2016

Automated Tool Selection and Tool Path Planning for Free-Form Surfaces in 3-Axis CNC Milling using Highly Parallel Computing Architecture

Andrey Balabokhin
University of South Carolina

Automated Tool Selection and Tool Path Planning for Free-Form Surfaces in 3-Axis CNC Milling using Highly Parallel Computing Architecture

by

Andrey Balabokhin

Bachelor of Science
Omsk State University, 2006

Master of Science
Omsk State University, 2008

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Mechanical Engineering

College Of Engineering and Computing

University of South Carolina

2016

Accepted by:

Joshua Tarbutton, Major Professor

Ramy Harik, Committee Member

Dmytro Konobrytskyi, Committee Member

Stephen McNeill, Committee Member

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

Acknowledgments

I would like to thank Dr. Tarbutton for believing in me for the last four years and for helping me to do the proper things and to understand how to be a scientist. I would also like to thank the staff and students at McNair who supported my work as well as all my friends and family who did not hear from me as often as they should.

Abstract

This research presents a methodology to automatically select cutters and generate tool paths for all stages in 3-axis CNC Milling of free-form surfaces. Tools are selected and tool paths are planned in order to minimize the total machining time. A generalized cutter geometry model is used to define available cutters and an arbitrary milling surface is initially defined by a triangular mesh.

The decisions made by process engineers in selecting cutting geometry and generating tool paths for milling dramatically influence the final result. Often, the resulting tool path is non-optimal, because the engineers cannot consider all the available information. However, making these decisions can be delegated to a computing system that can find a better result.

The developed methodology selects the cutters to use for milling from the set of all available cutters, assigns milling zones to every selected cutter, based on its performance, and builds iso-scallop and contour parallel tool paths for every cutter and its milling zone. After generating all tool paths for both milling stages (rough milling and finishing), the tool selection sequence is defined and all the tool paths for one tool are connected into the single tool path. The tool paths should be connected in the best possible manner in order to minimize the time of CNC non-cutting motions. This is similar to the travelling salesman problem with constraints. A heuristics solution is provided here. At the end, the total machining time for one tool set is calculated. Finally, the set of cutters used is changed to minimize the total machining time.

A digital, voxel-based model is used to represent a workpiece and the available

tools. This model is selected so that the algorithms is simpler and they can be easily paralleled for thousands of computing cores. The parallel processing framework is implemented to work with multiple graphics processing units. Tool paths generated from this framework are post-processed into G-code and the representative part is machined.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the introduction of Computer Numerical Control (CNC) machines it is possible to mill almost any free-form surface in all industries and fields. Process engineers rely on years on manufacturing experience to select cutting parameters in order to machine a part. These parameters include material piece selection, cutting conditions, tool materials, tool geometry, tool sequence, cutting fluid selection, tool path, etc. The selection of these parameters influences the machining cost, surface quality and the machining time. Traditionally, CAM software transfers the task of selecting most of these parameters to the operator.

In this work we propose a method to automatically select cutters and generate tool paths for 3-axis CNC milling of free-form surfaces in order to minimize the total machining time. The methodology includes assigning milling zones for the cutters, based on their performance, and generating the contour parallel, iso-scallop tool paths for every selected cutter and its milling zones for all stages of milling. Then, the set of cutters is changed to minimize the total machining time. The machining is comprised of milling time, rapid motion total time and the time to change cutters. The methodology to select the order of the cutters to mill and to connect tool paths on different milling layers for every used cutter is also provided.

The input for the method is composed of an arbitrary part geometry (defined by a triangular mesh), a desired tolerance size, a set of available generalized cutters with their milling parameters (feed rate and depth of cut) and CNC parameters (acceleration/deceleration, the average time to change a cutter and the rapid feed rate). The output is a G-code for the optimal tool path and the optimal set of cutters to use. The discretized, voxel-based representation is used for the part and the tools. The discretized model is used to make calculations of different surface point characteristics (tool performance, maximum tool depth without gouging, etc.) easier. Because of the complexity of some of the algorithms used in this work, several graphics processing units (GPUs) are employed to accelerate calculations. The time-consuming algorithms are designed in such a way, that they can be easily paralleled on thousands of cores for several GPU devices, clusters or int the cloud. The main steps for the whole approach to minimize the total machining time are shown in Fig. 1.1. The main idea behind the algorithm is that different sets of available cutters lead to different machining times. Thus, by changing the set of available cutters the machining time can be minimized.

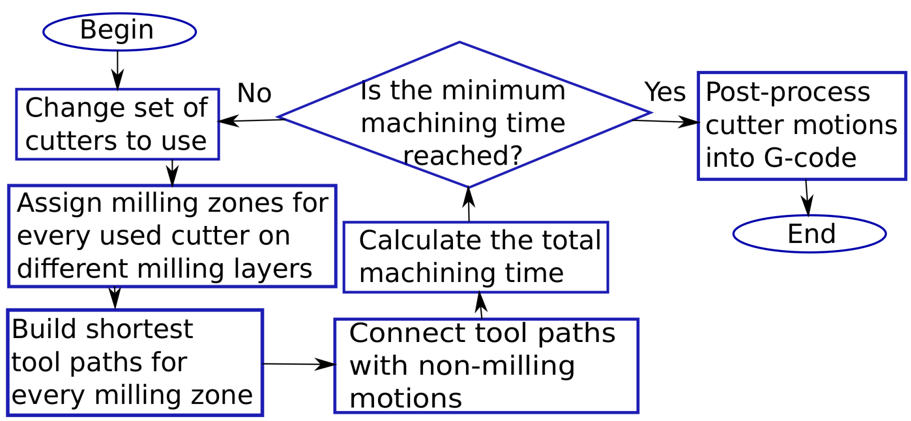

Figure 1.1: Main steps to minimize the total machining time

The main algorithms in the approach are:

- Assigning milling zones for the available cutters, based on their performance
- Building the shortest tool paths for every milling zone and its cutter on different milling layers
- Connecting tool paths together and calculating the total machining time

Each of these algorithms are addressed in corresponding chapters. Chapter 2 provides the background for the tool path generating approaches, tool selection approaches and general-purpose computing on a GPU (GPGPU) used in CNC simulation and related areas. Chapter 3 explains the methodology to select cutters for finishing, assigning milling zones for the cutters based on their performance, the part and tool representation models, and the result of the algorithm of building milling zones for the representative tool set and the part. Chapter 4 describes the method for generating tool paths for the finishing stage if milling. Chapter 5 shows the adaptation of the main proposed algorithms (assigning zones for milling and tool path planning) for the rough milling stage. Chapter 6 describes the methodology to optimize the tool set in order to minimize the total machining time. Chapter 7 shows the results of the milling and provides the comparison of the milled parts for the developed g-code and traditional CAM generated g-code. Finally, Chapter 8 discussed the provided approach with its advantages and disadvantages and gives the potential improvements for the approach.

