Int J Adv Manuf Technol (2010) 50:1101–1111
DOI 10.1007/s00170-010-2580-z

ORIGINAL ARTICLE

Contour parallel milling tool path generation for arbitrary pocket shape using a fast marching method

Sandeep Dhanik · Paul Xirouchakis

Received: 28 September 2009 / Accepted: 11 February 2010 / Published online: 11 March 2010

Abstract Contour parallel tool paths are among the most widely used tool paths for planer milling operations. A number of exact as well as approximate methods are available for offsetting a closed boundary in order to generate a contour parallel tool path; however, the applicability of various offsetting methods is restricted because of limitations in dealing with pocket geometry with and without islands, the high computational costs, and numerical errors. Generation of cusps, segmentation of rarefied corners, and self-intersection during the offsetting operations and finding a unique offsetting solution for pocket with islands are among the associated problems in contour tool path generation. Most of methods are inherently incapable of dealing with such problems and use complex computational routines to identify and rectify these problems. Also, these rectifying techniques are heavily dependent on the type of geometry, and hence, the application of these techniques for arbitrary boundary conditions is limited and prone to errors. In this paper, a new mathematical method for generation of contour parallel tool paths is proposed which is inherently capable of dealing with the aforementioned problems. The method is based on a boundary value formulation of the offsetting problem and a fast marching method based solution for tool path generation. This method handles the topological changes during offsetting naturally and deals with the generation of discontinuities in the slopes by including an "entropy condition" in its numerical implementation. The appropriate modifications are carried out to achieve higher accuracy for milling operations. A number of examples are presented, and computational issues are discussed for tool path generation.

Keywords CAD/CAM · Contour parallel · 2.5D milling · Pocketing

S. Dhanik (✉) · P. Xirouchakis
Institute of Production and Robotics (STI-IGM-LICP),
Ecole Polytechnique Fédérale Lausanne,
ME A1 403, Station 9,
1015 Lausanne, Switzerland
e-mail: sandeep.dhanik@epfl.ch

P. Xirouchakis
e-mail: paul.xirouchakis@epfl.ch

1 Introduction

One of the most common operations in machining of metal parts is pocket milling. In pocket milling, all the material is removed inside a closed arbitrary boundary on a flat surface of a workpiece to a fixed depth [1]. Most of mechanical parts consist of faces parallel or normal to a single plane, and free-form objects require a 2.5D rough milling operation of the raw workpiece, making 2.5D pocketing one of the most important milling operations. Almost 80% of the milling operations to produce mechanical parts are produced by NC pocket milling [2]. There are two main tool path patterns commonly used in 2.5D end milling operations: direction parallel tool paths and contour parallel tool paths. The relative merits of direction parallel and contour tool paths for minimum machining time have long been studied, and according to a recent study [3], the best tool path depends upon the geometry of the part, the machining characteristics, and cutting conditions. The contour parallel tool paths are known to be coherent as the tool is always in contact with the material and thus reduces idle time spent in lifting, positioning, and plunging of the tool. Also, they maintain the consistent use of either up-cut or down-cut milling strategy. Contour parallel tool paths are, therefore, widely used as cutting tool paths

especially for the large-scale material removal in 2.5D end milling.

The generation of contour parallel tool path in a pocketing operation requires an offsetting operation of the inner boundary of the pocket and outer boundary of the island. Given a simple, closed planar curve $C_0(s) = [x(s), y(s)]^T$, where s is an arbitrary curve parameterization, the offset $C_d(s)$ with constant distance d is defined as:

$$C_d(s) = C_0(s) + N(s,0)d \tag{1}$$

Where, $N(s, 0)$ is the unit normal to the original curve at the parametric point s and is given by:

$$N(s,0) = \frac{1}{\left(x_s^2(s) + y_s^2(s)\right)^{1/2}}[-y_s(s), x_s(s)]^T \tag{2}$$

Where, $x_s(s)$ and $y_s(s)$ denote the partial derivatives with respect to arc length s.

This approach could be generalized for finding a whole class of offset curves. However, this mathematical formulation has some drawbacks. These drawbacks can be summarized as follows:

- If the original curve has corner points, where the derivative is not defined, the problem of determining the intersection of propagating lines arises (see Fig. 1a, the ambiguity at the singular point produces two normals). In case of expansion of the corner point, the insertion of an additional arc is required (see Fig. 1b, dotted line denotes the additional arc).
- The geometry of curve could give rise to self-intersection features in the offsetting curve (see Fig. 1c, the shrinking offset of the outer curve produces the self-intersecting feature).
- Cusps or points of local curvature discontinuity could form at the offset distance d if the local maximum curvature at the original curve is $1/k < d$ (see Fig. 1d).
- The topology of the closed region may change with the offsetting, i.e., a single bounded region could be divided into multiple bounded regions or vice-versa (see Fig. 1e, the shape is divided into two regions).
- If there are islands in the pocket, then a unique solution may not exist when the offsets of the pocket and island intersect each other (see Fig. 1f, for the same offset of pocket and island boundary the offset curve may intersect at more than one point).

2 Literature review

Offsetting is a general operation used in a variety of engineering applications such as computer-aided design (CAD), finite element modeling and analysis, robot path planning, tolerance analysis, mold design, computer graphics, and electrical circuit design. The literature over the offsetting methods could be broadly divided into two categories: exact methods and approximate methods. The exact methods are limited in terms of the geometry they could deal with and are used in some special cases [4–6]. The exact methods are not widely used, as the approximate

