






Simulation-Driven Computation of High-Speed Pocket Machining Toolpaths

Tathagata Chakraborty¹ , Chinmaya Panda² , Nitin Umap³ 

¹HCL Technologies, tathagata.chakr@hcl.com

²HCL Technologies, chinmaya.panda@hcl.com

³HCL Technologies, nitin.umap@hcl.com

Corresponding author: Tathagata Chakraborty, tathagata.chakr@hcl.com

Abstract.

Curvilinear toolpaths are the preferred choice for high-speed machining of 2.5D pockets when capable machine tools are available. Curvilinear toolpaths support high feed rates since the tool engagement and movement direction change only gradually. By maintaining a near-constant tool engagement, curvilinear toolpaths also increase the life of the tools. Automatically generating such toolpaths, however, requires complex algorithms.

In this paper, we describe a simulation-driven method for generating curvilinear toolpaths. A GPU executes the simulation by rendering the pocket and the tool cross-sections on a discrete frame buffer. The tool is moved in small increments, and a discrete tool engagement value is computed by comparing the framebuffer data before and after the tool is moved. The direction in which to move the tool is determined using a brute-force approach that explores a range of angles in the neighborhood of the previous move direction and selects the direction in which the tool engagement value is closest to a preferred value.

Keywords: pocket machining, high-speed machining, curvilinear toolpath, spiral toolpath

DOI: <https://doi.org/10.14733/cadaps.2024.88-103>

1 INTRODUCTION

Despite the popularity of 3-axis and 5-axis CNC machines, machining 2.5D pockets remains important. Large volumes of a given part can be quickly removed using 2.5D roughing [15, 1]. There are three main types of 2.5D roughing toolpaths - contour-parallel, direction-parallel, and curvilinear (sometimes also called a spiral toolpath). Amongst these, curvilinear toolpaths are best suited for high-speed machining. However, generating curvilinear toolpaths is not easy. Implementing a production-quality curvilinear toolpath library that can machine arbitrary pocket shapes can take several years of hard effort. Although computationally more expensive than a geometry-based approach, a simulation-driven method offers a quick and low-code approach to solving this problem.

1.1 Simulation-Driven Algorithms

Although Moore's law has slowed down in recent years, the availability of computing power has kept increasing, and its cost has continued decreasing [20]. This increase in the availability of computing power can be seen as a response to the demand for GPU-driven algorithms in artificial intelligence and machine learning [18]. Affordable GPU computing power has also made simulation-driven algorithms feasible and attractive. Simulation-driven methods may be slower than a traditional computational geometry-based approach for specific problems. However, these methods are well-poised to use newer hardware when available. More importantly, simulation-based algorithms are much simpler to implement because they replace some of the more complex parts of intelligent algorithms with either a statistical black box (like a neural network) or a brute-force computation.

1.2 Our Cutting Simulation Approach

In this paper, we describe a simulation-driven method for generating curvilinear toolpaths. The algorithm simulates the material-cutting process in 2D using the GPU to accelerate the simulation. The toolpath is generated by determining the next best direction in which to move the tool at each step, given its last movement direction and tool engagement constraints. The algorithm works for all pocket shapes without requiring complex heuristics. Furthermore, a simulation-based approach enables the toolpath developer to focus on higher-level meta-heuristics rather than lower-level geometric details. Before describing our method, we briefly review some 2.5D toolpath generation techniques.

2 RELATED WORK

This section discusses prior work on generating toolpaths for machining 2.5D pockets, focusing primarily on modern methods for generating curvilinear toolpaths for high-speed machining.

2.1 Contour-Parallel and Direction-Parallel Toolpaths

Contour-parallel techniques have been used for machining pockets for a very long time. Here the pocket is machined by driving the tool along curves offset from the pocket boundary (see Fig. 1 (left) for an illustrative example). The contour-parallel toolpath generally starts from a point near the center of the pocket and gradually advances toward the boundary.

Alternatively, direction-parallel techniques like zig and zig-zag approaches can also be used (see Fig. 1 (middle)). Here the tool is moved in a direction parallel to a given direction, either always moving in one direction (in the case of zig toolpaths) or moving alternately in opposite directions (in zig-zag toolpaths).

The toolpaths generated by these techniques are often optimized to reduce the number of retractions and the overall length of the toolpath. However, such toolpaths are still far from optimal since they completely ignore machine dynamics [4]. Both contour-parallel and direction-parallel toolpaths contain many sharp corners. During machining, the tool has to slow down before it approaches a sharp corner and then speed up again as it exits the corner. The need to decelerate and accelerate repeatedly can significantly slow down the machining process. Toolpaths with sharp corners also wear down the tools faster since the tool engagement increases abruptly in these corners.

2.2 Curvilinear Toolpaths

Curvilinear or spiral toolpaths are smoother versions of contour-parallel toolpaths used mainly for high-speed machining. High-speed machining techniques minimize sharp corners in toolpaths and maintain a near-constant tool engagement throughout the process.

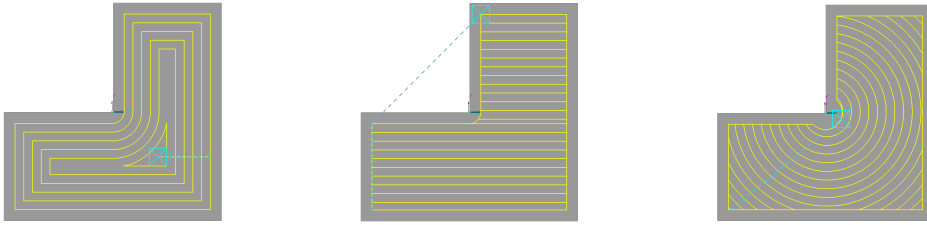


Figure 1: Left: Contour parallel toolpath for an L-shaped pocket. Middle: Zig-zag toolpath for the same pocket. Right: A naive spiral toolpath with sharp corners.