Chapter 2

Background

Free-form surfaces have been widely used in aerospace, automobile and the die/mold industry. A free-form surface is often defined as a composition of non-planar and non-quadratic surfaces [5]. There are many ways to define a surface, e.g. CAM package usually uses parametric surfaces. Another way to define a surface is to use a triangular mesh. Having small enough polygons in the mesh, the milling surface can have an arbitrary shape. In this work, the milling surface is defined with an arbitrary triangular mesh.

Milling the exact geometric surface is impossible for non-trivial surfaces, so the concept of surface tolerance is used to measure the required quality of the machined surface. Traditionally, for a surface to be machined, the scallop height should not exceed the maximum allowed tolerance. More broadly, the tolerance size can be defined as a maximum distance between any point on the milled surface and the closest point on the precise designed geometrical surface. As this definition is the more general one, it is used in this research. The other requirement on the milled surface is that it should be gouge free.

Stages to complete surface machining can be classified into rough, semi-finish, finish and clean-up [26] or into rough, finish and clean-up [37]. For the rough milling stage, bigger tools are used to remove as much material as possible. At the finishing stage, the "rough" surface is milled with smaller tools to reach the desired shape. In this work, milling stages are defined using a tolerance surface and it is enough to distinguish two stages of milling: rough-milling and finishing.

Cutter location points, tool shape and size, spindle speed, the tool feed rate and other parameters need to be considered to machine a free-form surface with a 3-axis CNC machine. The issues of selecting all these parameters are related to each other. However, it is difficult to solve the whole problem considering the optimality of all relevant aspects, such as path pattern, tool geometry selection, feed rate scheduling as well as the other objectives. As a result, many researches try to solve these problems separately. The background for the two most important issues (tool path generation and tool geometry selection) is shown in next sections.

Recently, the computational power of computers grew dramatically. Instead of increasing the performance of one computer, the trend now is to use parallel computing, such as multi-core processors, clouds, mainframes, GPUs, etc. To use multiple cores, new approaches from different research areas can be used in CNC milling. For example, image processing techniques were used by Mario et al. [30] to find optimal tools set for 2.5D milling. Tarbutton et al. [40] proposed a method of building tool path using GPU computing for voxel model.

2.1 Tool Path Generation

Tool path planning is an important issue in machining surfaces since it directly influences machining time. Different stages of milling require different approaches to achieve the optimal milling time and quality. For the finishing stage the usual goal is to minimize the machining time while the maximum scallop height remains below the predefined level. Smaller scallop height usually leads to longer tool paths, causing a trade-off between the precision and the machining time.

Approaches to generate a tool path entirely depends on the geometry representation for milling surfaces. Traditionally, modern CAD/CAM systems use Boundary Representation (b-rep) to represent the geometry. B-rep approach represents a boundary between the part material and empty spaces. Surface elements is usually defined by Non-uniform rational Basis splines (NURBS) surfaces or other analytical surface representation. Traditionally, CAD/CAM systems converts B-rep represented models into parametric surfaces in order to generate tool paths.

Tool path planning is composed of two aspects: path topology and path parameters. Path topology is defined by the pattern of moving the tool to produce the surface, and the path parameters are modeled by the side step between successive paths and the forward step in each tool path.

Two path topologies are most widely used for milling free-form surfaces: contour parallel and direction parallel. In a direction parallel path topology, path segments are parallel to a predefined line. This line could be parallel with or normal to the surface boundary or parallel to the axis of the specified coordinate system. Proper selection of the reference line directly affects the generated tool path length [39, 2]. The optimum path direction will result in longer individual paths and the minimum non-cutting movements of the cutting tools. A specific type of direction parallel path, zigzag path, is commonly used in CAM systems for roughing [29].

A contour parallel path is constructed by the boundary curves of the surface. Traditionally, each next path is built by offsetting the boundary of the surface. It can be built using the Voronoi diagram, pair-wise offsetting [19], or a pixel based approach [21]. The paths in the pattern could be spirally connected after building. Even though the optimal path topology depends on the surface and tool geometries, after comparing the machining time for direction- and contour- parallel paths, considering acceleration and deceleration [31, 20], Kim and Choi [20] mentioned that zigzag paths usually result in longer tools paths than contour parallel paths.

Traditionally, iso-parametric [15], iso-planar [3] and iso-scallop height [22] methods are used for tool path generation. In the iso-parametric method, by keeping one of the two parameters constant, contact points are generated along the other parameter of a parametric surface $S(u, v)$. The main drawback of the method is that while mapping a line in the parametric domain into Euclidean space, a constant step in the parametric domain can lead to unequal path intervals between adjacent paths in Euclidean space. Even though this method works only for parametric surfaces, the parametric surface can be reconstructed from a triangular mesh surface [46].

An iso-planar tool path is developed by intersecting the surface with parallel planes. The distance between parallel planes is decided based on the maximum scallop height constant. This method is very robust and can be used for parametric surfaces and triangular meshes. However, proper selection of the intersecting planes affects tool path length and the machining time. It is evident, that as in the iso-parametric method, the iso-planar method leads to unequal scallop height for adjacent tool paths.

In iso-scallop tool paths, the next cutter contact point is calculated based on the previous cutter contact point, so that the scallop height remains the same all over the surface. The iso-scallop method is considered to generate the shortest overall tool path for a given scallop height, however it is computationally expensive. Many researchers [9, 6, 10, 41] addressed the problem of generating iso-scallop tool path for paramertric surfaces. Usually, ball-end or flat-end cutters are used, however Chiou at al. [9] studied the problem for a generalized cutter.

