www.cad-journal.net
Computer-Aided Design AND APPLICATIONS™

A GPU Method and Error Analysis of Multi-Point Positioning of a Toroidal Tool for 5-Axis Machining

Mukhmeet Singh[1], Sanjeev Bedi[2], Stephen Mann[3]

[1]University of Waterloo, mukmit.singh@gmail.com
[2]University of Waterloo, sbedi@uwaterloo.ca
[3]University of Waterloo, smann@uwaterloo.ca

Corresponding author: Stephen Mann, smann@uwaterloo.ca

Abstract. Advantage in five-axis machines is realised by placing a cutting tool's surface in close proximity to the part's surface. This paper presents a procedural method that utilizes the GPU to position a toroidal tool on a curved surface. The proposed method is independent of the surface definition and has a fast computational speed. An accuracy assessment of the method is also presented. The accuracy measure shows that recursive use of a small GPU window can lead to faster computation of an accurate toolpath, and shows that the discretization to pixels used in our GPU method only results in negligible gouging. Furthermore, an example of using the accuracy measure to determine the discretization of the tool and the part surface is also given.

Keywords: CNC Machining, 5-Axis Machining, Tool Positioning, GPU, Error Analysis
DOI: https://doi.org/10.14733/cadaps.2024.819-833

1 INTRODUCTION

Machining curved surfaces, in the die and mold industry, is often done using 5-axis machines. Where 3-axis machines can move the tool or the work piece linearly in the x, y, and z directions, 5-axis machines have two additional degree of freedom that allow the tool to rotate and tilt relative to the work piece [15]. These additional degrees of freedom in 5-axis machines enables the machinist to position the tool to match the tool's shape to the surface being machined. Placing the cutting surface in proximity of the modeled surface produces wider strips, allowing surface tolerance criteria established by the user to be met. The increased width improves the material removal rate (MRR) and decreases the machining time during finish machining [15].

The machining time and MRR also depend upon the type of tool used for machining. There are various types of tools used for machining curved surfaces. The most common types are the ball nose end mill, flat end mill, and toroidal cutter or bull-nosed end mill. A toroidal cutter has circular cutting inserts that emulate

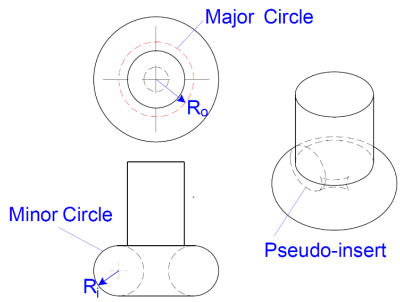


Figure 1: Geometry of toroidal cutter used for toolpath generation for 5 Axis machining.

the shape of the torus as the tool rotates about its axis. A toroidal cutter is defined by its, major and minor radius, R_O and R_i. R_i is the radius of the circular insert that revolves around the axis of revolution at a distance of R_O to form a tori. The circle that revolves around the axis of revolution emulates the physical insert on actual tool and hence is called a *pseudo insert*. The line through the center of the pseudo insert circle is the *pseudo insert axis*. A toroidal tool is shown in Figure 1.

The cutting tool is placed and moved over the surface (workpiece) to produce a gouge-free surface, i.e., it should not overcut the workpiece. The movement of the tool over the work piece is guided by the footprint of the toolpath. The footprint of the toolpath is the projection of tool positions on a convenient plane perpendicular to the tool axis. The footprint follows a certain pattern. The zigzag, xy parallel and contours footprints have commonly been used in literature. This paper uses the zigzag pattern that is defined by parameters sidestep and forward step [6]. A zigzag footprint is shown in Figure 2.

The placement of the tool over the surface for 5-axis machining is challenging as it must be placed to avoid gouging or overcutting. The tool and modeled surface involve geometry that results in complex simultaneous transcendental equations that must be solved to get gouge-free placement. In Multipoint machining methods (MPM) the toroidal tool touches the modeled surface (to be machined) at multiple points of contact. Positioning the tool to have multiple points of contact places the tool in closer proximity to the design surface than a tool positioned with just a single point of contact. This allows for a larger side step and gives better surface finish in shorter machining time.

There are number of methods to find MPM toolpaths [4, 1, 2, 3, 5, 6]. All these methods involve numerical methods and use the CPU to solve the non-linear transcendental-polynomial equations. These methods of obtaining the solutions for positioning the tool over the surface are time-consuming [1]. In addition, the solution results in multiple answers that satisfy the equations, but some of them gouge the surface (at other points on the tool) and necessitate the use of a gouge checker to isolate the gouge-free solution.

Warkentin et al. [15] introduced the concept of multipoint machining (MPM) using an optimization algorithm but the method was numerically sensitive and required a long time to converge to a solution. It was clear that unless the method could be simplified it would have little appeal for use on the shop floor.

Over the years the authors have tried to simplify the tool positioning strategy. The methods are demonstrated by machining a variety of surfaces to demonstrate robustness and reliability. However, these numerical methods rely on tolerances (usually set by the user) and an initial guess at the solutions. In some cases, a poor selection of tolerances or a poor choice of initial guess can lead to non-convergence in the numerical search or poor solutions (exceeding a maximum number of iterations for convergence) [3]. Duvedi et al. give an example where 60,000 initial guesses were required before a solution could be obtained [3]. In either case the user must modify the tolerances and initial guesses to get a better solution. This requires understanding numerical methods and tool positioning strategy, which are unrealistic expectations on the shop floor. In

addition, erroneous or missed tool positions may lead to damage of the part due to overcutting or worse. Furthermore, the produced toolpaths may require the added step of verification in a simulation package, consuming more time and effort. Although overcutting can be mended with welding, thermal destressing and machining, overcutting is expensive and can delay the project. Thus, it is best if a tool positioning method avoids the use of numerical techniques that require providing tolerances and initial guesses.

