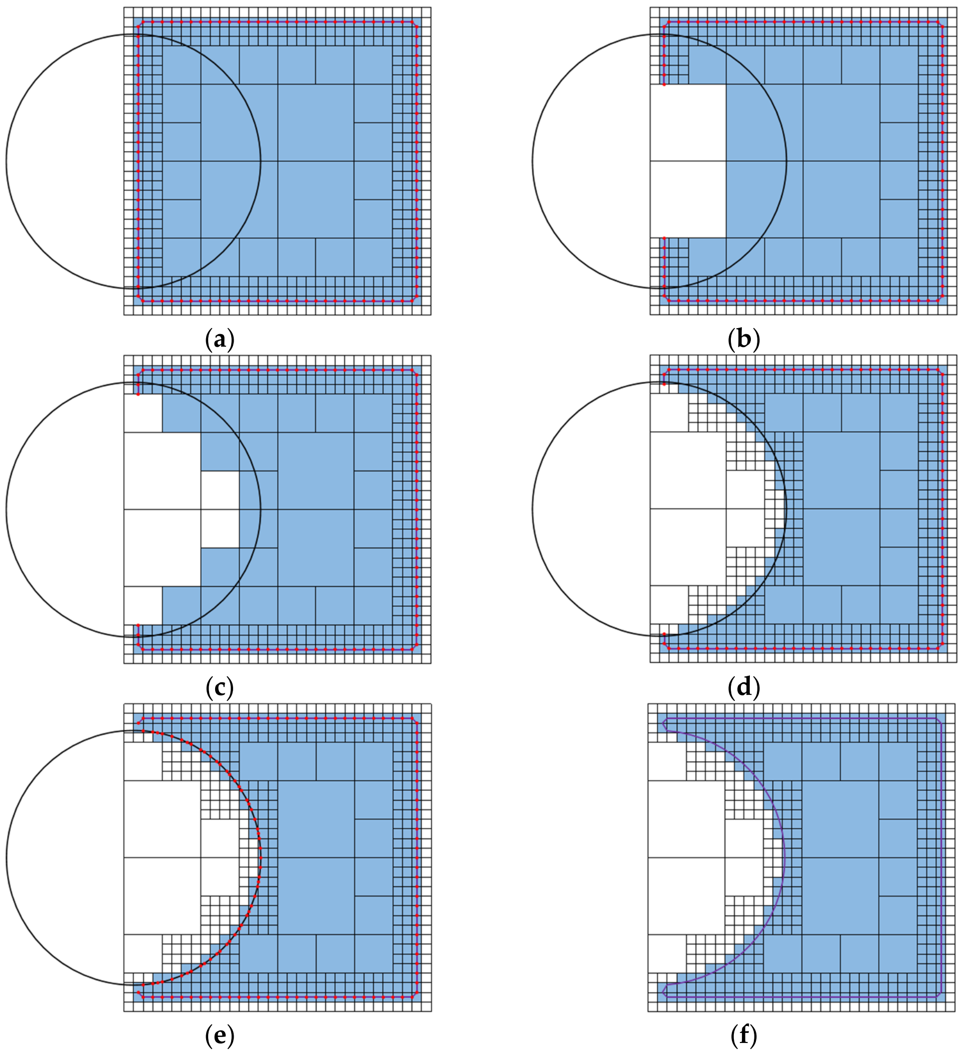


**Figure 4.** Workpiece update based on the tri-level grid: (**a**) blank model and cutting tool at current cutter location; (**b**) level-1 voxel update; (**c**) level-2 voxel update; (**d**) level-3 voxel update; (**e**) edge intersection point update; (**f**) workpiece surface reconstruction.

### *5.1. Solid Voxelization of Blank*

The proposed algorithm takes the triangle mesh representation of the blank as input because it is relatively simple to create a water-tight solid voxel model, and the intersection between a voxel edge and a planar triangle is easy to compute. In addition, a triangle mesh can be readily obtained from a CSG or B-rep model and easily rendered for visualization. Note that a collection of discrete triangles is utilized to approximate a solid surface by the triangle mesh model; therefore, the approximation error is inevitable during transformation. However, such an error can be mitigated by reducing the average triangle size.

As illustrated in Figure 5, the solid voxelization of the input shape is accomplished by two major procedures: surface voxelization and 3D filling. For the first step, the level-1 voxels intersecting a target triangular face are identified by traversing all voxel candidates inside the axis-aligned bounding box of the triangle. Then, each intersecting voxel is subdivided into a level-2 subgrid, where each intersecting voxel also needs to be activated and subdivided into a level-3 subgrid, with the finest intersecting voxels being activated. For each bottom-level intersecting voxel, all intersection points between its edges and one or more triangles need to be computed for the first time. Once the surface voxel model of the blank is achieved, all internal voxels of the triangle mesh at three levels need to be activated by counting the intersections between the triangle mesh and the ray emitted from the interior of the voxel. An odd number of intersections indicates an internal voxel that needs to be activated, and all its finer voxels, if they exist, need further detection; meanwhile, an even number of intersection points indicates an external voxel.

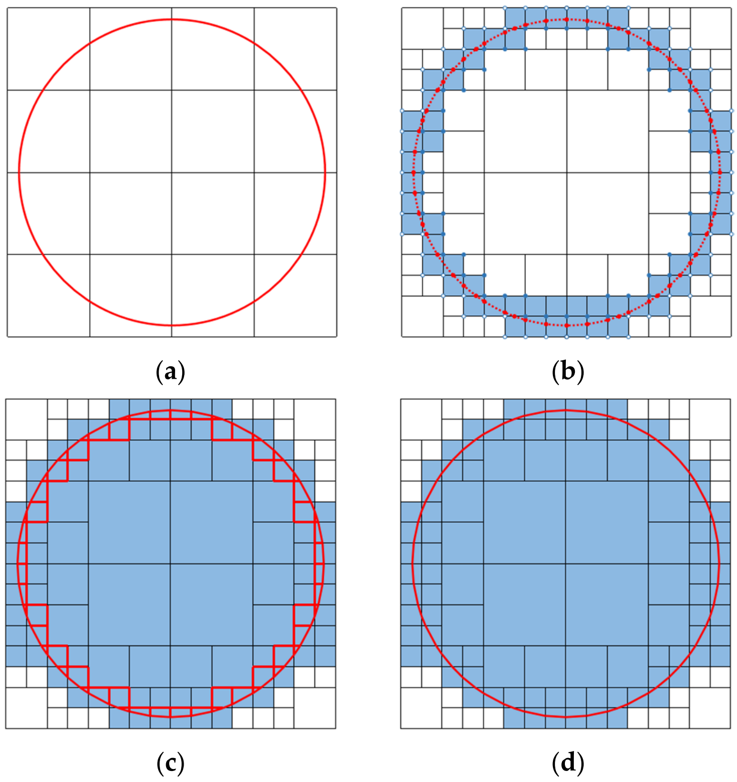


**Figure 5.** Solid voxelization of the blank: (**a**) triangle mesh model of the blank; (**b**) surface voxelization; (**c**) surface reconstruction and 3D filling; (**d**) solid voxel model.