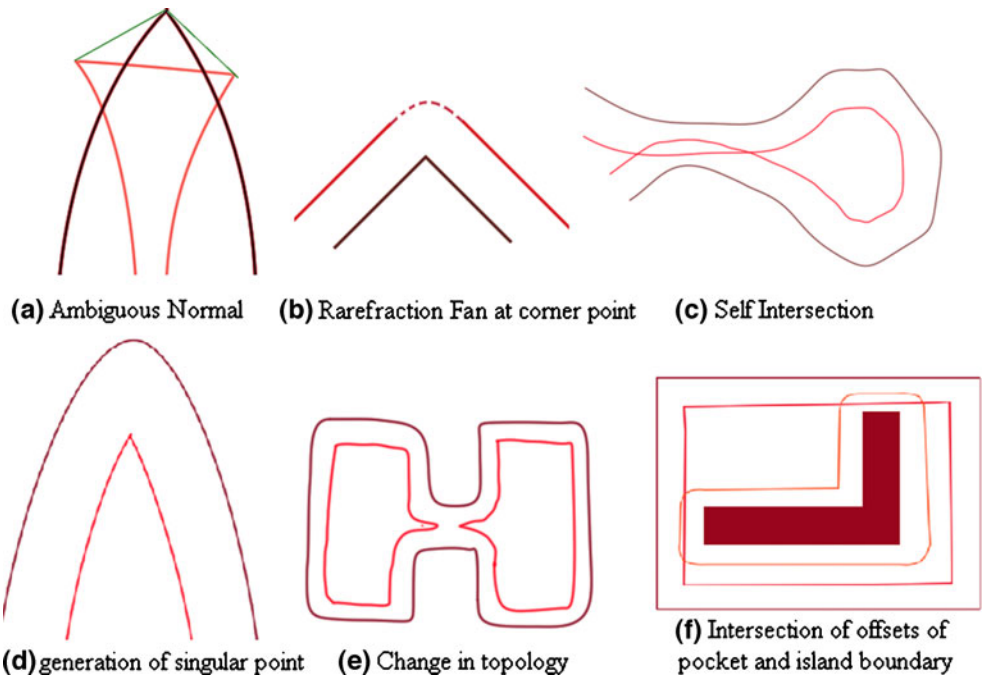

Fig. 1 Various problems in offsetting boundary closed curves

methods present sufficiently accurate solutions and are valid for more general cases. A number of approximate methods are now described as follows:

2.1 Iterative methods for spline offsetting

In CAD systems, in order to represent an arbitrary geometry, the standard parameterized curves such as line segments, circular arcs, or splines are widely used. However, even if the curve is regular or rational, the offsetting curve is not an exact polynomial or rational in general because of the denominator in Eq. 2. Thus, the offset curves of splines are not necessary splines. Thus, approximate methods are widely proposed to offset these general curves. Klass [7] offsets the B-spline segments in an iterative fashion calculating each time the tangent to the offset curve. The offset curve is checked for the error in the offset distance at various points, and if a large error is found, the curve is segmented and the process is repeated until the error is reduced to the required resolution. Tillar and Hanson [8] used a "rational B-spline representation" with control points located on the curve itself. The offset operation is then carried out by moving these control points to the offsetting distance. These methods require sophisticated procedures to deal with problems of loops, shocks, cusps, and self-intersection singularities. In this direction, Hoschek [9] applied nonlinear optimization techniques to find the offset approximation. He used an iterative geometric algorithm to find the self-intersecting portions and then eliminated the tails and loops that arose in the offset curves. Elber et al. [10] present a comparison of these contemporary methods and presented simulation results. Based on similar developments, Hansen and Arbab [11] made the pair-wise intersection scheme more efficient for pocket milling of circular and linear segmented boundaries. In summary, offsetting of a boundary in this manner is a two-stage process: (1) determining the offset of the spline in an iterative manner until the error in the offset is reduced below a user-defined limit and (2) detecting the singularities and redundant portions and correcting them (see Fig. 2a).

2.2 Voronoi diagram-based offsetting

The complex task of detection and correction methods has motivated alternate methods for bypassing the step two mentioned in the previous section. In this direction, Persson's approach [6], which is based on finding bisectors of lines and circular arcs using the Voronoi diagram, is one the earliest approaches to construct the contour parallel tool path. Held [2] also utilized the method for creating the tool path but restricted his approach for linear and circular segments because these representations can easily be fed to

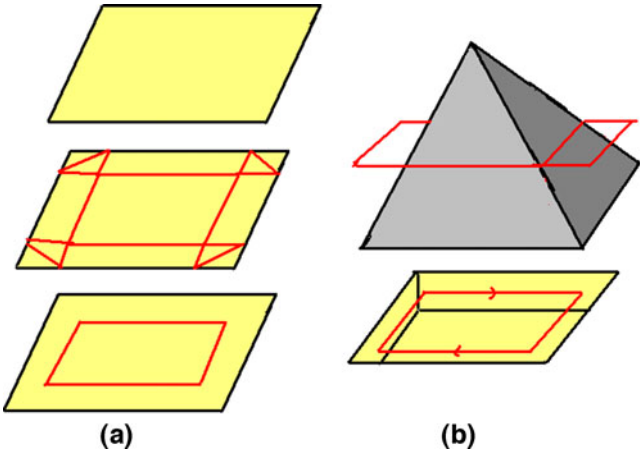

Fig. 2 **a** Pair-wise intersection and correction **b** Voronoi diagram

CNC machines. The polygon boundary of the pocket and its Voronoi diagram are shown in Fig. 2b; note that the equidistant offset can be achieved if the Voronoi mountain is sliced at a particular level. The Voronoi diagram not only helps in finding the contour offset but also the optimization of process in terms of multiple-tool selection and machining time minimization [12]. Seth and Stori [13] have extended this approach for the tool path generation for nonlinear segments also. However, the method based on Voronoi diagram employs costly two-dimensional (2D) Boolean set operations, relatively expensive distance calculations and an overhead of extraneous geometry [11].

2.3 Pixel simulation and image processing-based offsetting

Similar to bypassing the step to detect and remove the self-intersection features, a pixel simulation-based approach for the offsetting is developed by Choi and Kim [14], where the tool path is generated by successive sweeping of the tool. This method is based on the Z-map, and hence, high computational time and huge memory are required to achieve a desired level of precision due to its dependence on the resolution of the Z-map. An interesting alterative approach was discussed by Saeed et al. [15] which is based on mathematical morphology. The basic set operations, like erosion and dilation of mathematical morphology, are closely related to the offset surface generation and are shown to prepare the roughing milling paths. Recently, Molina-Carmona et al. [16] developed a mathematical morphology method for finding the offset of an arbitrary shaped tool. It should be noted that the related problems like skeleton finding, fat curves, and calculation of Euclidian distance maps for a plane curve in fields like computer graphics and image processing could make a significant contribution to the problem of the offsetting in milling. However, the adaptation of any method requires some pre-processing as well as post-processing of the

original method to make the generation of milling path efficient. For example, the Laplace-based parameterization and meshing of the domain to be machined produces smoother contour parallel tool paths [17, 18] than the conventional contour parallel tool paths, thus making this method suitable for the tool path generation for the roughing process; however, a feed rate optimization program is often required to compensate for the large variation in engagement conditions due to unequal spacing of the contours.