Various approaches were taken to simplify and improve the method [4, 1, 2, 3, 5, 6, 7]. Voxels-based models have been used in conjunction with the Graphics Processing Units (GPU) to generate 3-axis tool paths [9, 13]. The methods can rapidly generate toolpaths for 3-axis machining. GPU's have also been used to reduce the error in flank milling [8]. In addition to flank milling on 5-axis machines, GPU's have also been used to assist in 5-axis surface machining [7]. Others have also used the speed of GPU's to compute material removed as the tool rotates and to estimate the cutting forces encountered by the tool [12].

In this paper we build on the work of Duvedi et al. [2], who presented a method that only used a version of the ray firing based gouge checking to do both the drop and tilt steps in DTM. In their work they fired the rays from the surface towards the tool, in a transformed coordinate system, where the tool was centred at the origin and could only move along the Z-axis. In this configuration the rays intersect the toroidal tool at zero, one or two points and these cases can be easily identified using close-form equations. The method is, however, executed hundreds of times and could be time consuming.

The method proposed in this paper is based on a physical model and does not require the solution of simultaneous equations or initial guesses. The new method emulates the tool dropping on a surface [4]. The drop distance is calculated using a GPU based algorithm. The method presented here evolved from an earlier work, where the earlier method was used for gouge checking (a minor role in tool positioning). This gouge-checking method is modified to speed up its computational efficiency and to produce 3-axis tool positions. In addition, the GPU-based tool drop method is repeatedly used with a rotation algorithm to obtain multiple contacts between the tool and the design surface and produce gouge-free 5-axis tool positions.

Many of these tool positioning methods use a triangular approximation for surface and tool. These approximations impact the tool position relative to the surface. The paper studies the impact approximation can have and identifies limits to the approximations. These limits can be used as guides for setting the accuracy of triangulation for curved surfaces.

2 GPU BACKGROUND

The Graphics processing unit (GPU) is a specialized hardware component used to accelerate graphics rendering. The GPU is used to manipulate graphic elements and is more efficient than the CPU (Central Processing Unit) in processing huge blocks of data in parallel.

In the simplest GPU operation, the objects (typically triangles) are defined in an internal global reference system representing the modelled world. As the modelling world is infinite, a cuboid, called the view volume, is created that limits the represented world. The top face of the cuboid is called the near plane and the bottom face is called the far plane. All objects or partial objects within the view volume are processed for visualization. All objects outside the view volume are ignored. To view the space within the view volume an eye or observer is defined. The view volume as seen from the eye position is displayed on the view plane; location of the view plane is arbitrary but in this work, for simplicity, the view plane is placed at the near plane. There are two type of images that can be created on the view-plane, perspective and orthographic. For this work an orthographic view of the world as seen from the eye is created on the view-plane. The eye position and the normal to the view plane determine a location and a direction.

Although not implemented this way on the GPU, in essence several uniformly spaced rays, around the eye, are defined. These rays are perpendicular to the view-plane and originate in it. The originating points represent the pixels of the view screen. The rays go into the view volume and intersect with the objects in it. The parameters of the first point of intersection of the ray are stored in a buffer or map. The intersection of

all the rays creates a uniformly discretized depth map. The discretization maps to the pixels on the computer screen. The GPU stores all the depth information in a buffer known as the *z-buffer*. The color and texture are stored in their own buffers. For visualization color and texture are associated with the pixels. The view displayed shows all the visible objects in color and with appropriate texture and other properties. With depth test enabled, the GPU can compare the depth buffer (z-buffer) of two separate views. The GPU compares two buffers using parallel computation and can generate a composite scene with appropriate fragments in front of the other fragments. This composition is done automatically by the rendering engine. We used the ability to store and compare z-buffers of the objects rendering with a GPU, using OpenGL, for gouge-free tool positioning for machining curved surfaces.

3 GOUGE CHECKING

The method used for gouge checking is called by many names including z-map and mow-the-grass method [11]. Vertical rays at the user-specified spacing (possibly uniform), parallel to the tool axis, are fired from a plane below the surface being machined. The rays are trimmed where they intersect the surface. The tool is positioned above the trimmed rays and the rays are checked for intersection with the tool. The intersections identify gouges. In addition, the method with some modifications can also be used to study surface finish or to estimate the height of scallops left in machining.

4 CONCEPT

The view volume in the GPU [10] and the z-map method share similarities, see Figure 2. In both cases rays (simple linear equations) are fired from a base plane in a known direction. In both cases the rays are trimmed against objects. Both methods allow for a precise estimation or visualization. The gouge-checking method is like the orthographic projection method used by the GPU to display graphics on a computer screen. These similarities between the two present an opportunity to use the GPU for gouge checking. This paper exploits the similarity and uses the ability of the GPU to do fast computations to facilitate MPM tool positioning strategies. Furthermore, the gouge-checking method is modified into a method for estimating tool drop distance for generating gouge-free 3-axis tool positions. The paper further presents a method that builds on the ability to rapidly calculate the tool drop distance using the GPU to generate a gouge-free 5-axis tool position based on the Drop and Tilt method for Multi-Point Machining [4, 1, 2].

The method is independent of the kind of surface used to describe the part, as a surface must be triangulated for use in OpenGL. As CAD packages provide tools for triangulation of many surfaces, this method works on all those surfaces. For a demonstration of the concept, cubic Bézier surfaces are used in this work. A concave sample surface is shown in Figure 3.