## *5.2. Workpiece Geometry Computation*

In order to reconstruct the surface of the final machined part, the in-process workpiece model needs to be updated accurately at each sampled cutter location. This multi-level update is accomplished by three levels of voxel updates and one level of edge intersection point update. The level-1 voxel update aims to carry out the fast bulk material removal from the workpiece, which needs to identify all coarse voxels that may possibly intersect the tool instance. A simple method is to traverse every voxel inside the axis-aligned bound box of the tool instance; however, this approach may introduce a large number of unnecessary detection computations, as many voxels are clearly far away from the cutter surface. Therefore, an oriented bounding cylinder is employed by this work, whose internal space can provide the minimal voxel candidates that need to be checked for updates. Note that the interior of the bounding cylinder can be extracted very efficiently by implementing the direct voxel tracing algorithm [24].

Theoretically, the geometric relationship between an affected coarse voxel and the cutter surface needs to be determined by checking all its vertices, which is practically inefficient since some voxels definitely inside or outside the cutter can be quickly ruled out by a relaxed proximity check [27]. Assuming $v_c$ is the coarse-level voxel size and the distance between the voxel center and the cutter surface is $d_{vc}$, then a voxel is deemed to be a near-field voxel, or a possible intersecting voxel, for the tool instance if $d_{vc} < \sqrt{3}v_c/2$, where $\sqrt{3}v_c$ is the thickness of this near-field region of the cutter surface which lies in the middle of this region, as shown in the dashed boundaries in Figure 6. Once a near-field voxel is recognized, its contact conditions are detected accurately by putting all its vertices into the implicit function of a generalized cutting tool.

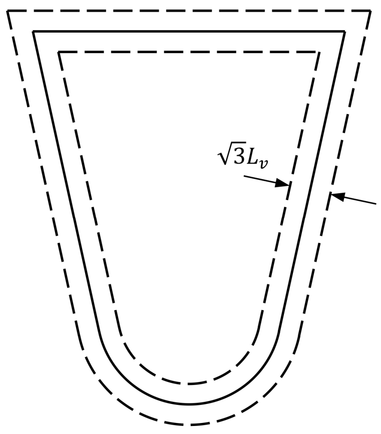


**Figure 6.** Near-field of a tool instance.

Each possible intersecting voxel at the top level is subdivided into a finer level-2 subgrid, where a voxel definitely inside the tool instance is deleted directly, and an intersecting voxel is decomposed into the finest level-3 subgrid. Within each level-3 subgrid, an internal voxel of the cutter is removed, while both voxel edge intersection points and vertex states need to be updated immediately inside each intersecting voxel at the bottom level. Specifically, for an edge with both ends inside the cutter, its two vertex states are set to false, while for an edge with one end inside the cutter and the other outside the cutter, its intersection with the tool instance needs to be updated appropriately. The pseudocode of this multi-level update process is outlined below Algorithm 1:

**Algorithm 1.** Workpiece Geometry Computation

```
for each level-1 voxel v1 in oriented bounding cylinder of tool instance
  if v1 inside cutter
    deactivate v1
  else if v1 intersects cutter
    subdivide v1 into level-2 subgrid
    for each level-2 voxel v2 in subgrid
      if v2 inside cutter
        deactivate v2
      else if v2 intersect cutter
        subdivide v2 into level-3 subgrid
        for each level-3 voxel v3 in subgrid
          if v3 inside cutter
            deactivate v3
          else if v3 intersects cutter
            for each voxel edge of v3
              if both vertices inside cutter
                deactivate two vertices
              else if one vertex inside cutter, the other one outside cutter
                if both vertices outside cutter before
                  update edge intersection point
                  compute cutting depth and immersion angle
                else if both vertex states stay the same as before
                  if edge intersection point inside cutter
                    update edge intersection point
                    compute cutting depth and immersion angle
```

## *5.3. CWE Extraction*

During the update process of the workpiece at the subvoxel level, the intersection points between the edges of the finest intersecting voxels and the tool instance need to be computed and updated appropriately, which have managed to depict the instantaneous contact surface. Therefore, an accurate and efficient way to compute this region is to fuse these discrete intersection points directly, as shown in Figure 7. In order to obtain the CWE diagram suitable for cutting force prediction, the cutter is uniformly discretized into a group of cylindrical disks first, and their thickness is set equal to the finest voxel edge length for balanced computational performance. Then, for each newly computed intersection point, its cutting depth $d_c$ measured from the tool tip and immersion angle measured from the $y$-axis of the tool coordinate system (TCS) are calculated, and the newly determined angle is put into all elementary disks that fall into the range of $\left[d_c - \sqrt{3}v_f/2, d_c + \sqrt{3}v_f/2\right]$, where $v_f$ is the finest voxel size. After all intersection points are determined, the discrete immersion angles stored in each disk are fused by connecting any two neighboring angles with an interval smaller than $2\tan^{-1}\left(\sqrt{3}v_f/\left(2r_d - \sqrt{3}v_f\right)\right)$, where $r_d$ is the radius of the cutting disk. Finally, each group of connected immersion angles in a disk forms a CWE arc, whose first and last angles correspond to the entry and exit angles, respectively. This fusion process achieves the desired resolution and precision of the computed result at a subvoxel level while introducing a very limited number of extra computations.

