COMPUTER-AIDED DESIGN & APPLICATIONS, 2018
VOL. 15, NO. 1, 76–89
https://doi.org/10.1080/16864360.2017.1353730

5-Axis tool path planning based on highly parallel discrete volumetric geometry representation: Part I contact point generation

Dmytro Konobrytskyi [a], Mohammad M. Hossain [b], Thomas M. Tucker [c], Joshua A. Tarbutton [d] and Thomas R. Kurfess [b]

[a]Uber Technologies Inc., USA; [b]Georgia Institute of Technology, USA; [c]Tucker Innovations Inc., USA; [d]University of South Carolina, USA

ABSTRACT

While 5-axis CNC machines improve the manufacturing productivity, the tool-path programming demands human expertise and tremendous time investment to generate collision-free optimal tool trajectories. To address the challenge of multi-axis CNC programming, this work presents a new discrete volume based geometry representation that naturally enables highly parallel geometry processing. Then it formulates and discusses a methodology for tool path planning algorithms designed around the formulated geometric representation for 5-axis CNC milling machines and ball end tools. This paper is the first of two papers that deals with finding the XYZ locations of ball end mill centers that govern the tool trajectories. The experimental 5-axis machining using the resulting tool paths demonstrates that the designed algorithms are capable of producing collision- and gouge- free tool paths with a desired surface quality for a given target geometry by using the selected ball end tools.

KEYWORDS

Computer-aided manufacturing; CNC tool-path planning; GPU acceleration

1. Introduction

Tool path planning for multi-axis milling CNC machines is a complicated problem that requires knowledge of the material removal process, selecting multiple appropriate strategies and highly accurate calculations. CAM (computer-aided manufacturing) software lies in between and allows interaction between human and computer. This approach can solve almost every problem that appears in modern manufacturing, but it requires two important components: a trained engineer and a significant amount of time even for simple parts. This initial use of resources becomes smaller for high volume production however for a small number of parts personnel and time costs become extremely critical. In the case of low volume production, time of an engineer may cost many times more than actual machining cost. As a result, today the low volume market is occupied by usually additive Rapid Prototyping technologies such as 3D printing which allow manufacturing of a part almost without machine-human interaction. However existing RP technologies cannot provide a set of cost, surface quality and available material properties found in traditional subtractive CNC machining.

The key for this change is reducing the time required for tool path planning. This time includes two components: the time used by a computer for calculations and the time used by an engineer for selecting the right machining approach. Although these components look completely independent, they are parts of the same performance related problem. Time used by a computer for calculation obviously depends on the performance of the computer and on the ability of the programmer to use the available resources efficiently, which is a challenge. The time invested in selecting the right machining strategy is highly dependent on the experience of the engineer in addition to any complexity of algorithms used to select this strategy. Therefore, in order to solve the tool path planning problem, the computational performance problem has to be solved along with the use of novel automated path planning algorithms.

The computational problem can be solved by hardware that has enough computational performance if the available resources are used efficiently. At the time when further increasing of processors clock frequency is almost impossible, both requirements are pretty much identical and mean support for parallel processing and ability to use multiple cores, devices and even computers simultaneously. Although parallel processing itself is not a complicated idea, the parallelization of existing geometry processing algorithms and data structures is not

CONTACT Dmytro Konobrytskyi dkonobr.cv@gmail.com; Mohammad M. Hossain mhossain7@gatech.edu; Thomas M. Tucker tommy@tuckerinnovations.com; Joshua A. Tarbutton jat@cec.sc.edu; Thomas R. Kurfess kurfess@gatech.edu

a trivial process. In order to simplify this process, this work proposes the fundamentally parallel geometry representation. The idea behind it is to move parallelization complexity from an algorithms design level to a data structure design level. As a result, every algorithm that uses the described geometry representation can be easily parallelized.

This paper provides a methodology for designing parallel algorithms by reformulating path planning problems in a way that they can be described in terms of operations supported by the developed geometry representation. As a proof of concept this paper will describe a complete and fully automated 5-axis tool trajectory planning system capable of machining almost any possible part geometry. This paper first describes a highly parallel GPGPU based volume offset calculation approach using the underlying hardware-informed data structure. The volume offset calculation discussion is followed by a description of the surface filling algorithm that is used as a foundation for two tool-center trajectory planning algorithms. These two fully automated and robust 5-axis tool path planning algorithms are used for path planning of both roughing and finishing processes with ball-end mills. All of the algorithms described here are designed for parallel GPU hardware and can run on multi-GPU system. This paper summarizes the results of these algorithms by demonstrating their performance on real parts machined using a 5-axis milling machine.

2. Related work

CNC milling has progressed in the last 40 years from fully mechanical machine controls, punch cards and paper tape to modern fully computerized controllers, programmed via variants of the G-code programming language. Programming these machines has advanced from inefficient handwritten programs to powerful CAM systems capable of generation complex multi-axis trajectories, based on strategies selected by operator and precise virtual milling simulation.

Significant research has been focused on key areas such as tool path planning, tool orientation selection, and selection of tool geometry. Many researchers have addressed tool path planning using traditional methods such as iso-planar [2, 11, 17] or iso-parametric approaches [18]. Results of these approaches generate paths that achieve certain accuracies, or surface characteristics, but that may not be optimal with respect to other process parameters, such as production time. In order to improve performance of traditional methods, the iso-scallop approach was introduced by Suresh and Yang [25] and Lin and Koren [22]. It produces a constant scallop height of a machined surface. Popularization of 5-axis milling and milling of non-parametric surfaces has resulted in the development of new approaches resolving specific 5-axis problems and further reducing milling time. These approaches can be classified [13] as curvature matched milling [6, 14, 15], iso-phote based method [4, 9, 31], configuration space methods [3, 19], region based tool path generation [9], compound surface milling [29, 30] and methods for polyhedral models and point clouds [21, 24]. With respect to tool orientation selection, traditional methods such as fixed orientation, principal axis methods [32] or multi point milling [28] have been developed. Furthermore, in the past 10 years, more advanced path planning methods such as the rolling ball [7, 8] and arc intersect methods [5] as well as earlier C-space based approaches [3, 20] were successfully deployed. Furthermore, research addressing tool geometry selection [1, 12], and implementation of automatic tool selection in commercial products does not exist or is very limited when addressing optimized tooling parameter selections. While significant progress has been achieved over the last several decades, a plethora of issues to be addressed that will reduce production time and improve / guarantee component quality still exist.

Throughout the literature, it is clear that computation time is a major limitation of most, if not all, of the proposed algorithms. One solution for this problem is the employment of high performance computing, in particular the GPU (Graphical Processing Unit) platform to accelerate the processing. Development and popularization of a general purpose GPU (GPGPU) approach and platforms like Compute Unified Architecture (CUDA) have resulted in promising results for deploying GPGPU functionality in a manufacturing environment. Tukora and Szalay presented an approach for GPGPU accelerated cutting force prediction [26]. Hsieh and Hsin proposed a GPU accelerated particle swarm optimization approach for 5-axis flank milling [10]. Furthermore, new approaches for geometry representation used in CNC area were recently proposed. Guang and Ding proposed employing a quadtree-array for representation of a workpiece in 3-axis milling [16]. Wang and Leung described the use of layered depth-normal images for solid modeling of polyhedral objects [27].