2.4 Laplace-based formulation and offsetting

The approach of Bieterman and Sandstrom [17] for a Laplace-based tool path generation method was motivated by the mathematical methods used in design and control of aerospace vehicles. They demonstrated their method for calculating offsets and then modifying the offsets for the spiral path generation. But, the capability of the method was demonstrated for convex pockets and pockets without islands. Chunag and Yang [18] use a similar approach, but demonstrated the use of Laplace-based formulation for the tool path generation for arbitrary geometry with a number of islands. The Laplace-based methods are suitable for the tool path generation as they make the contours smoother and continuous; however, the workpiece is sometimes over-machined depending upon the shape and the distance between the two neighboring contours which could not be controlled.

In this paper, the contour tool path generation based on a boundary value partial differential equation (PDE) formulation is presented. The formulation of the problem is similar to Laplace-based methods. For the solution of the equation, the modified fast marching method is utilized. It should also be noted that the initial value formulation of this problem for milling tool path generation is already done by Krimmer and Bruckstein [19]; however, the solution based on this formulation is relatively slow, and also, their method suffers from large error in the diagonal direction. Although the initial value formulation and solution using level set method [19] could be used for both expanding and contracting the moving fronts, in contour parallel machining, the main objective is to find the offset which is strictly contracting for the pocket. This condition of strictly contracting or expanding could be exploited by the boundary value formation and fast marching method, which reduces the cost of computation considerably. The fast marching method based on the unwinding nature of the solution is developed by Sethian [20]. Unlike a finite element-based solution for the Laplace equation, this method utilizes the finite difference scheme and an appropriate causality condition to guarantee the fast result. The proposed method tries to find out the minimum distance value from the initial pocket boundary at each grid point, and hence, the topological issues are handled naturally as each grid point could have only one distance value. For example, as the boundary of the pocket shrinks during offsetting or the boundary of the island expands during the same offsetting, the offsetting curves can break into many closed curves or can merge into one. Furthermore, the intrinsic geometrical properties of the offsetting curves can be determined easily in the formulations. The determination of such intrinsic properties could lead to the generation of smoother contour profiles such that the dynamics of machine tool could be respected. Also, the time complexity of the calculation is less as compared with the other discretized methods which is made possible by the use of a fast marching method and the appropriate sorting algorithm. The method is much faster and very straightforward in higher dimensions also. But, the method suffers from the inaccuracy in the diagonal direction in a rectangular grid. Thus, further modifications inspired by the work of Hassauna and Farag [21] are implemented in the original fast marching method proposed by Sethian [20] to prepare a system for contour parallel milling. In the following sections, the offsetting problem is formulated as an initial value PDE and for modeling the contour parallel tool path generation. The numerical method is demonstrated, and the system implementation is presented. Some general contour parallel tool paths for a general pocket or a pocket with an island are demonstrated.

3 Mathematical formulation

In many contour parallel tool path generation methods mentioned above, the offsetting of the pocket and the island are considered as different entities, which results in a problem when the inward offsets of the pocket and outward offset of the island meet. In the proposed formulation, both boundaries are treated as a single entity bounding a closed domain. If the closed domain can be identified, there is no need to track and correct the different offset of pocket boundary and the island boundary [refer case (*f*) of Fig. 1]. Thus, the method can consider *n*-number of islands in pocket machining. The closed domain interior of the pocket boundary and exterior to the islands boundary is the region where the milling should take place (see Fig. 3). The boundary of the closed domain at any time instant is represented by $C_t(s)$; it is understood that $t=0$ represents the original boundary and s is the curve parameter. Now, in order to offset the closed boundary, it is needed to travel perpendicular to the boundary curve. The position of the offsetting curve could be characterized by the arrival time $T(x, y)$. Then, ∇T will

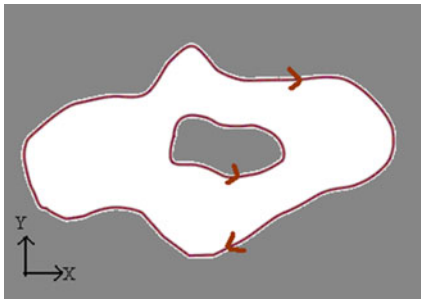

Fig. 3 An arbitrary pocket with islands

be perpendicular to the level sets of T, and its magnitude is inversely proportional to the speed. Hence,

$$|\nabla T|F = 1, \quad \text{where } T = 0 \text{ on the boundary } C_{(t=0)}(s) \tag{3}$$

Let scalar F be the speed of the inward normal direction, hence $C_t^{'}(s) \times n = F$ where n is the unit normal vector to the curve and $C_t^{'}(s)$ is the derivative of $C_t(s)$ with respect to time at time t. Using the Lagrangian formulation as the direct numerical approximation, the solution for a smooth curve beyond a certain time step could develop swallowtails as shown in Fig. 4a [20]. Thus, there is no guarantee of any order of smoothness if the interface moves with a constant speed F=1. Even when offsetting with C^{∞} continuity (such as for a sinusoid function), singular points could quickly develop (as shown in Fig. 4a). However, if a smoothing term ε is added to the speed function, such that $F = 1 - \varepsilon\kappa$, where κ is the curvature of the corresponding point on the curve, the offsetting curve will always be a smooth curve. But, even though the swallowtails are removed from the solution, the formulation does not allow the production of corner points and smoothens them out. The generation of singular points could simply be a property of an original boundary undergoing expansion. Thus, this form of formulation with a speed function