Smoothing corners of toolpaths generated using contour-parallel techniques can be used as a starting point for generating high-speed toolpaths (see [19] [21]). In [4], a method is described where the toolpath starts as a spiral near the center of the pocket and gradually morphs into the pocket shape as it approaches the part boundary. In [8], a method based on maintaining constant tool engagement while machining the external contour of a given part is described. In [22], a method that limits the maximum curvature in the toolpath is described. In [6], the generation of a curvilinear toolpath is framed as an optimization problem where the machining time and the forces on the tool are minimized. In [3], a continuously morphed spiral toolpath consisting of splines is proposed. A technique where the pocket is first mapped onto a circular domain and then a guide spiral in the circular domain is inversely mapped to the pocket is proposed in [23]. In [16], an analytic method for morphing a spiral toolpath to fit the shape of the pocket is presented. In [10], [11], and [2], the authors propose different spiral toolpath generation methods based on the medial axis transform.

In [14] and [12], discrete and geometry-based simulation methods for analyzing and modifying toolpaths are described. In [24], a method for generating smooth toolpaths is described based on extracting iso-curves from a blurred version of a discrete medial axis transform image of the pocket.

2.3 Summary of Existing Techniques

For a review of the various traditional techniques and limitations of pocket-machining, see [15] and [9]. Traditional pocket machining techniques are still commonly used due to a conservative familiarity with the techniques and sometimes due to the limitations of the CNC machine and software licensing costs. At this time, however, it is clear that a combination of a spiral toolpath and trochoidal moves for machining the corners and slots are the best-known strategies for high-speed machining. As noted above, several possible curvilinear toolpaths exist for a particular pocket shape, and there are several ways of generating such toolpaths. This paper presents a novel simulation and metric-driven method for computing similar toolpaths. Our method requires much less code (all significant parts of the algorithm are included in this paper in pseudocode) and is thus easier to implement than other methods.

3 PIXEL-BASED TOOL ENGAGEMENT

For simulation, we use a hardware-accelerated 2D rendering engine, using which the pocket, tool, and plunge holes are rendered on screen or onto an off-screen framebuffer. An illustrative screenshot is shown in Fig. 2 (left), where the pocket cross-section is rendered in grey, and a plunge-hole from where to start the machining process is rendered in black.

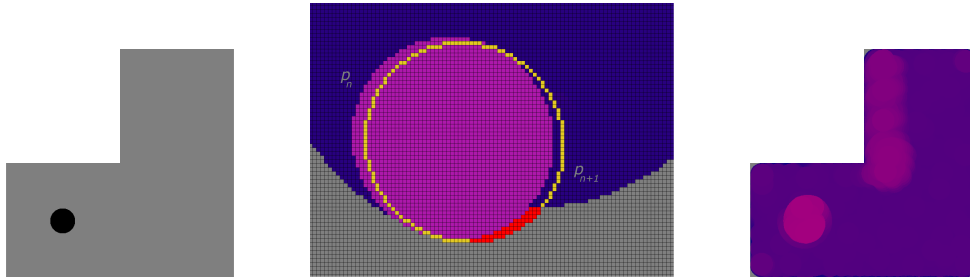


Figure 2: Left: Simulation environment showing an L-shaped pocket with a plunge-hole. Middle: Pixel engagement computation. Right: Cutting simulation result showing tool engagement.

3.1 A Discrete Toolpath

The toolpath is computed as a polyline consisting of small segments of equal length. The tool engagement over each segment is approximated by counting the difference in the number of pocket pixels in the framebuffer when the tool is rendered at the start and then at the end point of the segment. Such a toolpath with many small segments may not be appropriate for older CNC machines. As noted in [8], this problem can be somewhat mitigated using the Douglas-Peucker or similar polyline simplification algorithms. Modern NC controllers can automatically smooth toolpaths consisting of small segments when the G64 G-code instruction is specified.

The pixel-based tool engagement computation is illustrated in Fig. 2 (middle). Here the pocket material is shown in grey, material that has already been cut is shown in blue, the tool located at position p_n is shown in fuchsia, and the tool located at the next position p_{n+1} is shown in outline in yellow. The additional material that gets cut when the tool moves from position p_n to p_{n+1} is indicated in red. The count of red pixels forms our measure of tool engagement for this move. During simulation, the tool is rendered in graded shades of purple, with increasingly pink values indicating larger tool engagements and bluer shades indicating smaller tool engagement values. This enables quick visual inspection of the changes in the tool engagement value as the pocket is machined, as shown in Fig. 2 (right).

3.2 Toolpath Step, Engagement Angle and Rendering Resolution

The tool engagement angle can also be approximated from the framebuffer data. However, when the tool is moved smoothly and in equal increments, the pixel engagement count is directly proportional to the engagement angle under most circumstances and can be used as a reliable proxy. An important concern is determining the delta measure by which the tool should be moved in each step. A tiny move may cut only a handful of pixels in each step giving only a rough estimate of the tool engagement that can be very noisy. A large delta move, on the other hand, will result in a faceted and inaccurate toolpath. Therefore, an appropriate delta move needs to be carefully chosen depending on the required engagement angle and the resolution of the framebuffer at which the simulation is being rendered. For example, a smaller engagement angle will require a framebuffer with a larger resolution for a given toolpath delta step.