5 IMPLEMENTATION

In the MPM method [2], positioning the tool is done in two steps. In the first step, the first point of contact is obtained by dropping a tool over the modeled surface from a certain height atop of the surface. The drop distance for the calculation of the gouge-free tool position is computed using the GPU. The rendering requires the definition of the view volume and the view window. Since the goal is accurate tool positioning, the view direction is set along the tool axis and the view volume is set to enclose the tool. The view window is limited only by the size of the computer screen and is commonly left to the user to define; in this paper, we give some guidance on setting the size of the view window to achieve a desired error tolerance in the simulation.

The window size impacts the time taken to compute the tool positions and the accuracy of the tool position. The tool is rendered using Orthographic projection, and the resulting z-buffer is saved; a second rendering with the same view direction and view volume is done for the surface, and again, the resulting z-buffer is saved. These two z-buffers are compared with each other and the minimum difference in z-value

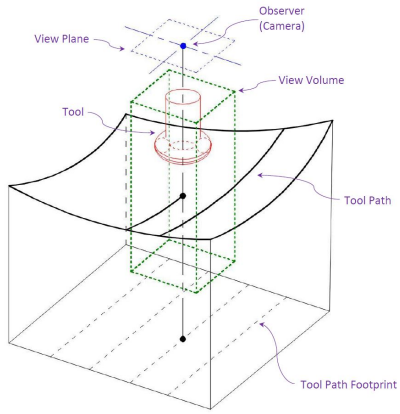


Figure 2: The tool, enclosed in a view volume, is dropped onto the surface beneath it. The camera is placed above the view volume and the orthographic projection is used to speed up computations in a GPU.

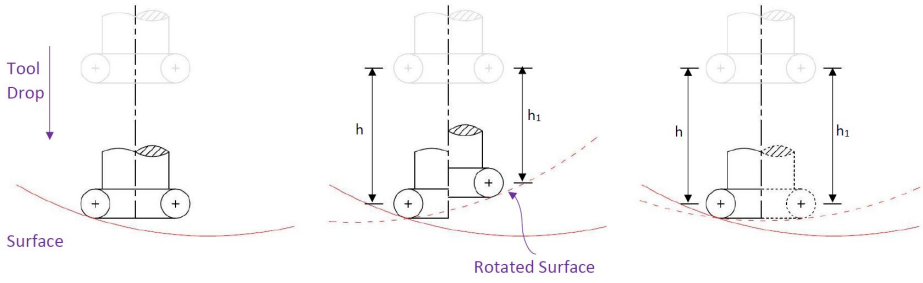


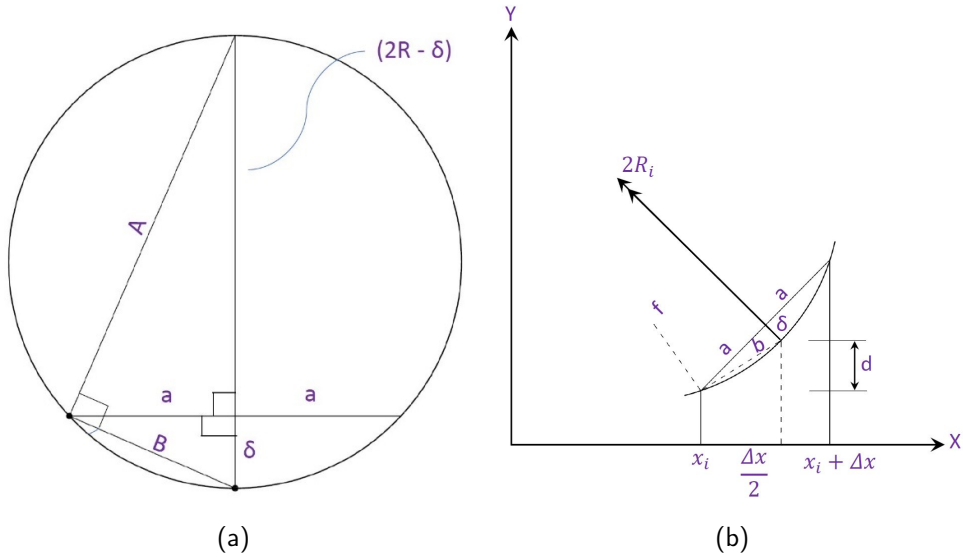
Figure 3: Tool Drop used for computing tilt angle.

or depth value gives the drop distance with which the tool is moved towards the surface to touch it without any gouging. This drop distance concept is fast to compute, and its accuracy can be determined based on discretization and world modeling parameters. After an initial drop to obtain the first point of contact, the center and axis of the pseudo insert at that point of contact are obtained.

The process is illustrated in Figure 3. Figure 3(a) shows the tool dropping on the surface to determine the first point of contact at a height h. In Figure 3(b) the surface is rotated about the pseudo insert axis, and the tool is dropped onto this rotated surface at a height h_1. The surface is rotated iteratively until a rotation angle, theta, is found where $h_1 = h$. Rotating the surface by angle, θ, results in two points of contact as shown in Figure 3(c). The surface rotation about the pseudo-insert ensures the toroidal tool axis stays aligned with the vertical. The details are given in the Appendix. In this method tool drop with the z-buffer method is used repeatedly to find both points of contact.

6 ERROR DUE TO PIXELIZATION

The speed of the GPU-based method depends on the number of pixels in the window. A larger number of pixels results in a larger time taken by the algorithm. However, as the pixels in the window increase the accuracy



(a) (b)

Figure 4: An insert and the rays fired towards. The discretization results in an approximation that can be evaluated.

of representation and the tool positioning also increases. The accuracy of tool positioning and the size of the window are inversely related. The computation time for one tool position using a 625×625 window is 408 milliseconds and is about 10 times larger than the computation time for a 50×50 window. The relationship between error and window size is studied in this paper.