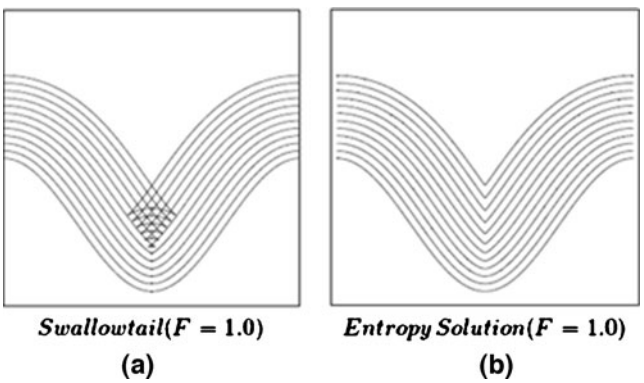

Fig. 4 The evolution of curve **a** Lagrangian approach **b** entropy-based construction [20]

$F = 1 - \varepsilon\kappa$, although removing the overlapping portions (i.e., swallowtails), does violate the condition that the solution must be derived from the original interface information only, i.e., it should generate the corner points if boundary is meant to produce them. This agrees with the traditional definition of entropy that concerns the amount of randomness to be added, i.e., the new information to be added during the evolution. In the case of offsetting, the entropy condition describes the constraint on the addition of randomness, i.e., the new information cannot be added during interface motion.

Thus, if a smoothing term is added, the solution violates the entropy condition. However, if $\varepsilon \rightarrow 0$, the solution can be regarded as the correct entropy satisfying solution. This limit is known as viscosity limit, and the solution which satisfies this limit is known as viscosity solution.

Eq. 3 is a form of the well-known Eikonal equation, and the notion of viscosity solutions is intimately connected to this equation, and according to Sethian [20], the use of monotone, consistent schemes will lead to schemes that select the correct viscous limit of the partial differential equation. In this approach, a rectangular grid is generated which contains the boundary to be offset. The objective is to find the distance value at each grid point of this rectangular domain. The numerical scheme for the appropriate solution will be described in the next section.

3.1 Initialization process

Based on the above mathematical formulation, the contour parallel tool path could be generated once the signed distance from the pocket boundary to the grid points of the discretized zone are known. To initiate the solution process, initialize the grid points neighboring the pocket boundary. In computer-aided design, a boundary is represented in parametric form, where the x and y values of the curve are represented by some function of the scalar parameter s. This curve can be represented in the form of a signed function $\phi(x(s), y(s)) = 0$, a point inside the curve will have ϕ<0 and outside ϕ>0. As the curve may not always pass through the grid points and may cross the grid lines instead, there is a need to initialize the neighboring grid location of the Cartesian grid with the appropriate value of the distance function. Thus, as shown in Fig. 5, the green points are the interior points to the curve, and the red points are the exterior to the curve. The error during the initialization depends upon the type of method used to initialize the neighboring grid point and on the grid spacing (h).

Krimmer and Bruckstein [19] discussed the initialization for linear and circular segmented boundary by first finding the intersection points with all the grid lines and then the linear approximation for the closest grid points and as an approximation, implemented gray scale image based ini-

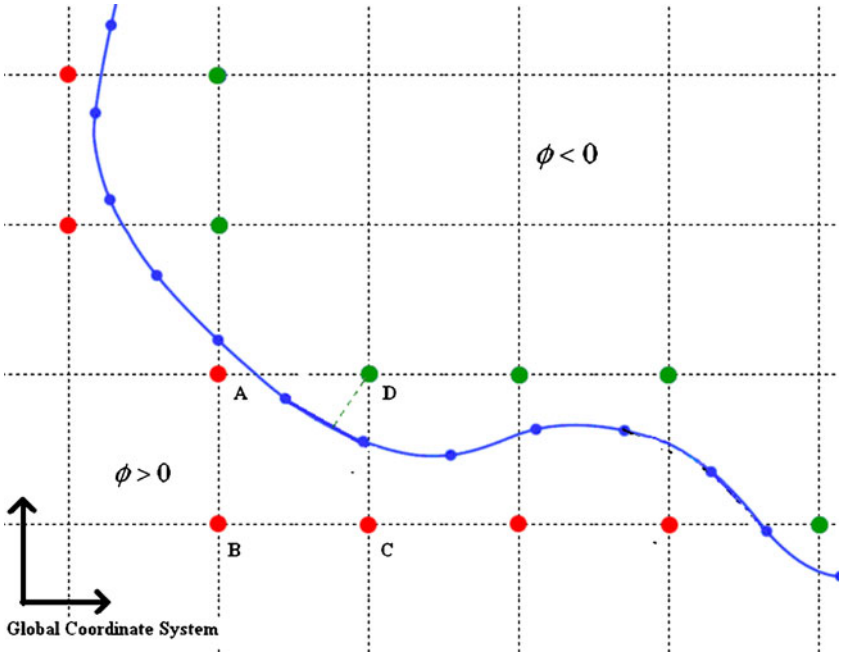

Fig. 5 The closest neighboring points for initialization of distance function

tialization which simply assigns to closest grid points the distance value zero. The initialization could be done by any of the methods listed above. Although the above procedure yields a fast result, the error in the initialization is in the order of grid resolution. In this paper, another method for initialization of neighboring grid point is developed considering the tolerance specified by the user. The following algorithm is devised for this purpose:

- Find neighboring grid points to the pocket boundary curve: The curve is discretized with a discretization interval of $\Delta s=(res)h$, where $res \leq 1$. Each discrete point on the boundary of pocket belongs to a cell which is identified by its left-most corner point (for example *ABCD* is a cell and B is the left-most point). Thus, based on the point on the boundary curve ($x(s)$, $y(s)$) the coordinate of the grid cell is identified as $\left(\text{floor}\left(\frac{x(s)}{h}\right), \text{floor}\left(\frac{y(s)}{h}\right)\right)$. All the four grid points of this cell are stored in an array containing the neighboring grid points.
- Convert to unique set of neighboring grid points: The duplicate grid points (when a grid point is shared by two neighboring cells) are removed. Note, by using this procedure, the grid points are not exactly but approximately arranged according to the arc length parameter of the pocket boundary curve.
- Initialization of the grid points: Each grid point from the above-mentioned set of distinct neighboring grid points is selected, and the grid point distance from the boundary curve is calculated using Newton–Raphson method. As the points are approximately arranged with increasing magnitude of the pocket boundary arc length parameter s, the good initial value guess in Newton–Raphson scheme can be estimated. In this way, the distance value of each neighboring point within a specific tolerance limit is obtained.