4 DEVELOPING A SIMULATION-BASED STRATEGY

The pixel tool engagement data can be used to develop various strategies for computing a simulation-driven toolpath. A simulation-based approach enables us to use a low-code [17] method for computing complex toolpaths, which would otherwise require a lot of time and effort. Our method is most similar to the ones described in [8] and [14], with the difference that we develop the toolpath from the ground up. In contrast, these previous methods rely on modifying an existing toolpath. In [8], the authors describe a simulation-driven

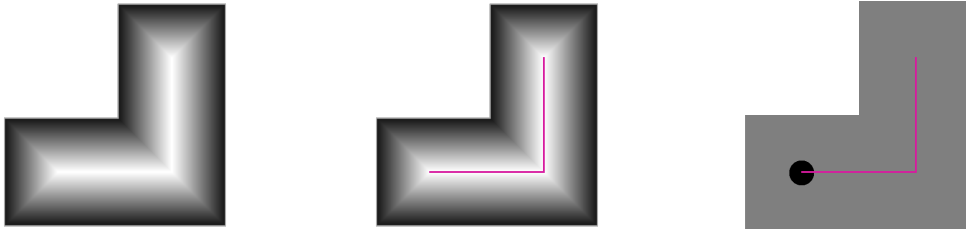


Figure 3: Left: Euclidean distance transform of the L-shaped pocket. Middle: Discrete Medial Axis obtained by thresholding the maximum values in the transform. Right: Plunge hole created at the first point of the skeleton.

strategy for machining the outer contour of a given part; however, the strategy fails to maintain smoothness when contouring near the part surface where the toolpath has to morph into the shape of the part. Also, it is not clear if and how their strategy can be used to machine complex pockets with or without islands. Similarly, in [12] and [13], the authors describe a geometric simulation-driven method for computing a local curvilinear toolpath, but they do not describe an overall strategy for computing the complete toolpath for a general 2.5D pocket.

4.1 Our Strategy in Brief

We start by computing the discrete medial axis of the pocket shape. Next, a point with the largest value on this medial axis (a point farthest from the pocket boundary) is chosen, and a plunge hole of diameter a little larger than the tool diameter is made in that location. Multiple plunge holes may also be made depending on the length of the medial axis. The tool is then placed in the plunge hole and advanced incrementally in small movements in a pre-determined direction until it starts cutting into the material.

Once the tool starts cutting material, further moves are chosen such that the tool engagement always lies within a specific narrow range, and each toolpath move makes only a small angle from the previous move direction. These two constraints are sufficient to create a spiral toolpath that stops when it hits the pocket boundary where it first fails to maintain sufficient engagement. Therefore, additional heuristics are needed to keep the tool moving to machine the corners and other regions of the pocket. This can be done by relaxing the engagement constraints and allowing the tool to follow the boundary of the pocket when the engagement criteria cannot be met. The method exits when the tool completes a full traversal of the outer boundary of the pocket without cutting any additional material.

At this stage, the toolpath is neither optimal nor smooth and contains numerous redundant non-cutting moves. Several post-processing steps are required to fix these issues. The toolpath is first segmented into regions where it is cutting material and where it is not. Next, the cutting regions are grouped to form trochoidal sections by adding linking moves. The non-cutting regions are also grouped and connected with the trochoidal sections to create the complete toolpath. Finally, the sharp corners in the toolpath where the non-cutting and linking moves join the cutting regions are smoothed out. We explain these steps in more detail in the sections below. We also show how the overall strategy can be developed starting from a more naive approach.

4.2 Discrete Medial Axis Computation

Fig. 3 shows an L-shaped pocket's discrete medial axis. The discrete medial axis is obtained by first computing the Euclidean distance transform using the grassfire algorithm and then thresholding the maximum value in this transform [5] to create a skeleton (shown in pink in Fig. 3 (middle)).

Depending on its length, one or more plunge holes may be created on the medial axis. Depending on where the plunge hole is made on the medial axis, the resulting toolpath can be very different. Given enough computational resources, multiple toolpaths may be created by choosing different plunge hole locations, and the best amongst the resulting toolpaths can be selected. To focus on the paper's main ideas, we chose the first point in the medial axis for creating the plunge hole, as shown in Fig. 3 (right).

Algorithm 1: Simple spiral toolpath strategy

```

Input:  $P_{tp} = [p_1, p_2, \dots, p_n]$  // points comprising the toolpath
          $\theta_{max}$  // the maximum deviation angle
          $\delta_{step}$  // toolpath step size
          $\varepsilon_{min}, \varepsilon_{max}$  // tool engagement limits

Function CheckEngagement( $p$ ):
   $FB_{before} \leftarrow readPixels()$  // read the framebuffer data
   $renderTool(p)$  // render the tool in the next location
   $FB_{after} \leftarrow readPixels()$ 
   $\varepsilon \leftarrow pocketPixels(FB_{before}) - pocketPixels(FB_{after})$  // count change in pocket pixels
  return  $\varepsilon$ 

Function GetEngagement( $P_{tp}, \theta$ ):
   $\vec{d} \leftarrow p_n - p_{n-1}$  // direction of the last toolpath segment
   $\vec{d}_\theta \leftarrow R_\theta(\vec{d})$  // rotate the last direction by  $\theta$ 
   $p \leftarrow p_n + \vec{d}_\theta * \delta_{step}$ 
  return ( $p, checkEngagement(p)$ )

Function FindNextMove( $P_{tp}, \theta_{max}$ ):
   $A_\theta \leftarrow [0^\circ, +1^\circ, -1^\circ, +2^\circ, -2^\circ, \dots, +\theta_{max}, -\theta_{max}]$ 
  for  $\theta \in A_\theta$  do
    ( $p, \varepsilon$ )  $\leftarrow GetEngagement(P_{tp}, \theta)$ 
    if  $\varepsilon_{min} \leq \varepsilon \leq \varepsilon_{max}$  then
      return  $p$ 
    end
  end

```

4.3 Spiral Toolpath with Simple Heuristics

A spiral toolpath that terminates when the tool hits the pocket boundary can be obtained using simple heuristics. However, such a strategy can only cut a circular region within a pocket. Over the following few sections, we show how this strategy can be extended to handle arbitrary pockets.