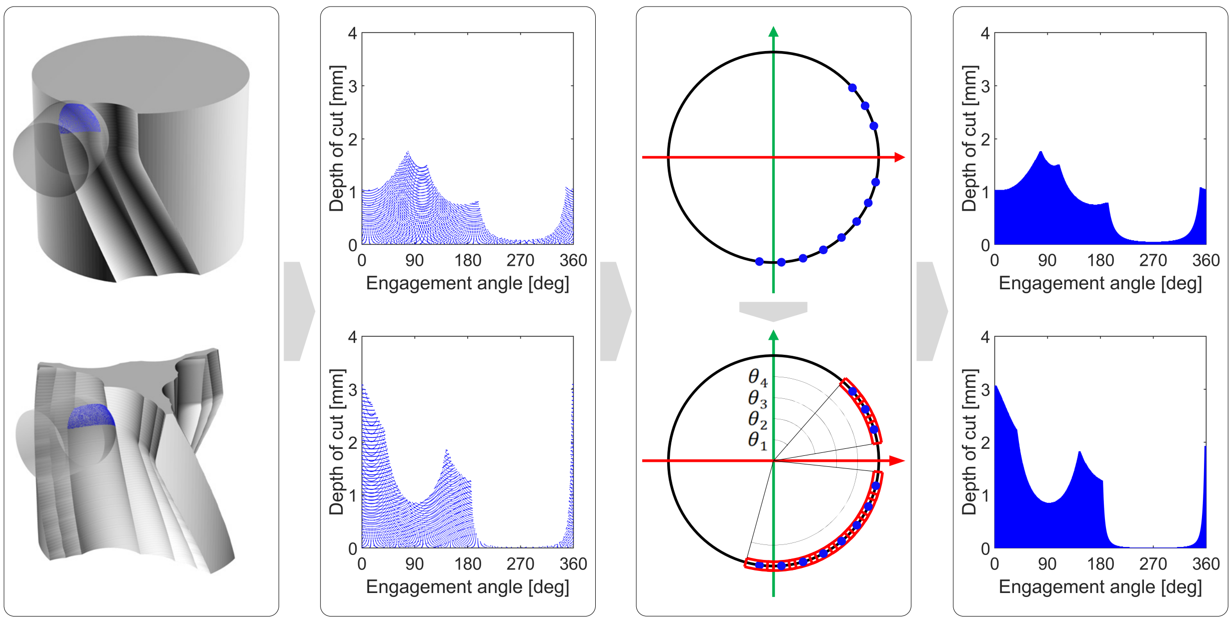


**Figure 7.** Illustration of CWE determination.

## *5.4. CWE Trimming by Envelope Theory*

In this work, the computed CWE arc may be slightly larger than its theoretical counterpart because only internal voxels are deleted for workpiece update. Therefore, the so-called feasible contact arc (FCA) is employed in this work to trim the computed result [3], as shown in Figure 8. The grazing curve is a collection of points on the cutter surface that remain on the enveloped surface during the milling process. The entire cutter surface is then partitioned into two parts by this grazing curve. The part facing the cutter moving direction is called the front-facing part, and it is responsible for removing the workpiece material, either in part or in whole; the other part will never be involved. Then, each sliced circle, which is obtained by slicing the cutter with a plane perpendicular to its axis, is divided into two arcs by its intersection points, namely grazing points, with the grazing curve. The arc within the front-facing part is called the feasible contact arc, which defines the largest arc that may engage with the workpiece.

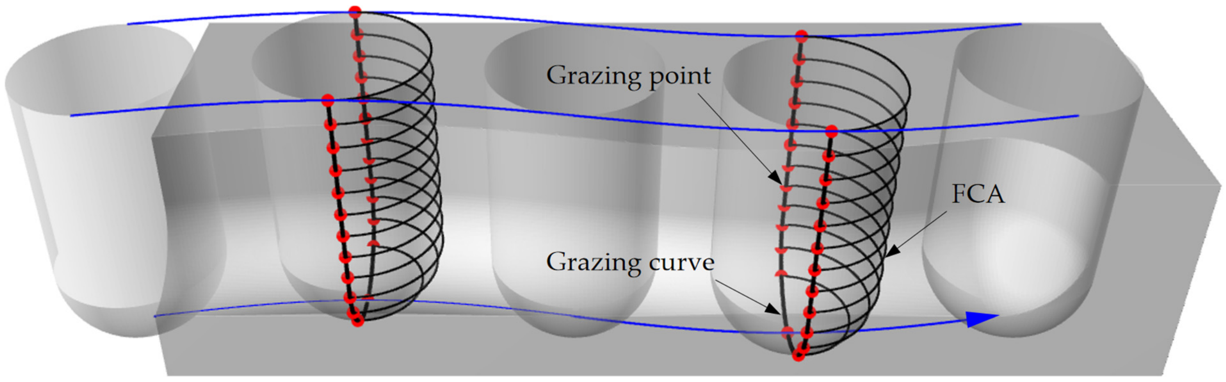


**Figure 8.** Illustration of grazing point and feasible contact arc on a moving tool.

Assume the trajectory of the tool tip can be described by a general curve $\boldsymbol{P}(t)$ in 3D space, as illustrated by Figure 9; then, the tool coordinate system located at the tool tip can be determined as follows:

$$\begin{cases} \boldsymbol{o}_t = \boldsymbol{P}(t) \\ \boldsymbol{x}_t = \boldsymbol{y}_t \times \boldsymbol{z}_t \\ \boldsymbol{y}_t = \frac{\boldsymbol{z}_t \times \boldsymbol{P}'(t)}{\|\boldsymbol{z}_t \times \boldsymbol{P}'(t)\|} \\ \boldsymbol{z}_t = \boldsymbol{A}(t) \end{cases} \tag{8}$$

where $\boldsymbol{A}(t)$ is the unit axis vector of the cutter and $\boldsymbol{P}'(t)$ is the tool tip velocity, which can be computed based on the CL data file:

$$\boldsymbol{P}'(t_i) = \frac{\boldsymbol{C}(t_{i+1}) - \boldsymbol{C}(t_i)}{\Delta t} \tag{9}$$

where $\boldsymbol{C}(t_i)$ is the tool tip position, and $\Delta t$ is equal to one.

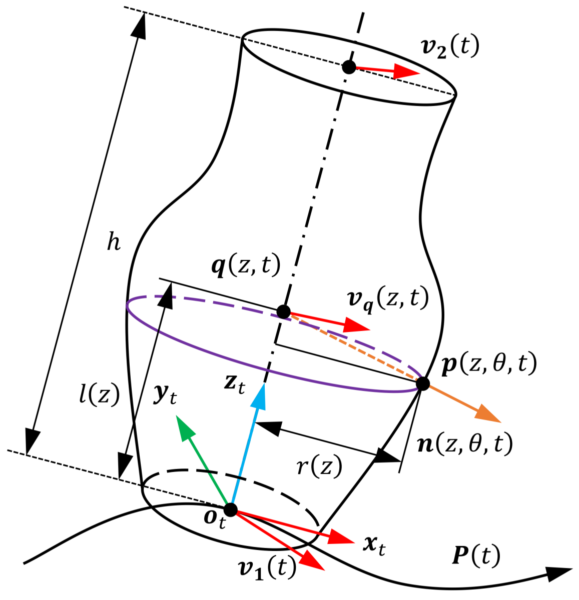


**Figure 9.** The enveloping characteristic of a moving tool.

According to research [3], a 3D point located on the cutting tool surface can be expressed as follows:

$$\boldsymbol{p}(z,\theta,t) = \boldsymbol{P}(t) + z\boldsymbol{A}(t) + r(z)\sin\theta\boldsymbol{x}_t + r(z)\cos\theta\boldsymbol{y}_t \tag{10}$$

where $z$ is the axial height with respect to the tool tip, $r(z)$ represents the cutter radius along the cutter axis, and $\theta$ is the oriented angle measured from the $y$-axis of TCS.