Once the initialization is done, the close neighboring points to the boundary curve are initialized for signed distance valued using fast marching formulation. The fast marching process and various steps involved will now be discussed in the following sections.

3.2 Updating process

Once the neighboring green grid points are initialized, to find the value at any grid point inside the domain, the two stencils of a grid point and its four neighbors are considered as in Fig. 6. In the original approach of Sethian [20], only the horizontal and vertical neighboring points in the Cartesian coordinate system are considered to update the distance function; however, this procedure produces an error of the order of $\left(1+1/\sqrt{2}-\sqrt{2}\right)h$ [20], which

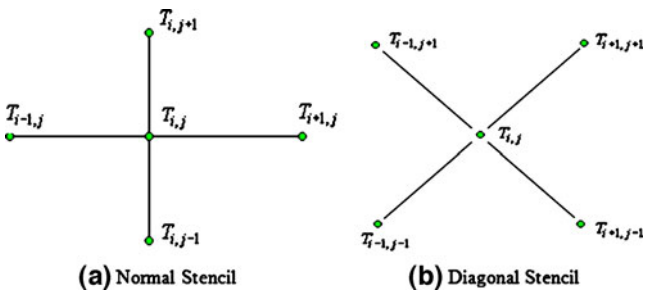

Fig. 6 Updating a grid point: multi-stencil approach

mainly results due to inaccuracy in diagonal direction. Thus, the accuracy in the diagonal direction could be improved if another stencil is used in the diagonal direction. If ∇T in Eq. 3 is approximated by the first-order finite difference scheme, then the above equation can be written as:

$$\max\left(\frac{T-T_1}{\Delta x},0\right)^2+\max\left(\frac{T-T_2}{\Delta y},0\right)^2=\frac{1}{F_{ij}^2} \tag{4}$$

$$\text{Where } \begin{matrix} T_1=\min\left(T_{i-1,j},T_{i+1,j}\right),\\ T_2=\min\left(T_{i,j-1},T_{i,j+1}\right),\end{matrix}$$

and $\Delta x=\Delta y=h$

However, if the solution is based on only the above stencil, the method suffers from numerical errors along the diagonal direction as described above. Thus, a solution along the diagonal direction is sought to make the method more accurate. The formulation for the first-order scheme is as follows:

$$\max\left(\frac{T-T_1}{\sqrt{2}\,\Delta x},0\right)^2+\max\left(\frac{T-T_2}{\sqrt{2}\,\Delta y},0\right)^2=\frac{1}{F_{ij}^2} \tag{5}$$

$$\text{Where } \begin{matrix} T_1=\min\left(T_{i-1,j-1},T_{i+1,j+1}\right),\\ T_2=\min\left(T_{i+1,j-1},T_{i-1,j+1}\right),\end{matrix}$$

It is observed that the above equation is quadratic in nature for T, assuming that the neighboring grid values are given. An iterative algorithm to solve the above equations is used. For each point (i,j), the arrival time T is calculated using both stencils approach. The minimum of the two values was chosen as the value of arrival time.

As it is needed to calculate the arrival at each grid point in a set of N grid points and again having an optimistic view of N iterations for a solution to converge, the complexity of above computation is $O(N^2)$ for 2D application. To reduce the time complexity, the idea of ordering the selection of the grid points during the computation of the solution which is similar to Dijkstra's shortest path solution was introduced by Sethian [20]. This idea exploits the upwind difference structure of the above equation that the information propagates "one way"—from smaller values of T to larger values. In short, it is causality relationship which states that the arrival time T at any point depends only on the adjacent neighbors that have smaller values. The updating algorithm is described as follows:

1. During initialization, the nearest grid points to the boundary are initialized and simultaneously tagged as *Alive* points (see Fig. 7). The grid points which are one grid away from the *Alive* points are tagged as *Narrow band*. All the other grid points are assigned status of *Far Away*.
2. The signed distance value of the narrow band is calculated by using the minimum time calculated by either Eqs. 4 or 5.
3. LOOP: select the minimum arrival time grid point from the *Narrow band*, and change its tag as *Alive*.
4. Find its nearest neighbors, and select the grid points with either *Far away* or *Narrow band* tag. Update their arrival time value according to the minimum arrival time calculated by either Eqs. 4 or 5.
5. Go back to step (3) until all the grid points are tagged as *Alive*.

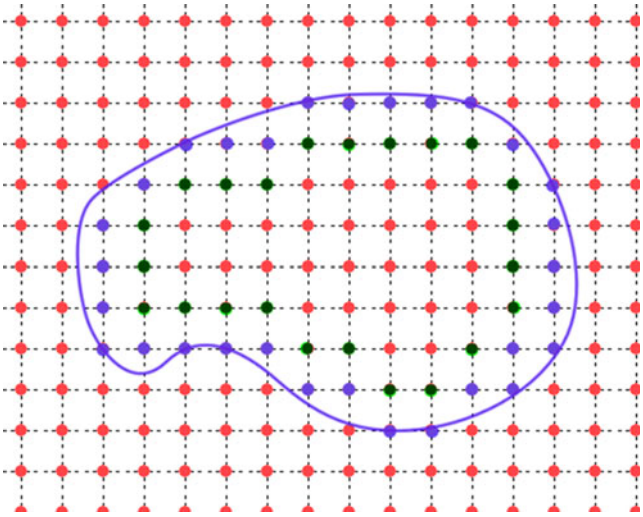

Fig. 7 The "Far away" (*red*), "Alive" (*blue*), and "Narrow band" (*green*) points

The key in the efficiency of this method is to find the minimum arrival time in the narrowband grid points in the step (3) of above-mentioned algorithm. This is done by a standard heap sort algorithm with a min-heap data structure. It is important to note that the total work for one call is O (log N), where N is the size of the heap. Thus, the worst case total operation cost is $O(N \log N)$, where N is the number of total grid points.