4.3.1 Pseudocode Description

Algorithm 1 shows the simple spiral strategy in pseudocode. The function *FindNextMove* computes the next point to which the tool should be moved given the previous points comprising the toolpath P_{tp} and the maximum deviation angle θ_{max} from the last direction. This algorithm is called repeatedly to compute the complete toolpath. The function *GetEngagement* returns the tool engagement value for a particular angle and has been refactored to make the algorithm more readable. We will keep referring to this function in later

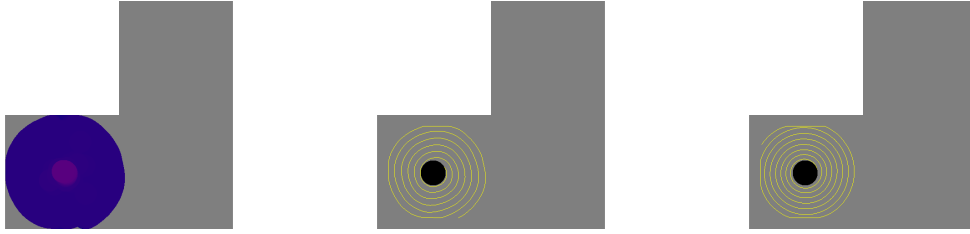


Figure 4: Left: Material cut and tool engagement using Algorithm 1. Middle: Toolpath for the result on the left using Algorithm 1. Right: Choosing the best angle results in a better spiral toolpath.

improvements to the algorithm. We have also tried to keep the notation consistent and avoid repeating some inputs and constants in later algorithms.

Here P_{tp} contains the points representing the current toolpath as a line strip. θ_{max} is the maximum angle by which a toolpath segment can deviate from the previous segment. θ_{max} along with the step size δ_{step} controls the smoothness of the generated toolpath with smaller values resulting in smoother toolpaths. The δ_{step} also depends on the resolution of the simulation framebuffer and will have a minimum value beyond which it cannot be decreased (see section 3.2). The *checkEngagement* function computes the tool engagement as the tool moves from the position p_n to p_{to} . The algorithm greedily accepts the first valid move with the smallest angular deviation from the previous direction.

The results of the simulation-driven toolpath using Algorithm 1 are shown in Fig. 4. Fig. 4 (left) shows the material that has been cut using the toolpath displayed on the right. The cut material is rendered in shades of blue and pink, where increasingly pink values indicate increasing tool engagement. While very simple, such a toolpath is unsatisfactory since it can machine only a small portion of the overall pocket.

Algorithm 2: Simple brute-force/non-greedy toolpath strategy

Input: ε_{tar} // target required engagement

Function FindNextMove(P_{tp}, θ_{max}):

```

 $A_\theta \leftarrow [0^\circ, +1^\circ, -1^\circ, +2^\circ, -2^\circ, \dots, +\theta_{max}, -\theta_{max}]$ 
 $\delta_{best} \leftarrow \varepsilon_{tar}$  // the smallest difference from target engagement
 $p_{best}$  // the best position to move to
for  $\theta \in A_\theta$  do
     $(p, \varepsilon) \leftarrow GetEngagement(P_{tp}, \theta)$ 
    if  $|\varepsilon - \varepsilon_{tar}| < \delta_{best}$  then
         $\delta_{best} \leftarrow |\varepsilon - \varepsilon_{tar}|$ 
         $p_{best} \leftarrow p$ 
    end
end
return  $p_{best}$ 

```

4.3.2 Some Minor Improvements

The initial large tool engagement can be avoided by letting the algorithm explore a larger range of angles since a somewhat higher curvature is initially required for a spiral toolpath. Additionally, on closer observation, it can be seen that the toolpath is not exactly spiral. This is because Algorithm 1 is a greedy algorithm where we choose the first valid angle. A better approach is to choose the angle that results in an engagement closest

to the desired tool engagement. The modified algorithm is shown in Algorithm 2, and the toolpath is shown in Fig. 4 (right).

If we further increase the maximum deviation angle, then we can move in straight segments along the boundary (see Fig. 4 (right)). During these moves, the tool engagement falls well below the target engagement and must be relaxed. However, further exploring this line of thought does not result in a robust overall strategy. Our solution is to separate the strategy into two parts. The first part tries to find a move within the engagement limits. If such a move cannot be found the second part tries to find a move along the boundary of the pocket. We explain this strategy in detail in the next section.

5 A BOUNDARY FOLLOWING STRATEGY

Our strategy for constructing the initial toolpath consists of two parts (a) the part that computes the best cutting move and (b) the part that ensures that the tool moves along the boundary when a valid cutting move cannot be found.

Algorithm 3: Best Spiral Move Function

Input: ε_{tar} // target engagement
 δ_{tol} // max preferred change from target engagement

Function FindBestMove(P_{tp}, θ_{max}):

```

 $\delta_{best} \leftarrow \varepsilon_{tar}$ 
 $\theta \leftarrow -\theta_{max}$ 
while  $\theta \leq \theta_{max}$  do
     $(p, \varepsilon) \leftarrow GetEngagement(P_{tp}, \theta)$ 
     $\delta \leftarrow |\varepsilon - \varepsilon_{tar}|$ 
    if  $\delta < \delta_{tol}$  then
        | return  $(p, true)$ 
    end
    if  $\delta < \delta_{best}$  then
        |  $\delta_{best} \leftarrow \delta$ 
        |  $p_{best} \leftarrow p$ 
    end
    if  $\delta_{best} < \varepsilon_{tar}$  and  $\delta > \delta_{best}$  then
        | return  $(p_{best}, true)$ 
    end
     $\theta \leftarrow \theta + 1$ 
end
return  $(p_{best}, \delta_{best} < \varepsilon_{tar})$ 

```

5.1 Best Cutting Move

The function to compute the best spiral move is shown in Algorithm 3 with various early return optimizations. The function returns a tuple with the best-found move p_{best} and another variable indicating its validity. For each angle in the range $-\theta_{max}$ to θ_{max} , we compute the tool engagement ε and the magnitude δ of its difference from the target engagement ε_{tar} . If this difference δ is less than a small amount δ_{tol} , then return the found position. Else if this difference is smaller than the smallest difference δ_{best} , we update δ_{best} and the best move position p_{best} . In the last *if* block, we check if we have reached a local minima and return early if

the δ difference starts increasing beyond δ_{best} . We may reach the end of the loop if no δ that is less than δ_{tol} is found. In this case, we return *true* if a positive tool engagement was found.