3. Developed irregularly sampled volume representation

Existing geometry representation approaches provide a wide range of tradeoffs between accuracy, memory usage, parallelizability and scalability; but do not offer a perfect choice for GPU-computing. The main reason why existing geometry representations are not a good match for GPGPU is because the focus is on reducing the overall

number of operations required for geometry processing without trying to make these operations independent from each other. This is a perfectly valid strategy for traditional CPUs where fewer operations almost always means faster algorithms, but it does not work with GPUs which are optimized for performing operations in parallel. As a result, there is an opportunity to create a GPGPU optimized geometry representation and a corresponding data structure that can be used for the 5-axis CNC milling simulation and tool path planning.

Existing geometry representations are not well suited to parallel computation compared to volumetric approaches which can be discretized into volume elements that can be processed independently in parallel. A fundamental limitation is that a naïve, regularly sampled volumetric approach (such as voxels represented in a 3D array) requires gigantic storage. To realize this storage bottleneck, consider the representation of a 500mm cube with 2 micron elements (as an example of a work area and accuracy found in modern 5-axis machines) and 1 byte per element, a simple part occupying this volume would require ~350 TB (6 sides * (500 mm /side / 0.002 mm)^2) of data just for the surface representation of the part. Current generation of GPUs provide a maximum memory up to 24 GB (about 15,000X smaller than required). Although current memory limits cannot be overcome there is still a practical application for the many cases where extremely high accuracy is completely unnecessary, for instance, a volumetric approach that could produce parts with the same accuracy of commodity 3D printers.

After accepting the fact of reduced accuracy for volumetric data representations, the next step is to make a decision about the tradeoff between memory usage, accuracy and parallelizability. It is worthwhile to mention a few relationships between these parameters. Generally, more complicated data structures provide higher accuracy for a given amount of memory. For example, deeper trees provide more efficient memory usage for a given accuracy. Relatively complicated data structures with non-predictable density such as k-d trees are less suitable for this research due to GPGPU specific load balancing (the problem similar to the BREP) and editing problems. Although generally they provide higher efficiency, their processing algorithms are more complicated, often have non-linear memory access patterns and have a higher branch divergence. These properties result in significant performance penalties on modern GPUs. One of the simplest possible tree-based volumetric representations is a tree with nodes where each node represents a regularly sampled volume and a list of references to its children. The octree is a classic example of this type of geometry representations with 8 children per node.

One of the most important steps is the selection of a number of children and amount of geometrical data stored in each tree node. While the storage scales with the the number of children, abundant parallelism is exposed by efficiently processing each child of a node concurrently. If this is done by a warp (term used by NVidia for representing a group of GPU processing threads (usually 32) that perform the same command on different data), it makes memory access more efficient by storing child data in a continuous memory block which can be read linearly in one memory access operation. Considering the amount of geometrical data stored in a node it is possible to say that more data approximates geometry better but uses more memory. On one side of this tradeoff, each node contains a complete mathematical description of all geometry elements. And on the other side, it is possible to use only one bit to store information about presence of material in a node's volume (or store a byte that describes a distance to a surface or material density as it is done in the traditional voxel model).

All these tradeoffs were considered resulting in a volumetric data structure designed for GPGPU accelerated multi-axis CNC tool path planning (Fig. 1). This geometry representation is a 2-level hybrid of the tree and the dense block of voxels. It uses a 3D array of cells that represents a regularly sampled volume. Each cell stores 2 bits of geometrical data and a pointer to an array of 4096 children (similar to a tree). Cells children (called "subcell") represent a regularly sampled (16×16×16) volume and store 2 bits of geometrical data but do not store pointers to their children. 2 bits geometrical data is used for 3-color scheme for geometry representation. They represent 3 possible states of a cell or subcell:

- Cell is completely filled by material
- Cell is completely empty
- Cell state is unknown and it probably contains a boundary

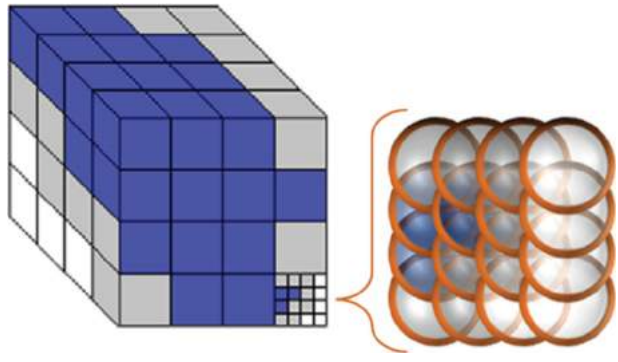

Figure 1. Developed geometry representation model.

In contrast to traditional cubical voxels, cells or subcells represent spheres circumscribed around traditional cubes calculated by volume subdivision (Fig. 1). Figure 2 demonstrates a surface representation example with the 2D version of the described geometry representation and square cells.

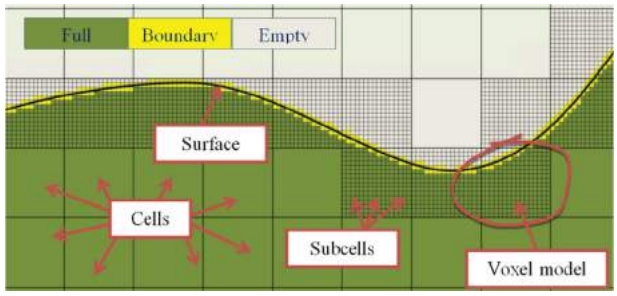

Figure 2. 2D example of the developed model surface representation.

From a hierarchy point of view, it can be viewed as a two level tree as shown in Fig. 3. It is important that the low level nodes that represent subcells are stored as a dense block of voxels. But, information about high level nodes is stored in a list. As a result, links between nodes are not really stored anywhere as done in traditional trees but the model still has a tree like hierarchy. First level links are represented by indexes in a cells list and second level links are represented by indexes in voxel models. This approach allows saving significant amounts of memory relative to a traditional linked tree based approach. From a memory point of view, the model looks like the diagram shown in Fig. 4.

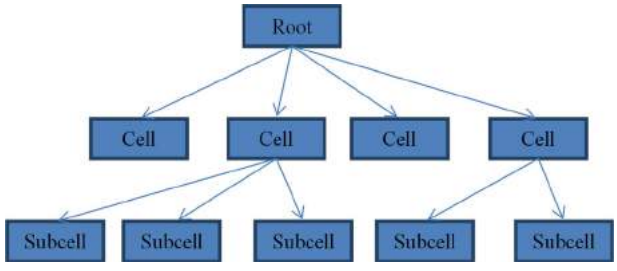

Figure 3. HDT hierarchy.

The rationale behind the selected design is an attempt to combine parallelizability and scalability of the voxel model and memory efficiency of tree based geometry representations. The 2-level design provides much better memory efficiency than the dense voxel representation. The reason for the selection of the 2 bits geometry representation and spherical cells is an attempt to use as simple as possible geometry processing algorithms with the lowest number of branches. The design also provides scalability. Since all cells are independent, they can be stored on multiple GPUs (and possibly on multiple computers) and can be processed independently with the near-linear performance improvement.

