

Enhanced Dual Contouring for Continuous Surface Reconstruction: A Novel Method in Real-Time Geometric Visualization for CNC Machining Simulation

Chang Su

Ocean University of China

Yong Han

`suchang5022@stu.ouc.edu.cn`

Ocean University of China

Guoxian Song

Ocean University of China

Qingchen He

Ocean University of China

Article

Keywords: CNC Simulation, Computer Graphics, Enhanced Dual Contouring Continuous Surface Reconstruction, Real-time Geometric Visualization, Virtual Manufacturing, 6-Adjacency Encoding Octree

Posted Date: April 9th, 2024

DOI: <https://doi.org/10.21203/rs.3.rs-4120831/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.
[Read Full License](#)

Additional Declarations: No competing interests reported.

Enhanced Dual Contouring for Continuous Surface Reconstruction: A Novel Method in Real-Time Geometric Visualization for CNC Machining Simulation

Chang Su¹, Yong Han¹, Guoxian Song¹, Qingchen He¹

Corresponding author: Yong Han; suchang5022@stu.ouc.edu.cn;

Author affiliation:

1.Department of Information Science and Engineering, Ocean University of China, OUC 266100, CN

Abstract

This study introduces Enhanced Dual Contouring Continuous Surface Reconstruction (EDCCSR) to improve CNC simulation for complex surface multi-axis machining. By integrating advanced computer graphics algorithms and octree encoding, EDCCSR enhances real-time visualization and efficiency in geometric model reconstruction. It converts the outer rough model into a triangulated mesh through voxelization, ensuring consistency. Innovatively, it employs a crack table method for seamless integration of rough and cutting surface meshes, facilitating uniform mesh transitions and accurately visualizing dynamic geometric transformations. This method not only optimizes simulation efficiency but also minimizes errors and delays in machining processes. It underscores computer graphics' potential in manufacturing and supports the industry's digital transformation, highlighting its wide applicability in industrial contexts.

Keywords CNC Simulation; Computer Graphics; Enhanced Dual Contouring Continuous Surface Reconstruction; Real-time Geometric Visualization; Virtual Manufacturing; 6-Adjacency Encoding Octree

1 Introduction

In an era of intensifying global market competition, the manufacturing industry is increasingly challenged to meet rigorous market demands. The rapid evolution of computer technology and graphics algorithms has positioned virtual manufacturing as an innovative solution for the traditional manufacturing sector. This method facilitates product performance assessment and processing strategy optimization, leading to significant cost reductions and shorter research and development cycles. At the heart of this virtualization lies CNC simulation, which replicates the actual machining process through computer models, enabling engineers to visualize the tool's trajectory and anticipate potential issues effectively.

This paper primarily focuses on enhancing the real-time geometric visualization in CNC machining simulation, particularly for complex surface mold and multi-axis machining. We conduct a comprehensive study of key algorithms and introduce the Enhanced Dual Contouring Continuous Surface Reconstruction (EDCCSR) method, a novel method designed to improve efficiency and real-time visualization in these simulations. The EDCCSR method offers two significant advancements: Firstly, it employs an optimized surface reconstruction algorithm, which ensures the rapid and accurate construction of voxel models for the workpiece in various machining states. Secondly, it innovatively applies a crack table method based on vertex states, facilitating a seamless integration between the rough and cutting surface meshes. This method effectively addresses the limitations of traditional Dual Contouring algorithms in continuous surface reconstruction, thereby enabling smoother transitions on the cutting surface and ensuring an uninterrupted visualization process. Ultimately, the EDCCSR algorithm exemplifies the successful application of computer graphics technology in manufacturing process simulation, highlighting its vital role in interdisciplinary research and signaling a leap forward

in the digital transformation of the manufacturing industry.

2 Related Work

CNC machining simulation technology is primarily divided into two categories: geometric simulation [1][2] and physical simulation [3]. Geometric simulation primarily focuses on the movement and geometrical shapes of the tool and workpiece without considering cutting parameters, cutting forces, or other physical parameters. Its purpose is to verify the correctness of NC programs, reduce production costs, and shorten the design and manufacturing cycles of workpieces [4][5][6][7][8]. In contrast, physical simulation predicts tool wear and vibration by simulating physical quantities in the actual cutting process to optimize cutting techniques [9], thereby increasing the production success rate of the workpieces. This paper primarily addresses the first category, namely, geometric modeling.

International research on CNC machining simulation technology began in the early 1970s. During the simulation process, the shape of the workpiece continuously changes, necessitating ongoing modeling. Due to limitations in computer technology at the time, initial simulations used wireframe models [10], which easily caused interference between the tool path and the wireframe model, affecting clarity and reducing resolution. Entering the 1980s, solid modeling techniques began to emerge. With the development of computer technology, CNC machining simulation has been divided into the two widely used categories today: simulation methods based on solid modeling and approximate simulation methods based on discrete models [11]. Yan et al. [3] proposed the Constructive Solid Geometry (CSG), which uses the lower voxel elements as the main base elements and applies geometric operations to obtain the solid target model. Spence et al.[13] used CSG in CNC machining simulation. Nehring [14] et al. proposed a high-performance, scalable CSG model based on embedded octrees, and Tzoumas et al.[15] extended the traditional CSG model using projection operators and supported the computation of gradient and topological properties. The disadvantage of such methods is the huge computational effort, having a time complexity of about $O(n^4)$, according to Chappel [16] et al. Therefore, using such methods in

scenarios with real-time requirements is not easy. Boundary-representation (B-rep) has more point-line surfaces utilizing their corresponding topological information. EI-Mounaryri [17] et al. developed a three-axis CNC machining simulation system based on the boundary representation model, which uses Bessel curves to represent the tool and determines the scan body based on the toolpath code, which performs Boolean subtraction operations with the workpiece model. Boz[18]proposed an improved method based on the boundary representation to solve the contact area between the free-form five-axis milling tool and the workpiece model. The disadvantages of this method are the poor ability to describe the object as a whole, the large amount of data required to describe the object, and the fact that the representation is not unique. These problems are often fatal for high-speed machining because although these establish a solid geometric CSG model using a boundary representation B-rep that accurately represents objects, the number of Boolean operations in the CNC simulation is often between thousands and tens of thousands. If only Boolean subtraction is used, the calculation cost will increase rapidly with the increase of tool machining times, which is challenging to the real-time requirements. Currently, numerical control simulation modeling methods mainly include the spatial partition, Z-Buffer, discrete vectors, boolean operation, and the surface modeling.

(1) Spatial partition: The spatial partition method represents an object by dividing a whole model area into a series of spatial units, which are planes or cubes of preset accuracy size. The computer can represent the model by the presence or absence of spatial units in a certain area, and all the existing

spatial units constitute the complete model. The simplest spatial partition is the Voxel representation, but the computer's most usual spatial partitioning method is the octree structure. Karunakaran et al. [19] applied the octree model to the field of CNC machining simulation. By representing the workpiece blank model as a collection of several hexahedra, the index structure of the octree can significantly reduce spatial complexity. This method greatly simplifies the fading of Boolean operations, but the storage space of the octree structure shows an order of magnitude increase as accuracy increases. On this basis, Liu [20] and Li[21] developed a three-axis CNC machining simulation system using the quadtree expression method. The quadtree has a simpler data structure, which simplifies CNC code verification. Later, some scholars combined quadtree with octree to accelerate the generation of geometric entities with hybrid tuples, simplifying the representation of these geometric entity variants.

(2) Z-Buffer: Z-Buffer is a depth buffer, which mainly saves the distance from each pixel to the corresponding spatial point. It is a method proposed by Van Hook [22] to quickly observe the dynamic process of cutting by first determining a fixed viewpoint, after which a ray is sent inward from the screen, and the intersection of the ray with the scanned body of the tool is calculated, reducing the Boolean operation to a local Boolean difference operation on one dimension. However, this method does not change the viewing angle during machining. Later, Huang [23] improved this structure by proposing the depth element model (Dexel), which converts the solid model into a Dexel model by establishing a coordinate system and using scan lines to achieve arbitrary viewpoint conversion in the CNC simulation process. Saito [24] developed the G-Buffer method based on Z-Buffer and successfully applied it to the CNC machining simulation system. Then, Lee et al.[25] introduced the Tri-Dexel model based on the Dexel model to represent the solid model. Tri-Dexel was employed in the five-axis machining simulation, with the corresponding experiments verifying the method's effectiveness. The disadvantage is that the original data is converted to pixel information, and thus machining error detection cannot be performed.

(3) Discrete vectors: The main idea of discrete vectors is to discretize the part surface into a series of points, extend its normal vector to intersect with the blank surface with these points as the starting point, and intersect the tool scan body and vector during the simulation machining process. If there is an intersection, modify the vector length and reduce the vector to the intersection. Chappel used this method for cutting simulation, but this method can only be used to improve the efficiency of the 3-axis cutting simulation. Later, Kiswanto et al. [26] combined the point vector method with analytical modeling, where a set of normal vectors discretizes the surface of the workpiece. Then the intersection of the tool and the workpiece is calculated by using the parametric equations of the tool and the direction with the surface equations of the workpiece in conjunction with the intersection of the tool and the workpiece during cutting. This method avoids collision in a five-axis CNC machining and has a high simulation effect.

(4) Boolean operation: This scheme directly performs a Boolean operation using triangular meshes, with its output being a mesh model. In addition, the Boolean operation affords high accuracy and an appealing visual effect, presenting good efficiency[27][28][29]. Although more suitable for simulating dynamic geometric changes in models, the complex geometric models involved in cutting simulation has significantly increased the number of triangular models, substantially increasing the computational burden to apply a direct intersection and imposing an unacceptable memory consumption.

Taking into account the unique challenges associated with the dynamic cutting process of complex industrial products, it can be difficult to accurately represent the real-time geometric modeling of these processes. Therefore, it is essential to evaluate the advantages and disadvantages of various modeling methods in order to select the most suitable method, while also focusing on optimization and improvement of existing techniques.

Marching cube (MC) [30] algorithm. This classic surface rendering algorithm approximates the iso-surface through linear interpolation in a 3D discrete data field. The method's details are as follows: each grid cell in a 3D discrete data field acts as a voxel, and each vertex has a corresponding scalar value. If the value on the voxel vertex is greater than or equal to the iso-surface value, the vertex is located outside the iso-surface value and is marked as "0". If the value on the voxel vertex is less than the iso-surface value, it is within the iso-surface value and is marked as "1". Since each voxel element has 8 vertices, there are $2^8 = 256$ cases. Researchers enhanced the algorithm's fitting accuracy and efficiency through various optimizations and expansions[31][32][33]. Theoretically, suppose the algorithm can be applied to the simulation of the CNC machining process. In that case, the number of triangles in the machining process will not increase greatly, and the calculation efficiency will not degrade because of the increase in the number of triangles. This is an advantage of the Boolean operation method based on the tool-swept volume. Nevertheless, many problems still need to be solved when considering CNC machining process simulation. Since the Marching Cubes algorithm only calculates the coordinates of the intersections of the voxel cells and constructs the geometric model within the voxel cells based on the triangular surface pieces connected to these intersections, if there are features of the geometric model (edges, angles) within the voxel cells, the final geometric model constructed by the Marching Cubes algorithm will lack these features.

To solve the problems posed by the Marching Cubes (MC), Ju et al. introduced Dual Contouring [34]. Dual Contouring is a 3D reconstruction algorithm that converts discrete voxel data to surface models. It has exceptional benefits such as generating high-quality surfaces and fewer topological changes. When utilized in dynamic cutting of geometric models, it presents the following advantages:

1.Reduced triangle mesh count: Dynamic cutting leads to continuous updates and an increase in the triangle mesh count of the model. However, the Dual Contouring algorithm reduces this count by retaining key points in the mesh and merging the mesh when necessary.