Furthermore, the intersection point $\boldsymbol{q}$ between the normal vector crossing point $\boldsymbol{p}$ and the cutter axis is expressed as follows:

$$\begin{gathered}\boldsymbol{q}(z,t) = \boldsymbol{P}(t) + l(z)\boldsymbol{A}(t) \\ \text{with } l(z) = r(z)r'(z) + z \\ r'(z) = \begin{cases} \begin{cases} 0 & \alpha = 0 \\ \cot\alpha & \alpha \neq 0 \end{cases} & 0 < z \leq h_{lc} \\ \frac{f-z}{\sqrt{r^2-(f-z)^2}} & h_{lc} < z \leq h_{lc} + h_t \\ \tan\beta & h_{lc} + h_t < z < h \end{cases}\end{gathered} \tag{11}$$

Thus, the unit normal vector crossing point $\boldsymbol{p}$ can be written as

$$\boldsymbol{n}(z,\theta,t) = \frac{\boldsymbol{p}(z,\theta,t) - \boldsymbol{q}(z,t)}{\|\boldsymbol{p}(z,\theta,t) - \boldsymbol{q}(z,t)\|} \tag{12}$$

According to the envelope theory in [13], a grazing point should meet the requirement below:

$$\boldsymbol{v}_q(z,t)\cdot\boldsymbol{n}(z,\theta,t) = 0 \tag{13}$$

where $\boldsymbol{v}_q$ is the velocity of point $\boldsymbol{q}$ on the cutter axis, which can be computed by linearly interpolating the velocities of two endpoints on the cutter axis:

$$\boldsymbol{v}_q(z,t) = \boldsymbol{v}_1(t)\frac{h-l(z)}{h} + \boldsymbol{v}_2(t)\frac{l(z)}{h} \tag{14}$$

where $h$ is the overall cutting-edge length, and the velocities $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ can be easily determined by connecting the endpoints at two adjacent cutter locations.

Finally, the equation below can be obtained by substituting Equations (12) and (14) into Equation (13):

$$\begin{gathered}A_0\cos\theta + A_1\sin\theta = A_2 \\ \text{with } \begin{cases} A_0 = r(z)\boldsymbol{v}_q(z,t)\cdot\boldsymbol{y}_t \\ A_1 = r(z)\boldsymbol{v}_q(z,t)\cdot\boldsymbol{x}_t \\ A_2 = r(z)r'(z)\boldsymbol{v}_q(z,t)\cdot\boldsymbol{A}(t) \end{cases}\end{gathered} \tag{15}$$

Based on this equation, the grazing points, and hence the feasible contact arc, on each sliced circle can be represented by $\theta$:

$$\begin{cases} \theta_1 = 2\tan^{-1}\frac{A_1+\sqrt{{A_0}^2+{A_1}^2-{A_2}^2}}{A_0+A_2} \\ \theta_2 = 2\tan^{-1}\frac{A_1-\sqrt{{A_0}^2+{A_1}^2-{A_2}^2}}{A_0+A_2} \end{cases} \tag{16}$$

It should be noted that the above equation can be built only if ${A_2}^2 \leq {A_0}^2 + {A_1}^2$. Then, any point on the sliced circle can possibly be a grazing point under the condition of $A_2 < -\sqrt{\left({A_0}^2 + {A_1}^2\right)}$, but cannot be a grazing point under the condition of $A_2 > \sqrt{\left({A_0}^2 + {A_1}^2\right)}$.

## 6. Case Studies

In this section, a series of practical machining cases with sufficient complexity have been conducted in order to compare the computational performance of the proposed method with that of the tri-dexel representation and the two-level grid in different aspects. The tri-dexel model is used as a comparison benchmark because it has been recognized as providing the best combination of computational accuracy, efficiency, and flexibility among all modeling methods. Table 1 lists the geometric information needed to implement these two five-axis milling cases.

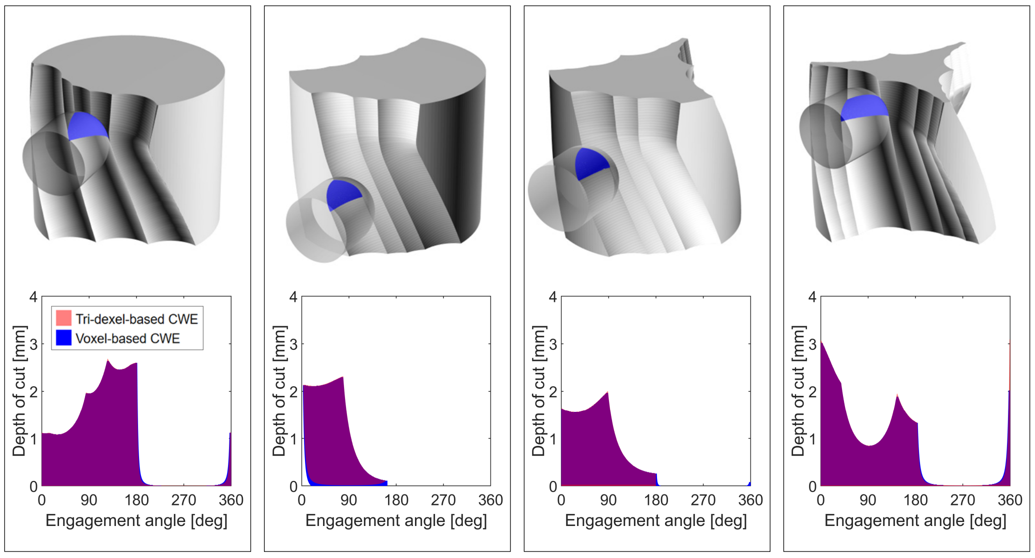
**Table 1.** Case summary.