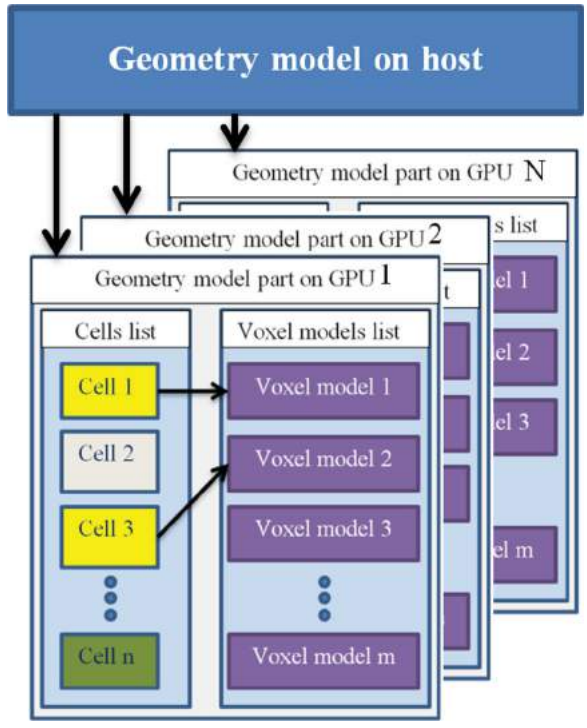

Figure 4. Geometry model from a memory point of view.

4. Low-level parallel Geometry processing algorithms

Before discussing the specific application of path planning, it is important to understand some basic operations provided by this underlying data structure. A flexible and powerful basic operation is the "containment" test that can be performed for each cell. The containment test uses two user provided expressions that determine if a sphere defined by a center position and radius is 1) completely inside or 2) completely outside of a target shape. These expressions are calculated independently for each cell and their results are used to update the cell state based on predefined rules as shown in Algorithm 1. If a cell fails both tests it is assumed that a cell potentially has a boundary.

The flexibility of the containment test allows it to be used as a main component for designing many useful algorithms such as machining simulation, volume offset calculation or contour offset path planning. But what is more important is that any derived algorithm is always highly parallel. For example, one of the most important operations in this work is the volume surface

Algorithm 1 Containment test

1: **For** each cell in parallel:
2: Calculate containment expressions
3: Generate a finishing tool path by combining all generated curves
4: **End For**

intersection calculation. It takes two independent volumes and outputs a set of points that contain volume boundaries in both volumes. Then it uses a post processing algorithm to convert the point cloud into a list of continuous curves. The idea behind the post-processing algorithm is to start with a random boundary point and to use a wave approach to iteratively connect neighbor points. By calculating a center of each wave for all iterations it is possible to get a continuous curve that describes the actual volume intersection curve. The described operations are shown in Algorithm 2. It is important to notice that only step 1 of the Algorithm 2 can be easily parallelized while the other steps are iterative and cannot readily be performed in parallel. However, this should not be a problem since they always process a reasonably small subset of all cells that represents a curve.

Algorithm 2 Volume surface intersection

1: Find all cells that have boundary states in both volumes
2: **While** there are non-processed cells:
3: Select a random cell from 1
4: Initialize new intersection curve with center of cell from 3
5: **While** there are non-processed neighbors around selected cell:
6: Mark all neighbors as current wave cells
7: Calculate center of the current wave
8: Append a wave center to a current intersection curve
9: **End While**
10: **End While**

The basic operations described above form a basis for finding the contact point locations of a path planning algorithm. Additional algorithms for selecting the tool orientation and performing machining simulation are required to implement a full tool path planning system. These orientation and simulation algorithms require use of the same data structure and low level algorithms presented here but are out of the scope of this paper.

5. Volume based parallel algorithms design methodology and limitations

The containment test and volume intersection operation are the main tools for finding a sequence of contact points for path planning. However, solutions for tool path planning problems should accommodate the methods used in practice. For example, a simple iso-planar [1–3] approach that uses the intersection between a sequence of parallel planes and the part surface as a path of contact points can be easily implemented in a parallel fashion in two ways. First, the intersections between the part surface and the sequence of planes can be represented as the intersection between part volume and a sequence of parallelepipeds. Second, the containment test can be applied where a target shape is actually a sequence of planes. Although both approaches do the same task, they are quite different and use different tools. But they both have two important benefits. First, there is no need to care about special cases, singular points, discontinuities, etc. Second, both approaches can be implemented in a highly parallel way and run on highly parallel hardware.

A key concept is to reformulate operations with surfaces as operations with volumes that can be represented by independent operations with the volume's cells. Although reformulation of algorithms in volumetric fashion is usually not too complicated, this approach requires caution due to some limitations of the underlying discrete geometry representation. The first and probably the most significant limitation is related to the volume boundary position. It is important to recognize that an actual surface position is only known to within a tolerance. Each cell stores only 2 bits of information that represent 3 states and only 2 of 3 states fully define the cell. If a cell has a completely empty state, all positions inside the bounds of the cell are known to be empty and vice versa for a completely full cell. However, if a cell does not hold either of these states, portions of the cell may be empty and other portions full. The boundary surface of the volume may or may not pass through the cell. In most cases and for most applications, it is safe to assume that such cells actually contains a surface. However, even if cell does contain the volume's boundary surface, where the surface geometry passes through the cell is unknown. One must select cell sizes at or below manufacturing tolerances in order to have adequate resolution in applying this data structure and algorithm to path planning.

The second limitation is the use of finite difference derivative approximations. Since the underlying geometry representation is of a discrete nature, the calculation and use of finite difference derivatives is readily available. The finite differences can supply information as surface gradients and surface normals. The value of these

finite difference derivatives are subject to the cell resolution. For example, surface normals used in the rendering process for lighting calculations are estimated from a finite differences in runtime and provide adequate normals for effectively rendering a 3D visualization of the data structure. However, the surface normal produced by this method is not sufficiently accurate for critical calculations in the path planning process.

6. Offset volume calculation

Despite the described limitations, many complex geometrical problems can readily be solved by using the geometry representation describe in this paper. One of these problems is the offset surface generation problem. The offset surface is defined as a surface at equal distance from an original surface (Fig. 5). It is often used in tool path planning processes as a surface where the center of a ball-end mill cutting tool may move freely without producing overcuts and yet remaining in contact with the part surface at the outer radius of the ball. By replacing tool contact point trajectory planning with tool center trajectory planning it is possible to eliminate a complicated gouge prevention process and make tool path planning algorithms simpler.

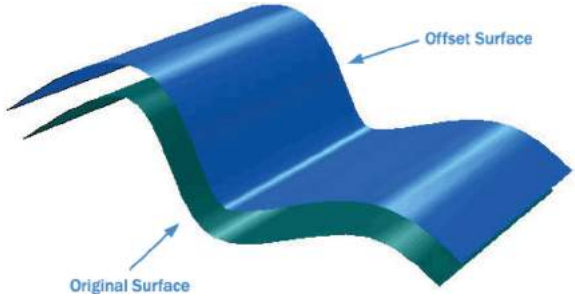

Figure 5. Offset surface.