In spite of the fact that the vast majority of CAD/CAM systems use parametric surface representation to build tool paths, tool path planning for triangular mesh models (also called polyhedral, tessellated or faceted models) have also become popular in CAD/CAM systems. These models can be created from parametric surfaces, cloud points or designed in dedicated software directly. Nearly all CAD/CAM packages have tessellation algorithms that produce tessellated surfaces with the desired surface accuracy. The tessellation algorithms are robust and capable of combining any number of surfaces into a single triangular mesh. Sometimes, machining of non-parametric or non-implicit surfaces is inevitable, e.g. if the surface is reconstructed from cloud points. Thus, the tool path algorithms should be adapted for polyhedral models. All basic tool-path generation methods can be used with polyhedral models: iso-parametric [46], iso-planar [34, 7] and iso-scallop [25, 27].

Recent research emphasis has shifted from 3-axis milling to 5-axis milling. Many researchers addressed the problem of gouge-free tool path generation for triangular meshes milling [4, 14] and for parametric surfaces [17, 47] for 5-axis milling. However, the tools of simple shapes are usually used: ball-end, flat-end and radiused cutters.

2.2 Tool Geometry Selection

Tool geometry selection is a process that is usually performed by the operator. It plays an important role among other milling parameters. If the tool is selected incorrectly, it can lead to dimensional errors. However, the main objective for tool geometry selection is to reduce machining cost. Machining cost primary depends on machining time and tool cost.

Different approaches of tool geometry selection can be applied for different stages of milling: roughing and finishing. The goal of rough milling is to remove the excess material as fast as possible. The usual method for roughing free-form surfaces is a layer-by-layer approach. The largest possible cutter is selected for every layer and the optimal set of tools is selected for all the layers [42]. Flat-end tools are most widely used for rough milling [24].

As for roughing, the main objective for finishing is to minimize machining time. However, the goal in this case is to traverse the whole milling surface with the set of available tools as fast as possible, so that the finished surface is gouge free and within some tolerance limit. The traditional approach for finishing is to calculate the minimum radius of surface curvature and match it with the largest possible tool. For tools of the same shape (e.g. ball-end tools), the feasible regions for every tool can be detected, and the optimal set of tools can be built [13, 44]. If the tools of different shapes are employed for milling, the feasible regions are not enough to build the optimum set of tools. To guarantee that the best tool for the different surface zones is used, the performance of different tools in different cutter locations should be assessed. This was partially done by Patel et al. [35]. They provided the metrics to calculate the finishing area at a point for ball-end and radiused cutters in predefined cutter location points. Tool performance in a cutter location point depends not only the tool shape but also on milling direction. The issue of optimizing the tool path based on milling direction in different cutter location points was also studied [23, 28].

In addition, the problem of tool selection and tool set optimization was addressed for some special cases. The methodology to select cutters and optimize the tool set was introduced for complex 2.5D parts [32] and for complex islands [48]. The approach to select tools and build tool paths for impeller channels was also shown by Han et. al. [18].

In this work, tools are assigned to different milling zones based on their performance at different CLs. The performance criterion is defined based on the tool and the part geometries for both milling stages (finishing and roughing). As a consequence, the tool paths generated for different milling layers can be treated in a uniform manner. This approach allows direct connection of all the tool paths simplifying the global tool set optimization strategy to a single pass of connected paths.

Chapter 3

Tool selection for finishing[1]

Finish milling can take significant time, and the set of selected cutters dramatically affects the milling time. Even more important is to use the most efficient cutters to machine different zones on the milling surface. In this chapter, the problem of assigning the milling zones for the provided cutters, based on their performances, is addressed. First, the general criteria for cutter selection is shown. Then, data structures for tools and part representation are provided. Note, that the same data structures are used later in other chapters, e.g for tool path building algorithms. At the end of the chapter, the result of the proposed method and a performance analysis for the most time consuming algorithms are shown. In Chapter 5 we provide the adaptations of the algorithms from this chapter to rough milling stage.

3.1 Criteria in cutter selection

The factors that influence cutter selection include the following: production cost, geometric constraint, machining quality, cutting tool life, machining accuracy and machine tool performance. In this work, we focus on production cost, geometric constraint and machining quality parameters. The material left for the finishing stage is assumed to be small enough that the CNC machine can support the given feed rates for all the available tools. Therefore, the formulation of the optimization

[1]Andrey Balabokhin and Joshua Tarbutton. "Generalized Cutter Selection for Finishing of Free-Form Surfaces in 3-Axis CNC Milling by "Surface Tolerance and Tool Performance Metrics"". Submitted to *International Journal of Advanced Manufacturing Technology*, 04/03/2016.

problem is to achieve the required machining quality with the minimum production cost while geometric constraints are satisfied.

Surface quality has a direct effect on machining time. Machining scallop height is usually used to define the surface quality. The distance between two adjacent tool paths is called the step over size (g); the unmachined material is called scallop, and scallop height (h) is defined as the max height of the scallop. The scallop height depends on the tool geometry, milling surface geometry and step over size. Because the scallop height defines the milling surface quality, the step over size is calculated using the known scallop height, tool geometry and milling surface geometry. Fig. 3.1 shows the different values of the scallop heights with constant step-over, but different tool and part geometries.