2.Ensuring accurate geometric information of model: The Dual Contouring algorithm extends the Marching Cube algorithm by not only identifying the intersection of the iso-surface with the 3D spatial grid but also determining the normal vector at the intersection location. Unlike the Marching Cubes algorithm, Dual Contouring places mesh vertices in the voxel's interior rather than on the edges. This results in a more regular mesh that retains accurate shape information without losing sharp edges and vertex. However, it's important to note that this feature may require adjustments when applied to CNC machining to ensure the accurate representation of sharp edges caused by intersecting tool paths.

3.Efficient computation: Fast computation of surface curvature and normal vectors is necessary for the next processing stage during dynamic cutting. The Dual Contouring algorithm performs this computation efficiently and generates dynamic models.

4.Handling complex geometry: Dual Contouring can handle complex geometries such as models with internal cavities and voids. This feature makes it useful in dynamic cutting scenarios where the model shape often changes.

The Dual Contouring algorithm is an extension of the Marching Cube algorithm. In contrast to the volume metadata in the Marching Cubes algorithm, the Dual Contouring algorithm will not only find the intersection of the iso-surface with the 3D spatial grid but will also find the normal vector at the location of the intersection. In addition, the Dual Contouring algorithm does not place mesh vertices on the edges of the voxel data but instead places the vertices of the mesh model in the interior of the voxel, constructs a quadratic error equation based on the intersections obtained by linear interpolation on the edges of the voxel with its normal vector, finds its characteristic points as vertices and connects the vertices in four

adjacency voxels (a quadrilateral or two triangles). The Dual Contouring algorithm does not rely on look-up tables for topological connections and thus produces a more regular mesh and a mesh model that is always streamlined while retaining accurate information about the shape without discarding sharp edges and vertex. The Dual Contouring algorithm is implemented as follows. Its voxel pre-processing is the same as that of the Marching Cube algorithm, where the location of the intersection is obtained by linear interpolation but with the addition of the process of obtaining the normal vector. For any point located on the iso-surface, its gradient component along the tangent direction of the iso-surface is always zero, and then the direction of the gradient vector at that point represents the normal direction at that point on the iso-surface. It should be noted that the Dual Contouring algorithm is based on a regular voxel data field. Compared with the Marching Cubes algorithm, the Dual Contouring algorithm does not rely on a lookup table for topological adjacency, and the resulting mesh is more regular and the mesh model is always streamlined, while retaining accurate information about the shape without the loss of sharp edges and vertex. Fig. 1 illustrates the application of the voxel-based modeling in an aero-engine blade model, where the red and the green numbers represent the cube's eight vertex and twelve edges, respectively.

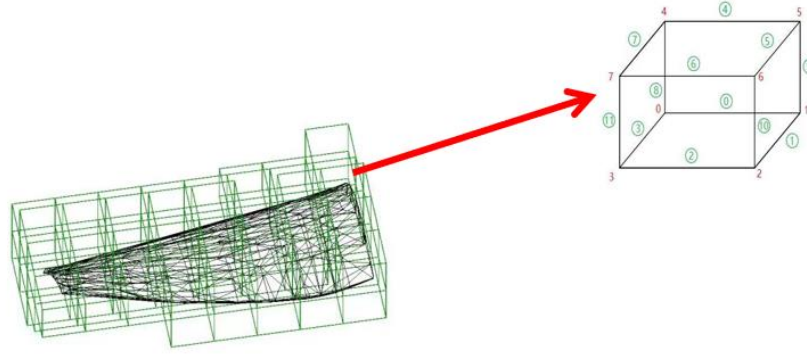


Fig. 1 Voxel-based modeling for aero-engine blade model reconstruction

3 Methodology

3.1 Fast Surface Reconstruction for CNC Model

Although surface reconstruction is a key mechanism for visualizing complex data structures, its core challenge lies in accurately representing the implicit boundaries of objects or phenomena. This boundary, often referred to as the 'iso-surface,' depicts the exact shape and contour of the model under discussion. Effectively and precisely constructing this iso-surface is crucial for ensuring that the reconstructed surface truly reflects the inherent features of the underlying data. The geometric state of product models in industrial manufacturing processes can be exceedingly complex, potentially including features like grooves, sharp edges, and complex curves. Therefore, applying the dual contouring algorithm to the voxelization of the model is an excellent method. The specific algorithm is as follows; According to the linear interpolation method, if the edge on which the intersection is located is on the x-axis, then the coordinates of the intersection are:

$$(i + \frac{C - f(i, j, k)}{f(i + 1, j, k) - f(i, j, k)}, j, k) \quad (1)$$

Where $f(i, j, k)$ is the value at (i, j, k) coordinates, and C is iso-surface value of the model

used for contact. Similarly, the coordinates of the intersection on the Y and Z axis are:

$$(i, j + \frac{C - f(i, j, k)}{f(i, j + 1, k) - f(i, j, k)}, k) \quad (2)$$

$$(i, j, k + \frac{C - f(i, j, k)}{f(i, j, k + 1) - f(i, j, k)}) \quad (3)$$

Next, the normal vector of each vertex of the body element is calculated using the central difference method:

$$g_x = [f(i + 1, j, k) - f(i - 1, j, k)] / 2\Delta x \quad (4)$$

$$g_y = [f(i, j + 1, k) - f(i, j - 1, k)] / 2\Delta y \quad (5)$$

$$g_z = [f(i, j, k + 1) - f(i, j, k - 1)] / 2\Delta z \quad (6)$$

where g represents the gradient value in each direction and (g_x, g_y, g_z) is the normal vector.

If the prismatic edge on which the intersection is located is on the x-axis, then the normal vector of the intersection is:

$$N = N(i, j, k) + \frac{[C - f(i, j, k)][N(i + 1, j, k) - N(i, j, k)]}{f(i + 1, j, k) - f(i, j, k)} \quad (7)$$

Similarly, if the intersection is on the Y and Z axes of the prism, the normal vectors of the intersection are:

$$N = N(i, j, k) + \frac{[C - f(i, j, k)][N(i, j + 1, k) - N(i, j, k)]}{f(i, j + 1, k) - f(i, j, k)} \quad (8)$$

$$N = N(i, j, k) + \frac{[C - f(i, j, k)][N(i, j, k + 1) - N(i, j, k)]}{f(i, j, k + 1) - f(i, j, k)} \quad (9)$$

where N is the normal intersection vector, $N(i, j, k)$ is the coordinate (i, j, k) vector value, and

C is iso-surface value of the model used for contact.

After finding the intersection of the iso-surfaces in the voxel and the normal vectors corresponding to the iso-surfaces according to the formula in (1)-(9), the characteristic points in the voxel can be calculated using a quadratic error function. The specific algorithm is divided into the following steps:

First, vertex coordinates are generated using a quadratic error function. For each voxel cell intersecting an iso-surface, a vertex coordinate is generated by minimizing the quadratic error function, defined as follows:

$$E(x) = \sum_i (n_i \bullet (x - p_i))^2 \quad (10)$$

where p_i is the position of the intersection and n_i is the normal direction of the intersection.

The error function can be written in matrix form as follows:

$$E(x) = (Ax - b)^T (Ax - b) \quad (11)$$

Expanded to:

$$x^T A^T x - 2x^T A^T b + b^T b \quad (12)$$

The row vectors of matrix A are normal to the intersection n_i , and each element of vector b is $n_i \cdot p_i$

. Solving formula (12) is complex. Therefore, this paper uses the QR matrix decomposition method to reduce computational complexity. By solving the characteristic points is based on the orthogonal matrix Q so that Q and [A b] are multiplied into the upper triangular matrix form, as presented in (13):

$$\begin{pmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots \end{pmatrix} = \begin{pmatrix} A' & b' \\ 0 & r \\ 0 & 0 \\ \dots & \dots \end{pmatrix} \quad (13)$$

where A' is an upper triangular matrix of $3 * 3$, b' is a vector of length 3, and r is a scalar. Then the error function can vary as:

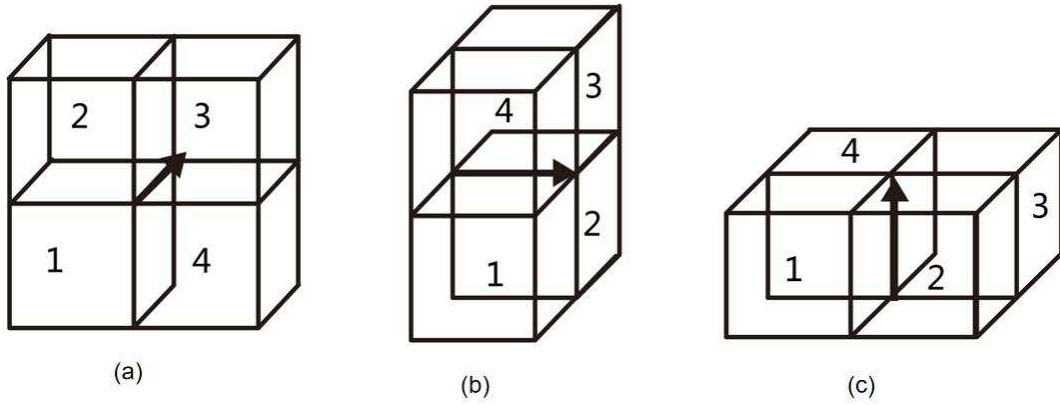
$$(Ax - b)^T (Ax - b) = (Ax - b)^T Q^T Q (Ax - b)$$

$$= (QAx - Qb)^T (QAx - Qb)$$

$$= (A'x - b')^T (A'x - b') + r^2 \quad (14)$$

Subsequently, the extremum points are calculated based on the above formula. If such an extremum point is located inside the voxel cube, it is then used as the vertex coordinate of the voxel. If the calculated extremum point lies outside the voxel cell, the vertex coordinate is replaced with the centroid position of the intersection points inside the voxel. After calculating the coordinates of the vertices located within the voxel mesh, we can proceed with the mesh connection. The connection in the Dual Contouring algorithm mainly targets the voxel edges with intersections, connecting the vertices of the four adjacent voxel cells sharing this edge, to generate either a corresponding quadrilateral mesh or two triangular meshes (in our case, triangular connections are used). For each voxel edge intersecting with the isosurface, the four adjacent voxel cells containing this edge will inevitably have vertices that can be connected, as shown in Fig.2, connecting triangles 1, 2, 3 and 1, 3, 4. In the X-axis direction, adjacent quadruplets of voxels are connected in sequence as shown in figure (a), and for the Y and Z axes, they are connected as per the sequences in figures (b) and (c), respectively. Our algorithm, if directly optimized using an octree structure, is not suitable for traversing voxels layer by layer, column by column, and row by row in a three-dimensional data field. To achieve fast voxel connection, we have optimized the coding of the octree and designed an adjacency relationship table. The adjacency relationships in the octree are used to quickly query the four adjacent voxel cells, enabling the efficient skipping of queries for empty voxel

270 units, thereby enhancing the efficiency of connections.



272 **Fig. 2** Triangular connection method in Dual Contouring

271

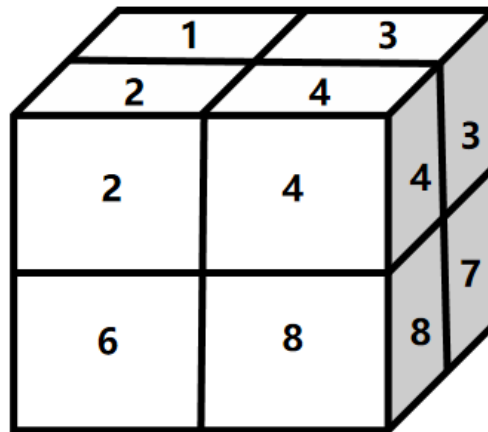
272

273

274 As mentioned above, the construction of iso-surface requires the connection of the vertices of four
 275 adjacent voxels, surface reconstruction involves only the voxels in contact with the tool, the others are
 276 not involved in the reconstruction. Therefore, surface reconstruction by traversing all voxels of the model
 277 is very time-consuming and complex. It cannot meet the requirement of fast reconstruction of cutting
 278 surfaces. Therefore, we propose the use of octrees to help us search and locate the related voxels quickly,
 279 thus optimizing the vertex computing time in our algorithm.