Although the offset surface makes path planning algorithms simpler, finding an offset surface is not a trivial problem for most geometry representations. Problems in special cases such as holes and self-intersections [23] as shown Fig. 6 are characteristic. The offset surface finding approach in this work eliminates the self-intersection problem completely and allows the use of models with holes that are smaller than the offset distance. It is important to notice that this approach uses triangular meshes as an input geometry representation but similar algorithms can be implemented for other data structures.

The main idea behind the surface offset algorithm is to work with volumes and not strictly with surfaces, so it is more accurate to describe it as an "offset volume" algorithm. Here the offset volume represents a volume

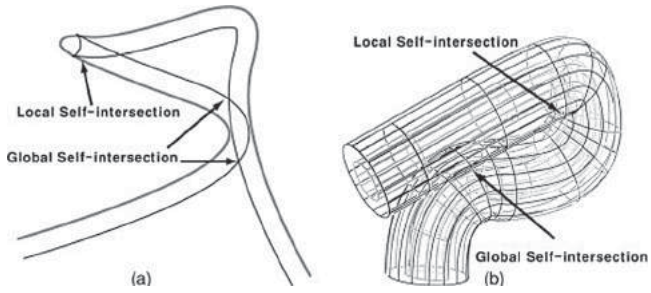

Figure 6. Offset surface self-intersections [23].

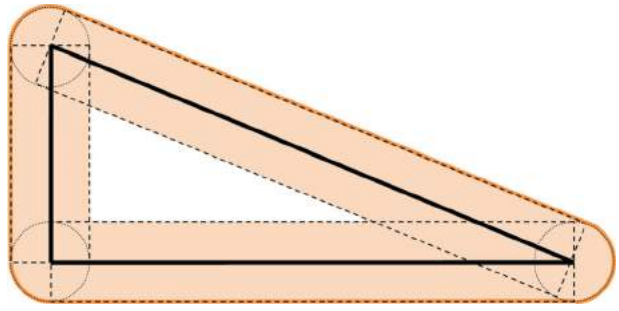

Figure 7. 2D offset surface decomposition.

that contains all points that are closer than an offset distance to initial surface. In 2D case an offset curve calculation can be replaced by offset area calculation as shown in Fig. 7. In order to construct an offset volume efficiently, it can be represented as a composition of topological elements associated with the original surface. For the 2D case (Fig. 7) every point is associated with a circle and every line is associated with a rectangle. For a 3D model and triangle mesh geometry representation there is a similar association list:

- Vertex – Sphere centered on vertex with radius equal to offset amount.
- Edge – Cylinder with axis along edge, radius equal to offset amount, and bounding by start and end points of edge.
- Face – Prism generated by extruding the triangle face in both directions by offset amount.

As a result, a triangular mesh may be converted to a list of volumetric primitives that can be composed together and represent an offset volume. Then every cell of geometry model can be tested against this list of volumetric primitives and marked as a part of an offset volume if it passes the containment test with one of these primitives. The offset volume calculation algorithm that combines all these steps is shown in Algorithm 3.

Loops 1, 4 and 7 in Algorithm 3 are completely independent and can be easily parallelized. Loop 10 is actually a part of the belonging test described before and can be

Algorithm 3 Volume offset calculation

```
For all vertexes in triangle mesh in parallel:
    Add sphere to primitives list
End For
For all edges in triangle mesh in parallel:
    Add cylinder to primitives list
End For
For all faces in triangle mesh in parallel:
    Add prism to primitives list
End For
For all cells in geometry model in parallel:
    For all primitives in list:
        If cell belongs to primitive:
            Mark cell as an offset volume cell
        End If
    End For
End For
```

easily parallelized as well since all cells are always completely independent in the developed geometry model. Figure 8 demonstrates offset volumes of the impeller mesh produced by the described algorithm. The top left picture demonstrated the original model while other pictures are computed with various offset distances.

7. Surface filling algorithm based on 3D contour offset approach

The example of the offset volume calculation algorithm has shown that it is possible to use methodology and geometry representation described here to solve challenging computational geometry problems. This section will discuss how these algorithms can be used to for automatic tool path planning. First of all, it is important to note that modern CAM systems support a large variety of tool path planning strategies (such as iso-parallel, spiral, contour offset, etc.) that produce efficient tool paths for a variety of different situations. Although there are many possible options for tool path planning, these solutions are usually quite specialized and do not work well as a true general purpose solution for all possible situations. This limitation requires the labor costs of a skilled engineer working "surface by surface" and selecting a sequence of appropriate strategies with appropriate parameters. The goal in this work is to create a foundational and robust strategy for a fully automated tool path planning system.

The idea behind the developed robust path planning strategy is generalizing the 2D contour offset strategy often used in modern CAM software to three dimensions. Although 2D and 3D versions are conceptually similar (in fact a 2D version is a special case of a 3D algorithm), there are some important differences related to where and how they generate a tool path. The traditional contour offset approach calculates a tool path on a plane which is orthogonal to a tool direction. It iteratively offsets a contour and uses offset curves as tool path components. Usually a sequence of parallel planes is used for removing most of volume during a roughing process. The 3D version does perform very similar steps but does not require using a planar surface (although it can use a plane and in this case it becomes a 2D contour offset approach). It uses any possible user selected surface called "Target surface". The problem here is an additional dimension. As a result, offsetting a contour creates a tube like shape that cannot be used for path planning (Fig. 9: b). As a solution for this problem, an additional step is required – calculation of the intersection between a tube and a target surface (Fig. 9: c). By calculating the intersection, it generates a curve that lies on a constant distance from an original contour and can be used for further path planning. The important property of the contour offset approach is preserved – the distance between path components is constant in most cases and always bounded. This property allows controlling a scallop height of the machined surface by controlling distance between path components.

Iteratively performing the contour offset algorithm until an entire surface (or a surface part) is covered (Fig. 10). generates a sequence of curves that completely fill a target surface in a way that a distance between them is no shorter than offset distance and no longer than two offset distances. The developed implementation determines that an entire surface is processed if the next intersection between a curve offset volume and a target

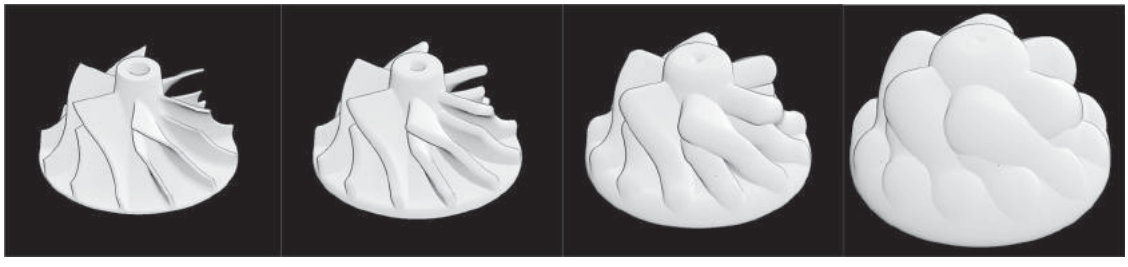

Figure 8. Offset volumes of an impeller.

