





Contour-Type Cutter Path Computation Using Ultra-High-Resolution Dixel Model

Masatomo Inui¹  and Nobuyuki Umezu² 

¹Ibaraki University, masatomo.inui.az@vc.ibaraki.ac.jp

²Ibaraki University, nobuyuki.umezu.cs@vc.ibaraki.ac.jp

Corresponding author: Masatomo Inui, masatomo.inui.az@vc.ibaraki.ac.jp

Abstract. We propose a novel algorithm for computing a contour-type cutter path for a mold CAD model in the polyhedral representation. Our algorithm uses a Minkowski sum shape of the mold model and a cutter model in the inverted orientation. To realize a robust Minkowski sum computation, a grid-based shape representation called a two-directional dixel model is used. This method can represent the 2D object with a much smaller amount of memory than the regular square-mesh-based method. Using this advantage, we realize a two-directional dixel models based on an ultra-high-resolution grid. Parallel processing with many cores of graphics processing unit (GPU) is also introduced for reducing the computation time. An experimental path computation software is implemented, and its performance is analyzed.

Keywords: NC Milling, Minkowski Sum, Solid Modeling, GPU, CAM

DOI: <https://doi.org/10.14733/cadaps.2020.621-638>

1 INTRODUCTION

Cutter path computation for numerical-controlled (NC) three-axis machining is still an important topic for computer-aided manufacturing (CAM) systems. In the machining of a mold part with a complex curved surface, a contour-type cutter path with very small vertical intervals is required. The Development of an algorithm for a robust, accurate, and fast cutter path computation is required by CAM software companies. In this paper, we propose a novel algorithm for computing the contour-type cutter path for mold CAD models in the polyhedral representation.

Our algorithm uses a Minkowski sum of the mold CAD model and a cutter model in the inverted orientation in the cutter path computation. The Minkowski sum for a polyhedral object can be obtained by computing the Minkowski sum shapes for the individual polygons (triangles) of the object and by computing their Boolean union shape. In the conventional boundary representation CAD modeling, the difficulty of the Minkowski sum computation increases significantly in the case of a complex model with many polygons, because topological reconstruction by trimming and reconnecting the surface elements into a closed model is iterated many times.

To overcome this difficulty, a grid-based shape representation with no topological information is used. Grid-based shape representation can be considered as the sampling of the surface points according to the spatial cell structure. To reduce the shape error caused by the sampling interval, the use of a cell structure based on an ultra-high-resolution grid is necessary. This approach results in an inevitable increase in the memory usage and processing time required. A two-directional dixel model is introduced for reducing the memory consumption in the model representation. Parallel processing with many cores of graphics processing unit (GPU) is also used for reducing the computation time. An experimental path computation software is implemented, and its performance is analyzed.

In the next section, related studies on the cutter path computation are briefly explained. In Section 3, a two-directional dixel model for representing the Minkowski sum shape and a GPU accelerated parallel algorithm for computing the Minkowski sum shape are illustrated. The computation method of the contour-type cutter path using the Minkowski sum shape is given in Section 4. The experimental computation results are provided in Section 5, and we summarize our conclusions in Section 6.

2 RELATED STUDIES

Because cutter path computation is a core function of the CAM software, several corresponding methods have been studied. A survey paper [5] presented a classification of the cutter path computation methods developed from the period 1989 through 1994. In this paper, the path computation methods for three-axis milling are classified as iso-parametric methods, methods for pocketing, tool-positioning methods, and Minkowski-sum-based methods (offset-based methods in [5]). Although several methods have been developed since then, they can be classified into any of these four groups. In the contour-type path generation, either the tool-positioning method or Minkowski-sum-based method are used. In the former method, it is necessary to repeat the convergence calculation when determining the appropriate tool position, which generally requires a longer processing time. Therefore, recently, the path computation based on the Minkowski sum has become more popular.

Incorrectly computed paths may result in the undesirable results of milling too deeply (gouging) or leaving too much material on the workpiece surface. These problems can be avoided by computing a cutter location (CL) surface before the path generation. The CL surface represents a trajectory surface of the reference point of a cutter when the cutter is slid over the mold surface. Figure 1(a) illustrates a CL surface for a rather large cutter that is typically used in the initial rough milling stage. Once the CL surface is obtained, the computation of a gouge-free cutter path is easily realized by controlling the reference point of the cutter such that it is always on or above the CL surface. Let us consider a local coordinate frame at the reference point of the cutter and generate its inverted shape by transforming point \mathbf{p} in the cutter to $(-\mathbf{p})$. The CL surface corresponds to the top surface of the Minkowski sum shape of the part surface and the inverted cutter as shown in Figure 1(b).

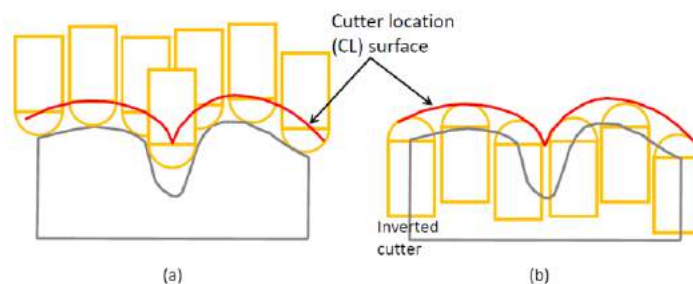


Figure 1: Cutter location surface (a) and Minkowski sum of a mold part and the inverted cutter (b).

In this method, the path computation process can be subdivided into two steps, the Minkowski sum computation step for obtaining the CL surface (Step 1) and its succeeding step for tracing the cutter path in the CL surface (Step 2). In the contour-type path computation, the latter step is an iteration of the slicing operation of the CL surface with horizontal planes. As the process in the first step is more critical than the slicing operation, we review previous studies on the Minkowski sum computation in this section. The Minkowski sum of a solid object and an inverted ball-end cutter can be obtained using the offset computation. Offsetting is one of the most fundamental operations in geometric modeling [15]. The offset computation of a 3D object in the boundary representation is a complex process, because it must handle both the offsetting of individual surfaces in the model as well as topological reconstruction by trimming and connecting the offset surfaces into a closed model.

Earlier techniques for offsetting 3D models [6, 15, 17] were often computationally expensive, and model reconstruction can be unstable. To overcome these difficulties, new offset computation methods based on the discrete representation of the 3D model have become popular. Known representation schemes utilize points, voxels, dexels [19], rays [13], and layered depth images (LDIs) [18], and various improvements, e.g., triple-dexels [1], have been reported. As discrete 3D models do not have surface elements, the topological reconstruction step, which is the most critical process in conventional offsetting, is unnecessary. After the offsetting, a polyhedral model of the offset shape is derived by applying a surface extraction technology, such as marching cubes [12] or dual contouring [19], to the discrete model.

Let us consider a 3D object in a box-like space. The distance field is the spatial grid structure in which the distance from the point to the closest surface of the object is recorded at each grid point. Many researchers have presented distance field-based offsetting methods [2, 3, 7, 11]. For some offset radius r , the offset surface of the model goes across an edge connecting a grid point with a distance greater than r with another point at a distance less than r . After detecting such edges, a marching cubes method (or similar) can be used to determine the polygonal offset surface. A similar idea was used in [22], wherein several filtering methods were developed for reducing the computation cost of the distance field.

Li and McMains discussed a voxelized Minkowski sum computation with culling techniques [9, 10]. Their method first generates possible surface elements of the Minkowski sum shape of two objects. The voxels corresponding to the Minkowski sum shape are then selected according to the surface elements. The offsetting method proposed by Wang and Manocha uses LDIs to record the object shape and temporal result of the offset computation [21, 23]. Zhao et al. developed a compact LDI (CLDI) approach, which offers improved data storage technology to reduce the amount of memory required [24]. These works also use the parallel processing capability of GPUs to accelerate the computation.