3.3 Contour finder

In order to compute the tool path at any offset distance, a contour finder program is required. Once the initialization and the updating processes are over, a simple contour program is written based on the marching square algorithm, which is the special case of marching cube widely used in computer graphics [22]. For each grid point (i,j), there are two possibilities, either the $T(i\Delta x, j\Delta y)$ is greater or equal to zero or it is less than zero. With this understanding, there are 16 types of cases for each cell. These 16 cases could be

reduced to four unique cases, if the symmetry is taken into account. However, there is an ambiguous case for the last configuration in Fig. 8; the segmentation can produce alternative solution. This case is taken care by further dividing the cell into four small cells and continuing until the last case disappears.

3.4 Implementation

3.4.1 Development of tool path generation

A system in C++ is prepared which takes the parametric form of geometric profiles of the pocket and island and tool radius as an input. A rectangular grid is generated which is based on the user-defined meshing resolutions. In the rectangular grid, each grid point can be identified by its *x*-coordinate, and the *y*-coordinate and distance value *T*[*i*][*j*] could be accessed, where *i*=floor(*y*-coordinate of grid point/grid resolution) and *j*=floor(*x*-coordinate of grid point/grid resolution). The initialization for the neighboring grid points is done by traveling along the boundary curves and finding the grid points closest to the boundary curve as mentioned in the section. The updating code is then utilized to find the unique arrival time at each grid point, using the fast marching method. Once the updating process is carried out up to the last grid point and when no grid points are left, it is terminated. This stage is shown as the preparation step in flow chart for tool path generation in Fig. 9. At this stage, the signed distances function of pocket boundary is obtained which is essentially a two-dimensional matrix where each data value refers to the minimum distance value from the boundary of pocket. This signed distance function will be shown as [*Pocket*] which refers to the pocket boundary. Note, in order to find any offset of the boundary of the pocket, a simple scalar subtraction operation is sufficient. The first offset from the pocket should be at the distance of the radius of tool, which makes the material boundary conformed to the required final pocket shape and is given by:

$$[\text{Boundary\_Conformed\_Path}] = [\text{pocket}] - R \tag{6}$$

A contour program (discussed in previous section) is then used to extract the 0-level contours of function [Boundary_Conformed_Path]. This contour program takes any signed distance function as an input along with the grid resolution and gives an ordered set of points which refers to the 0-level boundary of the given signed distance function.

This ordered list of coordinates of points is stored in an array Tool_path(*i*), with *i*= *1*.

The subsequent contour parallel path should follow the offset value specified as the input step over distance. As shown in Fig. 9, a loop is used for the offsetting with the specified step over value (*S*) such that, for each iteration, the tool path matrix is obtained as:

$$[\text{Tool\_path\_matrix}] = [\text{Tool\_path\_matrix}] - S \tag{7}$$

And, the tool path generation is terminated when the there are no positive valued data points in the matrix [Tool_path_matrix]. Otherwise, the iteration continues, and for each signed distance function [Tool_path_matrix], the contour program is used to extract the ordered set of *x* and *y* coordinates and stores them as an array in Tool_Path(*i*), where *i* is incremented by 1 for each new iteration. In this way, all the contour parallel tool paths are obtained for a particular pocket. The detailed flow chart for the tool path generation is shown in Fig. 9.

An example of a pocket with two islands is shown in Fig. 10. A pocket of 500×500 units with two islands of 100×100 units are taken as input, and the tool path is generated for the offset of 22 units.

Next, the machining of an arbitrary shaped pocket will be demonstrated. As described earlier, the method can yield contour parallel tool paths for any type of boundary. A number of methods available in the literature are limited in terms of the geometry they can handle. The feasibility of the method to machine an arbitrary pocket is demonstrated in Fig. 10 for the roughing process. Fig. 11a is an arbitrary 2D pocket, and Fig. 11b shows the corresponding contour parallel tool path.

As demonstrated in [16], some standard libraries of current CAD/CAM systems could yield erroneous results if the thin wall boundaries meet at an angle close to 2π. In the proposed method, the region is initialized into a signed distance function, which may not be accurate initially but

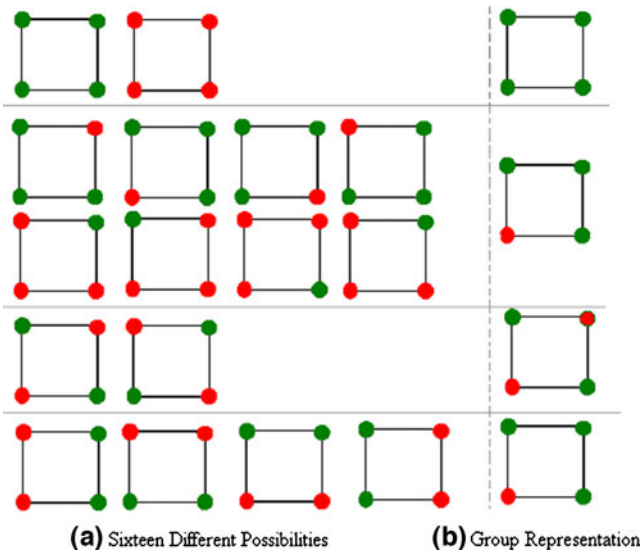

Fig. 8 **a** Different possibilities for the vertex leveling, *green* points corresponds to point with *T* value>0 and *red* points show points with value≤0 **b** group representation