In the z-buffer method described above, finely spaced rays are fired to determine the tool positions. The discretization of the rays introduces some errors. To estimate this error, we approximate the circular insert of the toroidal tool by linear segments connected to one another. The distance between two rays along x-axis is

$$\Delta_x = \frac{width_w}{width}$$

where $width_w$ is the width of the viewport in the world coordinate system and $width$ is the number of pixels in the viewport width. The largest deviation between the circle and the approximating segment is along the side of the torus Figure 4 shows the insert with a reference frame at the base of the insert or circle. Two neighboring rays at x_j and x_{j+1} are shown in the figure. The approximating line deviates from the circular insert. From right triangles in Figure 4 we can show

$$\delta = \frac{a^2 + \delta^2}{2R}$$

and as $\delta \ll 1$, we can drop δ^2 and still obtain a tight bound on δ:

$$\delta < \frac{a^2}{2R_i},$$

where a is half the length of the line joining the intersection of the ray with the insert circle. Knowing the equation of the circle, the length of the line segment between the intersection of two neighboring rays that

are Δ_x apart, represented by $2a$, can be calculated. The ray that is fired at $x = x_j$ and at $x = x_{j+1}$ intersect the circle at y_i and at y_{i+1}. Based on Figure 4 the intersection of the two rays is given by

$$\begin{aligned}(R_i - y_j)^2 &= R_i^2 - x_j^2 \\ (R_i - y_{j-1})^2 &= R_i^2 - (x_j - \Delta_x)^2\end{aligned}$$

and

$$\Delta y = y_j - y_{j+1} = \left(R_i - \sqrt{R_i^2 - x_j^2}\right) - \left(R_i - \sqrt{R_i^2 - (x_j - \Delta x)^2}\right). \tag{1}$$

In this equation Δx is constant. Δy is small at the bottom of the circular insert and is the largest at the side of the insert at $x = R_i$ where it is given by

$$\Delta y = \sqrt{2R_i\Delta x - \Delta x^2}.$$

Also, we have $(2a)^2 = (\Delta x^2 + \Delta y^2)$. Substituting these values in our approximation for δ, $\delta < \frac{a^2}{2R_i}$, gives:

$$\delta = \frac{\Delta x^2 + \Delta y^2}{8R_i} = \frac{\Delta x^2 + \left(\left(R_i - \sqrt{R_i^2 - x_j^2}\right) - \left(R_i - \sqrt{R_i^2 - (x_j - \Delta x)^2}\right)\right)^2}{8R_i}. \tag{2}$$

And at $x = R_i$, it gives

$$\delta = \frac{2R_i\Delta x}{8R_i} = \frac{\Delta x}{4}.$$

Although δ is strictly less than $\frac{\Delta x}{4}$, note that along the side of the insert this is a tight bound since in the above equations we have dropped δ^2 which is small compared to $\frac{\Delta x}{4}$.

The approximation introduced by linearization is approximately a quarter of the spacing between two rays in the worst case. Having more rays can improve the accuracy but having an accuracy better than the typical CNC machine unnecessarily increases the computation time. The accuracy of a CNC machine typically found in industry is about 0.01 mm. For our tool, $R_O = 6.7$ mm and $R_i = 6$ mm. The width and height of the view volume is taken to be 25 mm$\approx 2(R_O + R_i)$. To obtain an accuracy of $\delta < 0.01$ mm, implies that $\Delta x = 0.4$ mm. So for a view window of 25 mm we need 250.04=625 pixels; i.e., we need a view window with width and height of 625, thus meeting the accuracy requirements of industrial CNC machines.

7 MULTI-LEVEL RENDERING

The chosen window size is large enough to produce accurate tool positions that are within user-specified tolerance, but the computation time is slower than an equivalent CPU implementation [14]. The speed issue of the GPU method is due to slow transfer rates from the GPU to the CPU. To tackle this problem, a Multi-Level Rendering approach is used.

In the Multi-level rendering approach, the torus is rendered in two stages with lower window resolutions. In the first stage, the orthographic projection is kept the same, as explained in the previous section, except the window size is reduced. Since the width and height of the view volume is 25mm, a discretization of 50×50 results in a spacing of $\Delta x' = 0.5$ mm between two pixels. The above procedure is followed to obtain the drop distance and associated pixel position. Next, the discretization is locally refined around this approximate point of contact to improve the accuracy of the point of contact.

The approximate contact point is surrounded by other points in the grid; however, the algorithm ensures that the drop distance at these points is higher than at the approximate contact point. As the curvature of the torus is greater than the curvature of the surface, the lowest value lies in the region surrounded by the neighboring pixels. So, both the torus and the surface are rendered again with a zoomed view of the

	GPU		CPU	
Tool Positions	650×650 SL	50×50 ML	650×650 SL	50×50 ML
760	616(s)	8.5(s)	304(s)	14(s)

Table 1: Time comparison for single and multi-level GPU method.

surrounding area that focuses on a 3×3 area around the contact point. In this second stage of rendering, the camera is moved to the estimated point of contact, but the view vector is retained. The '*ToolDropMethod*' is re-applied for the newly rendered sections of torus and surface.

The zoomed-in view focuses on a (3×3) grid, whose parameters are obtained by redefining the orthographic projection with new viewport $width_w$ and $height_w$ for a second level of rendering. These values are

$$width_w = height_w = 2 \times \Delta x'.$$

With a view port of 50×50 pixels the spacing between rays is $\Delta x'' = \frac{width_w}{width} = \frac{2\times\Delta x'}{50} = \frac{2\times 0.5}{50} = 0.02$ mm.

A spacing of 0.02 mm for the rays gives an accuracy of 0.005 mm which is better than what can be machined on a typical CNC machine. This multi-level accuracy is equal to a single-level accuracy obtained from a view widow of 1250×1250. With the multi-level method, we reduce the view window size, but this two-stage process requires the drop distance to be calculated twice. The details are given in the Appendix.