The Minkowski sum computation with the use of discrete shape representation can be considered as a sampling operation of the surface points according to the spatial cell structure. To obtain the accurate result, the cell structure with a sufficiently small grid size, that is a grid of a sufficiently large resolution is necessary, which inevitably results in a large memory usage in the model representation. In the cutter path computation, an accuracy of 0.001 mm is required in the result path, and therefore, an ultra-high-resolution grid—for example, a resolution greater than 20,000—becomes necessary in the Minkowski sum computation. In the current implementation methods comprising the use of the distance field, voxels, LDI, CLDI or triple-dexels, it is difficult to realize a model representation comprising the use of such a high-resolution grid.

3 STEP 1: MINKOWSKI SUM COMPUTATION

We consider the contour-type cutter path computation for a milling process with a ball-end, flat-end, or radius-end cutter in the vertical spindle axis direction. The input data of the algorithm consists of a polyhedral model of a mold part, the shape data of a cutter (tool radius R and corner radius r in Figure 2), and a list of z coordinates specifying the height information of the reference

point of the cutter (see Figure 2) in the contour-type path generation. The majority of commercial CAD systems provide a function for outputting the model data as a group of triangular polygons, such as in the STL format. STL models are prepared such that the shape difference between the original model with the curved surface and the mesh model obtained by the tessellation is less than a predefined small value ε .

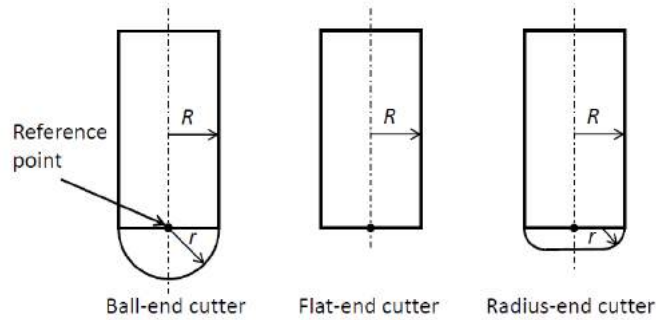


Figure 2: Cutter shape definitions.

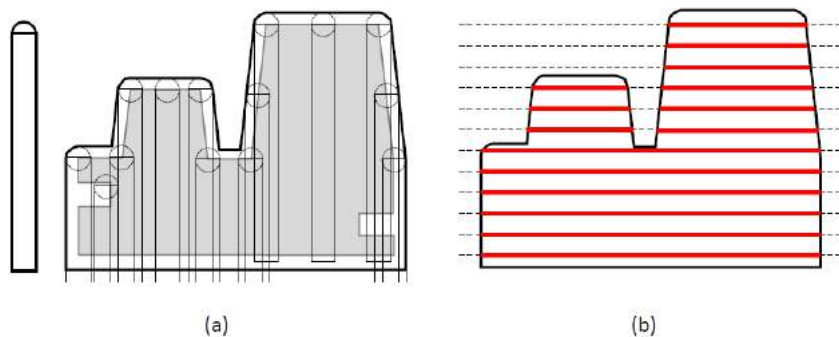


Figure 3: Contour-type cutter path computation process.

Our algorithm computes a Minkowski sum shape of the mold model and the cutter model in the inverted orientation (Figure 3(a)). The obtained shape is sliced by horizontal planes positioned at the heights specified in the given list (Figure 3(b)). The boundary curves of the cross-sectional figures correspond to the contour-type cutter path. In this section, the Minkowski sum computation algorithm is explained. To represent the Minkowski sum shape in the ultra-high-resolution discrete format within a limited memory capacity of an ordinary PC, a two-directional dixel model is introduced. The use of the parallel processing capability of the GPU for accelerating the computation is also explained.

3.1 Two-Directional Dixel Model

In the original dixel model, a 3D object is represented by a series of vertical segments (dexels) defined at each grid point in a regular square grid in the xy -plane [20]. Each segment corresponds to an overlapping range between the vertical ray that originates from each grid point and the object. In this method, near-vertical surfaces inevitably have staircase errors. A triple-dixel model was proposed to overcome this non-uniformity of the representation accuracy. In this representation, the 3D shape is not only defined by the z -axis-aligned (vertical) dexels, but also the x -axis-aligned dexels based on a grid in the yz -plane and y -axis-aligned dexels based on a grid in the zx -plane (Figure 4(a)) [1].

In the contour-type path computation, the Minkowski sum shape is sliced by a horizontal plane at a specific height. The boundary curves of the cross-sectional figure correspond to the path at

the height. To properly represent the section figures in the horizontal planes at the specific heights, we modify the grid definitions of the triple-dexel model. The Horizontal lines of the grids in the yz - and zx -planes are arranged such that their z -coordinates become equal to the height specifications for the contour-type path. As the z -axis-aligned dexels do not contribute to the cross-sectional figures in the horizontal plane, they are eliminated from the representation (Figure 4(b)). The elimination of the z -axis-aligned dexels and definition of horizontal lines of the grids only for specific heights facilitates a significant reduction in the amount of memory required for representing the model.

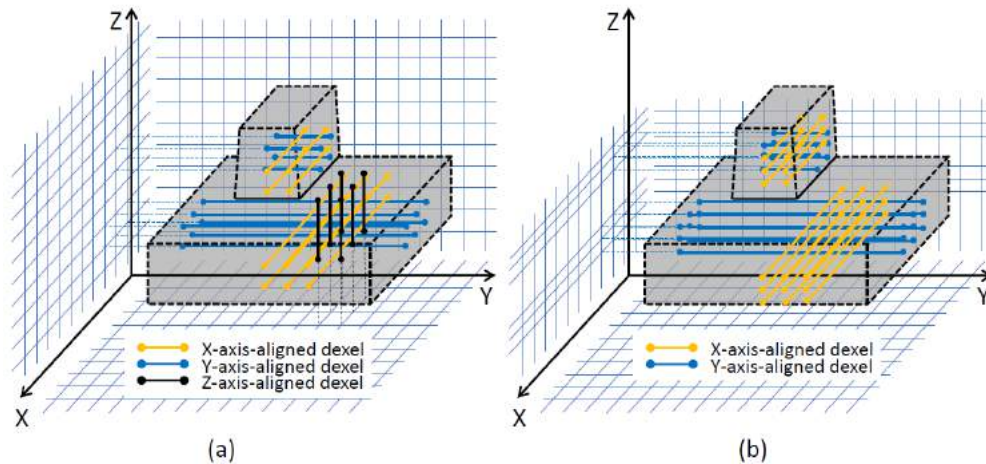


Figure 4: Triple-dexel model (a) and x-axis-aligned dexels and y-axis-aligned dexels for representing sectional figures (b).

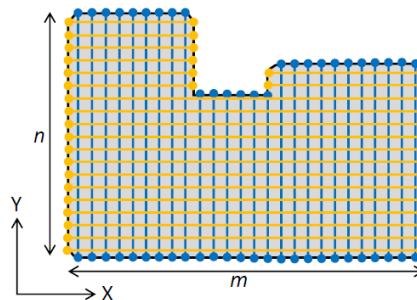


Figure 5: Two-directional dexel model for representing a cross-sectional figure in a slicing plane.

The cross-sectional figures in each slicing plane are represented by x-axis-aligned dexels and y-axis-aligned dexels. We call this representation method a two-directional dexel model. The two-directional dexel model is equivalent to a 2D figure representation method comprising the use of a regular square mesh. Let us consider a two-directional dexel model with a resolution m in the x -axis direction and n in the y -axis direction (Figure 5). In the two-directional dexel model, the amount of memory required for recording figures in the mesh is proportional to $m + n$, which is much smaller than mn and is necessary for recording the figures in the ordinarily square mesh. m (or n) can be larger than 20,000 in our current implementation. In the following explanation, we use the notation $[m, n]$ for describing the resolutions of the two-directional dexel model.