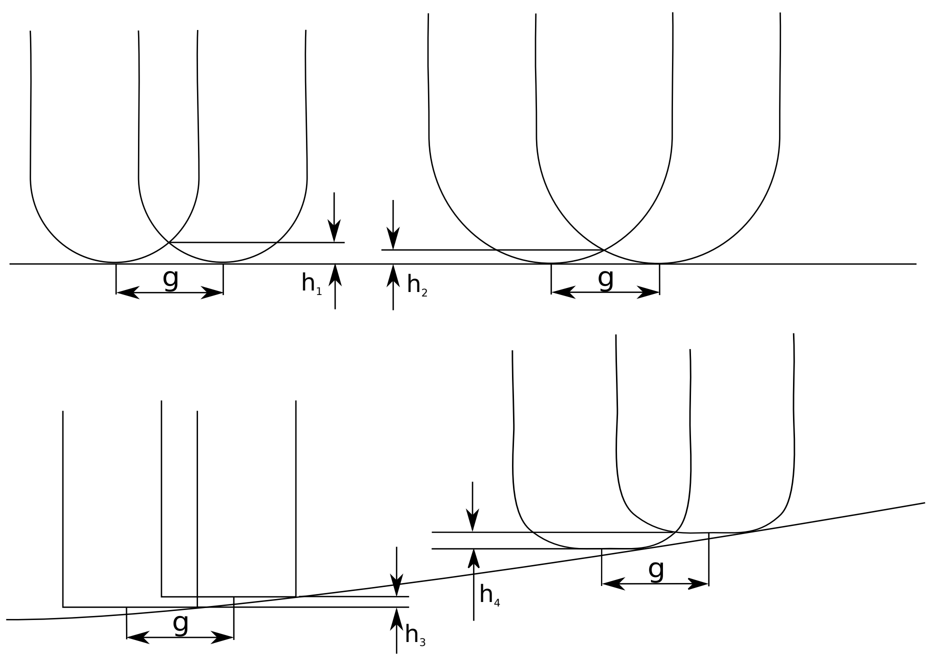

Figure 3.1: Variable scallop heights for different tools and milling surfaces configurations

Due to the fact that the actual tool path is not built in this chapter, the tolerance size is used as a criterion of surface quality instead of scallop height. The geometrical meaning of tolerance size is the maximum distance between any point in the milled surface and the closest point in the anticipated geometrical surface. Overall, tolerance size is similar to scallop height, with the exception that no tool path is needed for this metric.

The production cost of machining includes machining cost and tooling cost. The cost of the cutting tool is usually a very small portion in the total machining cost, so the machining cost is mostly determined by the production time. The cost of the cutting tool is not covered in this work. Therefore, the key criterion in cutter selection is to machine a workpiece with the minimum production time. The production time comprises milling time for every used tool and the time of changing tools. Thus, the first step is to define the optimal tool for every point of a milling surface from the set of available tools with their feed rates. In this work the optimal tool for a surface point is defined to be the tool that can mill the bigger surface area per unit of time (including the surface point) than any other available tool. Using this formulation, the total number of tool positions to mill the whole surface can be minimized which reduces production time.

In this research various tools with different geometries are tested for the given part geometry. For 3-axis milling the part and tool surface geometries can be represented as a function ($z(x, y)$), that defines z surface coordinate based on the x and y coordinates. The part and the tool are considered to locate above $z = 0$ plane and the tool direction is a negative z-axis direction. Fig. 3.2 shows such functions for a part and a tool in 2D.

For instance, for a ball-end tool the surface equation is:

$$z(x,y) = \begin{cases} \sqrt{r^2 - x^2 - y^2}, & x^2 + y^2 \leq r^2 \\ \infty, & otherwise \end{cases}$$

Overall, the input parameters for the algorithm to assign milling zones for the given tool set can be defined as follows:

- $z_p(x, y)$ - part surface geometry

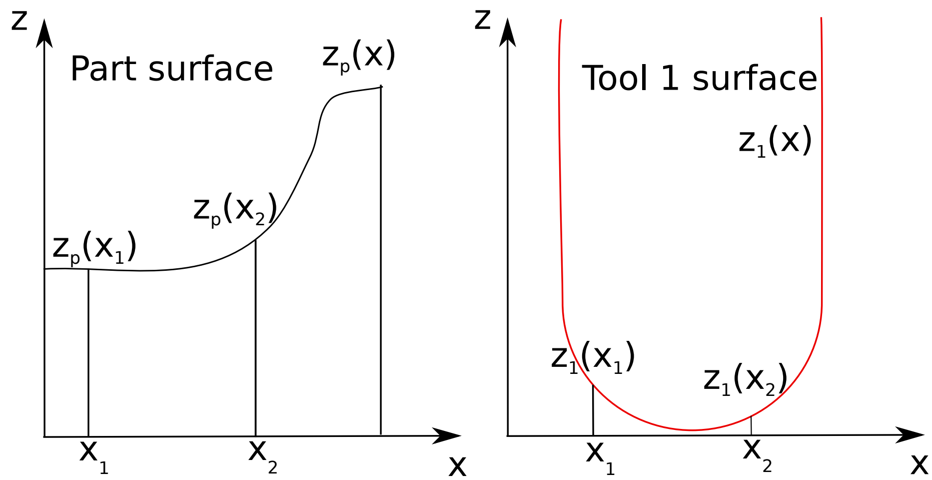

Figure 3.2: Part and tool surfaces as $z(x)$ functions

- h - maximum tolerance size
- N - number of tools
- $[z_i(x, y)]$ - set of tool geometries
- $[f_i]$ - set of corresponding tool feed rates
- $[r_i]$ - set of corresponding tool radii

The output can be expressed in the form of a tool index function $T(x, y) = T(z_p, h, N, [z_i], [f_i], [r_i], x, y)$. This returns the optimal tool index for a surface point (x, y) considering all the input parameters. To express $T(x, y)$ function, some other functions need to be introduced.

The first function describes the tolerance surface: $z_t(x, y)$. This surface bounds the volume, and all the material outside this volume must be machined. This function can be any function with the property:

$$\forall x, y : \begin{cases} \min_{\bar{x},\bar{y}}[l(x - \bar{x}, y - \bar{y}, z_t(x, y) - z_p(\bar{x}, \bar{y}))] \leq h \\ z_p(x, y) \leq z_t(x, y) \end{cases} \tag{3.1}$$

where $l(dx, dy, dz) = \sqrt{dx^2 + dy^2 + dz^2}$

In other words, the distance from any tolerance surface point $(x, y, z_t(x, y))$ to the closest point on the part surface should be less or equal than the tolerance value

(h), and the tolerance surface should be above the milling surface. More than one function with this property can be built. The bigger the volume is bounded by the tolerance surface the less volume need to be milled. For the ideal tolerance surface, the minimum distance from every tolerance surface point to the milling surface equals h. The easiest way to build the tolerance surface is to extend milling surface up on h, so the equation for z_t is:

$$z_t(x, y) = z_p(x, y) + h$$

However, it can be done more efficiently, by expanding z_p in all the directions. The exact way to build the tolerance function depends on the part surface representation. For the surface representation in this work, way to build the tolerance function is described later.