5.2 Best Boundary Move

The function to get the tool to follow the boundary of the pocket is shown in Algorithm 4. If a valid boundary move is found in the angle range $-\theta_{max}$ to θ_{max} , the function returns a tuple $(p, true)$ with the current position and validity set to *true*. If no valid boundary move is found, the function returns $(_, false)$, indicating strategy failure and stopping the initial toolpath generation process. The boundary can be found by comparing the values of the current engagement ε with that of the previous engagement ε_{prev} . If the previous engagement ε_{prev} is -1 (the *checkEngagement* function from Algorithm 1 and therefore the *GetEngagement* functions return -1 when a gouge is detected) and the current engagement is 0, then we return the current position. Else if the previous engagement ε_{prev} is 0 and the current engagement is -1 then we compute the position for the previous engagement and return.

Algorithm 4: Boundary Move Function

```

Function FindBoundaryMove( $P_{tp}, \theta_{max}$ ):
   $(_, \varepsilon_{prev}) \leftarrow GetEngagement(P_{tp}, -\theta_{max})$ 
   $\theta \leftarrow -\theta_{max} + 1$ 
  while  $\theta \leq \theta_{max}$  do
     $(p, \varepsilon) \leftarrow GetEngagement(P_{tp}, \theta)$ 
    if  $\varepsilon_{prev} = -1$  and  $\varepsilon = 0$  then
      return  $(p, true)$ 
    else if  $\varepsilon_{prev} = 0$  and  $\varepsilon = -1$  then
       $(p, \varepsilon) \leftarrow GetEngagement(P_{tp}, \theta - 1)$ 
      return  $(p, true)$ 
    end
     $\varepsilon_{prev} \leftarrow \varepsilon$ 
     $\theta \leftarrow \theta + 1$ 
  end
  return  $(_, false)$ 

```

A simplified version of the overall algorithm that computes the next move is shown in Algorithm 5, where the *FindBestMove* tries to find the best cutting move within the specified angle range (here $\pm 90^\circ$) and the *FindBoundaryMove* function tries to move along the pocket boundary in case a cutting move is not found.

Algorithm 5: Overall Move Function

```

Function FindNextMove( $P_{tp}$ ):
   $(p_{n+1}, valid) \leftarrow FindBestMove(P_{tp}, 90)$ 
  if not valid then
     $(p_{n+1}, valid) \leftarrow FindBoundaryMove(P_{tp}, 90)$ 
  end
  return  $p_{n+1}$ 

```

This strategy gives us a toolpath consisting of spiral and curvilinear moves (see Fig. 5 for some examples). However, it also contains a lot of redundant boundary moves. It is possible to avoid these redundant moves

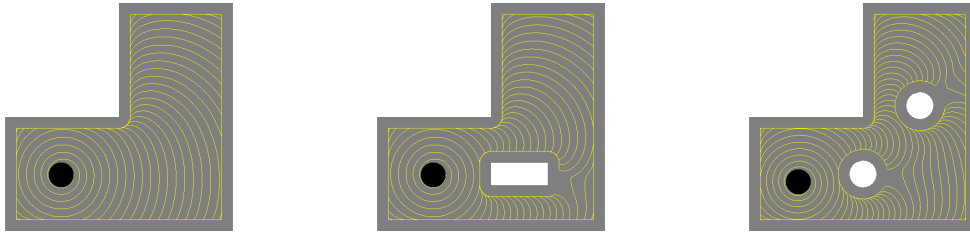


Figure 5: Boundary following toolpaths for variations of an L-shaped pocket with islands.

by using a more complicated strategy, but it will be difficult to generalize such a strategy across arbitrary pocket shapes. An easier and more robust way is to remove the redundant boundary moves and group the near-parallel curvilinear moves into trochoidal sections using post-processing steps.

6 POST-PROCESSING

In post-processing we first identify the cutting and non-cutting sections of the toolpath. This can be done by checking the tool engagement values at each toolpath segment and grouping contiguous cutting and non-cutting segments. The next step involves identifying trochoidal groups from amongst the cutting sections. This can be done by checking if two cutting sections are nearly parallel to each other and also approximately separated by the stepover distance (see Fig. 6 (left)). Finally, we also need to identify overlapping boundary moves in the non-cutting sections discarding all but the smallest such move in each overlapping boundary group (see Fig. 6 (middle)).

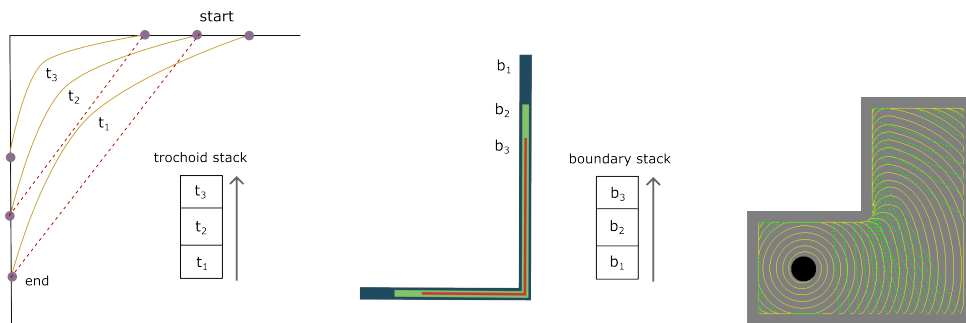


Figure 6: Grouping of trochoidal toolpath segments (left), grouping overlapping boundary segments (middle) and the composed toolpath (right).

Next, we connect the curvilinear moves in each trochoidal group to create the trochoidal toolpath in these regions (see the dashed lines in Fig. 6 (left)). This is followed by a composition step that connects the initial spiral and boundary moves with the trochoidal moves to create the complete toolpath (see Fig. 6 (right) where the dashed lines indicate linking moves). Lastly, the sharp moves in the toolpath are smoothed out using a smoothing step. The final toolpath can be checked for gouge, and the tool engagement variation can be inspected by using the simulation environment to step through the tool path and create a rendering like that shown in Fig. 2 (right).