Using the hierarchical cell structure such as Quadtree [16], a similar memory reduction is realizable in the 2D shape representation. However, this representation method is not suitable for representing the shape with a dynamic modification—for example, the Minkowski sum operation—because frequent reconstruction of the hierarchical structure is necessary. The memory required for a 2D shape representation can be reduced by recording only the cells on the object's boundary. In this method, the internal and external parts of the object are indistinguishable, which makes it difficult to obtain the Boolean union shape in the Minkowski sum computation.

3.2 Minkowski Sum Computation for Two-Directional Dixel Model

Our Minkowski sum computation algorithm is a modification of our offset computation algorithm using the triple-dixel model [8]. Let us consider the Minkowski sum computation of a polyhedral object and an inverted flat-end cutter of radius R and length l . We assume that l is sufficiently large (larger than the height of the target mold part). This processing can be decomposed into the computation of the Minkowski sum of individual triangular polygons constituting the object and the inverted cutter (see Figures 6 (b) and (c)). The Boolean union of the Minkowski sum shapes for all the polygons becomes the Minkowski sum shape of the target object as shown in Figure 6(d).

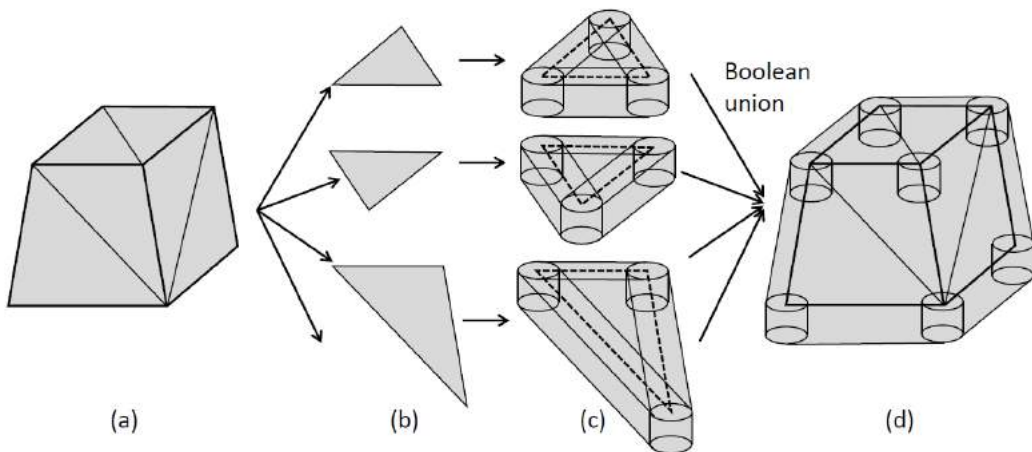


Figure 6: Computation process of Minkowski sum shape of a polyhedral object and an inverted flat-end cutter.

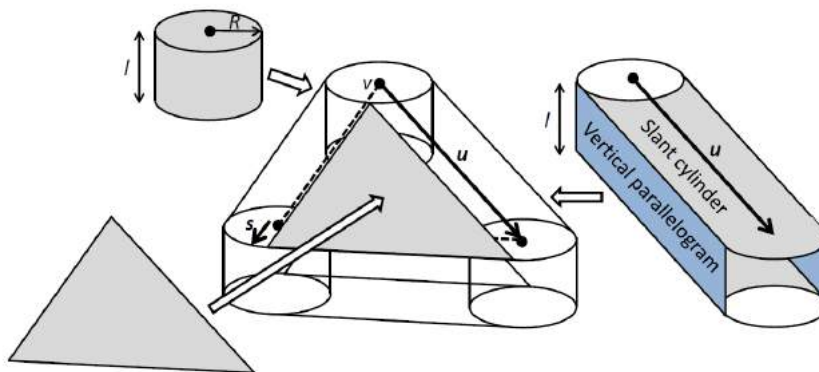


Figure 7: Surface elements for organizing the Minkowski sum shape of a triangle and an inverted flat-end cutter of radius R and length l .

The Minkowski sum of a single triangle and the inverted flat-end cutter is a composition of the following three types of surface elements (see Figure 7).

- For each vertex v of the triangle, the inverted flat-end cutter is placed such that the reference point of the cutter and v are coincident.
- For each edge e , ruled surfaces generated by sweeping the inverted cutter along e are placed. Let us consider a vector \mathbf{u} from one end point of e to the other end point. Using the vector \mathbf{u} , the surface of the inverted flat-end cutter can be classified to a region with a normal vector facing \mathbf{u} and the other region with a normal vector perpendicular to \mathbf{u} or facing the direction opposite to \mathbf{u} . We can obtain the ruled surfaces by linearly sweeping the border curves between these two regions along e . They are a slant cylinder connecting two end points of e , another slant cylinder locating l lower than the first slant cylinder, and two vertical parallelograms connecting two slant cylinders.
- Two triangles that are obtained by shifting the original triangle f by a vector \mathbf{s} and its opposite vector $-\mathbf{s}$, where \mathbf{s} is given by projecting the normal vector \mathbf{n} of the triangle to the horizontal plane, and then adjusting its length such that it is equal to R . The latter triangle is further moved in the negative z-axis direction by l .

For each x-axis-aligned line or y-axis-aligned line of the square mesh in a slicing plane, the intersection points between the line and all the surface elements mentioned above are calculated. Since the Minkowski sum of a triangle and an inverted flat-end cutter has a convex shape, the points at both ends of the point sequence on the line correspond to the boundary of the overlapping range between the Minkowski sum shape and the line. This range is replaced to a dixel on the line. This operation is iterated for all the grid lines in the slicing plane, and a two-directional dixel model representing a cross-sectional figure of the Minkowski sum shape for a single triangle is obtained. The Boolean union of the cross-sectional figures for all the triangles of the mold CAD model corresponds to the cross-sectional figure of the Minkowski sum shape of the entire mold model and the inverted flat-end cutter. This computation is iterated for all the slicing planes. As a result, the Minkowski sum shape for the mold model is obtained as a stack of two-directional dixel models.

3.3 Parallel Minkowski Sum Computation Using GPU

In the computation process mentioned above, a number of intersection point calculations and Boolean union computations of dexels on the same line are executed. These computations on a line are independent of those on other lines. Thus, the intersection point calculation and the dixel-wise Boolean union computation can be performed in a parallel manner. To implement a parallel computation software, the compute unified device architecture (CUDA) is used [4]. Current GPUs are designed to have thousands of streaming processors (SP) on a single chip. Using CUDA, programmers can utilize a GPU as a general-purpose parallel processor in which each SP executes a unit computation (or thread).

As a preparation, the surface polygons of the input model are classified into small groups according to their proximity. Let us consider n polygons forming the model surface. An AABB (axis-aligned bounding box) [14] that tightly confines the polygons is defined by measuring the coordinate ranges of the polygons in the x-, y-, and z-directions. A root AABB is defined such that it holds all the polygons of the model. The polygons in the AABB are sorted in the x-, y-, or z-directions and subdivided into two groups with $n/2$ polygons according to the sorting result. For each polygon group, a smaller AABB is formed and registered as a descendant of the original AABB. The process of defining descendant AABBs is iterated until the number of polygons in a group becomes less than or equal to a predetermined number n_{max} , and a binary AABB tree is obtained. n_{max} is set as 32 in our current implementation.