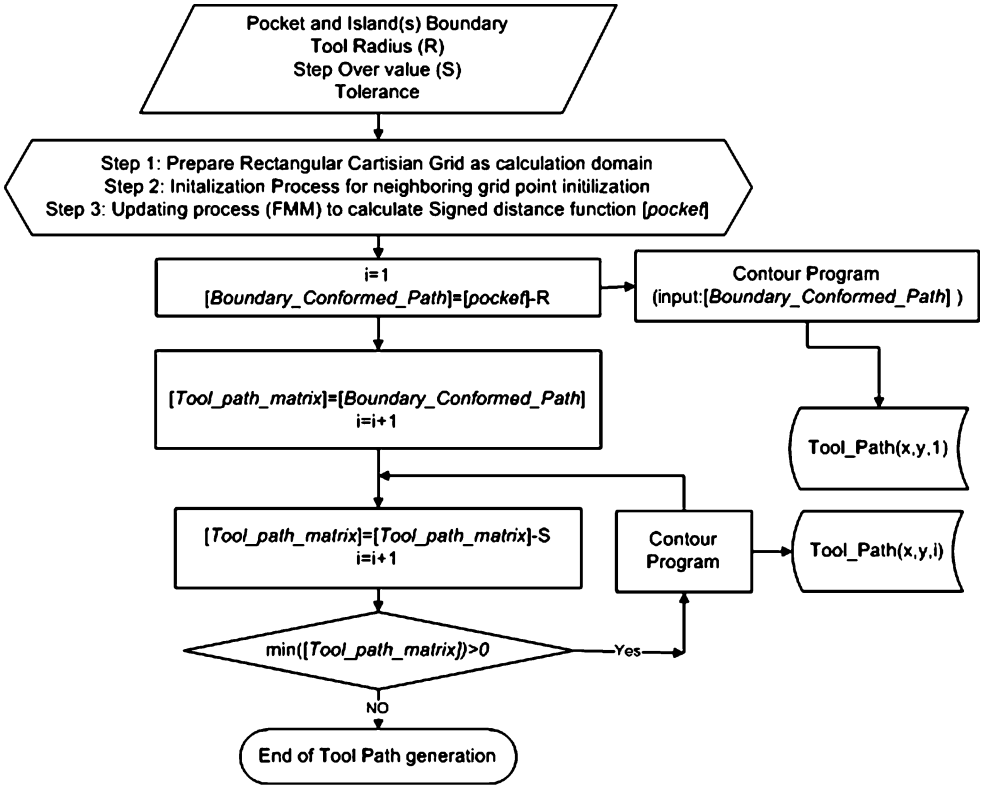

Fig. 9 The flow chart of tool path generation for contour parallel milling process

preserves the property of the bounded region. Thus, such errors are not possible in this method due to the signed distance initialization of the domain.

3.4.2 Error analysis

As mentioned earlier and in the contour parallel tool path generation using the level set method [19], the error reduces with the grid size, and accurate results could be achieved using the required resolution of the grid, but no analysis of the error is provided. The parameter *res* is selected as 1 for all the simulations. For the first-order scheme used to approximate ∇T, the error in computation will be of the order of spacing of the grid, i.e., $o(h)$. However, additional error in the diagonal direction is produced using the approach described in [19]. For the error analysis, the error in the offset position from the boundary is compared with the analytical result. An offsetting of a curve considered as a simple point is chosen which is in the middle of the domain (50×50 units). The expanding offset of a point will be circles, and hence, at each grid point, the exact distance value of the offset is known. The two standard error norms L_2 and $L\infty$ are considered, which are defined as follows:

$$L_\infty \text{ error} = \max\left(\left|T - T_{\text{analytical}}\right|\right) \tag{8}$$

and,

$$L_2 \text{ error} = \text{mean}\left(\left|T - T_{\text{analytical}}\right|\right) \tag{9}$$

L_∞ error reflects the maximum deviation on the domain from the exact solution, while L_2 error represents the mean error expected at any grid point of the domain. The accuracy of the original level set based contour tool path generation [19], and the modified method is compared in

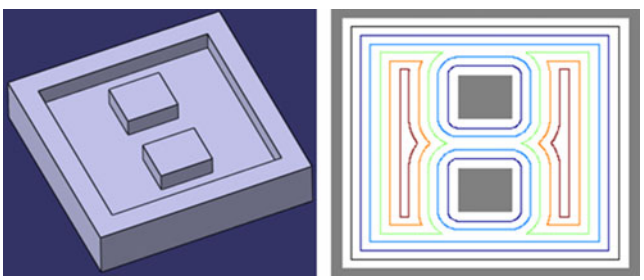

Fig. 10 Pocket with island

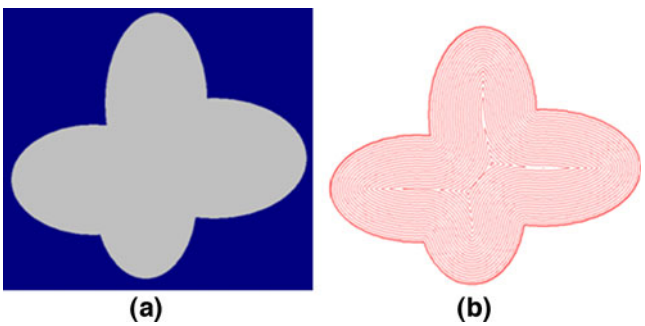

Fig. 11 Arbitrary pocket and contour parallel tool path

Table 1 The accuracy of different models based on standard error norms

Number	Grid size	Bounding box	L mean [19]	L mean (FMM)	L max [19]	L max (FMM)
1	1	50×50	0.640	0.226	1.094	0.447
2	0.5	50×50	0.372	0.145	0.658	0.289
3	0.25	50×50	0.221	0.090	0.386	0.180
4	0.1	50×50	0.109	0.046	0.188	0.08

Table 1. It can be concluded that the accuracies of the result indeed depend upon the resolution, and the error is reduced less than half by using the modified fast marching method. The error is also proportional to the grid size (h), and for the first-order two-stencil scheme, the error is bounded by the grid size.

3.4.3 Computational complexity

The time complexity of the proposed approach is $O(N \log(N))$; N is the total number of grid points. The time to execute the normal fast marching approach and the modified fast marching approach is now compared. Standard PC with Pentium 4 CPU 2.30 GHz is used for all the computations. Table 2 is prepared for the two methods; although the complexity of the both methods remains same, it appears that the execution time for modified fast marching method (FMM2) is 1.2–1.5 times higher than the ordinary FMM1. This increase in time is due to the additional logic added to the modified fast marching method.