280 Traversing the voxels layer by layer, column by column, row by row will generate huge data redundancy.
 281 Introducing dynamic octree will save a lot of time, but the time for each cutting surface reconstruction is
 282 also unacceptable for real-time machining. To achieve a fast connection of voxels, the octree structure
 283 was coded to use the adjacency relationships in the octree to achieve a fast look-up of four adjacency
 284 voxel units. This enables queries for empty voxel units to be skipped, speeding up the efficiency of joins.
 285 The encoding rules for octree structures are as follows: Each subnode of the octree is incrementally
 286 coded from 1 according to the order of the X, Y, and Z coordinate values from smallest to largest, where
 287 the root node is 0, as shown in Fig. 3. For each level the eight nodes of the cube are arranged in the
 288 corresponding spatial order. The positions of the cubes that are not dissected by the octree are also left
 289 with the corresponding values to provide a code for their dissection afterward:

290



291 **Fig. 3** Sub-node partition index

292

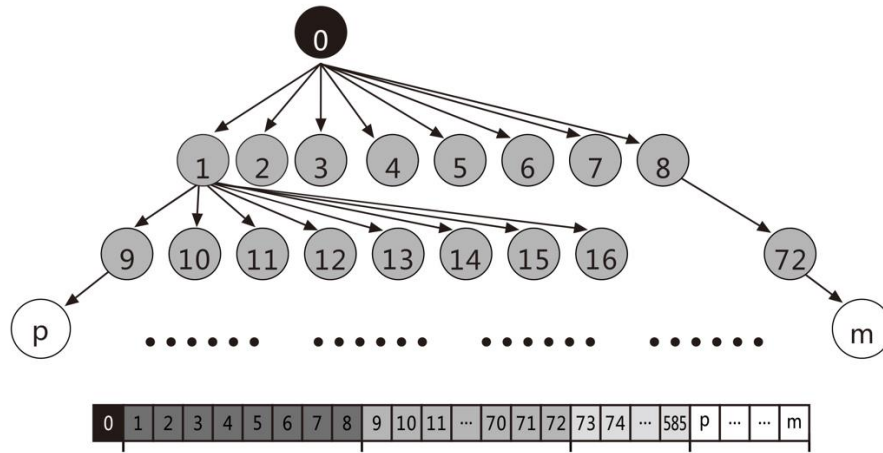


Fig. 4 Schematic diagram of octree structure coding

As shown in Fig. 4, 1-8 is the first level, 9-72 is the second level, and so on. With this octree encoding, the corresponding encoded values of the eight child nodes numbered i can be easily calculated as $8i+1$, $8i+2$, ..., $8i+8$ in the order of the coordinate X, Y and Z increments, respectively. In this case, the numbering increases gradually along the X-axis, i.e., child node No. 2 is located in the positive direction of the X-axis of child node No. 1. Afterwards, the numbering gradually increases along the Y and Z directions, i.e., child node No. 3 is located in the positive direction of the Y axis of child node No. 1. Nodes No. 5,6,7,8 are located on the upper level of nodes No. 1,2,3,4, respectively. With this encoding, the spatial location relationships within the octree structure can be obtained by numbering. Assuming

that the depth of an octree is n , the number of nodes in a complete octree is $\sum_{m=0}^{m=n} 8^m$ (m is the current depth of the node), and the nodes are numbered from 0 to $(8n-1)/7-1$. The corresponding information on the structure of the octree can be calculated as follows;

The index number of the primary node is $(i-1)/8$;

The index number corresponding to the sub-node is $8i+t$ ($t=1,2,...,8$);

The corresponding octree level for the node is $\log_8 (7i+1)$;

The octree structure constructed with this encoding rule is projected onto the XOY plane, the formula can calculate its encoding along the different coordinate increment directions and the encoding rules corresponding to the different levels of the profile depth 3, e.g., as depicted in Fig.5.

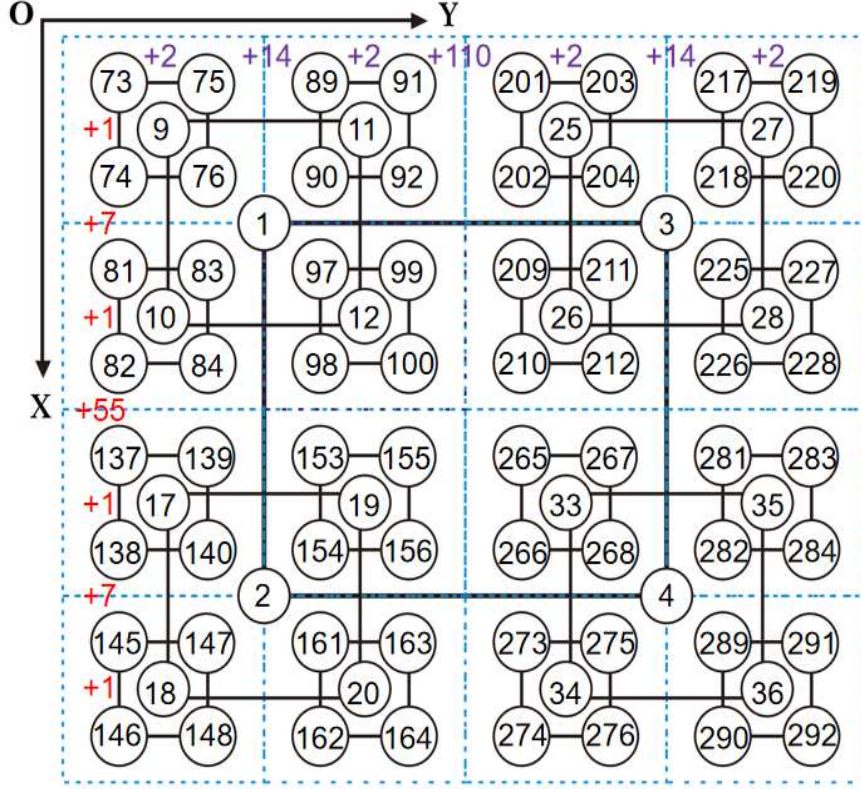


Fig. 5 Plane projection of octree coding

The increments of voxel coding can be summarized.

X-axis positive direction:

$$\Delta x(1) = [1]$$

$$\Delta x(m) = [\Delta x(m-1), \frac{6}{7}8^{m-1} + \frac{1}{7}, \Delta x(m-1)] \quad (15)$$

Y-axis positive direction:

$$\Delta y(m) = 2\Delta x(m) \quad (16)$$

Z-axis positive direction:

$$\Delta z(m) = 4\Delta x(m) \quad (17)$$

After encoding, the fast find traversal and deletion can be realized by using this information. By formulas (15) (16) (17), we can quickly locate the adjacent voxels according to their X,Y,Z direction increments after finding a voxel, thus significantly saving the time required to re-traverse.

As shown in Fig. 6, the hexahedral voxel structure in space corresponds to three types of topological relationships: 26-adjacency when any two hexahedra share a common point, as shown in (a); 18-adjacency when they share a common edge, as shown in (b); and 6-adjacency when any two cubic voxels share a face, as shown in (c). Therefore, the constraint of 6-adjacency is the strongest, while 26-adjacency is the weakest. Since the Dual Contouring algorithm is primarily used for finding adjacent voxels, this paper establishes a 6-adjacency lookup table for the nodes of the octree based on the aforementioned encoding rules. By calculating the relationship between the leaf node and its parent node numbers, we can quickly obtain the numbers of adjacent nodes, making the algorithm's connection easier and

improving the fitting efficiency.

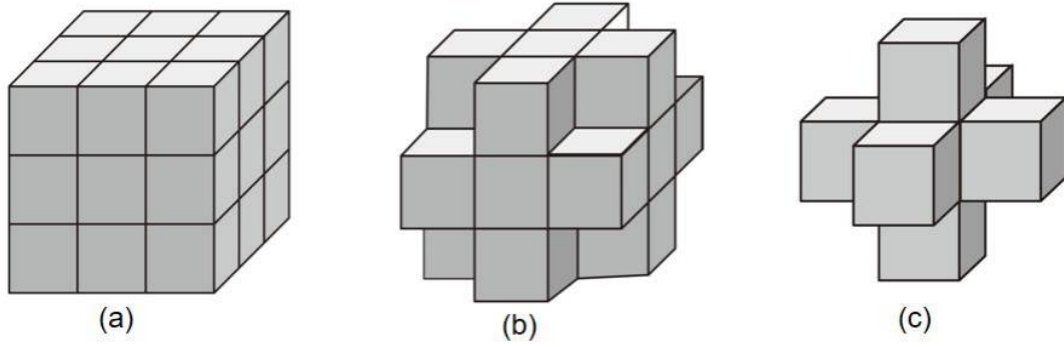


Fig. 6 Voxel adjacency relationship diagram

As shown in the voxel 6-adjacency model above, the adjacent voxel node numbers for any voxel node are set to default as -1. For root node voxels, their six neighboring voxels do not exist. When divided into more child nodes, the child nodes inherit the non-existence indicators on the corresponding faces. By analyzing the inheritance relationship between parent and child nodes, we can determine whether a 6-adjacency voxel node of any child node exists in three-dimensional space. If it does not exist, its value remains -1; if it exists, the corresponding coding value for the neighboring voxels is obtained according to the lookup table of adjacency relationships. The specific rules for the lookup table can be summarized as follows:

Table 1 Adjacency relationship along the X-axis direction

Satisfying the condition	Current child node number	Encoding increment in the positive X-axis direction	Encoding increment in the negative X-axis direction
The current child node's encoding value is modulo 2	Number is odd	Encoding value +1	Encoding value - a
	Number is even	Encoding value + a	Encoding value -1

Table 2 Adjacency relationship along the Y-axis direction

Satisfying the condition	Current child node number	Encoding increment in the positive Y-axis direction	Encoding increment in the negative Y-axis direction
The current child node's encoding value is modulo 8	Remainders are 1, 2, 5, 6	Encoding value +2	Encoding value - b
	Remainders are 3, 4, 7, 0	Encoding value + b	Encoding value -2

Table 3 Adjacency relationship along the Z-axis direction

Satisfying the condition	Current child node number	Encoding increment in the positive Z-axis direction	Encoding increment in the negative Z-axis direction
The current child node's encoding value is modulo 8	Remainders are 1, 2, 3, 4	Encoding value +4	Encoding value - c
	Remainders are 5, 6, 7, 0	Encoding value + c	Encoding value -4

Based on the coding rules of the octree and the summarized patterns, the values of increments a, b, c in the table exhibit a regular variation, which can be expressed by formula:

$$a = \frac{6}{7}8^p, b = 2\left(\frac{6}{7}8^m + \frac{1}{7}\right), c = 4\left(\frac{6}{7}8^n + \frac{1}{7}\right) \quad (18)$$

The values of P, m, n can be obtained through the following rules:

For the X coordinate direction, we analyze the encoding of the current octree structure. For any node's eight children, they are divided into two layers according to the X-axis coordinate increment direction (upper layer has smaller X coordinates, lower layer has larger X coordinates). The upper layer nodes are 1, 3, 5, 7 (all odd numbers), and the lower layer nodes are 2, 4, 6, 8 (all even numbers). Therefore, for all nodes in the upper layer in the X-axis direction, their adjacent nodes in the positive X direction share the same parent node, so the encoding value only needs to be increased by 1, corresponding to cases where p is 0. For all nodes in the lower layer, their parent node's level needs to be calculated to determine the p value. Ascend to their parent node and judge its level. If the parent node is in the upper layer, then the adjacent node in the positive X direction of this node shares the same grandparent node, and the p value is set to 1, corresponding to the initial value of p; if the parent node is also in the lower layer, then p is increased by 1 and the process continues recursively upwards until the parent node is in the upper layer. Since all upper layer nodes in the X-axis direction are odd numbers, and lower layer nodes are even numbers, the upper and lower layer relationships of the nodes can be determined by modulo 2. Simultaneously, the determination of the p value can be simplified to the consecutive even number occurrences between the parent node's encoding value and the child node's encoding value. The judgment for the negative direction of the X-axis is simply the opposite, considering odd numbers. Through such calculations, the encoding values of the adjacent nodes in the X direction can be quickly calculated, determining the adjacent voxels. For example, node number 74 has a remainder of 0, its parent node is 9, corresponding to a remainder of 1, with a respective p value of 1; node number 82 has a remainder of 0, its grandparents are 10 and 1, with respective remainders of 0 and 1, and a respective p value of 2.