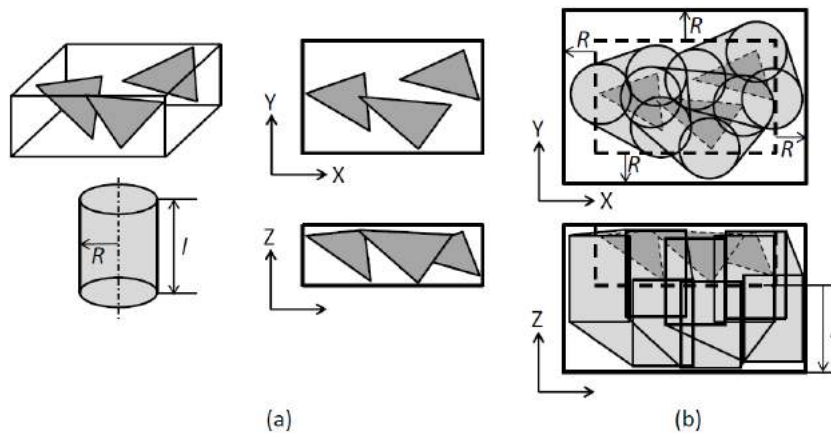


Figure 8: Expansion of an AAB for holding Minkowski sum shape of polygons within.

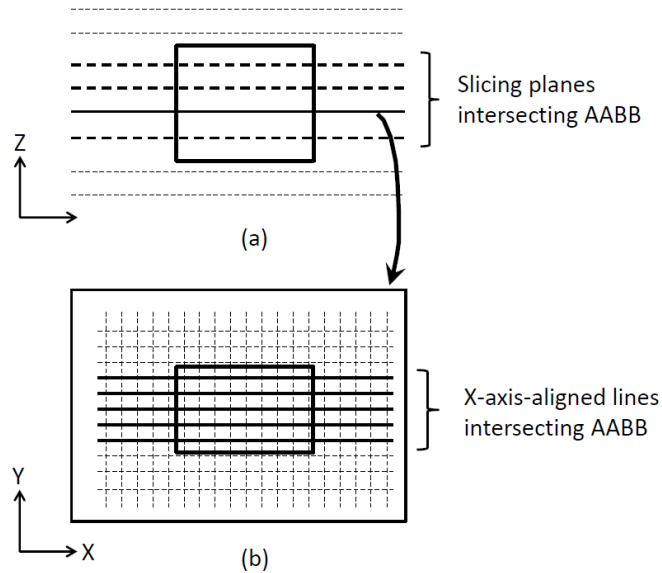


Figure 9: Slicing planes intersecting an expanded AAB (a) and x-axis-aligned lines on a slicing plane intersecting the AAB (b).

Each leaf AAB of the AAB tree is the basic processing unit of our parallel Minkowski sum computation. Let us consider the Minkowski sum computation for all polygons in a single leaf AAB with an inverted flat-end cutter of radius R and length l (Figure 8(a)). To hold the “expanded” polygons after the Minkowski sum computation, the AAB is horizontally expanded by R . The box is further stretched in the $-z$ direction by l . These operations are achieved by shifting four vertical rectangles of the AAB in the outward directions by R and then shifting the bottom side rectangle by l in the downward direction as shown in Figure 8(b).

The horizontal slicing planes intersecting the expanded AAB are checked. In Figure 9(a), four slicing planes are detected as the planes intersecting the box. For each slicing plane intersecting the box, the x-axis-aligned lines of the square mesh in the plane intersecting the expanded AAB are selected (see Figure 9(b)). For each selected line, a single CUDA thread is invoked for computing the overlapping ranges between the line and the Minkowski sum shapes of the triangles

in the box. In this computation, the algorithm explained in section 3.2 is used. After computing the overlapping ranges in the line, the ranges corresponding to their Boolean union are computed by the same thread. The dixel information in the same line is updated based on the computation result. After the processing for the x-axis-aligned lines, the same computation is executed for the y-axis-aligned lines of the square mesh intersecting the expanded AABB.

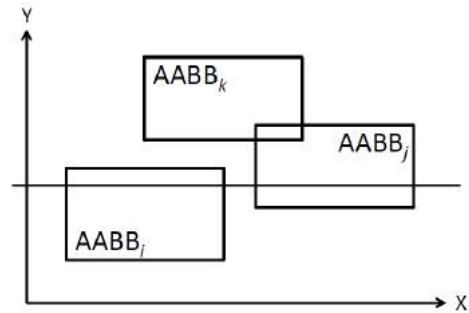


Figure 10: Selection of $AABB_i$ and $AABB_k$ for avoiding conflicts in the parallel dixel updating operation.

To further utilize the parallel processing capability of GPUs, the Minkowski sum computation for a single leaf AABB is extended to parallel computations with multiple AABBs. In this case, we need a mechanism to prevent the occurrence of conflicts in the dixel-wise Boolean operation. Let us consider the Minkowski sum computations for polygons in $AABB_i$ and $AABB_j$ being simultaneously invoked. If the expanded boxes of $AABB_i$ and $AABB_j$ have intersections in their projections to the yz-plane as shown in Figure 10, a CUDA thread for $AABB_i$ and another thread for $AABB_j$ may simultaneously try to update the dexels on the same x-axis-aligned line (line l in Figure 10). To prevent the occurrence of such conflicts, multiple AABBs must be selected such that the projections of their expanded shapes do not intersect. For example, $AABB_i$ and $AABB_k$ are selected for the box arrangement shown in Figure 10. We used the conflict resolution algorithm originally developed for our offset computation algorithm. The readers may refer to our previous paper [8] for details.

3.4 Minkowski Sum Computation for Ball-End Cutter

The Ball-end cutter shape can be considered as a composition of a vertical cylinder and a sphere as shown in Figure 2. The Minkowski sum computation of a polyhedral CAD model and an inverted ball-end cutter of radius r can thus be realized by a Boolean union computation of the following two 3D objects:

- A Minkowski sum shape of the CAD model and an inverted flat-end cutter of radius $R (= r)$,
- Another Minkowski sum shape of the same model and a sphere of radius r .

Minkowski sum shape of a triangle and a sphere is equivalent to a composition of three spheres, three cylinders, and a thick plate (slab) placed on the triangle as follows:

- Three spheres of radius r placed on three vertices of the triangle.
- Three open cylinders of radius r placed along each edge e of the triangle, with the center axis of the cylinders coinciding with the edges.
- A slab with the area of the triangle and thickness $2r$ placed on the triangle with the center plane of the slab coinciding with the triangle.

The two-directional dixel model of the Minkowski sum shape of the mold CAD model and the sphere can be obtained using a method that is similar to the Minkowski sum computation for the

inverted flat-end cutter. The result shape is then combined to the Minkowski sum shape for a flat-end cutter of radius R . The Minkowski sum computation for a radius-end cutter is discussed later.

4 STEP 2: CONTOUR-TYPE PATH COMPUTATION

4.1 Path Tracing Procedure

The contour-type cutter path is obtained by tracing the boundary of the cross-sectional figure in the two-directional dixel representation. Let us consider the grid lines defining the x-axis-aligned dexels and y-axis-aligned dexels in a slicing plane. These lines organize a regular square mesh in the plane. The two-directional dixel model is mapped on the square mesh. Each grid point of the mesh is then marked IN or OUT according to the following rule:

- If a grid point is within the range of an x-axis-aligned dixel or y-axis-aligned dixel, then the grid point is marked IN because it is located within the cross-sectional figure.
- Otherwise, it is located outside of the figure and is marked OUT.

After marking all the grid points, each square cell in the mesh is classified to four types according to the pattern of marks at its four corner points (Figure 11(a)). Directed short segments are inserted in the cell such that they connect two dixel points in the same cell (short red arrows in the figure). The segments are traced and the boundary curve around the cross-sectional figure is obtained as a contour-type path (Figure 11(b)).