Table 1 gives a comparison of times for single-level (SL) and multi-level (ML) versions of both CPU and GPU versions of the method. The comparison uses a concave surface that was used in previous work [15] [16] [17]. The proposed algorithm was used to generate a tool path for machining this concave surface. The tool positions generated by the proposed algorithm are numerically equal to those generated by earlier methods. We refer the reader to the earlier papers for machining example and accuracy analysis of the machined part.

Although the single-level version of the CPU method is faster than the single-level version of the GPU method, the multi-level version of the GPU method is faster than the multi-level version of the CPU method. The additional performance gains of the GPU method can be attributed to transferring less data from the GPU back to the CPU.

8 DISCUSSION

In this work the tool and the part surface are approximated by planar facets. The triangulation simplifies the mathematical formulation but requires it to be executed many times. Accuracy requires fine triangulation and computational efficiency favours coarse triangulation. This is an ideal optimization problem in which user-acceptable error can be used to determine triangular accuracy. Facet approximation introduces errors in the tool position. The work in the previous section bounds the error due to the planar approximation of the toroidal tool. The impact of the error due to approximation of the torus is further discussed here. For this discussion, we use as an example the tool we used for machining in earlier papers. This tool, with $R_i = 6$ mm and $R_O = 6.7$ mm, is illustrated in Figure 1; these values of R_i and R_O are also used in the plots of δ in Figure 6.

Equation (2) bounds the error δ as being strictly less than $\frac{\Delta x}{4}$. However, this is a worst-case error occurring when the point of contact is at the edge of the tool (at the point labeled 6.0 in Figure 5, for example). When the point of contact is at other locations on the tool, the error will be less. In Figure 6(a), to illustrate the error, x_j and Δx are plotted as continuous values. As x_j moves away from the center, the value of δ increases, with the error remaining below 0.01 until the point of contact is very close (more than 95% of the way from the center to the edge) to the edge of the tool. To better illustrate the error in the region not as close to the edge, in Figure 6(b) we have restricted the plot of the error to $x_j < 5.5$.

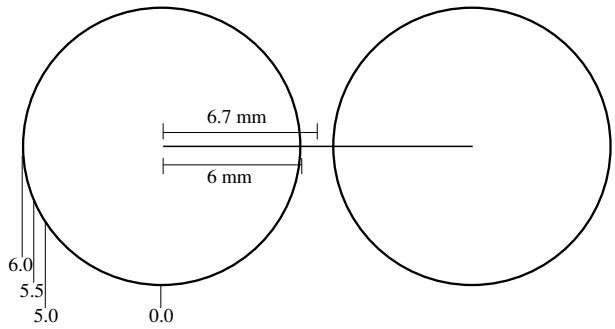


Figure 5: The key points between the tool bottom and tool side on a toroidal tool.

Experience shows that in MPM, the contact points lie close to the bottom of the tool, and the tool tilt is limited to within 20 degrees of the normal and often much less: for the surfaces machined in [15], the maximum angle was 6.5 degrees. The error in a 20-degree region around the normal is smaller than the accuracy of a typical CNC machine used in industry and thus cannot be measured. The GPU-based method can be used to ensure $\Delta X \ll 0.1$, and the ensuing linear approximation will introduce errors that are smaller than those introduced by a typical CNC machine. Thus, the method presented here can be reliably used to machine parts if the first contact point does not lie on the side of the insert. This condition is not very restrictive when machining surfaces but must be kept in mind when creating a general solution to tool path planning.

Regardless, even using the $\frac{\Delta x}{4}$ bound for the error, a pixel size of 0.04 mm (as used in our earlier example) is required to accurately position the tool on the surface. What our bound on the error shows is that using a pixel size this small will result in gouging of less than 0.01 mm, and thus the amount of gouging due to the pixelization in the process is well below machining tolerances. Additionally, in the next section, we use this $\frac{\Delta x}{4}$ on the error to provide guidelines on how finely to tessellate the tool and the design surface.

9 DISCRETIZATION OF THE TOOL AND DESIGN SURFACE

In OpenGL, the torus representing the tool and the surface are both discretized into triangles before rendering. The triangulation of the tool and the surface introduce an error, ε_t and ε_s respectively. The impact of these approximations is studied next.

The surface finish on the machined part depends on the discretization of the tool and surface. The surface finish, in the most conservative case, is given by $\epsilon_{user} = \delta_{tool} + \delta_{surf}$ and is plotted in Figure 8. The contours show all possible values of δX_{tool} and ΔX_{Surf}. The desired surface finish can be controlled by the number of triangles in the triangulation.

Figure 7 is modeled by dividing the circular circumference of the sphere into $\pi\sqrt{\frac{R}{2\delta_{Sphere}}}$ equal segments of arc length $2a$. The sphere is divided into a same number of segments in parallel planes. The corresponding points are connected to create trapezoidal faces that are divided into two triangles each. The upper estimate of the total number of triangular faces is given by $2\pi\sqrt{R/(2\delta_{Sphere})}\pi/4\sqrt{R/(2\delta_{Sphere})} = (\pi^2 R)/(4\delta_{Sphere})$.

The number of tool positions is proportional to the number of triangles in the surface and thus proportional to the computational time required for building the toolpath. The number of triangles in the tool surface impact the time taken to compute one tool position. Thus, ideally minimum number of triangles should be used to model the surface and the tool.