| Cutter Type | Ball-End Mill | Bull-End Mill | Taper Ball-End Mill |
|---|---|---|---|
| Cutter parameter (mm) | $d = 10, r = 5, e = 0$<br>$f = 5, \alpha = \beta = 0°, h = 50$ | $d = 10, r = 2.5, e = 2.5$<br>$f = 2.5, \alpha = \beta = 0°, h = 50$ | $d = 9.6569, r = 5, e = 0$<br>$f = 5, \alpha = 0°, \beta = 2°, h = 50$ |
| No. of CL points | 33,456 | 62,600 | 239,290 |
| Finest voxel size (mm) | 0.01 | 0.05 | 0.05 |

### *6.1. Accuracy Comparison*

The computational performance of a voxel-based method highly depends on the voxel size at the bottom level or the ratio of the cutter diameter to the finest voxel size once the cutting tool is selected in the first place. To reveal the computational performance of the proposed method under the condition of extreme-scale resolution, the diameter of the ball-end mill in the first case is thus set to 1000 times the finest voxel size in the initial process. More importantly, a large ratio can help demonstrate how the simulation efficiency of a voxel-based method is influenced by its fine-level branching factor, which can be adjusted in a wider range.

To demonstrate the accuracy of the proposed method, four groups of CWE maps with IPW mesh models of the first case are shown in Figure 10, where the engagement regions computed by the proposed method are drawn in blue while the results determined by the tri-dexel method are drawn in red for comparison. The good similarity, as well as the high complexity of the contact areas, solidly demonstrates the high precision and robustness of the developed method.



(**a**) (**b**) (**c**) (**d**)

**Figure 10.** CWE comparison of Case 1: (**a**) CWE maps at CL#1220; (**b**) CWE maps at CL#2025; (**c**) CWE maps at CL#3710; (**d**) CWE maps at CL#5148.

The second case is accomplished by a bull-end mill and a taper ball-end mill for its roughing and finishing processes, respectively. Two groups of CWE maps from the roughing process and another two groups from the finishing process are shown in Figure 11, where the 3D contact regions can be seen clearly in close-up images. The good agreement further demonstrates the sufficient accuracy of the present method.

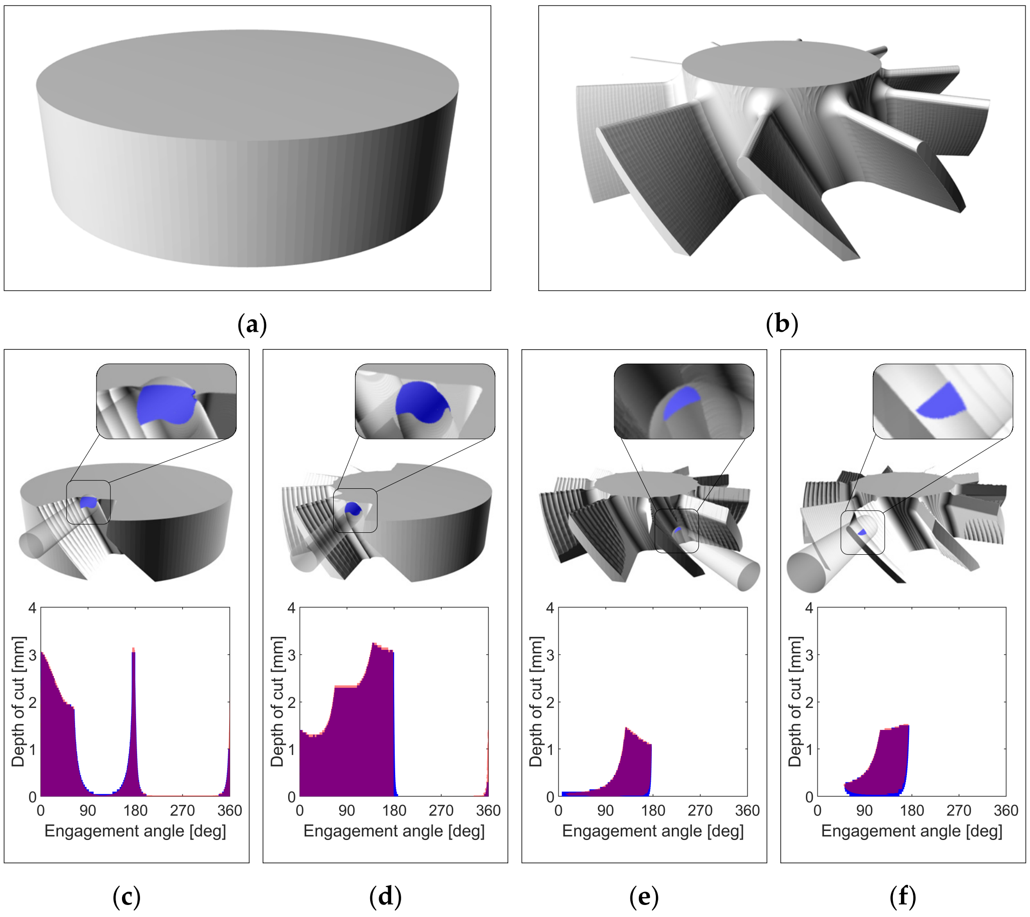


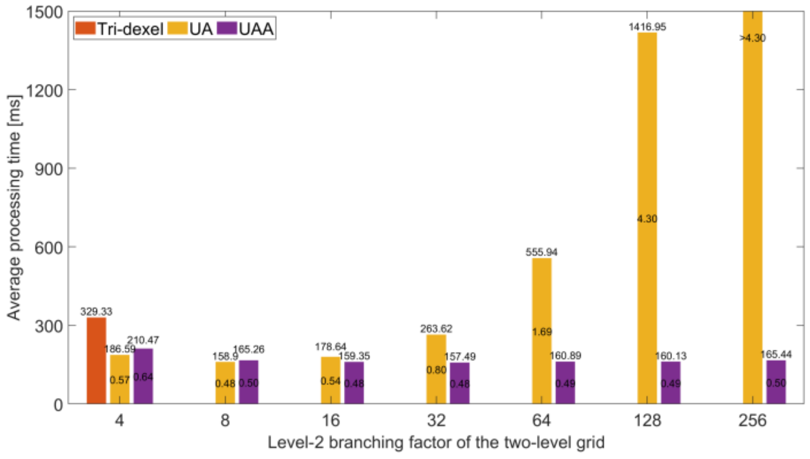
**Figure 11.** CWE comparison of Case 2: (**a**) blank mesh model; (**b**) machined part; (**c**) CWE maps at CL#6000; (**d**) CWE maps at CL#30,440; (**e**) CWE maps at CL#70,600; (**f**) CWE maps at CL#97,040.