The second function is a tool maximum depth without gouging ($d_i(x, y)$) for a cutter location (CL) point (x, y). It depends only on the tool and the surface geometries as denoted in Equation 3.2. In other words, after going through all the surface points under the tool in CL position (x, y), the deepest z position of the tool without gouging is defined as the maximum of the difference between z values of the surface point and the tool point above. The point that has this maximum value is the cutter contact (CC) point. The maximum depth of cutter without gouging is shown below in Fig. 3.3.

$$d_i(x, y) = \max_{\substack{x_l \in [-r_i, r_i] \\ y_l \in [-r_i, r_i]}} z_p(x + x_l, y + y_l) - z_i(x_l, y_l) \tag{3.2}$$

Then, the "finished area" ($A_i(x, y)$) function needs to be calculated. "Finished area" is the area of the surface that can be milled by the tool in the given CL positions within the tolerance surface. It depends on both tool and part geometries and on tolerance surface function as shown in Equation 3.3. This equation denotes that the area of every surface point $(x + x_l, y + y_l)$ that can be milled from the CL point

$(x, y, d_i(x, y))$ within the tolerance surface is added to the total "finished area" of the CL point.

$$A_i(x, y) = \iint_{(x_l, y_l) \in S_i} \begin{cases} dx_l dy_l, & h_1 + h_2 \leq h_3 \\ 0, & otherwise \end{cases} \tag{3.3}$$

where

S_i the circle with the center at the origin and the radius r_i,

h_1 $= d_i(x, y)$, cutter depth in a cutter location point,

h_2 $= z_i(x_l, y_l)$, the height of the tool surface point,

h_3 $= z_t(x + x_l, y + y_l)$, the height of the tolerance surface.

Fig. 3.3 shows the tolerance surface, the tool depth and the finished area in a cutter location for a complex surface and the ball-end tool in 2D.

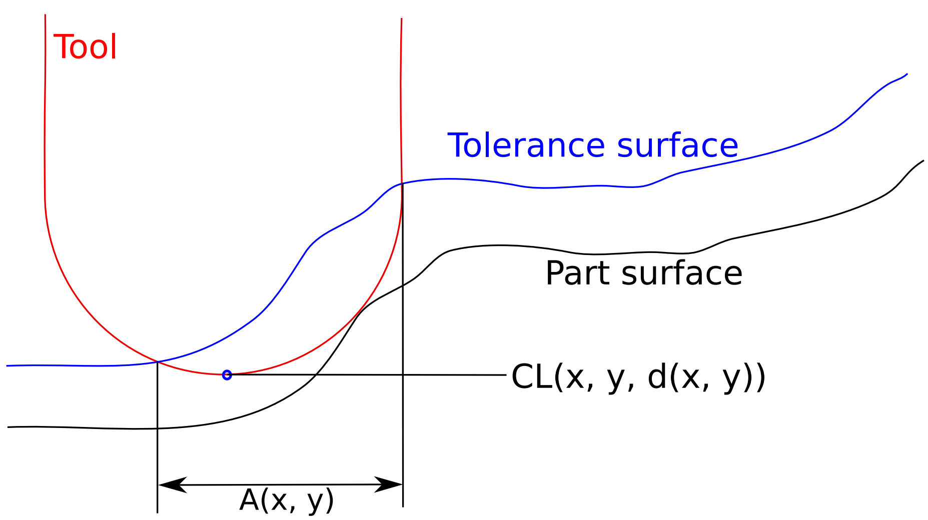

Figure 3.3: The maximum depth of cutter and its "finished area" for a CL point(x, y) for the tolerance surface

Now, the tool performance as a function of "finished area" and the tool feed rate can be considered. A finishing tool performance in a CL point can be calculated as shown in Equation 3.4. The bigger the performance value in a cutter location point, the better. Thus, the performance criterion for a cutter location states: if one tool can traverse a bigger area from the given cutter location point than another tool per unit of time, the first tool is better. The unit of the finishing performance is cubic meters per second. In this work, different performance values are compared to each other and are not used to produce other metrics.

$$P_i(x, y) = A_i(x, y) * f_i \tag{3.4}$$

To perform finishing milling, the material in every surface point must be milled within the tolerance surface ("covered" by a tool). So, with a given surface point (x, y) the set of all cutter locations (Ω) can be found. This set depends only on tool and part geometries and the tolerance surface function and can be calculated using Equation 3.5. Such a set for a complex milling surface and a ball-end tool is shown in Fig. 3.4.

$$\Omega_i(x, y) = \bigcup_{(x_l, y_l) \in S_i} \begin{cases} (x + x_l, y + y_l), & h_1 + h_2 \leq h_3 \\ \emptyset, & otherwise \end{cases} \tag{3.5}$$

where

S_i the circle with the center at the origin and the radius r_i,

h_1 $= d_i(x + x_l, y + y_l)$, cutter depth in a cutter location point,

h_2 $= z_i(-x_l, -y_l)$, the height of the tool surface point,

h_3 $= z_t(x, y)$, the height of the tolerance surface.

The function to find the best cutter $T(x, y)$ for the surface point (x, y) can be defined as in Equation 3.6. This function is used the set of CL positions $(\Omega_i(x, y))$ that can "cover" every surface point (x, y) and the finishing performance function $P_i(\bar{x}, \bar{y})$ for every CL position $(\bar{x}.\bar{y})$. Thus, for a given surface point, among all cutter locations of all the tools that can "cover" the point, the tool with the best performance from any cutter location is selected for the point. This is because the performance reflects the size of the milled area; the bigger the area, the more other surface points