The number of triangles can be chosen to minimize the computation time for the tool path. The tool path is comprised of several tool positions. The number of tool positions are a function of the feed forward and side

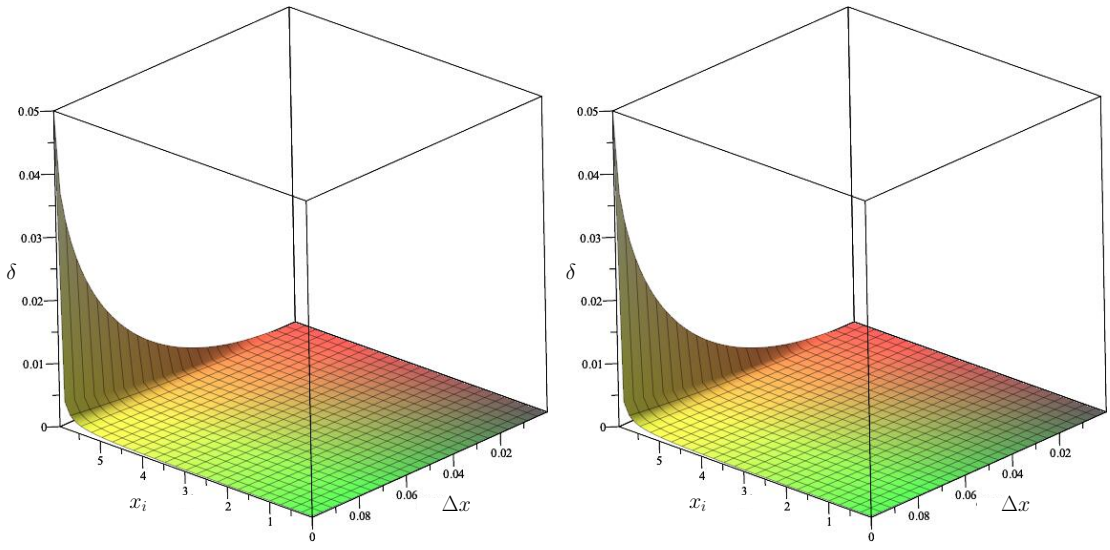


Figure 6: Error estimation in discrete approximation of toroidal tool as a function of location between tool bottom and tool side and discretization spacing.

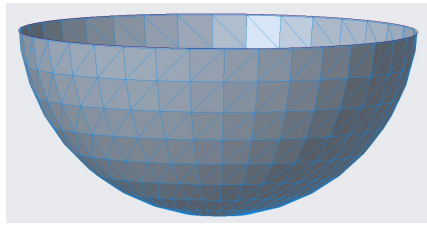


Figure 7: Triangular approximation of the cutting surface of a spherical end milling tool.

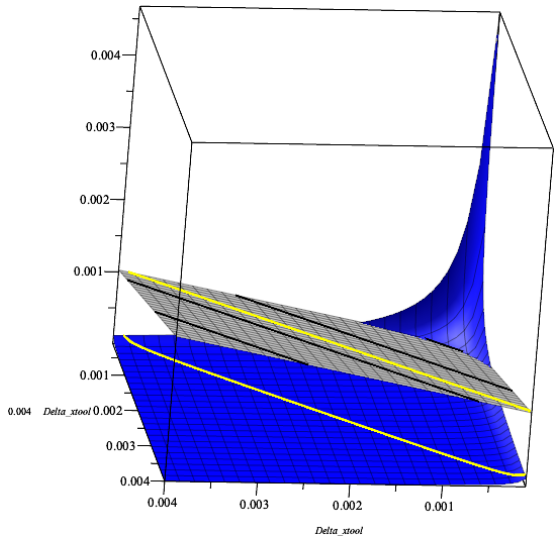


Figure 8: Error (Gray) and computation time (Blue) as a function of point spacing. Yellow curve shows the computation time for user acceptable error of 0.001 inch.

steps. The feed forward and side steps are related to the spacing of the rays, Δx_{Surf}, that are fired to model the surface and the tool. The larger the number of triangles, the higher the accuracy required in the tool path, and the feed-forward step needs to be small, just like Δx_{Surf}. The time required to compute one tool position is proportional to the spacing of the rays, Δx_{Tool}, fired in the GPU or the tool triangulation (number of triangles to represent the tool). The time for computing a tool position is also directly proportional to the number of tool positions in the tool path, thus the time is directly proportional to the surface triangulation (number of triangles to represent the surface). So, the time to compute the tool path comprising of a number of tool positions is given by

$$\begin{aligned} T_{ToolPath} &\propto N_{Surf} N_{Tool} \\ T_{ToolPath} &= C_{TP}(\pi^2 R_{Surf})/(4\delta_{Surf})(\pi^2 R_{tool})/(4\delta_{Tool}), \end{aligned}$$

where R_{Surf} is the average radius of curvature of the surface, R_{Tool} is the average radius of curvature of the Tool, N_{Surf} is the number of triangles in the surface model, N_{Tool} is the number of triangles in the tool model, δ_{Surf} is the sagittal error in the surface model, δ_{Tool} is the sagittal error in the tool model and C_{TP} is a constant of proportionality. For a spherical surface with $R_{Surf} = 200$, a tool with $R_{Tool} = 12$, and a user specified tolerance of $\epsilon_{user} = 0.001$, $T_{ToolPath}$ is plotted in Figure 8.

Figure 8 shows that for a user specified error a variety of combinations, such as the yellow line on the gray plane, are possible. The plane and line are shown clearly in Figure 9. For each of the combinations the computation time is estimated and is marked with the yellow curve on the blue surface. It is a flat U-shaped curve, indicating that there is a wide variety of Δx_{Tool} and Δx_{Surf} spacing that can be used without loss of surface finish or increase in machining time. Figure 8 will change as surface curvature and tool size change, but the shape will remain the same.