During the cutting process, the sampled tool instances along a tool path are used by this research to approximate the exact cutter-swept volume for workpiece update. As a result, some sampling scallops are left between adjacent tool instances, which can be observed in Figure 10. These scallops, however, are not evident since the distance between the original cutter location points has already been small enough.

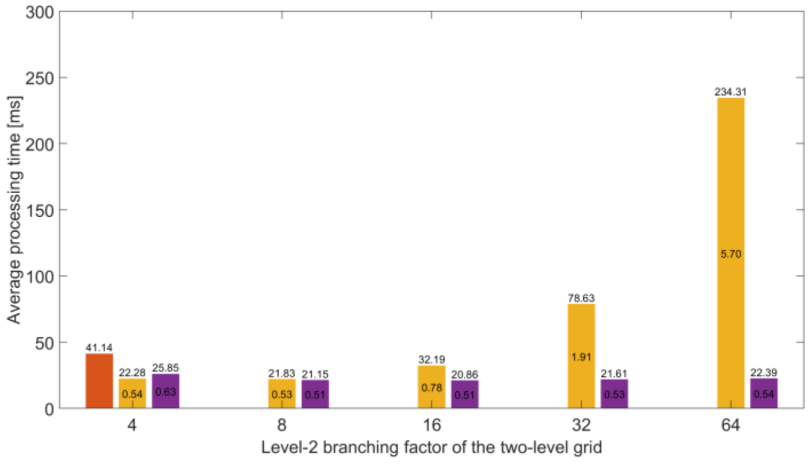
Generally, the triangle meshes of the in-process workpiece and the machined part created by the proposed method are all of good quality and thus useful for real-time rendering and visual verification in virtual machining. More importantly, the obtained meshes will become useful when performing a quantitative comparison against their desired models for identifying potential machining errors such as gouging and undercuts. Such good quality, in this work, is due to the small bottom-level voxel size, which is directly associated with the available grid resolution and hence the memory usage. The required memory is mainly used for the storage of the bottom-level voxels with edge intersection points, the containers, and the obtained mesh model. Therefore, the largest grid resolution or workpiece dimensions allowed will be limited by the available memory space of the hardware platform in use.

## *6.2. Efficiency Comparison*

The average processing time, obtained by dividing the overall simulation time by the total number of sampled cutter locations, is listed above each column in Figure 12 for both cases. To compare the modeling methods quantitatively, the execution time of the voxel-based method is divided by that of the tri-dexel method, and the computed result is listed in each column as the efficiency ratio. A smaller ratio indicates a larger improvement in efficiency. In the literature, even though vector modeling is able to accomplish workpiece updates efficiently by implementing clipping operations, it still lacks the ability to minimize these calculations since the workpiece vectors are always evenly spaced. This issue becomes more severe for a tri-dexel model because dexels along all three orthogonal directions need to be considered for possible calculations, which will greatly increase the processing time. Comparatively, because of the two levels of adaptive refinement in spatial decomposition, the proposed method is able to conduct the update process via the collective volume removal by batch processing at the coarse level before performing the finest computations, which effectively accelerates the bulk material removal simulation. With the fast removal of the workpiece material and the identification of near-field voxels, the coarse voxels in the vicinity of the machined part surface can be reached very quickly for finer updates. Then, the time-consuming voxel update at the finest level and the edge intersection point update at the subvoxel level are accomplished only inside the affected coarse-level surface voxels, which reduces the computation time significantly.



(**a**)



(**b**)

**Figure 12.** Efficiency comparison: (**a**) efficiency comparison of Case 1; (**b**) efficiency comparison of Case 2.

Figure 12 also illustrates how the computational efficiency of the two-level grid is affected by its level-2 branching factor. Note that the bottom-level voxel size remains unchanged after being set initially. Thus, different level-2 branching factors correspond to different level-1 voxel sizes. It can be observed that the computation speed of the two-level grid reaches its highest level when the level-2 branching factor is around 8, but it becomes unacceptable once the factor exceeds 32. In contrast, the fluctuations in the computation time of the tri-level grid are rather small, which solidly demonstrates its strong ability to provide high and stable computation speed without being affected by the level-2 branching factor. Essentially, a large level-2 branching factor indicates a large number of top-level voxels will be accessed multiple times for update since a tool path is sampled very closely, which will lead to a sharp increase in traversal operations at the bottom level. As illustrated in Figure 13, where edge intersection computation is neglected, to remove all cutter internal voxels inside level-1 cell $A$ at the current location based on the two-level grid (step 1), all its finer voxels need to be accessed for deactivation, and each access indicates a proximity check and possible accurate detection. By contrast, all level-3 voxels inside cell $B$ are excluded from deactivation since this mid-level voxel has already been deleted during the level-2 update process of the tri-level grid. Once the cutter moves to the next location (step 2), all finer voxels in cell $A$, whose vertices are still inside the tool instance partially, need to be accessed again for the two-level grid, while the finer voxels in cell $B$ and cell $C$, which just falls into the cutter's interior are skipped when it comes to the tri-level grid. As can be seen, the computational efficiency of the traditional two-level grid is adversely affected by a large level-2 branching factor due to its repeated traversal operations, which, however, can be significantly reduced by adding another level of adaptive refinement.