The principle for determining the Y and Z coordinate directions is similar to that of the X-axis. Based on the respective coordinate increment directions, the eight children are divided into upper and lower layers. In the Y-axis direction, the upper layer corresponds to 3, 4, 7, 8, and the lower layer to 1, 2, 5, 6; in the Z-axis direction, the upper layer corresponds to 5, 6, 7, 8, and the lower layer to 1, 2, 3, 4. Neighboring voxels and voxels at the same level are directly added or subtracted by 2 or 4. The initial values of m and n are set to 1 and are assigned based on the corresponding relationship of the parent node's level. The specific calculation method can be optimized by taking the modulo operation of the encoding value with 8. The layer correspondence is determined based on whether the remainder of the parent node's encoding value is the same as the remainder of the child node's encoding value.

As shown in Tables 1, 2, and 3, by analyzing the coding of the octree and the corresponding adjacency relationships of the voxels, we can find that the six adjacent voxels in the 6-adjacency relationship of a node at different levels show a relatively regular arrangement rule. Simultaneously, there is a regular coordinate increment for neighboring voxels in different coordinate directions. Therefore, in the actual solving process, we do not need to calculate the values of a, b, c in real-time. To improve the corresponding computational speed, by simplifying the relationships among the above formulas, we can obtain the relationship of $b=2a$, $c=4a$. We only need to store the a values corresponding to each p value in a linear table. When selecting different p, m, n values, we can quickly obtain b, c values based on the multiple relationship with the a value in the linear table.

3.2 Continuous Surface Reconstruction for Machining Process Model

For smoothing transitions in multi-cut surface meshes, traditional methods often employ triangular mesh Boolean operations. The process begins by extracting the intersection line from two triangular meshes. Based on this line, overlapping triangular faces are re-triangulated, followed by the removal of redundant triangles to form a new mesh. However, tests show this method doesn't suit our needs well. Although it provides good transition effects in static models without losing mesh accuracy, CNC machining simulation requires high time efficiency. Therefore, this paper proposes a secondary fitting method based on DC algorithm principles, maintaining precision within the partitioned mesh size. This method focuses only on the common voxels of the two cutting surfaces, speeding up the mesh transition. A flag bit is added to the octree node data structure. Voxels involved in the first cutting surface fitting are marked as fitted. During the second cutting, voxels marked as unfitted undergo direct Boolean fitting. If a voxel is in the overlapping area of both cuts and marked as fitted, it needs secondary fitting. The state from the first cut is used as a basis for the second cut's state judgment and refitting, achieving smooth transition. The specific conditions are as follows:

- 1.If none of the old vertices on a voxel edge are marked as cut, use the new isosurface's intersection point value for the vertex state and assign intersection information.
- 2.If all old vertices on a voxel edge are marked as cut, no new isosurface cut state judgment or intersection value assignment is needed; maintain the old state.
- 3.If one vertex on a voxel edge is marked as cut, compare the new isosurface's ray-saved intersection information with the old isosurface's cut information and assigned intersection values. Assign vertex states and update intersection information based on the situation.

For example, consider a voxel edge perpendicular to the XOY plane and its intersection points:

- (1) If the lower vertex's coordinate value of a voxel edge is greater than the first intersection point but less than the second, and the upper vertex's coordinate value is greater than the second intersection point, then the lower vertex of the voxel edge is inside the second tool's cutting surface. The state value of this lower vertex needs to be set to 'cut', and the coordinate value of the second intersection point becomes the value to be assigned to this voxel edge. If the state of this lower vertex has already been set to 'cut' by the first cutting surface, then it's necessary to compare the second intersection point with the old intersection point Z. If the second intersection point's value is less than the old intersection value, then the new isosurface intersection value is outside the pseudo-cuboid, and the intersection value remains unchanged, as shown in Fig.7 (a). If the second intersection point's value is greater than the old intersection value, then the new isosurface value is inside the pseudo-cuboid, and the old intersection value should be replaced with the second intersection point's value, as shown in Fig.7 (b). If the upper vertex of this voxel edge has already been set to 'cut' by the first cutting surface, then both vertices of this voxel edge are cut by the new and old isosurfaces, and no intersection value needs to be set, as shown in Fig.7 (c).

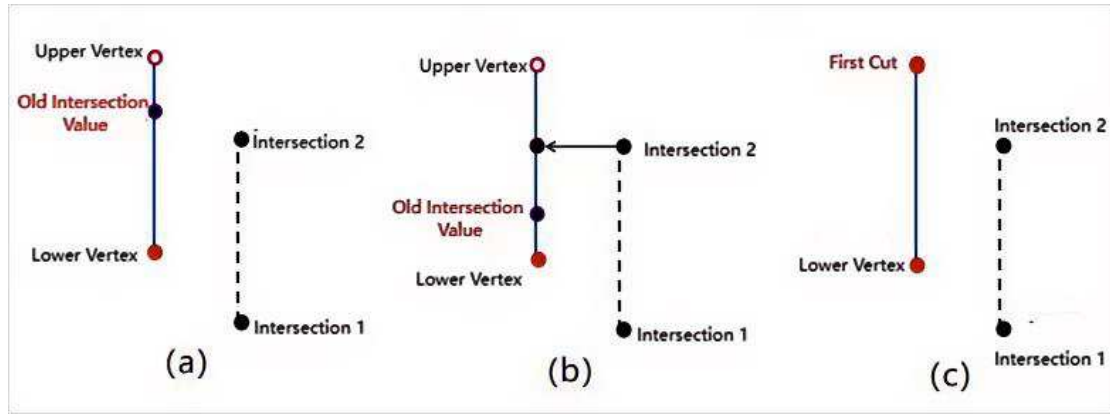


Fig. 7 Lower vertex of the new cut surface replaced by the intersection

(2) If the lower vertex's coordinate value of a voxel edge is less than the first intersection point, and the upper vertex's coordinate value is less than the second intersection point but greater than the first, then the upper vertex of the voxel edge is inside the second tool's cutting surface. The state value of this upper vertex needs to be set to 'cut', and the coordinate value of the first intersection point becomes the value to be assigned to this voxel edge. If the state of this upper vertex has already been set to 'cut' by the first cutting surface, then it's necessary to compare the first intersection point with the old intersection point Z. If the first intersection point's value is greater than the old intersection value, then the new isosurface intersection value is outside the pseudo-cuboid, and the intersection value remains unchanged, as shown in Fig.8 (a). If the first intersection point's value is less than the old intersection value, then the new isosurface value is inside the pseudo-cuboid, and the old intersection value should be replaced with the first intersection point's value, as shown in Fig.8 (b). If the lower vertex of this voxel edge has already been set to 'cut' by the first cutting surface, then both vertices of this voxel edge are cut by the new and old isosurfaces, and no intersection value needs to be set, as shown in Fig.8 (c).

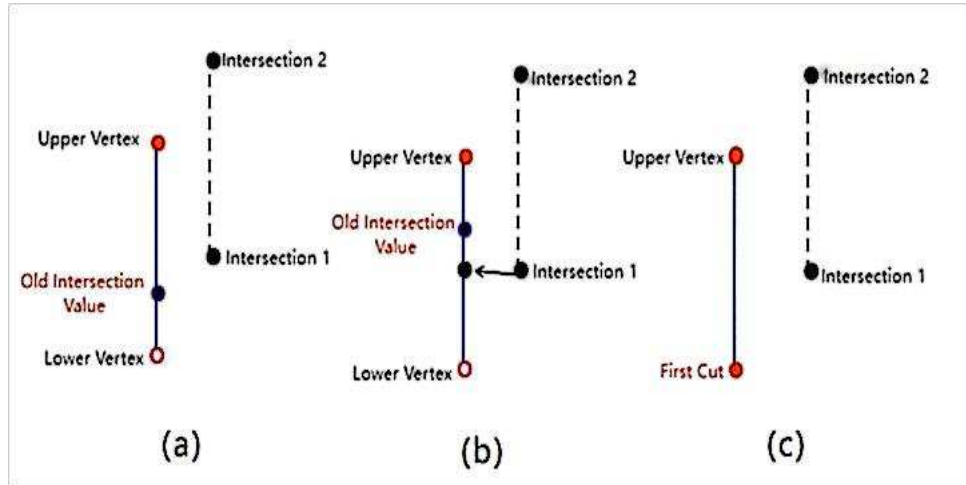


Fig. 8 Upper vertex of the new cut surface replaced by the intersection

The overall algorithm's execution on voxels is illustrated in Fig. 9. This method essentially performs the Dual Contouring (DC) algorithm based on a pseudo-cuboid, achieving a smooth transition between the first and second cutting surfaces. Subsequently, the triangulated mesh after the transition is reset as Triangulation 1, and the newly fitted triangulated mesh is set as Cutting Mesh 2. By repeating the aforementioned process, surface reconstruction under continuous cutting conditions based on the voxel

model can be achieved. However, as illustrated by the voxels, this substitution will result in a certain loss of precision.

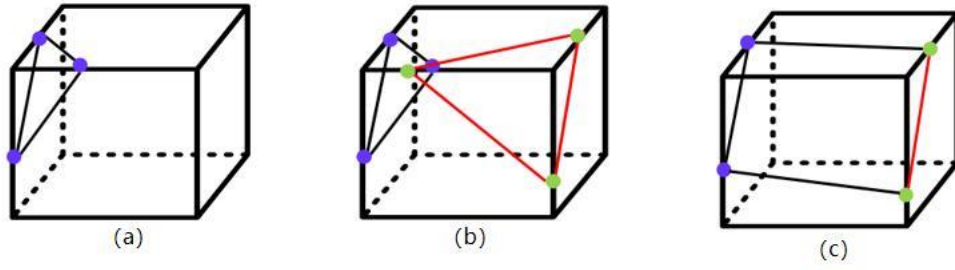


Fig. 9 Fitting of cutting surfaces in dynamic machining.

3.3 Transition of Model Cutting Surfaces During the Machining Process

As noted in 3.2, the algorithm presented in this paper focuses on fitting the cutting surface only in the regions where the tool operates, while the rough model of the workpiece is fitted using voxel data organized by an octree. This results in an inconsistency between the voxel model data and the triangular mesh data model. To achieve data consistency and facilitate unified processing, this paper adopts a face lookup table method to uniformly transform voxel data corresponding to the octree structure into surface triangulated mesh data. For the cubic cells of the workpiece's rough octree model, only the outermost surfaces of the model need to be displayed. The cubes located on the surface of the rough model can be categorized into three types: points, lines, and faces. By defining a face lookup table, we determine which faces of the smallest leaf node voxels in the octree structure are on the outer display surface, and then extract the triangular mesh from the rough model of the workpiece based on the different voxel types. In formula (19), $X_{\min}$, $Y_{\min}$, $Z_{\min}$ represent the coordinate values of the cubic voxel's index point 0 as defined above, while $X_{\max}$, $Y_{\max}$, $Z_{\max}$ correspond to the coordinate values of the cubic voxel's index point 6. $B_{\min}$ corresponds to the minimum coordinate value of the initial workpiece's minimum cubic bounding box, and $B_{\max}$ corresponds to the maximum coordinate value of the same.

$$\begin{cases} x_{\min} = B_{\min} \cdot x \\ y_{\min} = B_{\min} \cdot y \\ z_{\min} = B_{\min} \cdot z \\ x_{\max} = B_{\max} \cdot x \\ y_{\max} = B_{\max} \cdot y \\ z_{\max} = B_{\max} \cdot z \end{cases} \quad (19)$$

As shown in Fig. 10, for voxels with all three faces on the outer layer, they are precisely located at the eight corner points of the initial cubic rough model, classified as point type, corresponding to image (a); for voxels with two faces on the outer layer, they are precisely located on the 12 edges of the initial cubic rough model, classified as edge type, corresponding to image (b); the remaining voxels that only need to display one face are precisely located on the six faces of the initial cubic rough model, classified as face type, corresponding to image (c). By defining these three different types of voxels, the faces that appear on the exterior are triangulated, thus transforming the dynamically octree-organized rough model into a surface mesh model.