The error could be improved by having non-uniform triangulation, which places more triangles in high curvature zones, giving the same surface approximation for fewer triangles as a uniformly triangulated surface. However, for this work uniform triangulation is used. The optimal triangulation would correspond to the

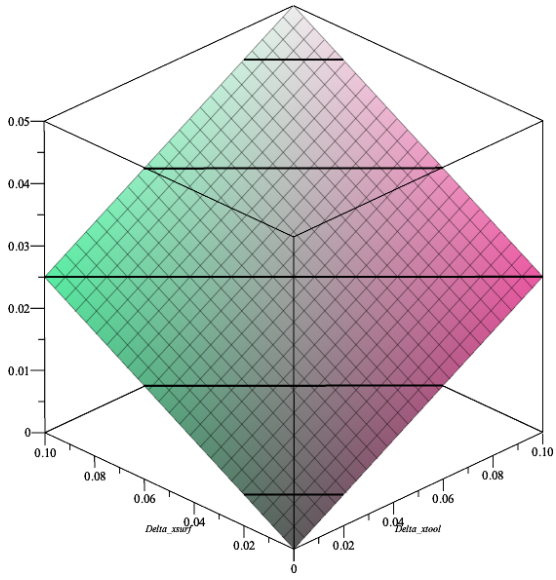


Figure 9: Surface finish as a function of tool and surface discretization. The contours show the possible values of discretization for a desired surface finish.

accuracy of the CNC machine. In this work the machine accuracy is taken to be 0.025mm or 0.001 inch. With 0.025mm the length of the line approximating the arc can be calculated as $2a = 2\sqrt{\delta 2R_i} = 1$mm. A 1mm long arc along the circle of radius R_i subtends an angle of approximately 10 degrees at the center. Thus, in this work the insert circle is discretized into 36 sectors.

In the other direction, the largest circle has a radius of $R_O + R_i$, and this circle must be represented with 1mm segments as well. An 1mm long arc along the circle subtends an angle of approximately 4.8 degrees at the center. Thus, in this work, the circle along which the insert rotates is discretized into 80 segments. Fewer segments will increase the error, and more segments will increase the processing time.

The error in surface representation with triangles depends on the radius of curvature of the surface. In this work, the radius of curvature is large and is approximated by 200 mm. The length of the line segment between two neighboring points on the surface can be approximated by $2a = 2\sqrt{\delta 2R_i} = 5$mm. Uniform discretization with spacing between points of 5mm will suffice. However, in this work a uniform grid of 100×100 in the parametric space was used and the spacing is 1.5 mm. This grid is finer than that required and thus with less sagittal error. When machining a convex surface, the surface triangulation should be done to ensure that $|\varepsilon_S + \varepsilon_t|$ is less than the acceptable tolerance. With $\varepsilon_S < 0.025$mm and $\varepsilon_t < 0.025$mm, the machined surface accuracy is given by $|\varepsilon_S + \varepsilon_t| < 0.05$mm. The achieved accuracy is better than the precision that can be achieved on today's CNC machines.

The above error calculations are based on the tool positions only. While machining a surface the tool moves from one tool position to another and it is assumed that if the tool positions are close to one another the deviation between the design surface and the surface produced as the tool moves (swept surface) are small. These calculations do not consider the swept surface.

Triangles are the most convenient method of describing objects in OpenGL. In this work Bézier surface are first triangulated and then used for tool path planning. The GPU method can thus be used for any type of surface by first triangulating the surfaces to approximate the surface within an acceptable tolerance and then using the GPU method to plan a toolpath.

10 CONCLUSIONS

The procedural method proposed in this paper does not require the solution of simultaneous equations or initial guesses and is faster in implementation as compared to previous attempts. The key difference between this method and the recent other is that rays are fired from a common plane towards the part surface and the tool surface using the GPU hardware and software. The new method emulates the tool dropping on a surface [3].

In addition, it has been shown that the efficient tool drop using GPU can also be algorithmically used to obtain multiple contact points between the tool and the design surface to produce gouge-free 5-axis tool positions. The presented method does not depend on the type of surface used to describe the part and thus can be used both on parametrically and algebraically defined surfaces without occlusion. Furthermore, to reduce the computational burden and to speed up the computational process an adaptive discretization method is presented.

The proposed method at its heart uses triangular approximation for surface and tool. It has been shown that the impact of the approximations on tool positioning is within the accuracy to which parts can be machined on a typical CNC machine. Simple methods of determining the tool path parameters such as discretization distances, number of pixels and surface finish achievable are presented. These limits can be used as guides for setting the accuracy of triangulation for curved surfaces. While our examples show how to achieve an accuracy of 0.01 mm, our methodology can be used to determine the appropriate discretization to achieve any level of software accuracy.