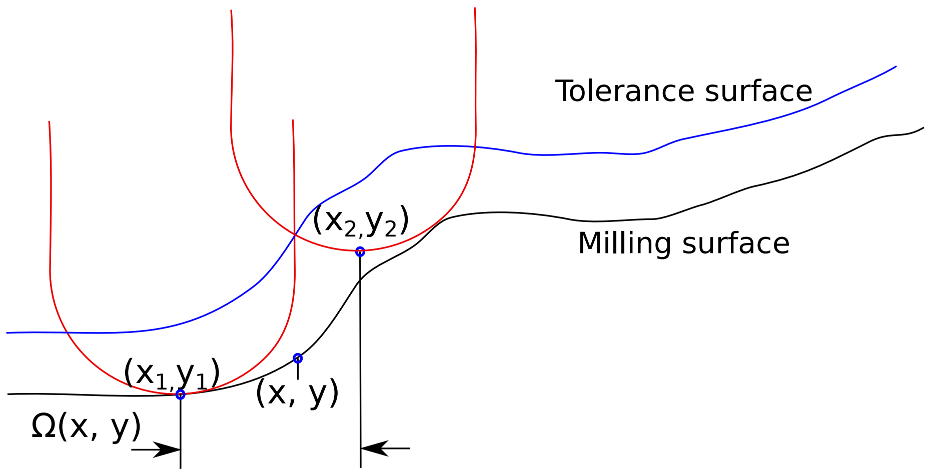

Figure 3.4: Cutter locations set to cover a surface point:
$\Omega(x, y) = (x_1, y_1) \cup (x_2, y_2) \cup ...$

can be "covered" from the same cutter location and the less overall cutter location points are needed to traverse the whole milling surface.

$$T(x, y) = \arg\max_{i=1..N} \max_{(\bar{x},\bar{y})\in\Omega_i(x,y)} P_i(\bar{x}, \bar{y}) \tag{3.6}$$

In reality, optimizing milling time is a more complicated problem than a "coverage" problem, described here. More parameters influences it, such as cutter swipe direction and the exact selection of CL points. However, the given approach of defining the best tool for every surface point is the fitting first step for further optimizations. The main hypothesis is that neighboring surface points have identical best tool indices and the whole surface can be divided into milling zones. As shown below, the whole surface can be zoned with a low frequency pattern.

In this section $\Omega_i(x, y)$, $P_i(x, y)$, $d_i(x, y)$ and $z_t(x, y)$ functions have been defined, but the exact implementation depends on the representation of the milling surface and tools geometries. In the next section, the models that are used in this work are discussed.

3.2 Tools and part representation models

As is stated in the previous section, four functions should be implemented in order to perform the whole optimization process:

- $z_t(x, y)$ - defining the tolerance surface, such that all the material outside this volume bounded by the surface should be machined.
- $d_i(x, y)$ - defining the maximum depth for a tool in the cutter location point (x, y) without gouging
- $A_i(x, y)$ - defining the surface area that a tool can mill from the cutter location (x, y) within the tolerance
- $\Omega_i(x, y)$ - defining the set of all the cutter locations that can mill the material from the surface point(x, y) within the tolerance.

Defining the maximum tool depth from the cutter location is a usual problem that has to be solved to build any toolpath for 3-axis CNC milling. Despite the fact that it can be done analytically for a generalized cutter and a free-form surface [45], computing z_t, A_i and Ω_i functions analytically is a difficult problem. On the other hand, digital models and the discrete approach avoid the complexity associated with the analytical approaches. Values for every surface or CL point with some discretization step can be calculated and located into a map that is done for this works. E.g. the tool and the part geometries are represented as depth maps. A depth map is a regular grid, stored as a 2D array, containing the information about the distances from the surface of interest to a plane. It is assumed that the milling surface is defined by an arbitrary polygon mesh and the milling axis is z-axis. To build a depth map, a discretization step (s) should be selected. The mesh is located above the plane $z = 0$ and the maximum distance from points $(x * s, y * s, 0)$ to a mesh polygon along the positive direction of the z-axis is measured and placed into the corresponding (x, y) element of the depth map. The smaller s is, the higher the resolution of the depth map, and the more accurate it represents the mesh. However with decreasing s the depth map occupies more space in memory and takes more computation power to process. Thus, it is a trade-off between precision and computation speed.

The important property of the depth map, used in this work, is that all the depth distances are multiples of s. This allows integers to be stored in the array and all the main computations are performed with integers. This dramatically increases the computation speed in comparison to operations with floating point numbers. To build such a depth map, the process of voxelization (converting a mesh into a voxel grid) is used.

A voxel represents a value in a regular grid in three-dimensional space. The position of a voxel is inferred based upon its position relative to other voxels. A simple voxel grid can be represented as a 3D array with voxel values as elements. To represent a polygon mesh, two voxel values are enough: one if there is a boundary of the mesh in the voxel position, and another if there is no boundary. They are called empty and boundary voxel respectively. Similar to depth map building, the discretization step should be selected. For the purpose of this research, it is the same as for the depth map (s). During the voxelization, for every (x, y), the highest z position of the boundary voxel is written into the (x, y) depth map element.

The most common way to voxelize a polygon mesh is to rasterize all the mesh polygons [12]. The rasterization can be performed quickly on modern GPUs, but there might be small dimensional errors on rasterizing polygons' edges or vertices. In this work voxelization is done, using polygon-box overlap testing [1]. To convert a polygon mesh into voxels, the virtual spacial grid is created in order to have the whole mesh inside. One cell is a cube with a side of s and the center in $(x*s, y*s, z*s)$ point and the overlapping of every mesh polygon and every cell is tested. If overlapping, the maximum value of z coordinate is located in the (x, y) depth map element. Overall, the part depth map representation can be considered as columns of the part material, with the lengths of the values from the depth map. An arbitrary 2D mesh in a voxel grid is shown in Fig. 3.5.

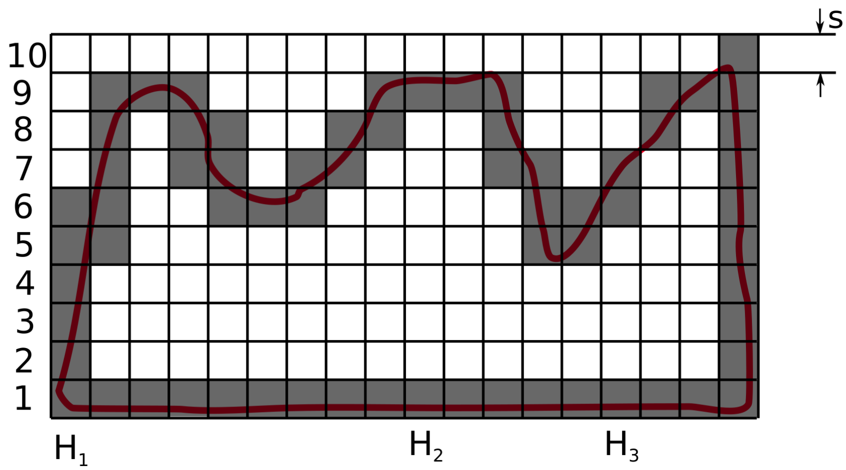

Figure 3.5: A mesh in a voxel grid. Black cells are boundary voxels and white are empty ones. Depth map values are: $H_1 = 6, H_2 = 9, H_3 = 6$.