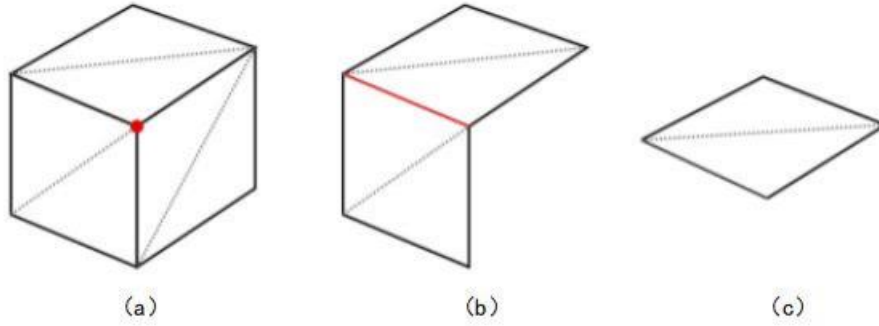


Fig. 10 Different Types of Voxel Surfaces

As shown in Fig.11, for point-type voxels, the three faces sharing that point are divided into six triangles; for edge-type voxels, the two faces sharing that edge are divided into four triangles; and for face-type voxels, the face is divided into two triangles.

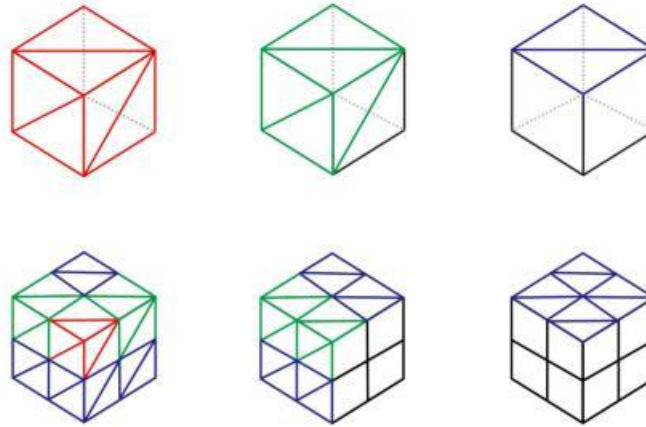


Fig. 11 Different Types of Voxel Division Rules

In the previous introduced, the inconsistency between voxel models and triangular mesh models was addressed. However, not all voxel models can be directly converted into triangular meshes. For voxel models that are completely outside the tool, a direct conversion is feasible; for those entirely inside the tool, the voxels can be simply deleted. However, for the cubes involved in cutting surface fitting, neither displaying the voxels (which would obscure the cutting surface mesh) nor directly converting them into surface meshes (which would lead to overlap with the cutting surface mesh) is viable. Therefore, additional triangles are needed to fill this area. On this basis, this paper introduces the concept of a crack table. Fig.12 illustrates the current problem.

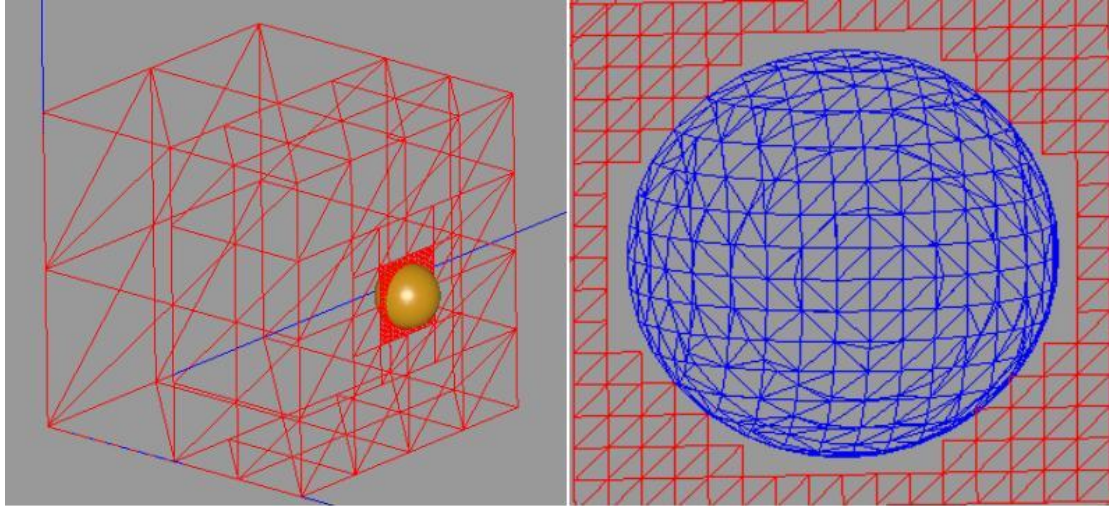


Fig. 12 Voids in Voxel Surface Mesh and Cutting Surface Mesh

Our crack table, inspired by the DC method's method to connecting corner points in different states, is applied to planes, as the patching process mainly concentrates on the outermost plane. The cubic voxels needing patching are those fitting the cutting surface triangular mesh, hence retaining state information at their corners. For a planar quadrilateral, there are a total of 24 possible internal and external corner states, but this can be simplified into three categories due to symmetry (in Fig. 13, the quadrilateral corner states are opposite to those in the DC algorithm, with solid dots representing outside the iso-surface and hollow dots inside).

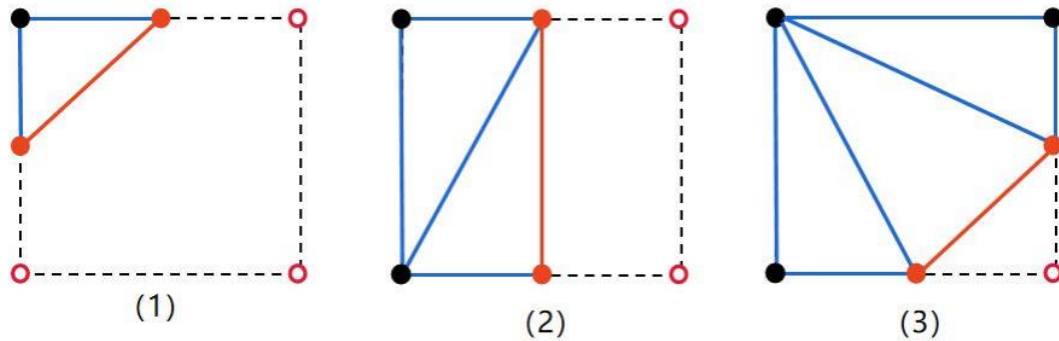


Fig. 13 Basic Connection Rules of Crack Table

Similar to the DC process, our crack table first defines the connections for different states. Next, based on the index values returned by the quadrilateral, it looks up the corresponding topological connections in the crack table to achieve the stitching of the surface voxel with the tool's cutting surface. After each cutting operation, repeating this process yields a complete surface triangular mesh model of the workpiece. Fig.14 shows a schematic of the area where the crack is stitched.

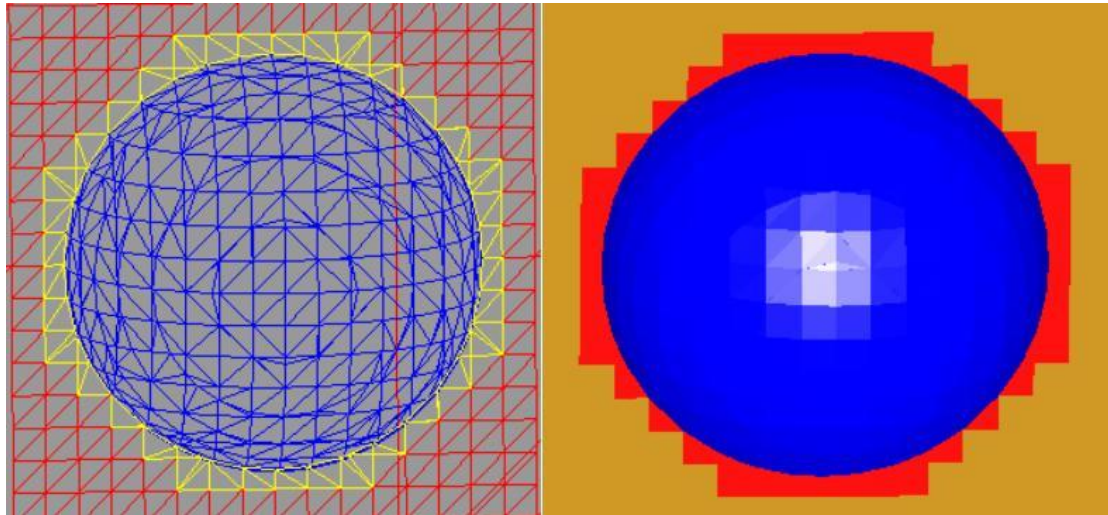


Fig. 14 Schematic Diagram of Crack Surface Repair

Thus, the overall workflow of the cutting operation algorithm based on the rough model of the workpiece cube is shown in Fig.15, and the mesh generated by the actual cutting process is shown in Fig.16, where red meshes represent the cubic surface mesh, blue meshes represent the cutting surface, and yellow meshes represent the patching surface.