6.1 Identifying Cutting and Non-cutting Regions

The cutting and non-cutting regions can be identified using the tool engagement data. Contiguous toolpath segments with zero or low engagement are grouped into a single non-cutting region, and contiguous segments with positive engagement are grouped into a single cutting region. A composite data-structure called a 'Region' is used, which consists of a polyline, a boolean indicating if the region is cutting or non-cutting, and an integer indicating the trochoidal or boundary group to which the region belongs. The procedure is shown in Algorithm 6 where the *FindRegion* function (not shown but trivial to implement) returns a contiguous region starting at a given index i_{start} . The algorithm's output is a list of regions used in the next post-processing steps.

Algorithm 6: Identifying cutting/non-cutting regions

Input: $E_{tp} = [\varepsilon_1, \varepsilon_2, \dots, \varepsilon_n]$ // tool engagement values at each point

```

struct {
  | polyline pl, bool cutting, int group
} Region

```

Function *SegmentRegions*(P_{tp}):

```

   $R = []$  // segmented regions
   $i_{start} \leftarrow 0$  // index from which to start segmenting
  while  $i_{start} < n$  do
    |  $r = \text{FindRegion}(P_{tp}, E_{tp}, i_{start})$ 
    |  $R \leftarrow R + r$ 
    |  $i_{start} \leftarrow i_{start} + \text{length}(r)$ 
  end
  return  $R$ 

```

6.2 Grouping Trochoidal Regions

Once the cutting and non-cutting regions have been segmented, we can identify the trochoidal groups from amongst the cutting regions. The potential trochoids are grouped into stacks and identified by checking if a cutting move is approximately parallel (with a distance equal to the stepover) to the top cutting move in an existing stack. If such a stack is found, then the cutting move is pushed into this stack, else a new stack is created with this cutting move.

Checking whether two cutting moves are approximately parallel can be done by sampling the two polylines and comparing the closest distances at these sample points. The procedure is shown in Algorithm 7. The *FindStack* function (not shown here) searches through the existing stacks to find the stack whose top polyline is approximately parallel to the input region. If an appropriate stack is not found, the *FindStack* function returns -1 . Figure 6 (left) illustrates a typical scenario where the cutting moves can be grouped into a trochoid.

Before the complete toolpath can be composed, we must connect the curves in each trochoidal region using linking moves. This can be quickly done by connecting one curve's end point with the next curve's start point, as shown in Fig. 6 (left). Note that connecting the trochoidal curves using straight line segments will result in sharp corners at the connecting ends that must be smoothed later.

Algorithm 7: Grouping Trochoidal Regions

Input: $R_{tp} = [r_1, r_2, \dots, r_n]$ // regions in the toolpath

```

Function GroupTrochoids( $R_{tp}$ ):
   $S_{trochoid} = []$  // trochoidal stacks
  for  $r \in R_{tp}$  do
    if  $r.cutting$  then
       $s \leftarrow FindStack(r, S_{trochoid}, R_{tp})$ 
      if  $s = -1$  then
         $S_{trochoid} \leftarrow S_{trochoid} + []$ 
         $s \leftarrow length(S_{trochoid})$ 
      end
       $r.group \leftarrow s$ 
       $S_{trochoids}[s] \leftarrow S_{trochoid}[s] + r$ 
    end
  end
  return  $S_{trochoid}$ 

```

6.3 Grouping Boundary Regions

Before composing the final toolpath, we also need to group the overlapping non-cutting or boundary regions. The process is very similar to one for grouping the trochoidal regions. In this case, we need to check if one non-cutting region is a subset of a larger region, in which case these regions must be grouped together. Fig 6 (middle) illustrates which overlapping boundary moves must be grouped. The boundary moves are rendered as thick lines to indicate that they are overlapping. When traversing the segmented toolpath, the overlapping boundary or non-cutting moves are encountered in the order of increasing lengths. Therefore, these overlapping boundary moves can be grouped by checking if the endpoints of the top segment in the boundary stack lie on the current boundary segment.

6.4 Composing the Complete Toolpath

The algorithm for composing the complete toolpath starts by looping over the regions. The first time a region belonging to a trochoid group is encountered, the whole trochoid group is added to the toolpath. Similarly, the first time a region belonging to a boundary group is encountered, the first and smallest region from the boundary group is added to the toolpath. Once a trochoid or a boundary group has been added, further encounters with the groups are ignored. Lastly, the composed toolpath is traversed to check for gaps between the segments, which are patched with linear moves. The procedure is shown in Algorithm 8.

6.5 Patching Gaps

At this stage there will still be some gaps in the toolpath. This is because we use the smallest boundary moves from the boundary stacks, and rarely are they directly connected to the trochoid groups. These gaps can be closed using straight linking moves in most cases. Sometimes, however, these linking moves may gouge the pocket boundary. Therefore a gouge check is performed on the connecting moves by computing the smallest distance of each linking move with the pocket boundary. If a gouge is detected, we must find an alternate non-gouging rapid move. Currently, we use a boundary following move that connects these segments; however, more elaborate and optimal strategies can be used.

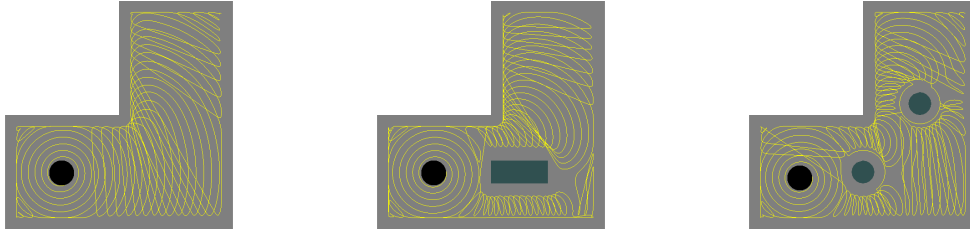


Figure 7: Smoothed toolpaths for variations of an L-shaped pocket with islands.