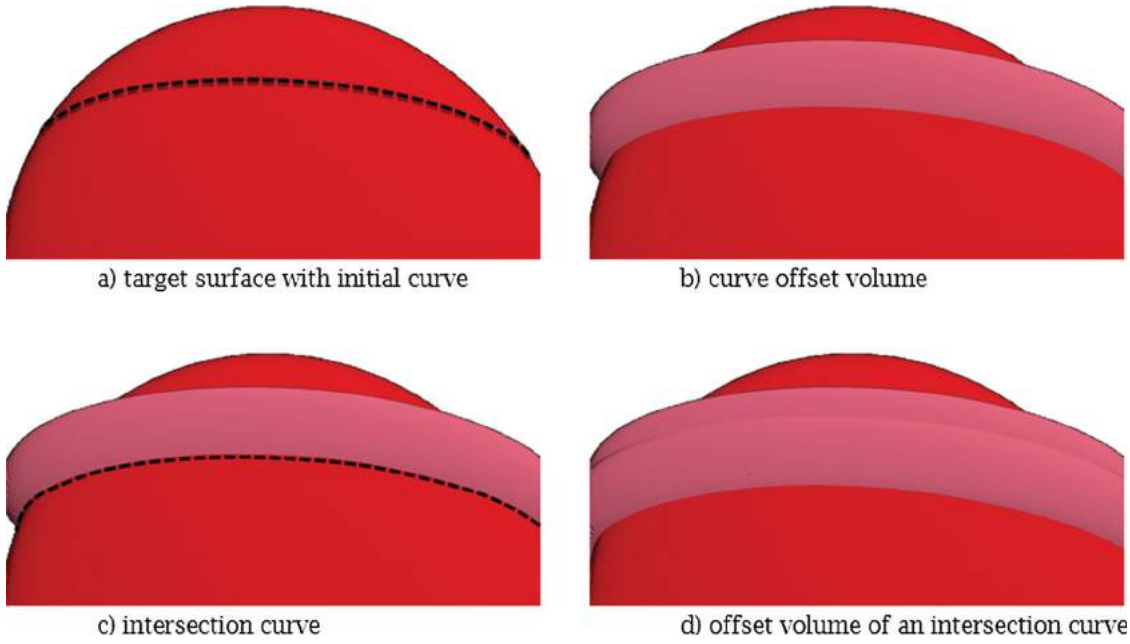

Figure 9. Curve offsetting.

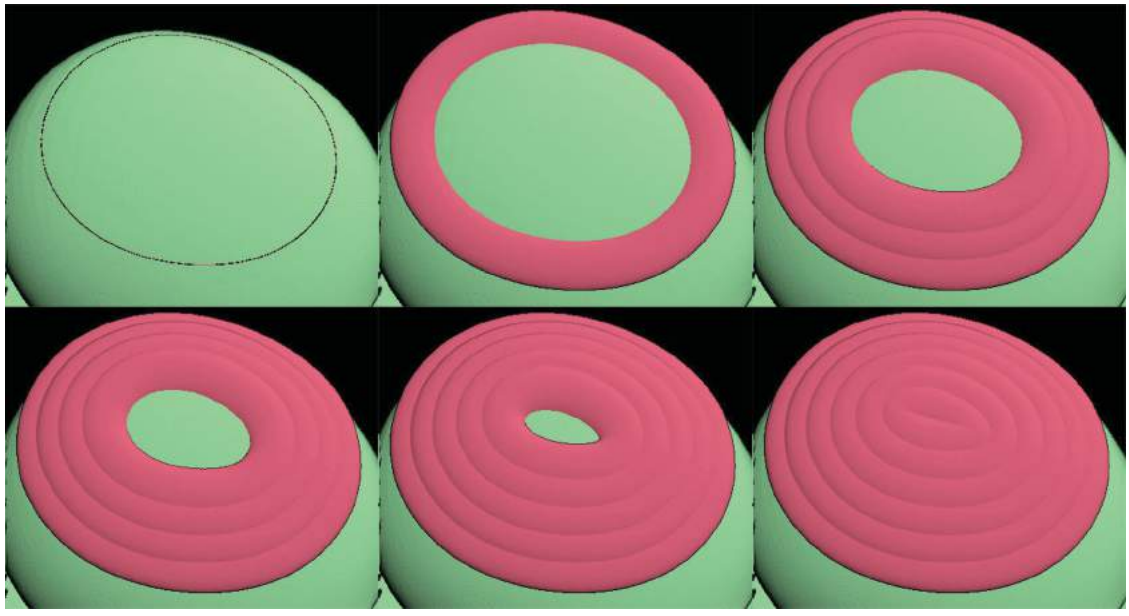

Figure 10. Iterative surface area filling.

volume is an empty volume. It is also important to notice that the curve offset volume combines all offset volumes calculated during previous iteration, so if a part of a target surface is already processed it will not be processed again.

Before describing the complete surface filling algorithm, it is important to discuss the third component used in this process, a "restriction volume". Boundary conditions, and also any required restrictions, are represented as a restriction volume that contains areas where tool movements are not desired or dangerous. For example, during roughing path planning for ball end tool, restriction volume includes an offset volume of a target geometry with an offset value equal to a tool radius plus some small extra distance. By not allowing path planning in areas that are too close to a part surface, it predicts overcuts because a tool center will never come closer than a tool radius and a tool surface will never intersect a part surface as a result. A restriction volume also limits filling algorithm in a way that only a desired part of a surface is processed even if an entire surface is not processed yet. This is useful for protecting against fixture collision.

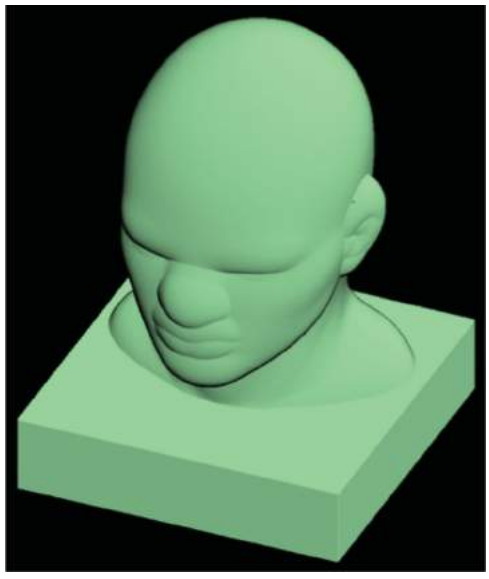

Figure 11. Restriction volume for the "head" model.

For example, Fig. 11 demonstrates the restriction volume for the "Head" model that contains two parts: offset volume of the model with offset distance equal the tool radius and a box volume in the bottom for protecting fixtures. The entire surface filling process is described in Algorithm 4.

The most important property of the developed surface filling algorithm is parallelizability. Since it is based on a volume offset and volume intersection algorithms, which are both parallel, the entire surface filling algorithm becomes naturally parallel and all algorithms that use it are also naturally parallel.