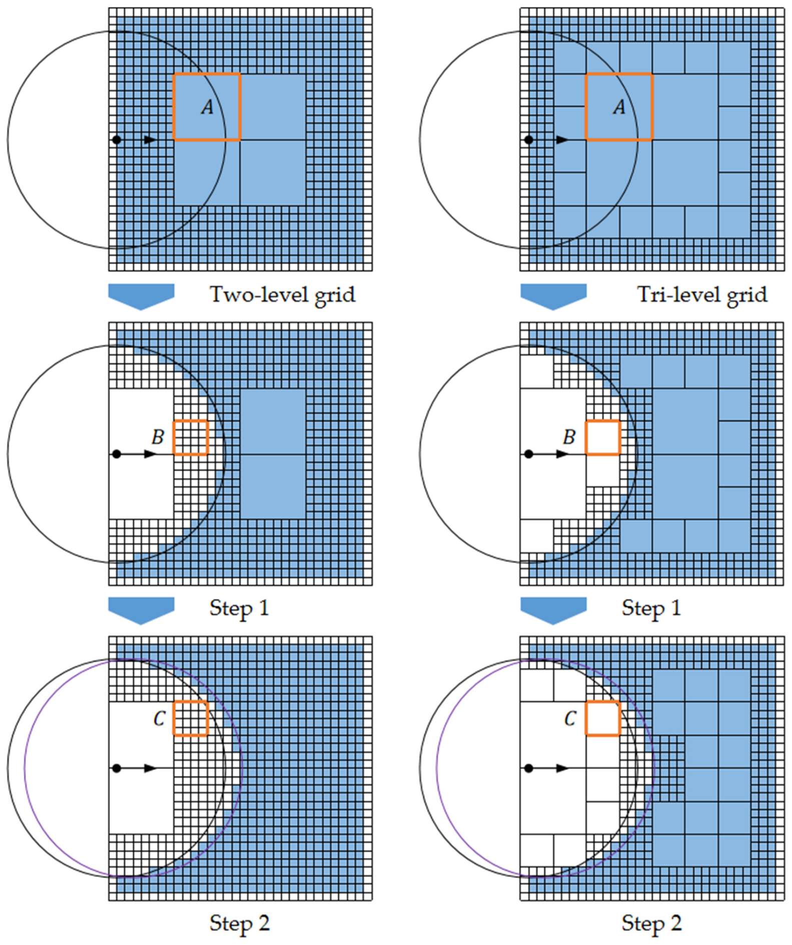


**Figure 13.** Comparison of the update processes based on different voxel grids.

Figure 14 shows the influence of the level-3 branching factor on the computational efficiency of the tri-level grid quantitatively with three different types of cutters. Since the level-3 voxel size remains unchanged in each case, the level-3 branching factor is reflected by the level-2 voxel size. It can be seen that the computation time of the tri-level grid reaches its lowest point when the level-3 branching factor becomes four or eight, and it rises quickly as the factor keeps increasing, also because of the rapidly increasing bottom-level traversal operations. Therefore, in order to obtain the maximum benefit from the tri-level grid, the level-3 branching factor should be set to four or eight in practice.

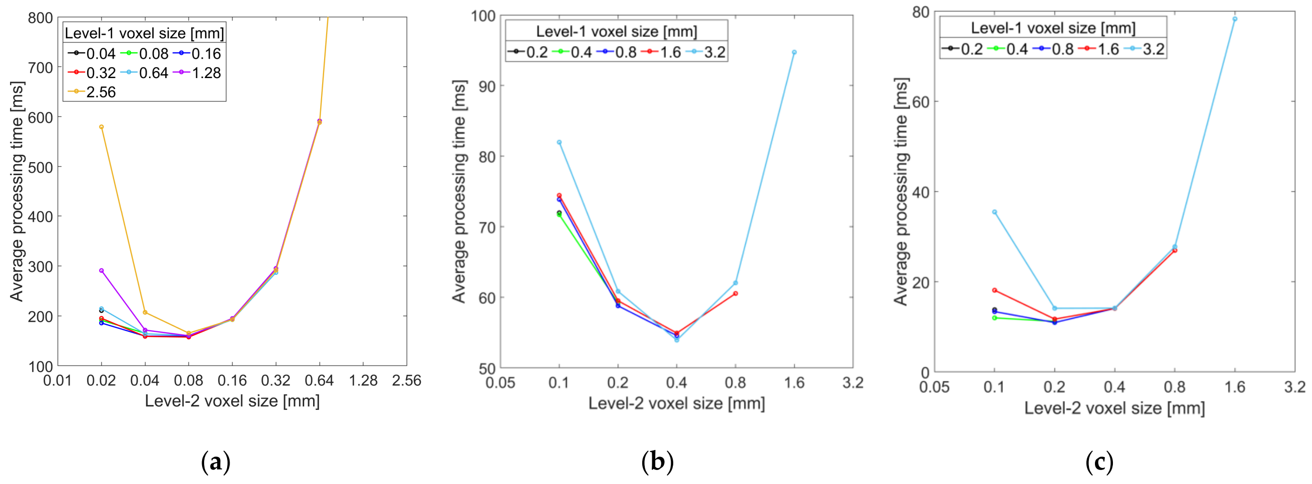


**Figure 14.** Efficiency characteristic of the tri-level grid: (**a**) average processing time by ball-end mill; (**b**) average processing time by bull-end mill; (**c**) average processing time by taper ball-end mill.

## 7. Conclusions and Future Work

This work presents an effective NC simulation method that can accurately compute machined part geometry and instantaneous CWE regions with higher and more stable computational efficiency under extreme-scale resolutions compared with the traditional two-level grid. The proposed method exploits its potential in high-precision machining by (1) eliminating the adverse impact of a large fine-level branching factor on the computational efficiency by adding another level of adaptive refinement in space decomposition and (2) calculating the immersion angles by fusing the edge intersection points, and trimming the computed results with feasible contact arcs determined by the envelope theory. Furthermore, in order to take full advantage of the tri-level grid, the mid-level subdivision factor should be selected appropriately according to its efficiency characteristic figures.

As discussed in case studies, minor sampling scallops are present on the workpiece surface due to the use of sampled tool instances. In order to obtain the mesh model with sufficient smoothness, the sampling interval needs to remain small, but this will hurt the overall simulation efficiency. Further study is thus needed to optimize the tool path sampling interval according to the fine voxel grid spacing or even eliminate the need to sample tool paths by directly calculating the intersection points between voxel edges and approximate linear cutter-swept volumes. Such a study will help researchers seek a balance between surface quality and computational efficiency in NC simulation.

**Author Contributions:** Conceptualization, methodology, software, validation, formal analysis, investigation, resources, data curation, writing—original draft preparation, visualization, Z.N.; writing—review and editing, supervision, project administration, funding acquisition, Y.Z. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the National Key Research and Development Program for Robotics Serialized Harmonic Reducer Fatigue Performance Analysis and Prediction and Life Enhancement Technology Research, grant number 2017YFB1300603.