Fig. 3.6 shows the representative mesh surface and the built depth map.

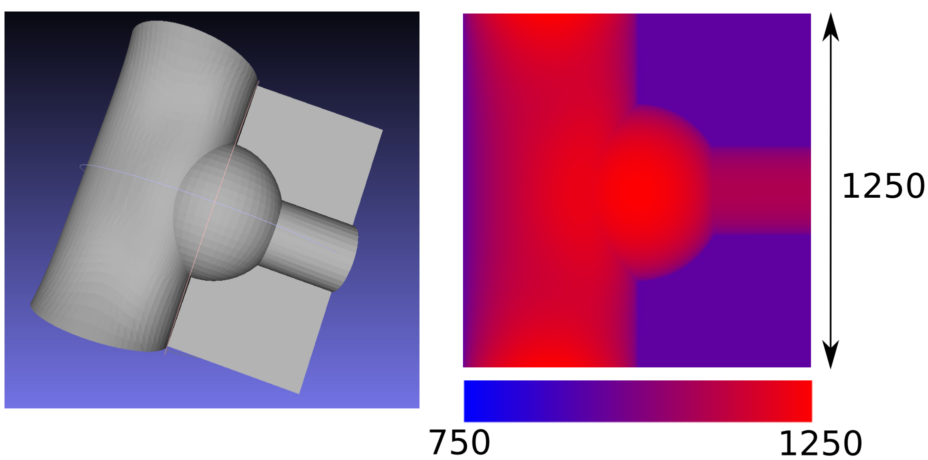

Figure 3.6: The representative part (5cm x 5cm x 5cm) with its depth map. The discretization step (s) is 0.04mm. Note that the size of the map is 1250x1250 (5cm/0.04mm) and the range of the depth is from 3cm (750*0.04mm) to 5cm (1250*0.04mm).

A tool geometry is also represented as a depth map with the same discretization step (s). The algorithm to build a depth map is different, because the tool is not represented as a polygon mesh, but as a generalized cutter. In the generalized cutter model, seven independent geometric parameters are used to define the tool geometry:

$D, R, R_r, R_z, \alpha, \beta, h$ [8]. Fig. 3.7 shows the generalized cutter with its parameters. The periphery of the milling cutter is divided into three zones and the radius can be defined as a function of z [16]:

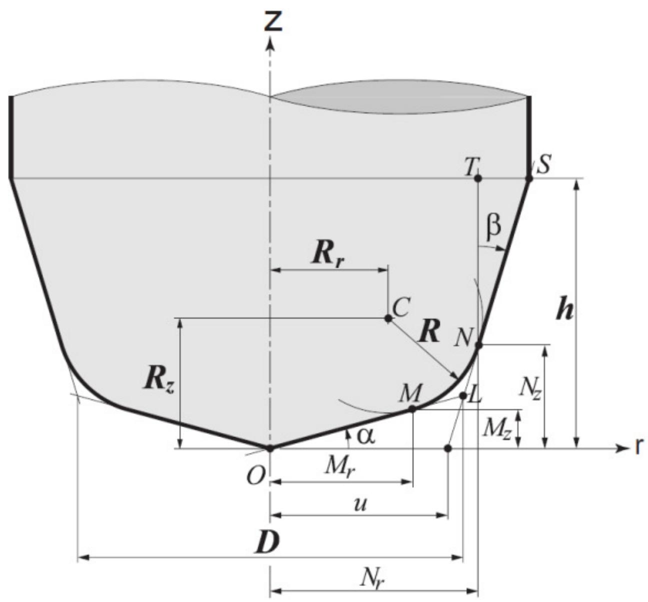

Figure 3.7: Generalized cutter geometry

$$r(z) = \begin{cases} \frac{z}{\tan\alpha}, & \text{for zone OM} \\ \sqrt{R^2 - (R_z - z)^2} + R_r, & \text{for zone MN} \\ \frac{D}{2}(1 - \tan\alpha\tan\beta) + z\tan\beta, & \text{for zone NS} \end{cases}$$

In contrast to the part depth map, the tool depth map represents the distance to the tool surface outside of the tool, so that the tool tip is always have the depth zero for the generalized cutter with any parameters. This difference between part and tool depth maps is clear from the Fig. 3.2. To build the tool depth map, two indices of the tool depth map i and j should take values from $-\lceil \frac{r}{s} \rceil$ to $\lceil \frac{r}{s} \rceil$, where r is a tool radius. The tool depth map value for the element $[i, j]$ is calculated as the minimum k value among all the points $(i * s, j * s, k * s)$ inside the tool:

$$\text{depth\_map}[i,j] = \min_{k=0..\lceil \frac{h}{s} \rceil} \begin{cases} k, & C \text{ is true} \\ \lceil \frac{\text{tool\_length}}{s} \rceil, & \textit{otherwise} \end{cases}$$

where $C = [(i*s)^2 + (j*s)^2 \leq (r(k*s))^2]$, the condition that the point $(i*s, j*s, k*s)$ is inside the tool.

Fig. 3.8 shows the result of the built depth map for a simple ball-end tool.