Algorithm 4 Surface filling

```
1: Current curve = Initial curve
2: Do:
3:     Offset current curve
4:     Calculate Intersection curve between Target surface and Offset volume
5:     If intersection curve exists:
6:         Save intersection curve as a tool path component
7:         Current curve = Intersection curve
8:     End If
9: Until: Intersection curve does not exist
```

8. Robust tool trajectory generation for 5-axis machines

The 3D contour offset algorithm is used both for the roughing and the finishing tool path planning by applying different target surfaces. In the case of finishing a model offset volume surface is used as a target surface. An offset value in this case is equal to a tool radius. A contour offset value controls path step and it is selected based

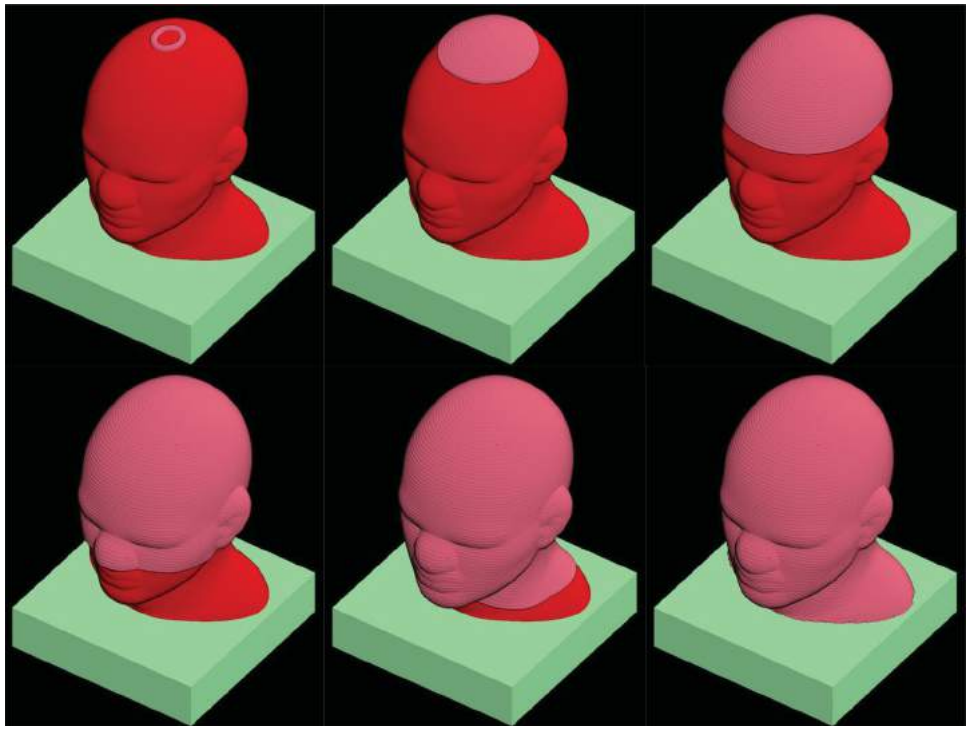

Figure 12. Surface filling for finishing tool path generation.

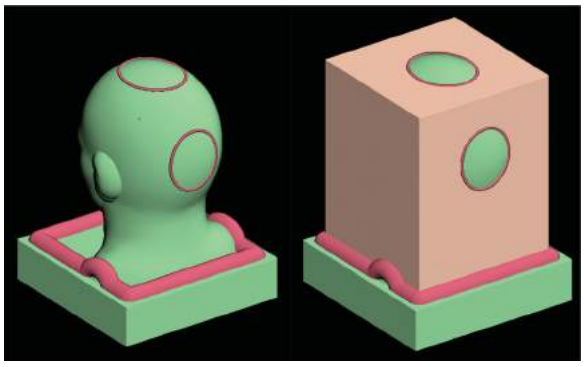

Figure 13. Initial curve selection for roughing process.

on a desired scallop height. As it was mentioned before, limiting tool center movements to an offset surface prevents overcuts by a ball part of a tool. For the finishing path generation, an initial curve can be selected in many ways but the current implementation uses an intersection between a horizontal plane and a top of an offset model.

Figure 12 demonstrates an example of the surface filling process used for a finishing tool path generation and Algorithm 5 demonstrates the algorithm steps. Intersection curves calculated during this process are used as tool center trajectory curves in a finishing tool path.

Algorithm 5 Finishing tool path generation

1: Calculate intersection curve between part offset volume and horizontal plane
2: Apply the surface filling algorithm starting with the intersection curve (Algorithm 4)
3: Generate a finishing tool path by combining all generated curves

The roughing tool path generation process is a bit more complicated than the finishing process because it has to process a volume, not a surface. There are three main differences. First, it uses an iterative approach to generate a tool path that removes material layer by layer until it reaches a part surface. Second, a target surface for roughing process is a workpiece material surface itself. Similarly to the finishing process, it uses surface filling algorithm for generating a set of curves on a material surface that are used as tool center trajectory curves. And finally, roughing algorithm selects initial curves differently. The current implementation uses the intersection between a workpiece and a model offset volume for selecting an initial curve. After the intersection is calculated, the longest intersection curve is selected (Fig. 13), and the surface filling algorithm is used. This process repeats until all intersection curves are processed. All roughing path planning steps are demonstrated in Algorithm 6.

Algorithm 6 Roughing path planning

1: Calculate part and fixtures offset volume (Algorithm 3)
2: **Do:**
3: Calculate intersection curves between workpiece and part offset volumes
4: **While** non-processed intersection available:
5: Select the longest intersection curve
6: Apply the Surface filling algorithm starting with the selected curve
7: Generate a roughing tool path for a layer by combining all generated curves
8: **End While**
9: **Until:** intersection curves exist
10: Generate a roughing tool path by combining all layers

Figure 14 demonstrates workpiece geometry after removing each layer of material during a roughing process with a tool path generated by the described roughing algorithm. It is also easy to see the exact tool trajectory on the first few layers. The finishing and roughing tool path planning approaches have some important properties that should be mentioned. First of all, these algorithms follow the developed methodology and essentially perform volume offset, volume intersection and surface filling operations. Since all these operations are naturally parallel because they employ the highly parallel data structure presented, the resulting tool path planning algorithms are also highly parallelized. The second important property is robustness. In context of this work, robustness refers to the ability to generate a valid tool path for any given geometry. It is easy to see that both algorithms just perform a set of steps without any a priori knowledge about geometry itself making them geometrically indifferent and able to produce paths for any possible 3D model. Both algorithms stop only after processing an entire surface or volume since it is part of exiting conditions.

9. Experimental results

All described tool trajectory planning algorithms were implemented using a combination of Python, C++ and OpenCL languages during the research project. The developed system was tested on a computer with 3 GPUs (2x NVidia GTX580 and 1x NVidia GTX480) and showed great parallelizability and almost linear scalability. The following results were achieved with the developed automated tool path planning system that also includes orientation selection and milling simulation algorithms which were not discussed in this paper.