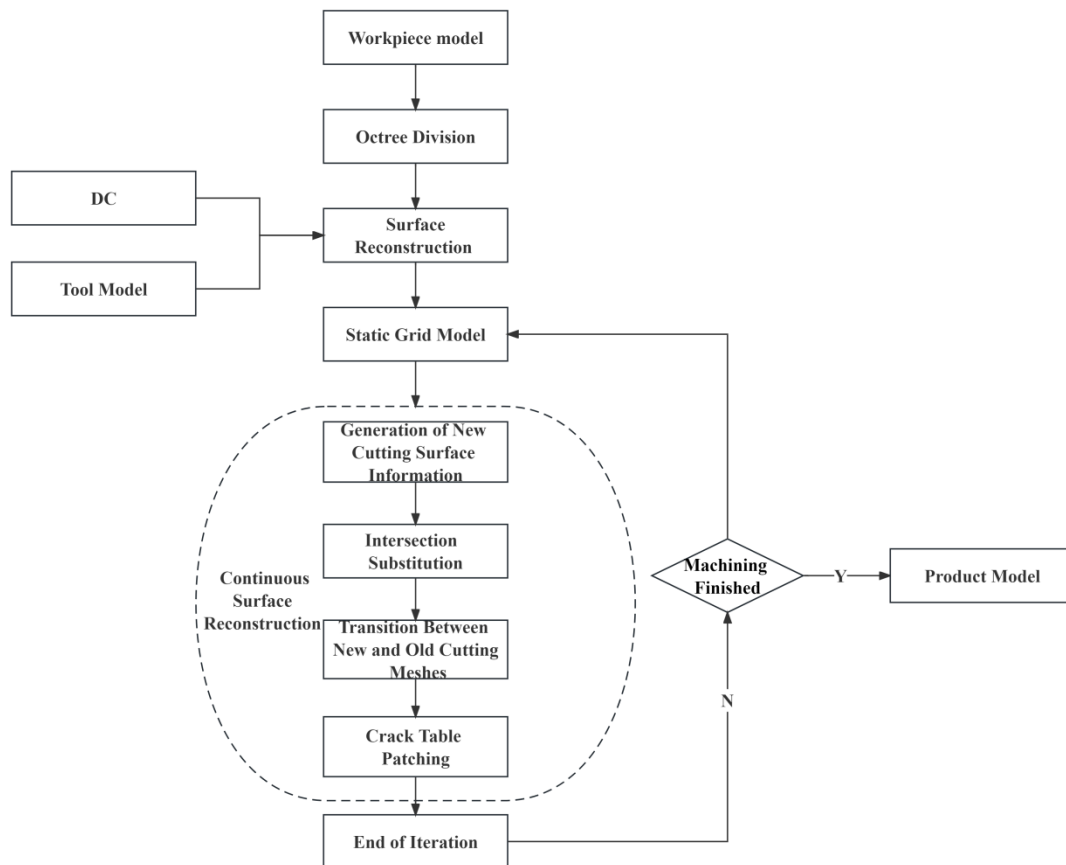


Fig. 15 Algorithm Flow Chart

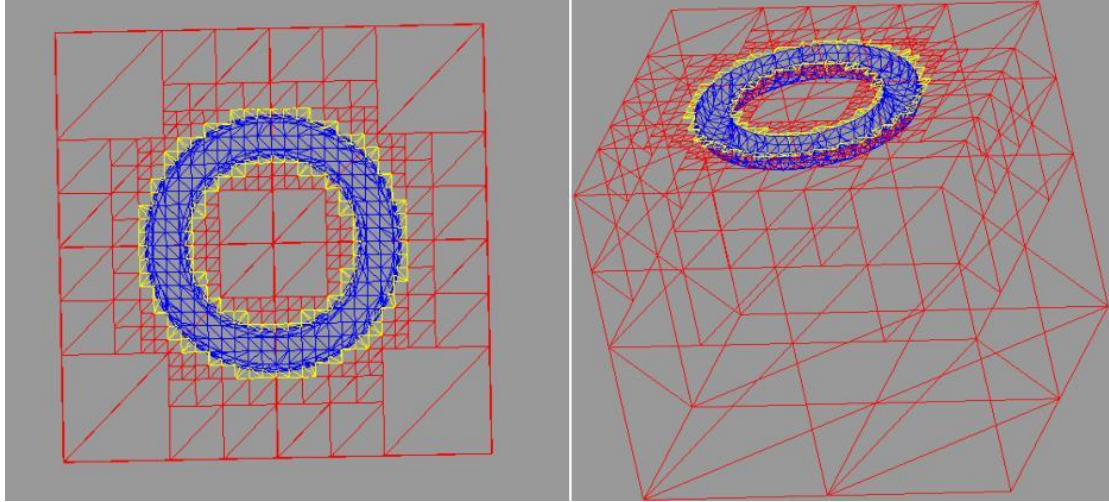


Fig. 16 Algorithm Rendering

4 Experiments and Results

We apply the algorithm to the dynamic cutting simulation of aero-engine blades. In this experiment, we have chosen to test on older hardware configurations to demonstrate the performance of our algorithm in a broader range of application scenarios. By utilizing a computer system equipped with an Intel Core i7-3770 quad-core CPU, 128GB memory, and NVIDIA GeForce GTX 750 Ti graphics card, we can ensure that the experimental results have universal applicability. If the algorithm performs well under this configuration, its performance will be further improved on newer and more powerful hardware. This helps to prove that our method has wider applicability and better performance in practical applications. The simulation method was programmed in the C++ language, and Open-CasCade was used for both the 3D modeling graphics library and the display library.

Firstly, we conducted experiments to validate the efficacy of dynamic vertex updating in achieving smooth transitions during the cutting process. Fig. 17(a) demonstrates that the cutting surface aligns well after implementing dynamic vertex updating in machining. Conversely, if vertex updating is not applied, the entire cutting surface is cluttered with numerous redundant triangles, as depicted in Fig. 17(b). This underscores the effectiveness of dynamic vertex updating in addressing issues inherent to continuous modeling in voxel-based simulations, ensuring good visualization effect of cutting simulation.

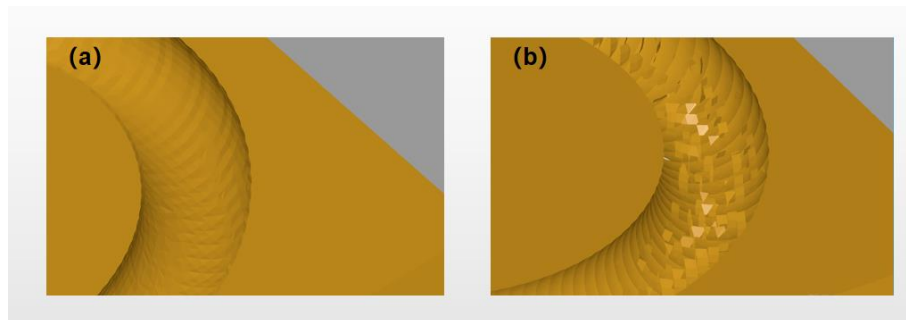


Fig.17 Cutting effect with or without dynamic vertex updating

The foundational algorithm of this paper was validated in a regular $20 \times 20 \times 20$ region. Initially, a voxelized rough model with dynamic octree indexing was established in the data region. Then, a ball-

nosed cutter with a radius of 1 was used to discretize the tool path along a radius 5 circular arc, performing one hundred tool position points discretization. The cutting effects at different octree subdivision depths and the efficiency comparison during the transition of the cutting surface were evaluated, as shown in Fig.18. Specifically, (1), (2), (3), and (4) in the figure represent experiments conducted in data scenes with octree subdivision depths of 4, 5, 6, and 7, respectively.

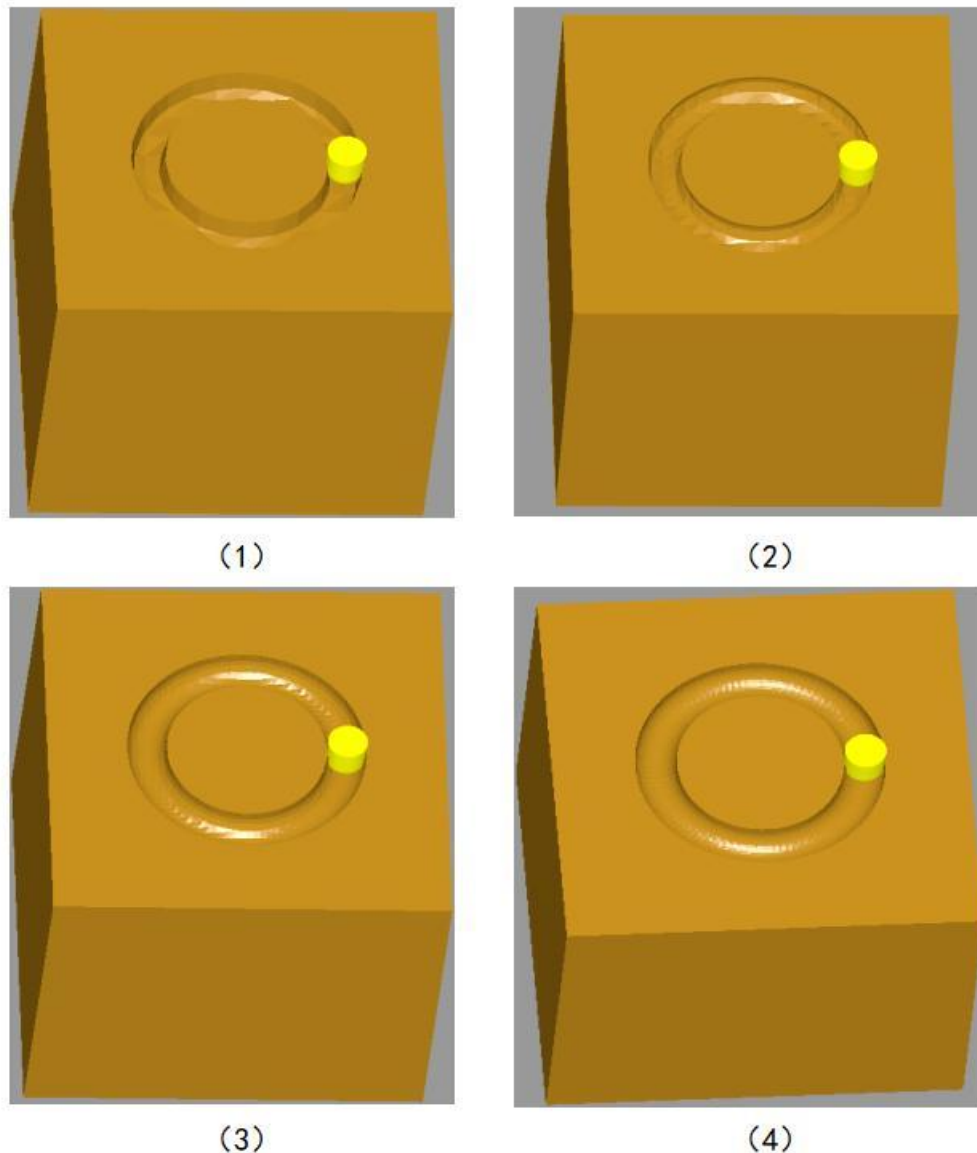


Fig.18 Visualization of cutting simulation

Table 4 shows the time consumed for transition calculations and the total time required for each cutting. In combination with the comparison in Fig.18, it can be observed that when the octree partitioning precision is set to 4 and 5, there are fewer triangles in the fitted cutting surface, resulting in less time consumption. However, the accuracy and visual effects of such models do not meet the requirements for modeling and effective visualization. When the partitioning precision is set to 6 and 7, the visual and accuracy requirements are met, but the total time consumption at 7 is three times that of 6. The reason for this is that each cutting requires voxelization and layer-by-layer data traversal, necessitating further algorithm optimization. This experiment demonstrates that the visualization effects achieved using the

voxelization method meet the requirements of the machining simulation fitting process.

Table 4 Calculation Efficiency with Different Accuracies

Division Depth	Cutting Triangle Quantity	Grid Transition/ms	Total Times/ms
4	256	0.27ms	11.59ms
5	874	1.44ms	35.52ms
6	3752	4.83ms	128.44ms
7	15588	15.29ms	459.37ms

After determining the effects of the octree subdivision depth, we proceeded to validate our method on industrial product objects. As shown in Fig. 19, the rough model of a aero-blade was constructed using traditional voxelization methods and set with an octree structure, which resulted in the loss of sharp edges as shown in image (a). However, using the dual contouring algorithm, the sharp edges in the blade rough model were well-preserved, as shown in image (b). Both had an octree division depth of 7, showing that the dual contouring algorithm can represent more accurate information.