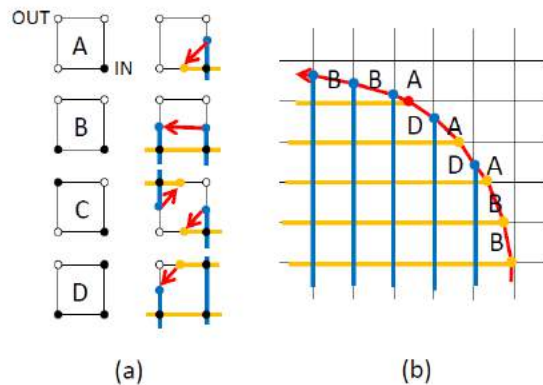


Figure 11: Boundary curve tracing method.

In our boundary curve tracing method, two points in the same cell are connected by a single segment; therefore, the sharp corners in the boundary curve cannot be reproduced appropriately. To improve the reproducibility of the sharp corners, we modify the segment insertion method in the cell. Before connecting two points \mathbf{p}_0 and \mathbf{p}_1 in a cell, the tangent vectors \mathbf{t}_0 and \mathbf{t}_1 at the points are checked. The segment insertion method is changed according to the angle θ organized by \mathbf{t}_0 and \mathbf{t}_1 as follows:

- If θ is larger than θ_0 and smaller than θ_1 , consider a tangent line l_0 passing \mathbf{p}_0 and another tangent line l_1 passing \mathbf{p}_1 , and compute their intersection point \mathbf{p} . Two segments $\mathbf{p}_0\mathbf{p}$ and $\mathbf{p}\mathbf{p}_1$ are inserted in the cell. \mathbf{p} corresponds to a sharp corner.
- Otherwise, \mathbf{p}_0 and \mathbf{p}_1 are connected by a single segment.

In our current implementation, we assign 5 degrees to θ_0 and 170 degrees to θ_1 . To properly compute the tangent vector at the point in the cell, we modify the Minkowski sum computation software such that the normal vector at the dixel's end point is computed and recorded in the two-directional dixel model construction process. Because our model uses an ultra-high-resolution grid in the computation, the cell size is sufficiently small. This simple segment insertion method can thus be used to reproduce the boundary curve accurately.

4.2 Resolution of Contradictory Dixel Arrangements

In the two-directional dixel model, the x-axis-aligned dexels and y-axis-aligned dexels are independently defined, and therefore, they may comprise a contradictory arrangement. Figure 12 illustrates a case. Two dexels near a grid point g of a cell are in a contradictory arrangement in this figure. Point g is contained within an x-axis-aligned dixel; however, it is located outside a y-axis-aligned dixel (see Figure 12(a)). The numerical errors at the intersection point computation can cause such contradictory dixel arrangement. In our grid point classification rule, g is judged as an internal point because the x-axis-aligned dixel contains this point within. Let us consider that all the other grid points of the cell are recognized as OUT. In this IN-OUT arrangement, our algorithm inserts a directed path segment, as shown in Figure 12(b). Since no dixel point is generated on the y-axis-aligned edge in the cell adjacent to g , one coordinate of the segment cannot be determined as shown in the figure. To solve this problem, our algorithm inserts a virtual point in the cell as an end point of the segment (see Figure 12(c)).

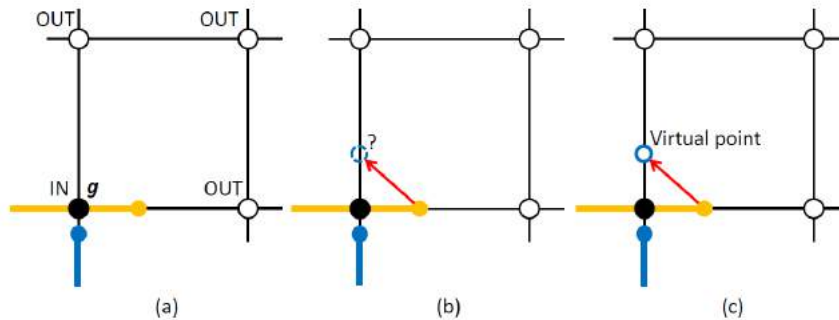


Figure 12: Contradictory arrangement of two dexels (a) and generation of a cutter path segment by inserting a virtual point (b, c).

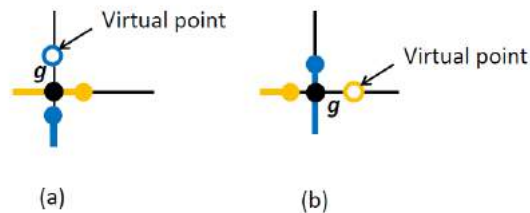


Figure 13: Virtual point insertion procedure.

Figure 13 illustrates two cases wherein the virtual point is necessary. In Figure 13(a), a grid point g has a contradictory arrangement of the dexels because it is contained within an x-axis-aligned dixel but it not within a y-axis-aligned dixel. Such a contradiction occurs when the surface of the Minkowski sum shape passes very close to a grid point, and therefore, the virtual point should be generated somewhere on the y-axis-aligned cell edge and sufficiently close to the grid point g . Figure 13(b) shows another contradictory arrangement wherein a grid point g is not within the x-axis-aligned dixel. In this case, the virtual point should be inserted on the x-axis-aligned cell edge as shown in the figure.

4.3 Cutter Path Generation for Radius-End Cutter

The shape of a radius-end cutter of radius R and corner radius r is equivalent to a Minkowski sum shape of a cylinder of radius $R-r$ and a small sphere of radius r . The Minkowski sum shape of a

polyhedral mold model and an inverted radius-end cutter can thus be obtained in the following two step process.

Step 1.1: The Minkowski sum shape of the mold model and a sphere of radius r are computed. The result shape is named M .

Step 1.2: The Minkowski sum shape of M and an inverted cylinder of radius $R-r$ are computed. Since the model M obtained in the first step is already in the two-directional dexel representation, we need a Minkowski sum computation algorithm for a model in that representation.

Instead of developing a new algorithm, we computed the contour-type path using the model M just after step 1.1, and then computed the final cutter path for the radius-end cutter by horizontally expanding the path by $R-r$. To realize the expansion, we take into consideration a vertical cylinder of radius $R-r$ moving along the cutter path. As the cutter path is a series of horizontal line segments, the swept volume of the moving cylinder is a Boolean union shape of vertical cylinders of radius $R-r$ placed at the end points of the segments and rectangular boxes of width $2R-2r$ placed along the segments as shown in Figure 14.

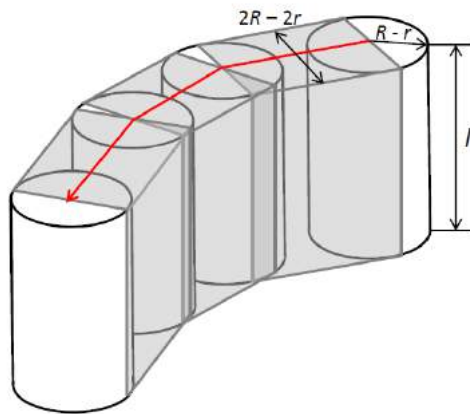


Figure 14: Placement of vertical cylinders of radius $R-r$ and rectangular boxes of width $2R-2r$ along the cutter path segments.

After the placements of the cylinders and rectangular boxes, their union shape is obtained in the two-directional dexel representation using a similar method for computing the Minkowski sum shape as explained in section 3. The final contour-type cutter path for milling the object with a radius-end cutter is obtained by tracing the boundary of the cross-sectional figure in the two-directional dexel representation.

5 COMPUTATIONAL EXPERIMENTS

We implemented a contour-type cutter path computation software using VisualStudio 2010 and CUDA 7.5, and we conducted several computational experiments with the software. A 64-bit PC with an Intel Core i7 Processor (2.8 GHz), 32-GB memory is used in the experiments. This PC is connected to an external GPU box with an nVIDIA GeForce RTX-2060 graphics card (6-GB graphics memory) installed.