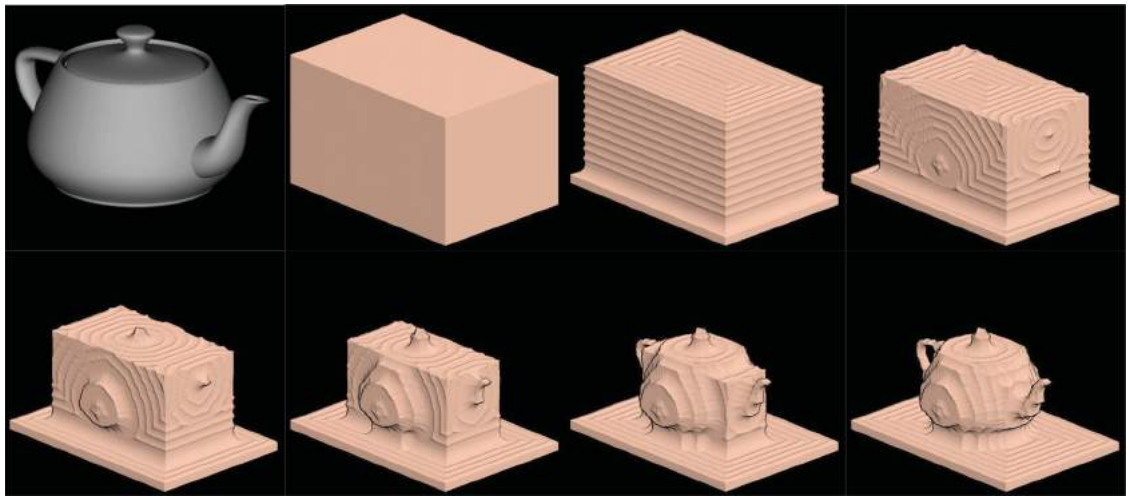

Figure 14. Layer by layer material removing during a roughing process.

In order to validate developed methodology and algorithms, the path planning system was used for generation G-code programs for multiple geometries. These G-code programs were feed on an Okuma MU500VA 5-axis milling machine. Although actual computation time depends significantly on target geometry and resolution, computing tool paths for demonstrated models took on average 10–30 minutes of computing time when the underlying model has a 50-micron resolution. For example, the "Fan" part took about 15 minutes of compute time in contrast to 2 hours of human time required to generate a similar tool path using traditional CAM software. This allows to get a significant speedup especially considering almost linear scalability and the fact that the software has being running on reasonably old GPUs (single modern GPU like Tesla V100 provides 3X more compute power (15 TFLOPS) than 3 used GPUs together). The Fig. 15 demonstrates simulation performance (it uses the same data structure and algorithms, so it exhibits the same performance properties as trajectory generation software) for various amount of available computational resources (it was measured by using various combinations of GPUs). It does not make sense to compare this software running on CPUs with GPUs since it was specifically designed for GPU architecture and CPUs will provide 2–3 orders of magnitude worse performance.

The following pictures (Fig. 16–18) demonstrate simulation and machining results for various test models and materials. It is easy to see the finishing path for the "Head" model on the Fig. 12 as the path is generated by intersecting the offset surface with the filling curve, so the path follows the curve.

Some of test parts are not completely finished due to various reasons. The "Fan" model on Fig. 17 contains

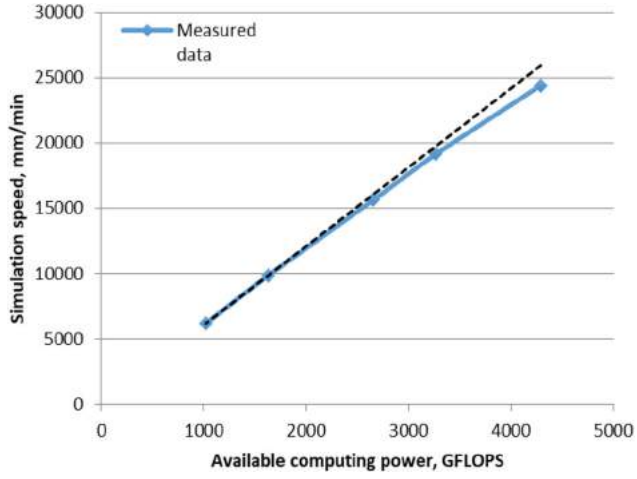

Figure 15. Simulation scalability.

material that was not completely removed due to a limited availability of fixture and milling tools at the time of the test. Basically, it was not possible to remove material safely under the fan using single setup with fixtures that were available. It is also important that the tool path planning algorithm has managed to consider this limitation and generate a tool path that removed all possible material on top of fan blades without overcuts. Considering two other models, entire tool paths were generated for both models but finishing tool paths were not machined completely due to significant time requirements. As result, the machined parts have different surface quality in different areas as shown. Considering described limitations of the testing process, the described algorithms have successfully generated valid tool paths for machining reasonably complicated geometry in a completely automated fashion almost without human interaction. It is important to note that the

(15 minutes planning type; 3 hours machining time)

Figure 16. Test model "head."

(15 minutes planning type; 1.5 hours machining time)

Figure 17. Test model "fan."

developed system does not require from a user to select a particular path mode (zig-zag, spiral, etc) but in most cases the finishing path will look like a contour-offset path (Fig. 12) while roughing path will look like a zig-zag path at the beginning and a more complicated trajectory at the end (Fig. 14). Since the developed system construct both roughing and finishing, the user is required to specify maximum scallop height that will be used to determine distance between two consecutive finishing passes.

(10 minutes planning type; 1.5 hours machining time)

Figure 18. Test model "puppy."

10. Conclusion

This paper has described the design methodology for a set of highly parallel algorithms for determining a sequence of contact points for 5-axis tool path. These algorithms include a solution for common computational geometry problems, such as offset surface calculations or volume surface intersections and a set of robust algorithms for multi-axis tool path planning used in CNC milling. Following the design methodology and using the developed highly parallel geometry representation have resulted in high parallelizability and scalability of these algorithms. These contact point sequence algorithms were combined with orientation determining algorithms discussed in a second paper to produce an automatic path planning system. The experimental results have demonstrated that a GPGPU approach can be used for acceleration and automation of the tool path planning process for CNC milling machines.

ORCID

Dmytro Konobrytskyi http://orcid.org/0000-0002-8700-4789
Mohammad M. Hossain http://orcid.org/0000-0003-4006-2831
Thomas M. Tucker http://norcid.org/0000-0002-4069-8269
Joshua A. Tarbutton http://orcid.org/0000-0002-0592-1971
Thomas R. Kurfess http://orcid.org/0000-0003-2356-9622

References

[1] Balasubramaniam, M.; Joshi, Y.; Engels, D.: Tool Selection in Three-Axis Rough Machining, International Journal of Production Research, 39(18), 2001, 4215–4238. https://doi.org/10.1080/00207540110055389

[2] Borrow, J.: NC Machine Tool Path Generation from CSG Part Representations? Computer-Aided Design, 17(2), 1985, 69–76. https://doi.org/10.1016/0010-4485(85)90248-9

[3] Choi, B. K.; Kim, D. H.; Jerard, R. B.: C-Space Approach to Tool-Path Generation for Die and Mould Machining, Computer-Aided Design, 29(9), 1997, 657–669. https://doi.org/10.1016/S0010-4485(97)00012-2

[4] Ding, S.; Mannan, M. A.; Poo, A. N.: Adaptive Iso-Planar Tool Path Generation for Machining of Free-Form Surfaces, Computer-Aided Design, 35(2), 2003, 141–153. https://doi.org/10.1016/S0010-4485(02)00048-9

[5] Fan, J.; Ball, A.: Quadric Method for Cutter Orientation in Five-Axis Sculptured Surface Machining, International Journal of Machine Tools and Manufacture, 48(7–8), 2008, 788–801. https://doi.org/10.1016/j.ijmachtools.2007.12.004