Algorithm 8: Composing the Complete Toolpath

Input: $T_{stack} = [t_1, t_2, \dots, t_m]$ // trochoid stacks
 $B_{stack} = [b_1, b_2, \dots, b_k]$ // boundary stacks
 $T_{region} = [T_1, T_2, \dots, T_m]$ // trochoid regions

Function $\text{Compose}(R_{tp}, T_{stack}, B_{stack}, T_{region})$:

```

 $C_{tp} \leftarrow []$  // composed toolpath
for  $r \in R_{tp}$  do
  if  $r.cutting$  then
    if  $\neg T_{stack}[r.group].added$  then
       $C_{tp} \leftarrow C_{tp} + T_{region}[r.group]$ 
       $T_{stack}[t.group].added = true$ 
    end
  else
    if  $\neg B_{stack}[r.group].added$  then
       $C_{tp} \leftarrow C_{tp} + B_{stack}[r.group].top$ 
       $B_{stack}[r.group].added = true$ 
    end
  end
end
return  $C_{tp}$ 

```

6.6 Smoothing Corners

When joining the trochoidal moves in a trochoid group and when joining the boundary moves with the trochoidal sections, the corners at the joins are unlikely to be smooth. Different methods can be used to smooth out these corners. Since our toolpath consists of small, approximately equal-length segments, a modified version of Chaikin's algorithm [7] can be used to smooth out the corners. The results of smoothing the toolpaths for the example L-shaped parts are shown in Fig. 7.

Smoothing the toolpaths requires merging all the cutting and the non-cutting regions. Maintaining the cutting/non-cutting status of the segments while smoothing can be complex and error-prone. A more straightforward approach is to recover the status of the segments and recreate the regions by running the toolpath through the simulation environment again and re-computing the engagement values for the toolpath at each segment.

The pixel engagement statistics and the angular deviation between adjacent toolpaths segments for the smoothed toolpath in Fig. 7 (left) are shown in Fig. 8. The horizontal axis represents the time step in the

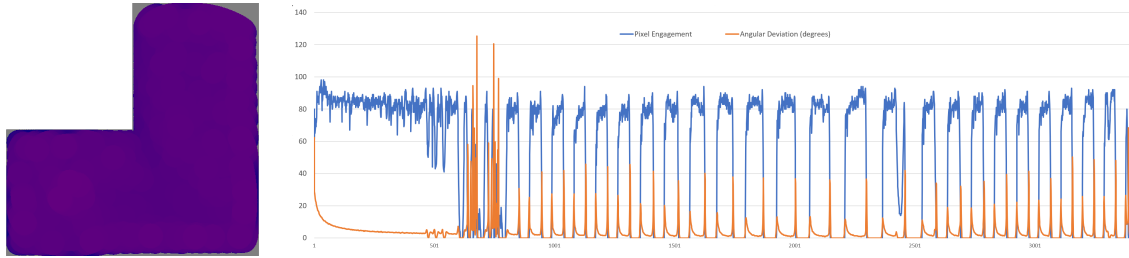


Figure 8: Pixel engagement (blue) and inter-segment angular deviation (orange) for the toolpath simulation result on an L-shaped pocket (left).

toolpath simulation, and the vertical axis indicates both the pixel engagement count and the angular deviation between consecutive toolpath segments in degrees, which are conveniently in the same range allowing us to overlay the two graphs. On the left of Fig. 8, the color-coded pixel engagement result from the cutting simulation is displayed, where one can see some material left along the boundary of the pocket due to toolpath smoothing.

The blue graph shows the pixel engagement value, and we can see that the engagement values are within a narrow range except where they fall to zero for the linking moves where no material is being cut. The angular deviation in degrees is plotted in orange and can be seen to be within 5-10 degrees in the cutting regions, except at the very start and in the two trochoidal regions near the start. The high angular deviation in the trochoidal regions is an artifact of the constant large individual steps in the toolpath. It is possible to reduce these sharp changes using an adaptive approach where the length of toolpath segments is reduced in the neighborhood where the linking moves connect with the cutting segments resulting in a smoother toolpath.

7 CONCLUSIONS

Toolpath generation algorithms are usually very complex and can take a very long time to develop into a form that can be used with production software. The proposed simulation-based method requires very little code to implement and minimal heuristics. Despite this, the method can handle complex pocket geometries and works for all pocket shapes. The method can also be extended easily to handle multiple plunge holes for more optimal toolpaths.

Unfortunately, at this time, a simulation-based approach like this is still computationally too expensive for real-time applications. It is also difficult to parallelize a simulation-based approach since only a single simulation environment is typically available. Maintaining multiple simulation environments and synchronizing them maybe even more expensive. More research is required to find a method to parallelize the simulation process. The brute-force exploration for the best tool movement direction is the most time-consuming part of the algorithm, and this can be considerably sped up using a learning algorithm.

ACKNOWLEDGEMENTS

This research was conducted as part of the CAMWorks project. CAMWorks is a popular CAM software used by small and mid-sized machining workshops worldwide. We want to thank everyone in the CAMWorks team, past and present, who made this research possible.