Figure 15 shows the cutter path computation results for a simple polyhedral model with 1,618 polygons. (a) is a cutter path for a ball-end cutter of 4 mm radius ($R = r = 4$ mm). (b) and (c) are cutter paths for a flat-end cutter of 4 mm radius ($R = 4$ mm, $r = 0$ mm), and for a radius-end cutter of 4 mm radius and 1 mm corner radius ($R = 4$ mm, $r = 1$ mm), respectively. This model has a size of 60 x 54 x 43 mm. The contour-type path is computed for 250 slicing planes. In the computation, a two-directional dexel model of [25,000, 22,500] resolution is used.

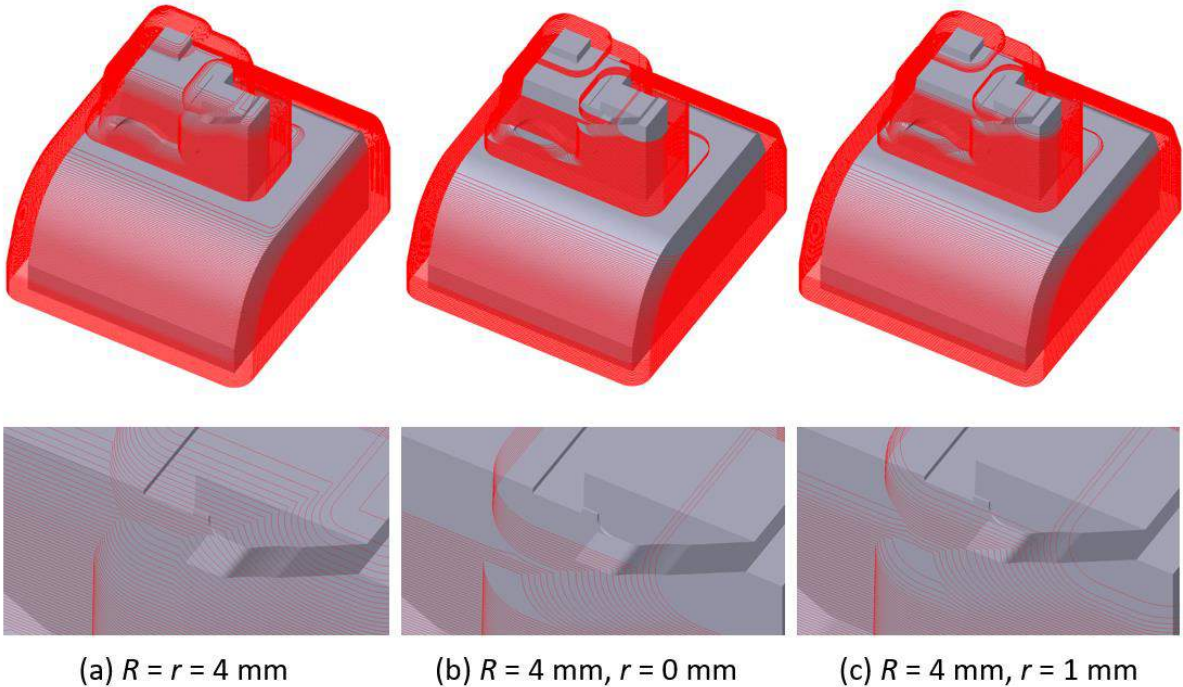


Figure 15: Contour-type cutter path computation results for a polyhedral solid model. (a) Path for a ball-end cutter of 4 mm radius, (b) path for a flat-end cutter of 4 mm radius, and (c) path for a radius-end cutter of 4 mm radius and 1 mm corner radius.

Table 1 shows the necessary computation time for obtaining the path. As shown in the table, the Minkowski sum computation (Step 1) for a ball-end cutter and flat-end cutter was achieved in a few seconds owing to the parallel processing capability of the GPU. A time that was 250 times longer was necessary for the tracing operation of the cutter path in the sling planes (Step 2). As the cutter path tracing operation is executed in our Minkowski sum computation process for the radius-end cutter, the necessary computation time for Step 1 for the radius-end cutter is almost the same as the total time of Steps 1 and 2 for the ball-end or flat-end cutter, as shown in the table.

	$R = r = 4 \text{ mm}$	$R = 4 \text{ mm}, r = 0 \text{ mm}$	$R = 4 \text{ mm}, r = 1 \text{ mm}$
Step 1	3.71 s	4.18 s	1123.50 s
Step 2	962.00 s	991.26 s	913.16 s
Total	965.71 s	995.44 s	2036.66 s

Table 1: Necessary computation time for obtaining the cutter paths given in Figure 15.

We applied the system to three polyhedral models of cavity and core of molds for injection molding as shown in Figure 16. We used a ball-end cutter of radius 10 mm ($R = r = 10 \text{ mm}$), a flat-end cutter of radius 10 mm ($R = 10 \text{ mm}$, and $r = 0 \text{ mm}$), and a radius-end cutter of radius 10 mm and corner radius 2 mm ($R = 10 \text{ mm}$ and $r = 2 \text{ mm}$) in the cutter path computation. Figure 16 illustrates the obtained paths for the ball-end cutter. The Cavity0 model (Figure 16(a)) has a size of 2529 x 1154 x 893 mm with 844,180 polygons. The Cavity1 model (Figure 16(b)) has a size of 2060 x 960 x 500 mm with 361,642 polygons, and the Core model (Figure 16(c)) has a size of 1550 x 950 x 350 mm with 250,494 polygons. In the contour-type path computation for Cavity0,

the [25,000, 11,409] resolution two-directional dixel models are generated in the slicing planes. The [25,000, 11,650] resolution two-directional dixel model and [25,000, 15,322] resolution two-directional dixel model are used for Cavity1 and Core model, respectively.

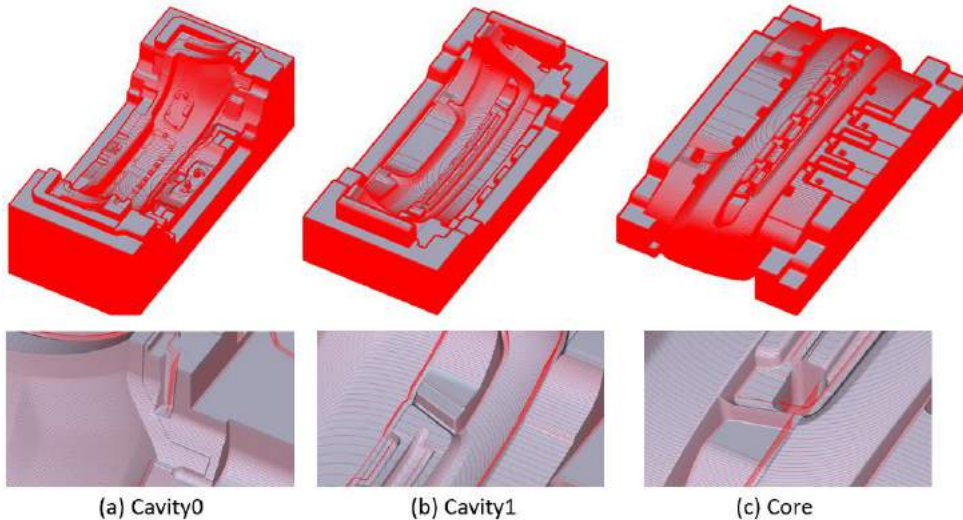


Figure 16: Contour-type cutter path computation results for cavity and core models.

Tables 2, 3, and 4 show the necessary computation time for obtaining the cutter path for these three models. The computation time in the tables shows a similar tendency to the data shown in Table 1. In the cutter path computation for the ball-end cutter or flat-end cutter, the Minkowski sum computation (Step 1) is achieved in a short time (less than a minute) for any model. In contrast, a much longer computation time (52 times more time for Cavity1) is necessary in the cutter path tracing operation (Step 2).