As compared with the narrow band level set method [17], it is important to note that there is no time step in the fast marching method. Thus, the fast marching method does not require the satisfaction of the Courant–Friedeichs–Lewy condition, which imposes a restriction on the time step based on the grid size, whereas the narrow band level set method does, and hence could be much slower than the fast marching method. Also, the heap operation time complexity is *ln (N)* in the worst case, and for practical problems, it is probably closer to $O(1)$. In case of the three-dimensional (3D) surface generation problem, this method based on the up-winding nature is useful for offsetting also. Thus, it could be useful for determining the offsetting surfaces in the higher dimension with the worst time complexity of $O(M^3 \ln(M))$, where M is the number of grid points in each direction.

4 Conclusion

A system to generate a tool path based on the fast marching formulation is presented in this paper. It can be seen that the appropriate grid resolution can be selected based on the required tolerance of the tool path. The method is found to be computationally efficient and is suitable for generating tool paths for the milling process. The advantages of this method could be summarized as follows:

1. The method naturally deals with the changing in the topology of the offsetting fronts; hence, it is an effective method for generating tool paths for pockets with islands as it is not required to track intersecting offsetting profiles of the pocket and islands.
2. The offsetting fronts may be self-intersecting or cusps or corners could form as the offsetting is taking place; in such cases, a viscosity-based weak solution is readily available; thus, it is not required to compute the self-intersecting sections separately. The corner points generated by the self-intersecting features are smoothened based on the updating scheme used for computation, and the c^1 continuity is maintained throughout the solution. Because of the smoothing of the sharp corners, the generated tool path is better in relation to machine dynamics.
3. The fast marching method and heap sort algorithm is developed in C++ and can be easily integrated with the ACIS kernel. The capability of the integrated system is demonstrated in the paper.
4. With every numerical method, there are always some numerical errors associated. As fast marching method

Table 2 The execution time for different fast marching methods

Number	Grid size	Bounding box	Execution time, ms(FMM1)	Execution time, ms (FMM2)	$N \log(N)$
1	1	50×50	94	156	1.9560e4
2	0.5	50×50	563	859	9.2103e4
3	0.25	50×50	5,047	6,390	42.387e4
4	0.1	50×50	149,440	166,778	310.73e4

calls for discretization of the domain, the numerical errors arise. The analysis of numerical error is compared with the possible analytical errors, and it is found that the standard error decreases with the reduction in the grid size. Furthermore, the fast marching method implementation is done on rectangular grid with uniform grid length. Very high grid resolution is needed for pocket boundary with high curvature sections. In order to reduce the computational time for high curvature section, an adaptive grid would be more beneficial.

The implementation of fast marching method is a first step towards the fast computation of contour parallel tool paths for a given arbitrary zone. The method presented here for 2D pocket boundary offsetting can be further extended for the contour surface offsetting in 3D contour pocket milling with consideration of motion along the additional Z direction. Furthermore, the smoothing could be controlled by using the motion based on curvature (i.e., speed function $F = 1 - \varepsilon\kappa$), and the uniform spacing between the two contours could be maintained. The further efforts will be directed towards the generation of contour parallel tool paths for optimization of the curvature of the tool path and the engagement condition between tool and workpiece.

References

1. Kramer TR (1992) Pocket milling with tool engagement detection. J Manuf Syst 11(2):114–123
2. Held M (2001) VRONI: an engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. Comp Geom Theor Appl 18(2):95–123
3. El-Midany TT, Elkeran A, Tawfik H (2006) Toolpath pattern comparison contour-parallel with direction-parallel. Proceedings of the Conference on Geometric Modeling and Imaging New Trends, 5–6 July, London, UK, p 77–82
4. Farouki RT (1985) Exact offset procedures for simple solids. Comput Aided Geom Des 2(4):257–279
5. Farouki RT (1992) Pythagorean-hodograph curves in practical use. Geometry Processing for Design and Manufacturing. SIAM Publishers, Philadelphia, pp 3–33
6. Persson H (1978) NC machining of arbitrary shaped pockets. Comput Aided Des 10(3):169–174
7. Klass R (1983) An offset spline approximation for plane cubic splines. Comput Aided Des 15(5):297–299
8. Tiller W, Hanson EG (1984) Offsets of two-dimensional profiles. IEEE Comput Graph Appl 4(9):36–46
9. Hoschek J (1988) Spline approximation of offset curves. Comput Aided Geom Des 5(1):33–40
10. Elber G, Lee IK, Kim MS (1997) Comparing offset curve approximation methods. IEEE Comput Graph Appl 17(3):62–71
11. Hansen A, Arbab F (1992) Algorithm for generating NC tool paths for arbitrarily shaped pockets with islands. ACM Trans Graph 11(2):152–182
12. Veeramani D, Gau YS (2000) Cutter-path generation using multiple cutting-tool sizes for 2-1/2D pocket machining. IIE Trans (Institute of Industrial Engineers) 32(7):661–675
13. Stori JA, Wright PK (2000) Constant engagement tool path generation for convex geometries. J Manuf Syst 19(3):172–183
14. Choi BK, Kim BH (1997) Die-cavity pocketing via cutting simulation. CAD Comput Aided Des 29(12):837–846
15. Saeed SEO, de Pennington A, Dodsworth JR (1988) Offsetting in geometric modelling. Comput Aided Des 20(2):67–74
16. Molina-Carmona R, Jimeno A, Rizo-Aldeguer R (2007) Morphological offset computing for contour pocketing. J Manuf Sci Eng, Transactions of the ASME 129(2):400–406
17. Bieterman MB, Sandstrom DR (2003) A curvilinear tool-path method for pocket machining. J Manuf Sci Eng, Transactions of the ASME 125(4):709–715
18. Chuang JJ, Yang DCH (2007) A Laplace-based spiral contouring method for general pocket machining. Int J Adv Manuf Technol 34(7–8):714–723
19. Kimmel R, Bruckstein AM (1993) Shape offsets via level sets. Comput Aided Des 25(3):154–162
20. Sethian JA (1996) A fast marching level set method for monotonically advancing fronts. Proc Natl Acad Sci USA 93 (4):1591–1595
21. Hassouna MS, Farag AA (2007) Multistencils fast marching methods: a highly accurate solution to the Eikonal equation on Cartesian domains. IEEE Trans Pattern Anal Mach Intell 29 (9):1563–1574
22. Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3D surface construction algorithm. Comput Graph (ACM) 21 (4):163–169