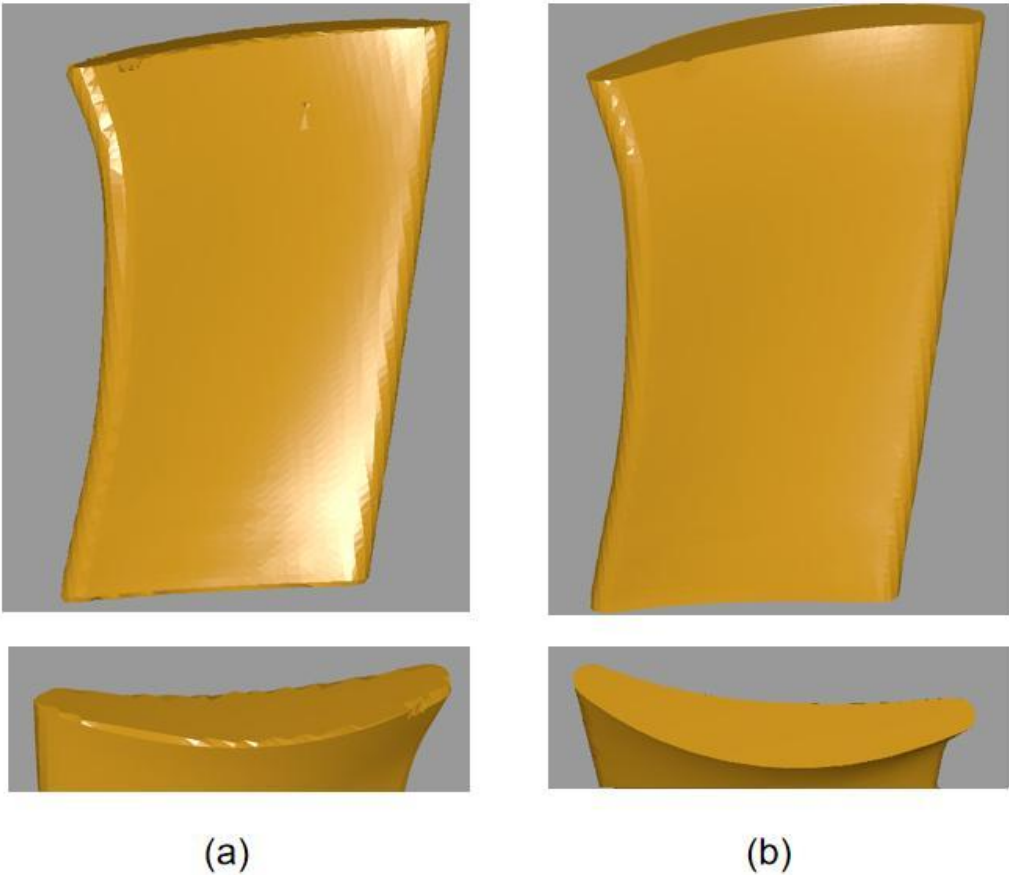


Fig.19 Aero-Blade Modeling Effect

After voxelizing the initial rough information using the dual contouring algorithm, the subsequent cutting process also employed this algorithm. Compared to the moving cubes method, the dual contouring algorithm, which adds normal information for fitting, achieves better display effects at a subdivision depth of 7. Since the dual contouring algorithm does not target individual voxels, this paper encoded the

octree indexing structure to enhance the connection efficiency of the dual contouring algorithm. This enabled the successful application of the dual contouring algorithm in CNC simulation. Table 5 shows the 4 times simulation process of average fitting time for the blade mesh during a single cut. As discussed earlier, a good rough model can be obtained at a subdivision depth of 7, so there is no need for higher precision divisions. Using the DC algorithm alone for voxelization fitting is highly inefficient due to the massive data volume. After optimization with the octree, the time was reduced by two-thirds but still did not meet the real-time visualization needs. This is due to the differing data structures of the octree and triangular mesh, requiring the octree to be regenerated after each cut. However, using EDCCSR enabled the dynamic matching of octree data with triangular mesh data, thereby enhancing computational time, and the final results met the requirements for real-time visualization.

Table 5 Voxel-based modeling method for continuous surface reconstruction

	Dual Contouring	Octree Optimization	Ours
1	324.80ms	173.97ms	38.69ms
2	303.02ms	153.26ms	37.45ms
3	318.94ms	169.01ms	37.50ms
4	319.56ms	143.16ms	38.66ms

We conducted comparative experiments of our algorithm with traditional simulation process visualization methods, namely, Boolean intersection operations, measuring the time required for single-cut visualization.

The results obtained are shown in Fig. 20. It was observed that the efficiency of Boolean operations gradually decreased as the machining process advanced, resulting in an increase in visualization time. Conversely, our method resulting in stable visualization time.

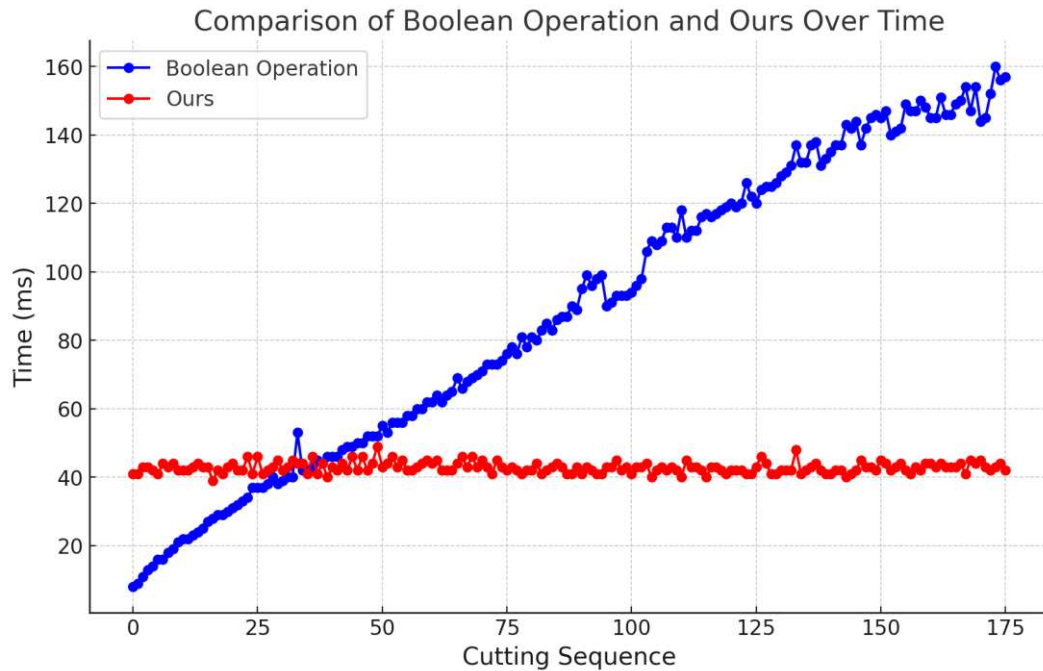


Fig. 20 Time comparison between Boolean operations and our algorithm per cutting

The Table 6 provides a statistical comparison of the time spent on the Boolean operation and our for over 176 cutting sequences. For the Boolean operation, the average time spent is approximately 88.09 ms with a standard deviation of 44.00 ms, indicating a moderate level of variability in the time spent. The minimum and maximum times observed are 8.00 ms and 160.00 ms, respectively. The Coefficient of Variation (CV), a measure of relative variability, is 0.50, suggesting a moderate level of stability in the time spent on the Boolean operation. On the other hand, for ours, the average time spent is around 42.82 ms with a standard deviation of just 1.58 ms, indicating a low level of variability in the time spent. The minimum and maximum times observed are 39.00 ms and 49.00 ms, respectively. The CV for our operation is 0.04, which is much lower than that for the Boolean operation. This low CV indicates a high level of stability in the time spent on our operation. In summary, while the Boolean operation and our operation show different averages and ranges of time spent, our operation exhibits a higher level of stability as evidenced by its lower CV.

Table 6 Statistical comparison of Boolean operation and ours

	Boolean Operation	Ours
Count	176 times	176 times
Mean	88.09ms	42.82ms
Standard Deviation	44.0	1.58
Min	8.00ms	39.00ms
Max	160.00ms	49.00ms
Coefficient of Variation	0.50	0.04

To further discuss our method, we counted the number of triangles during cutting, as shown in Fig. 21. This plot shows the number of surfaces over time along with its rolling mean and standard deviation. The rolling mean (in red) represents the average number of surfaces over a sliding window of 5 cutting sequences, which gives an indication of the trend over time. The shaded area represents the rolling standard deviation, which provides a measure of the variability of the number of surfaces over time. The plot shows that both the rolling mean and standard deviation are relatively stable, indicating that the number of surfaces is stable and does not show a significant increasing or decreasing trend over time.

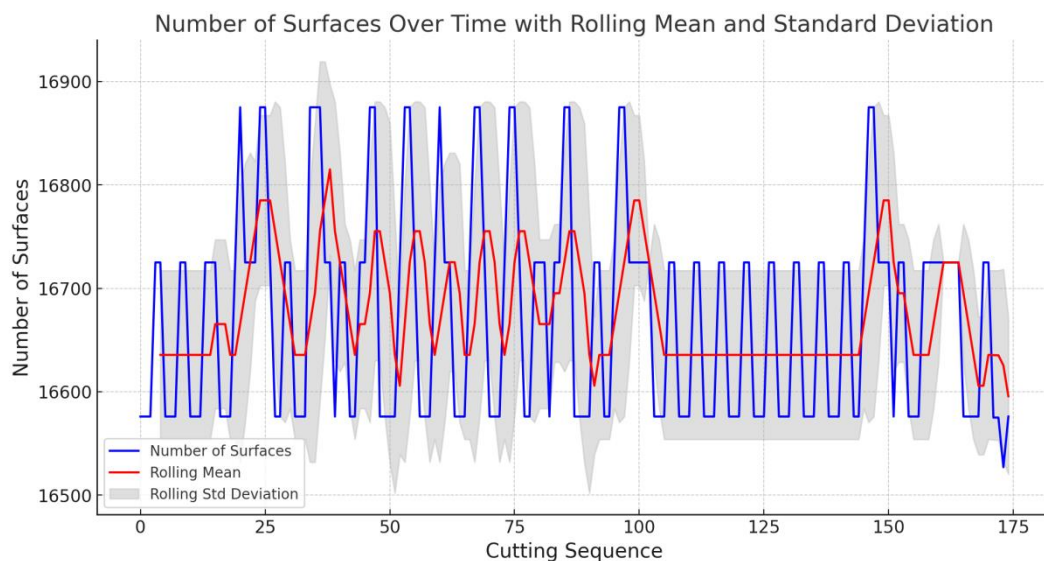


Fig. 21 Change in the number of surfaces per cutting

The cutting diagram based on the rough mesh is shown in Fig.22, where (a) represents the initial blade rough, (b) shows the mesh rough fitted with the Dual Contouring algorithm. The experiment used a ball-nosed cutter with a 3mm radius, cutting around the top back part of the blade rough mesh along the negative X direction. (c) and (d) show the specific cutting effects. Additionally, as for the accuracy of the model, we imported the 4 groups point cloud data generated from the actual blade product through 3D laser scanning into our system and compared it with the geometry model constructed using parameter knowledge, as shown in Fig. 23. We selected four cross-sections, namely the 1st, 3rd, 5th, and 9th, for error analysis. The results are presented in Table 7. By evaluating indicators such as leading and trailing edges, as well as chord curvature, we found that the knowledge-driven geometry model exhibited excellent accuracy performance. The errors were within an acceptable range, which meets the requirements of manufacturing process for geometry models.

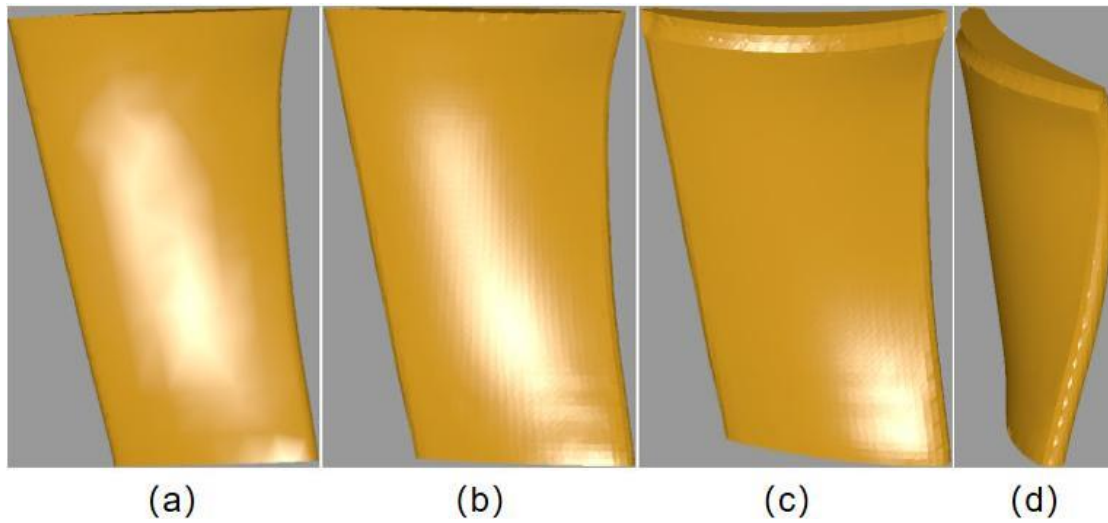


Fig. 22 Leaf blank cutting display



Fig. 23 Comparison of cross sections with geometry model for 4 groups of real point cloud data