Cavity0	$R = r = 10 \text{ mm}$	$R = 10 \text{ mm}, r = 0 \text{ mm}$	$R = 10 \text{ mm}, r = 2 \text{ mm}$
Step 1	43.87 s	43.88 s	823.29 s
Step 2	581.89 s	585.68 s	554.59 s
Total	625.76 s	629.56 s	1,377.88 s

Table 2: Necessary computation time for obtaining the cutter path for the Cavity0 model.

Cavity1	$R = r = 10 \text{ mm}$	$R = 10 \text{ mm}, r = 0 \text{ mm}$	$R = 10 \text{ mm}, r = 2 \text{ mm}$
Step 1	14.05 s	15.22 s	834.62 s
Step 2	776.30 s	791.32 s	677.60 s
Total	790.35 s	806.54 s	1,512.22 s

Table 3: Necessary computation time for obtaining the cutter path for Cavity1 model.

We investigated the relationship between the number of slicing planes and the computation time. The experiments were performed using the Cavity0 model. We applied the software for computing the cutter path for a ball-end cutter of 10 mm radius. In the computation, a [25,000, 11,409] resolution two-directional dixel model is used. The number of slicing planes was set as 250, 500, and 750 in the experiments. The necessary computation time is listed in Table 5. The obtained cutter paths for 500 slicing planes and for 750 slicing planes are illustrated in Figure 17. The

necessary computation time for Steps 1 and 2 increases in proportion to the number of slicing planes, as can be observed from the values shown in Table 5 and their graphs (Figure 18). The graph in Figure 18 also shows that the cutter path tracing process (Step 2) dominates the majority of the processing time in our cutter path computation software.

Core	$R = r = 10 \text{ mm}$	$R = 10 \text{ mm}, r = 0 \text{ mm}$	$R = 10 \text{ mm}, r = 2 \text{ mm}$
Step 1	23.38 s	32.26 s	847.72 s
Step 2	755.62 s	767.24 s	767.60 s
Total	779.00 s	799.50 s	1,615.32 s

Table 4: Necessary computation time for obtaining the cutter path for Core model.

In our computation method, the resolution $[m, n]$ of the two-directional dixel model in the slicing plane is a critical factor for the performance of the software. Using a model with a higher resolution, a more accurate path computation becomes possible; however, the time and cost for the computation also increase. We executed experiments to investigate the relationship between the model resolution and necessary computation time. Table 6 shows the necessary computation time when the resolution $[m, n]$ of the two-directional dixel model varies. In the experiments, the Cavity0 model was used. The computation time was measured in the contour-type path computation for a ball-end cutter of 10 mm radius. The number of slicing planes was set as 250.

Figure 19 shows a graph representing the relationship between the model resolution m and the necessary computation time for Steps 1 and 2. The computation time for Step 1 is proportional to the model resolution. Our software can compute the Minkowski sum shape for a two-directional dixel model of $[45,000, 20,536]$ resolution in 250 slicing planes in 70 s. In contrast, the time for the path tracing operation in Step 2 increases rapidly as m increases, and it becomes 1,800 s for the model at the same resolution. The two-directional dixel model in a slicing plane is equivalent to a square cell structure in the plane. The computation time for Step 2 is analyzed, and it is found to be proportional to the number of cells ($= m \times n$) in the plane. As can be observed from this computation result, an improvement in the processing algorithm of Step 2 is critical for the practical application of this technology.

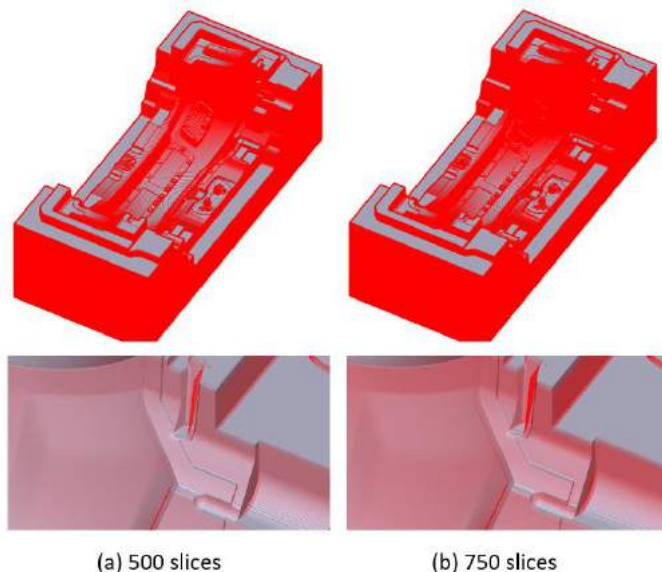


Figure 17: Contour-type cutter path for 500 slicing planes (a) and 750 slicing planes (b).

Cavity0	250 slices	500 slices	750 slices
Step 1	43.87 s	85.33 s	123.26 s
Step 2	581.89 s	1,163.23 s	1,729.59 s
Total	625.76 s	1,248.56 s	1,852.85 s

Table 5: Necessary computation time for obtaining the cutter path for the Cavity0 model for different numbers of slicing planes. Ball-end cutter of 10 mm radius and two-directional dixel model with a [25,000, 11,409] resolution are used in the computation.

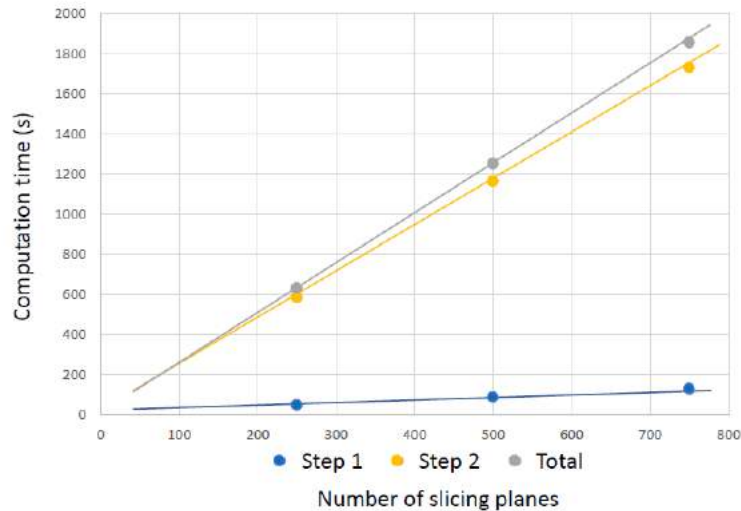


Figure 18: A graph of the necessary computation time for obtaining the cutter path for the Cavity0 model for different numbers of slicing planes.

Cavity0	[20,000, 9,172]	[25,000, 11,409]	[30,000, 13,691]	[35,000, 15,972]	[40,000, 18,254]	[45,000, 20,536]
Step 1	37.24 s	43.87 s	49.75 s	56.53 s	62.50 s	70.74 s
Step 2	374.23 s	581.89 s	821.81 s	1,136.44 s	1,455.59 s	1,836.59 s
Total	411.47 s	625.76 s	871.56 s	1,192.97 s	1,518.09 s	1,907.33 s

Table 6: Necessary computation time for obtaining the cutter path for the Cavity0 model when the resolution of the two-directional dixel models varies from [20,000, 9,127] to [45,000, 20,536].

6 CONCLUSIONS

In this paper, we propose a novel algorithm for computing a contour-type cutter path for a mold CAD model in the polyhedral representation. Our algorithm uses a Minkowski sum shape of the mold model and a cutter model in the inverted orientation. To realize the robust Minkowski sum computation, a grid-based shape representation called a two-directional dixel model is used. A two-directional dixel model is geometrically equivalent to a 2D shape representation method using a regular square mesh. It can represent the 2D object with much a smaller amount of memory than a regular square-mesh-based method. Using this advantage, we realized two-directional dixel models based on an ultra-high-resolution (resolution higher than 20,000) grid. A contour-type cutter path computation software was implemented to demonstrate the performance of the algorithm. In our current implementation, the Minkowski sum computation can be completed in a

very short time even for an ultra-high-resolution dixel model. The contour-type cutter path is obtained by tracing the boundary of the two-directional dixel model. The necessary time for the path tracing operation is proportional to the square of the resolution of the dixel model. An improvement in the path tracing algorithm is critical for the practical application of this technology. The use of the parallel processing capability of a GPU and a CPU with multiple cores will be examined in future studies for reducing the processing time in the path tracing.