**Data Availability Statement:** Data are contained within the article.

**Acknowledgments:** The authors thank the reviewers for their comments and the editors for all their work.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Lasemi, A.; Xue, D.; Gu, P. Recent development in CNC machining of freeform surfaces: A state-of-the-art review. *Comput. Aided Des.* **2010**, *42*, 641–654. [CrossRef]
2. Mali, R.A.; Gupta, T.V.K.; Ramkumar, J. A comprehensive review of free-form surface milling—Advances over a decade. *J. Manuf. Process.* **2021**, *62*, 132–167. [CrossRef]
3. Li, Z.-L.; Wang, X.-Z.; Zhu, L.-M. Arc-surface intersection method to calculate cutter-workpiece engagements for generic cutter in five-axis milling. *Comput. Aided Des.* **2016**, *73*, 1–10. [CrossRef]
4. Ma, H.; Liu, W.; Zhou, X.; Niu, Q.; Kong, C. High efficiency calculation of cutter-workpiece engagement in five-axis milling using distance fields and envelope theory. *J. Manuf. Process.* **2021**, *68*, 1430–1447. [CrossRef]
5. Yan, B.; Xu, G.; Lu, H.; Qin, S.; Zhu, C. Identification of milling information and cutter-workpiece engagement in five-axis finishing of turbine blades based on NURBS and NC codes. *J. Manuf. Process.* **2023**, *107*, 43–56. [CrossRef]
6. Dambly, V.; Rivière-Lorphèvre, É.; Ducobu, F.; Verlinden, O. Tri-dexel-based cutter-workpiece engagement: Computation and validation for virtual machining operations. *Int. J. Adv. Manuf. Technol.* **2024**, *131*, 623–635. [CrossRef]
7. Gao, S.; Duan, X.; Zhu, K.; Zhang, Y. Investigation of the tool flank wear influence on cutter-workpiece engagement and cutting force in micro milling processes. *Mech. Syst. Signal Process.* **2024**, *209*, 111104. [CrossRef]
8. Lin, M.; Wang, C.; Yue, T.; Guo, G.; Guan, W.; Shen, B. Deformation prediction in flank milling of thin-walled parts based on cutter-workpiece engagement. *J. Manuf. Process.* **2024**, *115*, 375–386. [CrossRef]
9. Li, X.; Lee, C.-H.; Hu, P.; Zhang, Y.; Yang, F. Cutter partition-based tool orientation optimization for gouge avoidance in five-axis machining. *Int. J. Adv. Manuf. Technol.* **2018**, *95*, 2041–2057. [CrossRef]
10. Ezair, B.; Elber, G. Automatic generation of globally assured collision free orientations for 5-axis ball-end tool-paths. *Comput. Aided Des.* **2018**, *102*, 171–181. [CrossRef]
11. Du, J.; Liu, P.; Zhi, H.; Ding, P. Global interference detection technology for five-axis machining of complex surfaces. *Int. J. Adv. Manuf. Technol.* **2019**, *102*, 4273–4287. [CrossRef]
12. Chiou, C.-J.; Lee, Y.-S. Swept surface determination for five-axis numerical control machining. *Int. J. Mach. Tools Manuf.* **2002**, *42*, 1497–1507. [CrossRef]
13. Gong, H.; Wang, N. Analytical calculation of the envelope surface for generic milling tools directly from CL-data based on the moving frame method. *Comput. Aided Des.* **2009**, *41*, 848–855. [CrossRef]
14. Yang, Y.; Zhang, W.; Wan, M.; Ma, Y. A solid trimming method to extract cutter-workpiece engagement maps for multi-axis milling. *Int. J. Adv. Manuf. Technol.* **2013**, *68*, 2801–2813. [CrossRef]
15. Boz, Y.; Erdim, H.; Lazoglu, I. A comparison of solid model and three-orthogonal dexelfield methods for cutter-workpiece engagement calculations in three- and five-axis virtual milling. *Int. J. Adv. Manuf. Technol.* **2015**, *81*, 811–823. [CrossRef]
16. Artetxe, E.; Olvera, D.; López de Lacalle, L.N.; Campa, F.J.; Olvera, D.; Lamikiz, A. Solid subtraction model for the surface topography prediction in flank milling of thin-walled integral blade rotors (IBRs). *Int. J. Adv. Manuf. Technol.* **2017**, *90*, 741–752. [CrossRef]
17. Chung, Y.C.; Park, J.W.; Shin, H.; Choi, B.K. Modeling the surface swept by a generalized cutter for NC verification. *Comput. Aided Des.* **1998**, *30*, 587–594. [CrossRef]
18. Tunç, L.T.; Ozkirimli, O.M.; Budak, E. Machining strategy development and parameter selection in 5-axis milling based on process simulations. *Int. J. Adv. Manuf. Technol.* **2016**, *85*, 1483–1500. [CrossRef]
19. Li, S.; Dong, Y.; Li, Y.; Li, P.; Yang, Z.; Landers, R.G. Geometrical simulation and analysis of ball-end milling surface topography. *Int. J. Adv. Manuf. Technol.* **2019**, *102*, 1885–1900.
20. Xiao, Y.; Ge, G.; Zeng, Z.; Feng, X.; Du, Z. An improved Z-MAP method based on the SQP algorithm for fast surface topography simulation of ball-end milling. *Int. J. Adv. Manuf. Technol.* **2023**, *128*, 1863–1878. [CrossRef]
21. Zhang, X.; Yu, T.; Wang, W. Modeling, simulation, and optimization of five-axis milling processes. *Int. J. Adv. Manuf. Technol.* **2014**, *74*, 1611–1624. [CrossRef]

22. Yousefian, O.; Balabokhin, A.; Tarbutton, J. Point-by-point prediction of cutting force in 3-axis CNC milling machines through voxel framework in digital manufacturing. *J. Intell. Manuf.* **2020**, *31*, 215–226. [CrossRef]
23. Nie, Z.; Feng, H.-Y. Integrated and efficient cutter-workpiece engagement determination in three-axis milling via voxel modeling. *Int. J. Adv. Manuf. Technol.* **2023**, *128*, 391–403. [CrossRef]
24. Nie, Z.; Feng, H.-Y. Efficient voxel-based workpiece update and cutter-workpiece engagement determination in multi-axis milling. *J. Manuf. Sci. Eng.* **2024**, *146*, 061003. [CrossRef]
25. Yau, H.-T.; Tsou, L.-S. Efficient NC simulation for multi-axis solid machining with a universal APT cutter. *J. Comput. Inf. Sci. Eng.* **2009**, *9*, 021001. [CrossRef]
26. Joy, J.; Feng, H.-Y. Frame-sliced voxel representation An accurate and memory-efficient modeling method for workpiece geometry in machining simulation. *Comput. Aided Des.* **2017**, *88*, 1–13. [CrossRef]
27. Joy, J.; Feng, H.-Y. Efficient milling part geometry computation via three-step update of frame-sliced voxel representation workpiece model. *Int. J. Adv. Manuf. Technol.* **2017**, *92*, 2365–2378. [CrossRef]
28. Lorensen, W.E.; Cline, H.E. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Comput. Graph.* **1987**, *21*, 163–169. [CrossRef]