Tathagata Chakraborty, <http://orcid.org/0000-0002-2752-2533>

Chinmaya Panda, <http://orcid.org/0009-0004-6096-9582>

Nitin Umap, <http://orcid.org/0000-0002-9063-1230>

REFERENCES

- [1] Abdullah, H.; Ramli, R.; Wahab, D.: Tool path length optimisation of contour parallel milling based on modified ant colony optimisation. *The International Journal of Advanced Manufacturing Technology*, 92(1), 1263–1276, 2017. <http://doi.org/10.1007/s00170-017-0193-5>.
- [2] Abrahamsen, M.: Spiral tool paths for high-speed machining of 2d pockets with or without islands. *Journal of Computational Design and Engineering*, 6(1), 105–117, 2019. <http://doi.org/10.1016/j.jcde.2018.01.003>.
- [3] Banerjee, A.; Feng, H.Y.; Bordatchev, E.V.: Process planning for floor machining of 2D/2d pockets based on a morphed spiral tool path pattern. *Computers & Industrial Engineering*, 63(4), 971–979, 2012. <http://doi.org/10.1016/j.cie.2012.06.008>.
- [4] Bieterman, M.B.; Sandstrom, D.R.: A curvilinear tool-path method for pocket machining. *J. Manuf. Sci. Eng.*, 125(4), 709–715, 2003. <http://doi.org/10.1115/1.1596579>.
- [5] Blum, H.: A transformation for extracting new descriptions of shape. *Models for the perception of speech and visual form*, 362–380, 1967.
- [6] Bouard, M.; Pateloup, V.; Armand, P.: Pocketing toolpath computation using an optimization method. *Computer-Aided Design*, 43(9), 1099–1109, 2011. <http://doi.org/10.1016/j.cad.2011.05.008>.
- [7] Chaikin, G.M.: An algorithm for high-speed curve generation. *Computer graphics and image processing*, 3(4), 346–349, 1974.
- [8] Dumitrache, A.; Borangiu, T.; Dogar, A.: Automatic generation of milling toolpaths with tool engagement control for complex part geometry. *IFAC Proceedings Volumes*, 43(4), 252–257, 2010. <http://doi.org/10.3182/20100701-2-PT-4011.00044>.
- [9] Hatna, A.; Grieve, R.; Broomhead, P.: Automatic cnc milling of pockets: geometric and technological issues. *Computer Integrated Manufacturing Systems*, 11(4), 309–330, 1998. [http://doi.org/10.1016/S0951-5240\(98\)00030-5](http://doi.org/10.1016/S0951-5240(98)00030-5).
- [10] Held, M.; Spielberg, C.: A smooth spiral tool path for high speed machining of 2d pockets. *Computer-Aided Design*, 41(7), 539–550, 2009. <http://doi.org/10.1016/j.cad.2009.04.002>.
- [11] Huang, N.; Lynn, R.; Kurfess, T.: Aggressive spiral toolpaths for pocket machining based on medial axis transformation. *Journal of Manufacturing Science and Engineering*, 139(5), 2017. <http://doi.org/10.1115/1.4035720>.
- [12] Jacso, A.; Matyasi, G.; Szalay, T.: The fast constant engagement offsetting method for generating milling tool paths. *The International Journal of Advanced Manufacturing Technology*, 103(9), 4293–4305, 2019. <http://doi.org/10.1007/s00170-019-03834-8>.
- [13] Jacso, A.; Szalay, T.: Optimizing the numerical algorithm in fast constant engagement offsetting method for generating 2.5 d milling tool paths. *The International Journal of Advanced Manufacturing Technology*, 108(7), 2285–2300, 2020. <http://doi.org/10.1007/s00170-020-05452-1>.
- [14] Jacso, A.; Szalay, T.; Jauregui, J.C.; Resendiz, J.R.: A discrete simulation-based algorithm for the technological investigation of 2.5 d milling operations. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 233(1), 78–90, 2019. <http://doi.org/10.1177/0954406218757267>.
- [15] Li, H.; Dong, Z.; Vickers, G.W.: Optimal toolpath pattern identification for single island, sculptured part rough machining using fuzzy pattern analysis. *Computer-Aided Design*, 26(11), 787–795, 1994. [http://doi.org/10.1016/0010-4485\(94\)90092-2](http://doi.org/10.1016/0010-4485(94)90092-2).
- [16] Romero-Carrillo, P.; Torres-Jimenez, E.; Dorado, R.; Díaz-Garrido, F.: Analytic construction and analysis of spiral pocketing via linear morphing. *Computer-Aided Design*, 69, 1–10, 2015. <http://doi.org/10.1016/j.cad.2015.07.008>.

- [17] Sanchis, R.; García-Perales, Ó.; Fraile, F.; Poler, R.: Low-code as enabler of digital transformation in manufacturing industry. *Applied Sciences*, 10(1), 12, 2019. <http://doi.org/10.3390/app10010012>.
- [18] Shalf, J.: The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A*, 378(2166), 20190061, 2020. <http://doi.org/10.1098/rsta.2019.0061>.
- [19] Stori, J.; Wright, P.: Constant engagement tool path generation for convex geometries. *Journal of Manufacturing Systems*, 19(3), 172–184, 2000. [http://doi.org/10.1016/S0278-6125\(00\)80010-2](http://doi.org/10.1016/S0278-6125(00)80010-2).
- [20] Theis, T.N.; Wong, H.S.P.: The end of moore's law: A new beginning for information technology. *Computing in Science & Engineering*, 19(2), 41–50, 2017. <http://doi.org/10.1109/MCSE.2017.29>.
- [21] Wang, H.; Stori, J.A.: A metric-based approach to 2d tool-path optimization for high-speed machining. In *ASME International Mechanical Engineering Congress and Exposition*, vol. 3641, 139–148, 2002. <http://doi.org/10.1115/IMECE2002-33610>.
- [22] Xiong, Z.; Zhuang, C.; Ding, H.: Curvilinear tool path generation for pocket machining. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 225(4), 483–495, 2011. <http://doi.org/10.1177/2041297510394085>.
- [23] Xu, J.; Sun, Y.; Zhang, X.: A mapping-based spiral cutting strategy for pocket machining. *The International Journal of Advanced Manufacturing Technology*, 67(9), 2489–2500, 2013. <http://doi.org/10.1007/s00170-012-4666-2>.
- [24] Xu, K.; Li, Y.; Xiang, B.: Image processing-based contour parallel tool path optimization for arbitrary pocket shape. *The International Journal of Advanced Manufacturing Technology*, 102(5), 1091–1105, 2019. <http://doi.org/10.1007/s00170-018-3016-4>.