APPENDIX

The pseudo-code for the proposed algorithm is given below.

```
  1.0 Initialize OpenGL and render objects: Initialize(orthogonal,Ro,Ri,P[n][n],GL)
  2.0 Generate Toolpath Footprint:
      FootPrint[k][2] = GenerateFootprint(sideset,forwardstep,xmax,ymax)
  3.0 Set zmax = torusZ+50; th=[0,0,1]
  4.0 For i=1; i<=k
      4.1 Obtain first point of contact and its surface normal:
          [Pi,npi,dmin] = ToolDropMethod(camera,projection,torus,surface,Footprint[i])
      4.2 Obtain the center of pseudo-insert:  Oi = Pi + Ri npi
      4.3 Set coordinate frame: [u1,v1,w1]
          w1=t; u1=(T-Pi) x w1/|(T-Pi) x w1|; v1=w1 x u1;
      4.4 Set Bmin = 0; Bmax=pi/4; B=Bmax; Blast=0;
      4.5 while |B-Blast| < eps
          4.5.1 Blast = B
          4.5.2 B=(Bmax+Bmin)/2
          4.5.3 Rotate Bezier Surface around u1 axis at O1 by angle B:
                surface'=rotate(surface,B,u1,O1)
          4.5.4 Obtain second point of contact, its surface normal, and drop distance
                [Qi,nQi,dmin] = ToolDrop(camera,projection,torus,surface',Footprint[i])
          4.5.5 If (dmin>eps) then Bmax=B else Bmin=B
      4.6 Rotate tool axis around global x axis at Origin by angle B
          t'=rotate(t,B,x,Origin)
      4.7 Output too position: (Pi,nPi,Qi,nQi,t')

Initialize(orthognal, Ro, Ri, P[n][n],GL)
```

```
    1.0 Define Window Size: GLwindow(width,height)
    2.0 Enable Depth test for obtaining Z-buffer: Enable GL
    3.0 Set orthogonal Matrix: orghographics(Ro,Ri)
    4.0 Set Projection Matrix: projectionMatrix(orthographic)
    5.0 Set Viewpoint and Camera:  camera(projectionMatrix)
    6.0 Define and Initialize Toroidal cutter object: torus(Ro,Ri)
    7.0 Define and Initialize Bezier surface: surface(P[n][n])

ToolDrop(camera,projection,torus,surface,FootPrint)

  1.0 moveTorus(FootPrint[1], FootPrint[2], 0)
  2.0 moveCamera(FootPrint[1], FootPrint[2], zmax)
  3.0 Define depth buffer arrays for Torus and Surface
  4.0 Read depth buffer of Torus: getDepth(depth(torus))
  5.0 Read depth buffer of Surface: getDepth(depth(surface))
  6.0 Get the pixel position with minimum diference of torus and surface depth buffers
     (pixel x, pixel y, difference]=compareDepthBuffers(depth(torus),depth(surface)
  7.0 Convert pixel position and difference into 3D Euclidean Space for surface model:
      P,np,dmin = screenToWorld(pixel x, pixel y, difference, projection, surface)
  8.0 return (P,np,dmin)
```

ACKNOWLEDGEMENTS

This work was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Mukhmeet Singh, https://orcid.org/0000-0001-5101-4312
Sanjeev Bedi, https://orcid.org/0000-0002-6993-9502
Stephen Mann, https://orcid.org/0000-0001-8528-2921

REFERENCES

[1] Duvedi, R.; Bedi, S.; Mann, S.: An efficient multipoint 5-axis tool positioning method for tensor product surfaces. Advanced Manufacturing Technology, 97, 279–295, 2018. http://doi.org/10.1007/s00170-018-1940-y.

[2] Duvedi, R.; Bedi, S.; Mann, S.: Numerical implementation of drop and tilt method of five-axis tool positioning for tensor product surfaces. Int J Adv Manuf Technol., 95, 219–232, 2018. http://doi.org/10.1007/s00170-017-1193-1.

[3] Duvedi, R.; Singh, M.; Bedi, S.; Mann, S.: Multipoint tool positioning of a toroidal end mill for five-axis machining of generalized tensor product Bézier surfaces. Advanced Manufacturing Technology, 111, 495–503, 2020. http://doi.org/10.1007/s0017-020-06006-1.

[4] Duvedi, R.K.; Bedi, S.; Batish, A.; Mann, S.: A multipoint method for 5-axis machining of triangulated surface models. Computer-Aided Design, 52(1), 17–26, 2014. http://doi.org/10.1016/j.cad.2014.02.008.

[5] Gray, P.; Bedi, S.; Ismail, F.: Rolling ball method for 5-axis surface machining. CAD, 35, 347–357, 2003. http://doi.org/10.1016/S0010-4485(02)00056-8.

[6] Gray, P.; Bedi, S.; Ismail, F.: Arc-intersect method for 5-axis tool positioning. CAD, 37, 663–674, 2005. http://doi.org/10.1016/j.cad.2004.08.006.

[7] Gray, P.J.; Ismail, F.; Bedi, S.: Graphics-assisted rolling ball method for 5-axis surface machining. Computer-Aided Design, 36(7), 653–663, 2004. http://doi.org/10.1016/S0010-4485(03)00141-6.

[8] Hsieh, H.; Chu, C.: Gpu-based optimization of tool path planning in 5-axis flank milling. In 2010 International Conference on Manufacturing Automation, 143–150, 2010. http://doi.org/10.1109/ICMA.2010.31.

[9] Kukreja, A.; Dhanda, M.; Pande, S.: Voxel-based adaptive toolpath planning using graphics processing unit for freeform surface machining. ASME. J. Manuf. Sci. Eng., 144(1), 2022. http://doi.org/10.1115/1.4051535.

[10] Ranjan, R.: Gpu acceleration of cfd codes, 2019. http://doi.org/10.13140/RG.2.2.11321.93286.

[11] Rao, F.I.S.B.N.: Tool path planning for five-axis machining using the principal axis method. International Journal of Machine Tools and Manufacture, 37(7), 1025–1040, 1997. http://doi.org/10.1016/S0890-6955(96)00046-6.

[12] Roth, D.; Ismail, F.; Bedi, S.: Mechanistic modelling of the milling process using an adaptive depth buffer. Computer-Aided Design, 35(14), 1287–1303, 2003. http://doi.org/10.1016/S0010-4485(03)00044-7.

[13] Sanfui, S.; Sharma, D.: A three-stage graphics processing unit-based finite element analyses matrix generation strategy for unstructured meshes. Numerical Methods in Engineering, 121(17), 3824–3848, 2020. http://doi.org/10.1002/nme.6383.

[14] Singh, M.: Fast and Robust Approach to Find the Gouge-free Tool Position of the Toroidal Cutter for the Bézier Surface in Five Axis Machining. Master's thesis, University of Waterloo, 2020.

[15] Warkentin, A.; Ismail, F.; Bedi, S.: Multi-point tool positioning strategy for 5-axis mashining of sculptured surfaces. CAGD, 17, 83–100, 2000. http://doi.org/10.1016/S0167-8396(99)00040-0.