Table 7 Error Between Geometry Model and Actual Blade

	Cross-section 1		Cross-section 5		Cross-section 9		Cross-section 13	
	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
CV	-0.0481	-0.0248	-0.0252	0.0055	-0.0129	-0.0040	-0.0133	-0.0016
CC	-0.0478	-0.0148	-0.0264	0.0006	-0.0185	-0.0016	-0.0123	-0.0009
LE	-0.0432	-0.0008	-0.0192	0.0294	-0.0147	0.0136	-0.0136	0.0156
TE	-0.0391	-0.0058	-0.0362	0.0351	-0.0232	0.0169	-0.0176	0.0364
MAX_Tolerance	0.1mm		0.1mm		0.1mm		0.1mm	

5 Discussion

Although the current methods for constructing solid geometric models, such as CSG (Constructive Solid Geometry) and B-rep (Boundary Representation), can accurately represent an object, the number of Boolean operations in CNC simulations often reaches thousands to tens of thousands, making it difficult to meet real-time requirements. Therefore, current CNC simulation modeling methods primarily adopt discrete modeling method. This paper starts from various methods of discrete modeling and summarizes the disadvantages of using voxel models alone: in voxel-based machining simulations, material removal is achieved by removing voxels. The higher the precision, the more voxels are used, increasing the memory resources and hardware requirements. Also, this method struggles to achieve good visual effects, making it unsuitable for high-precision cutting simulation. The z-buffering model has a contradiction between data structure and display, making it challenging to display effectively. Surface reconstruction requires more complex algorithms, so z-buffering does not meet the real-time needs of high-precision CNC simulation. When performing Boolean operations with triangular mesh models, the output is also a mesh model. Although this method can achieve higher accuracy and visual effects, direct intersection tests with numerous triangles are often time-consuming and unacceptable, making this method also unsuitable for real-time display in CNC machining. By comparing the advantages and disadvantages of these three methods, this paper combines their strengths to propose EDCCSR, introducing the dual contouring algorithm to solve the issue of sharp edges in complex industrial models. Also, to improve the fitting process of the dual contouring algorithm and enhance its connection efficiency, the paper uses 6-adjacency encoded octree indexing, making it more applicable to CNC simulation display. Converting the outer voxel model into a surface mesh to unify data formats and using voxel data structure for intersection substitution allows smooth transitions of cutting surfaces at different tool positions, overcoming the limitations of direct triangular mesh operations. Since all triangles carry a tree structure index, subsequent operations can be controlled in a small range of 3D data fields, completing the process in a shorter time. Finally, the paper utilizes the DC algorithm to extract crack tables from the boundary voxel data for patching the transition area between the voxel and cutting surface, achieving effective CNC milling simulation results.

Our experiments still have some problems that need to be further studied. Future research directions are proposed as follows:

The voxel-based CNC machining simulation algorithms proposed in this paper are all written and run on the CPU. DC support parallel processing, and the code can be improved to the parallel mode in the future to improve the operation's efficiency further. Finally, the modeling graphics library Open-CasCade(OCC) used in this paper has great advantages in the modeling direction but is less efficient in rendering. For

the problem of inefficient real-time rendering of large numbers of facets, the next step should be to consider the introduction of a professional rendering library to optimize the system and improve rendering efficiency, allowing for further visual optimization. Meanwhile, although our algorithm is verified in time and accuracy, the visual effect is not good because of the limitation of OCC itself, which cannot display the normal effect well, so we consider OpenSceneGraph as a display method.

6 Conclusion

In the field of industrial simulation, the visualization of simulation process models is a critical component. The challenge of matching a real-time visualization method to a complex and continuously changing geometric model during the simulation process is considerable. Our method offers a fresh perspective to address this challenge. The Enhanced Dual Contouring Continuous Surface Reconstruction (EDCCSR) ensures a rapid surface reconstruction of the geometric model, while the integration of Dual Contouring guarantees an accurate representation of the model's intricate geometric structure. A significant improvement of our method is 6-adjacency encoded octree for fast surface reconstruction and the surface meshing of the voxel model, using the Dual Contouring algorithm to create crack tables. This resolves the issue of transitioning cutting surfaces in visualization simulations. In summary, our method provides a new avenue for achieving real-time and accurate visualization during the simulation process.

Competing Interests

The authors declare that they have no competing interests or financial conflicts to disclose.

Funding: This research is supported by National Key R&D Program of China(2020YFB1710401).

Data Availability: The data that support the findings of this study are available from Aecc South Industry Company Limited, but restrictions apply to the availability of these data, which were used under license for the current study, and so are not publicly available. Data are however available from the authors upon reasonable request and with permission of Aecc South Industry Company Limited [Contacts: Mr.Han;E-mail: yonghan@ouc.edu.cn].

Reference

- [1] Aurich, J.C., Biermann, D., Blum, H. et al. Modelling and simulation of process: machine interaction in grinding. *Prod. Eng. Res. Devel.* 3, 111 – 120 (2009).
- [2] Zhou, Z.D., Zhou, J.D., Chen, Y.P., Ong, S.K., & Nee, A.Y.C. (2003). Geometric Simulation of NC Machining Based on STL Models. *CIRP Annals*, 52(1), 129-134.
- [3] Wöste, F., Siebrecht, T., Fast, M., & Wiederkehr, P. (2019). Geometric Physically-Based and Numerical Simulation of NC-Grinding Processes for the Calculation of Process Forces. *Procedia CIRP*, 86, 133-138.
- [4] Rajain, K., Sliusarenko, O., Bizzarri, M. and Barton, M., 2022. Curve-guided 5-axis CNC flank milling of free-form surfaces using custom-shaped tools. *Computer Aided Geometric Design*, 94, p.102082.
- [5] Machchhar, J., Plakhotnik, D. and Elber, G., 2017. Precise algebraic-based swept volumes for arbitrary free-form shaped tools towards multi-axis CNC machining verification. *Computer-Aided Design*, 90, pp.48-58.
- [6] Barton, M., Bizzarri, M., Rist, F., Sliusarenko, O. and Pottmann, H., 2021. Geometry and tool motion planning for curvature adapted CNC machining. *ACM Transactions on Graphics*, 2021, 40(4), 180

- [7] Duvedi, R.K., Bedi, S., Batish, A. and Mann, S., 2015. Numeric implementation of drop and tilt method of 5-axis tool positioning for machining of triangulated surfaces. *The International Journal of Advanced Manufacturing Technology*, 78(9-12), pp.1677-1690.
- [8] Kim, Y.J., Elber, G., Barton, M. and Pottmann, H., 2015. Precise gouging-free tool orientations for 5-axis CNC machining. *Computer-Aided Design*, 58, pp.220-229.
- [9] A. Arun Premnath,P. Suryatheja,A. Srinath,S. Karthikeyan. Analysis of Tool Wear while Milling Hybrid Metal Matrix Composites[J]. *Applied Mechanics and Materials*,2015,4240:813-814.
- [10] V. Dhanalakshmi, K. Sankaranarayan, Subramaniam Arunachalam, S.C. Lenny Koh. Extraction of simple and concatenated features from a wire frame model using IGES data[J]. *Int. J. of Manufacturing Technology and Management*,2006,9(3/4).
- [11] Goldfeather J , Hultquist J , Fuchs H . Fast constructive-solid geometry display in the pixel-powers graphics system[J]. 1986:107-116.
- [12] Yan, Q.W., Chen, C.L.P., & Tang, Z. (1994). Efficient algorithm for the reconstruction of 3D objects from orthographic projections. *Computer-Aided Design*, 26(9), 699-717.
- [13]]Spence, A. D., and Altintas, Y. (February 1, 1994). "A Solid Modeller Based Milling Process Simulation and Planning System." *ASME. J. Eng. Ind.* February 1994; 116(1): 61 – 69.
- [14] Nehring-Wirxel, J., Trettner, P., & Kobbelt, L. (2021). Fast Exact Booleans for Iterated CSG using Octree-Embedded BSPs. *Computer-Aided Design*, 135, 103015.
- [15] Tzoumas, G., Michelucci, D., & Foufou, S. (2015). Extending CSG with projections: Towards formally certified geometric modeling. *Computer-Aided Design*, 66, 45-54.
- [16] Chappel, I.T. (1983). The use of vectors to simulate material removed by numerically controlled milling. *Computer-Aided Design*, 15(3), 156-158.
- [17] El-Mouna Yri, H., EL .B Estawi, M.A., Spence, A.D., et al. (1997). General geometric modelling method for machining process simulation. *The International Journal of Advanced Manufacturing Technology*, 13(4), 237-247.
- [18] BOZ, Y., ERDIM, H., & LAZOGLU, I. (2015). A comparison of solid model and three-orthogonal dexelfield methods for cutter-workpiece engagement calculations in three- and five-axis virtual milling. *International Journal of Advanced Manufacturing Technology*, 81(5-8), 811-823.
- [19] KARUNAKARAN, K.P., SHRINGI, R., RAMAMURTHI, D., et al. (2010). Octree-based NC simulation system for optimization of feed rate in milling using instantaneous force model. *International Journal of Advanced Manufacturing Technology*, 46(5-8), 465-490.
- [20] Liu, S.Q., Ong, S.K., Chen, Y.P., & Nee, A.Y.C. (2006). Real-time, dynamic level-of-detail management for three-axis NC milling simulation. *Computer-Aided Design*, 38(4), 378-391.
- [21] LI, J.G., DING, J., GAO, D., et al. (2010). Quadtree-array-based workpiece geometric representation on three-axis milling process simulation. *The International Journal of Advanced Manufacturing Technology*, 50(5-8), 677-687.
- [22] Van Hook, T. (1986). Real-time shaded NC milling display. *ACM SIGGRAPH Computer Graphics*, 20(4), 15-20.
- [23] Huang, Y., & Oliver, J.H. (1995). Integrated simulation, error assessment, and tool path correction for five-axis NC milling. *Journal of Manufacturing Systems*, 14(5), 331-344.
- [24] Saito, T., & Takahashi, T. (1991). NC machining with G-buffer method. *ACM SIGGRAPH Computer Graphics*, 25(4), 207-216.
- [25] Lee, S.W., & Nestler, A. (2012). Virtual workpiece: workpiece representation for material removal process. *The International Journal of Advanced Manufacturing Technology*, 58(5-8), 489-499.

- [26] Kiswanto, G., Hendriko, H., & Duc, E. (2015). A hybrid analytical-and discrete-based methodology for determining cutter-workpiece engagement in five-axis milling. *The International Journal of Advanced Manufacturing Technology*, 80(9), 2083-2096.
- [27] Feito, F.R., Ogáyar, C.J., & Segura, R.J. (2013). Fast and accurate evaluation of regularized Boolean operations on triangulated solids. *Computer-Aided Design*, 45(3), 705-716.
- [28] Guo, K.B., Zhang, L.C., Wang, C.J., et al. (2007). Boolean operations of STL models based on loop detection. *The International Journal of Advanced Manufacturing Technology*, 33(5), 627-633.
- [29] Chen, M., Chen, X.Y., Tang, K., et al. (2010). Efficient boolean operation on manifold mesh surfaces. *Computer-Aided Design and Applications*, 7(3), 405-415.
- [30] Lorensen, W.E., & Cline, H.E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4), 163-169.
- [31] Zhou, M., Xiao, M., Zhang, Y., Gao, J., & Gao, L. (2022). Marching cubes-based isogeometric topology optimization method with parametric level set. *Applied Mathematical Modelling*, 107, 275-295.
- [32] Wang, X., Gao, S., Wang, M., & Duan, Z. (2021). A marching cube algorithm based on edge growth. *Virtual Reality & Intelligent Hardware*, 3(4), 336-349.
- [33] S. Zhang, Y. Hu and Z. Li, "Machining Simulation Application Based on Improved Marching Cubes Algorithm," 2023 12th International Conference of Information and Communication Technology (ICTech), Wuhan, China, 2023, pp. 197-201, doi: 10.1109/ICTech58362.2023.00048.
- [34] Ju, T., Losasso, F., Schaefer, S., et al. (2002). Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (pp. 339-346).