[6] Giri, V.; Bezbaruah, D.; Bubna, P.: Selection of Master Cutter Paths in Sculptured Surface Machining by Employing Curvature Principle, International Journal of Machine Tools and Manufacture, 45(10), 2005, 1202–1209. https://doi.org/10.1016/j.ijmachtools.2004.12.008

[7] Gray, P.; Bedi, S.; Ismail, F.: Rolling Ball Method for 5-Axis Surface Machining, Computer-Aided Design, 35(4), 2003, 347–357. https://doi.org/10.1016/S0010-4485(02)00056-8

[8] Gray, P. J.; Ismail, F.; Bedi, S.: Graphics-Assisted Rolling Ball Method for 5-Axis Surface Machining, Computer-Aided Design, 36(7), 2004, 653–663. https://doi.org/10.1016/S0010-4485(03)00141-6

[9] Han, Z.; Yang, D. C. H.: Iso-Phote Based Tool-Path Generation for Machining Free-Form Surfaces, Journal of Manufacturing Science and Engineering, 121(4), 1999, 656–664. https://doi.org/10.1115/1.2833094

[10] Hsieh, H.; Chu, C.: Particle Swarm Optimisation (PSO)-Based Tool Path Planning for 5-Axis Flank Milling Accelerated by Graphics Processing Unit (GPU), International Journal of Computer Integrated Manufacturing, 24(7), 2011, 676–687. https://doi.org/10.1080/0951192X.2011.570792

[11] Hwang, J.: Interference-Free Tool-Path Generation in the NC Machining of Parametric Compound Surfaces, Computer-Aided Design, 24(12), 1992, 667–676. https://doi.org/10.1016/0010-4485(92)90022-3

[12] Jensen, C. G.; Red, W. E.; Pi, J.: Tool Selection for Five-Axis Curvature Matched Machining, Computer-Aided Design, 34(3), 2002, 251–266. http:// doi.org/10.1016/S0010-4485(01)00086-0

[13] Lasemi, A.; Xue, D.; Gu, P.: Recent Development in CNC Machining of Freeform Surfaces: A State-of-the-Art Review, Computer-Aided Design, 42(7), 2010, 641–654. https://doi.org/10.1016/j.cad.2010.04.002

[14] Lauwers, B.; Kiswanto, G.; Kruth, J.: Development of a Five-Axis Milling Tool Path Generation Algorithm Based on Faceted Models, CIRP Annals - Manufacturing Technology, 52(1), 2003, 85–88.

[15] Lee, Y.; Ji, H.: Surface Interrogation and Machining Strip Evaluation for 5-Axis CNC Die and Mold Machining, International Journal of Production Research, 35(1), 1997, 225–252. https://doi.org/10.1080/00207549719606 4

[16] Li, J. G.; Ding, J.; Gao, D.: Quadtree-Array-Based Workpiece Geometric Representation on Three-Axis Milling Process Simulation, The International Journal of Advanced Manufacturing Technology, 50(5), 2010, 677–687. https://doi.org/10.1007/s00170-010-2530-9

[17] Li, S.; Jerard, R.: 5-Axis Machining of Sculptured Surfaces with a Flat-End Cutter, Computer-Aided Design, 26(3), 1994, 165–178. https://doi.org/10.1016/0010-4485(94)90040-X

[18] Loney, G.; Ozsoy, T.: "NC Machining of Free Form Surfaces," Computer-Aided Design, 19(2), 1987, 85–90. https://doi.org/10.1016/S0010-4485(87)80050-7

[19] Morishige, K.; Kase, K.; Takeuchi, Y.: Collision-Free Tool Path Generation using 2-Dimensional C-Space for 5-Axis Control Machining, The International Journal of Advanced Manufacturing Technology, 13(6), 1997, 393–400. https://doi.org/10.1007/BF01179033

[20] Morishige, K.; Takeuchi, Y.; Kase, K.: Tool Path Generation using C-Space for 5-Axis Control Machining, Journal of Manufacturing Science and Engineering, 121(1), 1999, 144–149. https://doi.org/10.1115/1.2830567

[21] Ren, Y.; Yau, H. T.; Lee, Y.: Clean-Up Tool Path Generation by Contraction Tool Method for Machining Complex

Polyhedral Models, Computers in Industry, 54(1), 2004, 17–33. http:// doi.org/10.1016/j.compind.2003.09.003

[22] Rong-Shine Lin, Y. K.: "Efficient Tool-Path Planning for Machining Free-Form Surfaces," Transactions of the ASME, 118(1), 1996, 20–28.

[23] Seong, J.; Elber, G.; Kim, M.: Trimming Local and Global SelfIntersections in Offset Curves/Surfaces using Distance Maps, Computer-Aided Design, 38(3), 2006, 183–193. https://doi.org/10.1016/j.cad.2005.08.002

[24] Sun, W.; Bradley, C.; Zhang, Y. F.: Cloud Data Modelling Employing a Unified, Non-Redundant Triangular Mesh, Computer-Aided Design, 33(2), 2001, 183–193. https://doi.org/10.1016/S0010-4485(00)00088-9

[25] Suresh K, Y. D.: Constant Scallop Height Machining of Free Form Surfaces, Journal of Engineering for Industry, 116(12), 1994, 253–259. https://doi.org/10.1115/1.2901938

[26] Tukora, B.; Szalay, T.: Real-Time Determination of Cutting Force Coefficients without Cutting Geometry Restriction, International Journal of Machine Tools and Manufacture, 2011, 51(12), 871–879. https://doi.org/10.1016/j.ijmachtools.2011.08.003

[27] Wang, C. C. L.; Leung, Y.; Chen, Y.: Solid Modeling of Polyhedral Objects by Layered Depth-Normal Images on the GPU, Computer-Aided Design, 42(6), 2010, 535–544. http:// doi.org/10.1016/j.cad.2010.02.001

[28] Warkentin, A.; Ismail, F.; Bedi, S.: Multi-Point Tool Positioning Strategy for 5-Axis Mashining of Sculptured Surfaces, Computer Aided Geometric Design, 17(1), 2000, 83–100. https://doi.org/10.1016/S0167-8396(99)00040-0

[29] Yang, D. C. H.; Chuang, J. J.; Han, Z.: Boundary-Conformed Toolpath Generation for Trimmed Free-Form Surfaces Via Coons Re-parametrization, Journal of Materials Processing Technology, 138(1–3), 2003, 138–144. https://doi.org/10.1016/S0924-0136(03)00062-1

[30] Yang, D. C. H.; Chuang, J.; OuLee, T. H.: Boundary-Conformed Toolpath Generation for Trimmed Free-Form Surfaces, Computer-Aided Design, 35(2), 2003, 127–139. https://doi.org/10.1016/S0010-4485(02)00047-7

[31] Yang, D. C. H.; Han, Z.: Interference Detection and Optimal Tool Selection in 3-Axis NC Machining of Free-Form Surfaces, Computer-Aided Design, 31(5), 1999, 303–315. https://doi.org/10.1016/S0010-4485(99)00031-7

[32] Yuan-Shin, L.: Admissible Tool Orientation Control of Gouging Avoidance for 5-Axis Complex Surface Machining, Computer-Aided Design, 29(7), 1997, 507–521. https://doi.org/10.1016/S0010-4485(97)00002-X