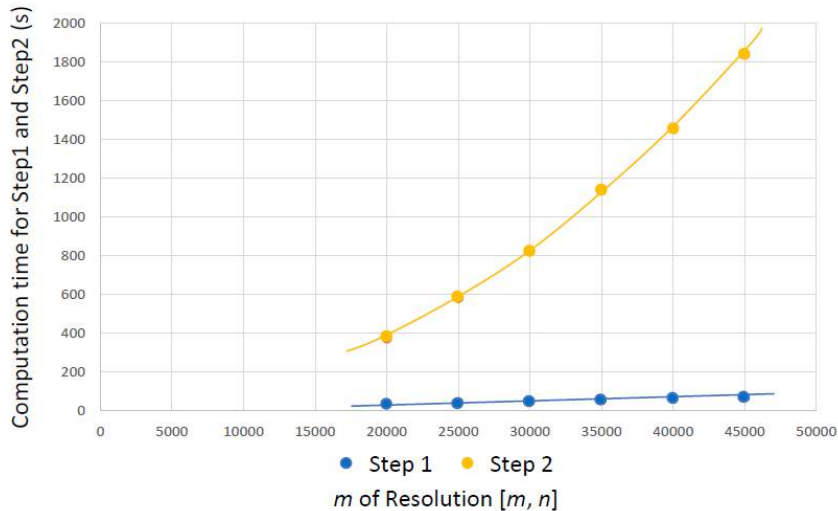


Figure 19: Necessary computation time for obtaining the cutter path for the Cavity0 model when the resolution of the two-directional dixel models varies.

ACKNOWLEDGEMENTS

This research work was supported by the JSPS KAKENHI Grant Number 17K06075.

Masatomo Inui, <https://orcid.org/0000-0002-1496-7680>

Nobuyuki Umezu, <https://orcid.org/0000-0002-7873-7833>

REFERENCES

- [1] Benouamer, M.O.; Michelucci, D.: Bridging the gap between CSG and Brep via a triple ray representation, Proceedings of ACM Symposium on Solid Modeling and Applications, 1997, 68–79. <https://doi.org/10.1145/267734.267755>
- [2] Breen, D.E.; Mauch, S.: Generating shaded offset surfaces with distance, closest point and color volumes, Proceedings of International Workshop on Volume Graphics, 1999, 307–320.
- [3] Breen, D.E.; Mauch, S.; Whitaker, R.T.: 3D scan conversion of CSG models into distance volumes, Proceedings of IEEE Symposium on Volume Visualization, 1998, 7–14. <https://doi.org/10.1109/SVV.1998.729579>
- [4] CUDA compute unified device architecture programming guide, http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, NVIDIA.
- [5] Dragomatz, D.; Mann, S.: A classified bibliography of literature on NC milling path generation, Computer-Aided Design, 29(3), 1997, 239–247. [https://doi.org/10.1016/S0010-4485\(96\)00060-7](https://doi.org/10.1016/S0010-4485(96)00060-7)

- [6] Forsyth, M.: Shelling and offsetting bodies, Proceedings of Third Symposium on Solid Modeling and Applications, 1995, 373-381. <https://doi.org/10.1145/218013.218088>
- [7] Huang, J.; Li, Y.; Crawfis, R.; Lu, S.C.; Liou, S.Y.: A complete distance field representation, Proceedings of the Conference on Visualization, 2001, 247-254. <https://doi.org/10.1109/VISUAL.2001.964518>
- [8] Inui, M.; Umezū, N.; Kitamura, Y.: Visualizing sphere-contacting areas on automobile parts for ECE inspection, Journal of Computational Design and Engineering, 2(1), 2015, 55-66. <https://doi.org/10.1016/j.jcde.2014.11.006>
- [9] Li, W.; McMains, S.: A GPU-based voxelization approach to 3D Minkowski sum computation, Proceedings of the 14th ACM Symposium on Solid and Physical Modeling, 2010, 31-40. <https://doi.org/10.1145/1839778.1839783>
- [10] Li, W.; McMains, S.: Voxelized Minkowski sum computation on the GPU with robust culling, Computer-Aided Design, 43(10), 2011, 1270-1283. <https://doi.org/10.1016/j.cad.2011.06.022>
- [11] Liu, S.; Wang, C.C.L.: Fast intersection-free offset surface generation from freeform models with triangular meshes, IEEE Transaction on Automation Science and Engineering, 8(2), 2011, 347-360. <https://doi.org/10.1109/TASE.2010.2066563>
- [12] Lorensen, W.E.; Cline, H.E.: Marching cubes: A high resolution 3D surface construction algorithm, Computer Graphics (Proceedings of ACM SIGGRAPH), 21(4), 1987, 163-169. <https://doi.org/10.1145/37401.37422>
- [13] Menon, J.P.; Marisa, R.J.; Zagajac, J.: More powerful solid modeling through ray representations, IEEE Computer Graphics and Applications, 14(3), 1994, 22-35. <https://doi.org/10.1109/38.279039>
- [14] Moller, T; Haines E.: Real-time rendering, A K Peters, 1999.
- [15] Rossignac, J.R.; Requicha, A.A.G.: Offsetting operations in solid modelling. Computer Aided Geometric Design, 3(2), 1986, 129-148. [https://doi.org/10.1016/0167-8396\(86\)90017-8](https://doi.org/10.1016/0167-8396(86)90017-8)
- [16] Samet, H.: The Design and Analysis of Spatial Data Structures, Addison-Wesley, MA, 1989.
- [17] Satoh, T.; Chiyokura, H.: Boolean operations on sets using surface data, Proceedings of ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, 1991, 119-126. <https://doi.org/10.1145/112515.112536>
- [18] Shade, J.; Gortler, S.; He, L.W.; Szeliski, R.: Layered depth image, Computer Graphics (Proceedings of ACM SIGGRAPH), 1998, 231-242. <https://doi.org/10.1145/280814.280882>
- [19] Tao, J.; Losasso, F.; Schaefer, S.; Warren, J.: Dual contouring of hermite data, Computer Graphics (Proceedings of ACM SIGGRAPH), 2002, 339-346. <https://doi.org/10.1145/566570.566586>
- [20] VanHook, T.: Real-time shaded milling display, Computer Graphics (Proceedings of ACM SIGGRAPH), 20(4), 1986, 15-20. <https://doi.org/10.1145/15886.15887>
- [21] Wang, C.C.L.: Computing on rays: A parallel approach for surface mesh modeling from multi-material volumetric data, Computers in Industry, 62(7), 2011, 660-671. <https://doi.org/10.1016/j.compind.2011.02.004>
- [22] Wang, C.C.L.; Chen, Y.: Thickening freeform surfaces for solid fabrication, Rapid Prototyping Journal, 19(6), 2013, 395-406. <https://doi.org/10.1108/RPJ-02-2012-0013>
- [23] Wang, C.C.L.; Manocha, D.: GPU-based offset surface computation using point samples, Computer-Aided Design, 45(2), 2013, 321-330. <https://doi.org/10.1016/j.cad.2012.10.015>
- [24] Zhao, H.; Wang, C.C.L.: Parallel and efficient Boolean on polygonal solids, The Visual Computer, 27(6-8), 2011, 507-517. <https://doi.org/10.1007/s00371-011-0571-1>