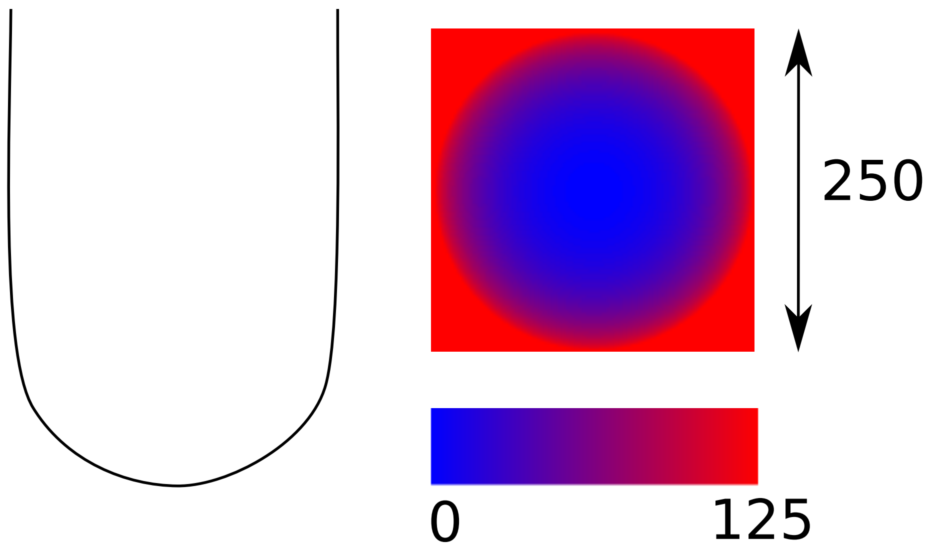

Figure 3.8: A 10mm ball-end tool with its depth map. The discretization step (s) is 0.04mm as before. Note that the size of the map is 250x250 (10mm/0.04mm) and the range of the depth is from 0 to 5mm (125*0.04mm).

After building the depth maps for every tool and for the part, calculating z value from functions $z_p(x, y)$ and $z_i(x, y)$ functions are as simple as taking values from the corresponding depth map array. Depth maps for the part and all the tools are denoted as $\text{depth\_map}_p[\bar{x}, \bar{y}]$ and $\text{depth\_map}_i[\bar{x}, \bar{y}]$ respectively. Coordinates $\bar{x}$, $\bar{y}$ and $\text{depth\_map}_i[\bar{x}, \bar{y}]$ are integers and are called "voxel coordinates". The correspondence between the real world coordinates $(x, y, z(x, y))$ and "voxel coordinates" $(\bar{x}, \bar{y}, \text{depth\_map}[\bar{x}, \bar{y}])$ are:

- $\bar{x} = \lfloor \frac{x}{s} \rfloor$
- $\bar{y} = \lfloor \frac{y}{s} \rfloor$

- $z(x, y) = s * \text{depth\_map}[\lfloor \frac{x}{s} \rfloor, \lfloor \frac{y}{s} \rfloor]$

However, for all the future computations in this work using only "voxel coordinates" is enough. For simplicity, later in this paper depth_map$[\bar{x}, \bar{y}]$ is denoted as $\text{dm}[x, y]$, where x, y and $\text{dm}[x, y]$ are integers starting from zero unless is stated otherwise.

3.3 Accuracy analysis for the provided models

The main source of the geometric error in the proposed models is the depth map used for part and tool geometry representations. In order to estimate the geometric error in using the depth maps the appropriate metric should be selected. In this work it is the difference in the distance between precise geometrical surfaces and the distance between depth maps. It can be seen that the depth maps for parts and tools are built in a manner so as to prevent overcut. The depth maps are constructed to be always outside of the precise geometrical surfaces. Therefore, if the distance between the depth maps is zero (which can happen in a CC point) then the distance between geometrical surfaces is greater or equal to zero.

Unfortunately, the undercut can easily happen and should be estimated. It is easy to calculate the maximum undercut distance in a CC point in order to estimate the maximum error. Recall that all the values in part and tool depths maps are multipliers of the discretization step (s), so the maximum undercut distance can be calculated as a function of s. Fig. 3.9 shows the maximum undercut distance between the part and the tool geometrical surfaces with zero distance between their depth maps.

From the figure, it is clear that the maximum error (e) is the maximum distance in the volume of two neighbour cubes with the side s. Thus, the maximum error can be calculated, using the equation:

$$e = \sqrt{6}s \approx 2.45s$$

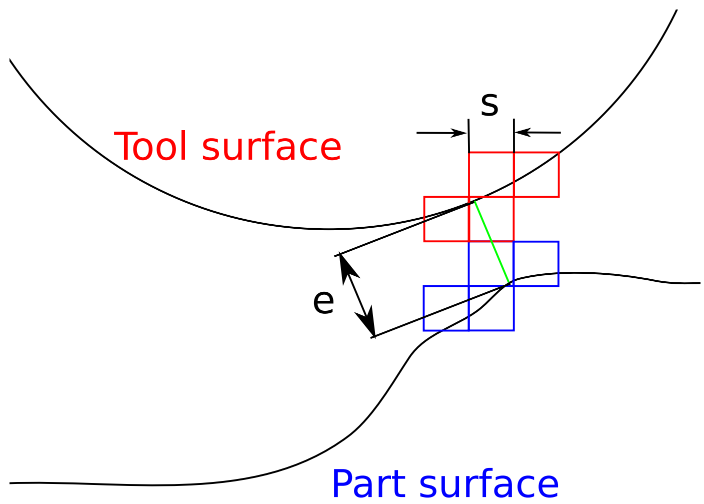

Figure 3.9: The maximum overcut distance (e) between geometrical part and tool surfaces with the zero distance between depth maps in 2D. Tool voxels are red, the part voxels are blue and the maximum error distance is green.

Thus, the smaller the discretization step (s) is, the smaller the maximum error is. In this work, the discretization step is 0.40 microns and the tolerance size is 120 microns for the simulation. Recall that the tolerance represents the maximum distance from the geometrical surface to the milling surface. Considering the error, the tolerance distance of the milled surface is 217 ($120 + \sqrt{6} * 40$) microns.

3.4 Implementation details

The algorithm of defining the best cutter for surface points is linear as shown in Fig. 3.10. However, this algorithm is repeated multiple times, for different tool sets, as a part if the total machining time minimization process. To complete the algorithm, the functions z_t, d_i, A_i, Ω_i and T should be expressed. Because part geometry function (z_p) and tool geometry functions (z_i) are already represented as depth maps, with the small discretization step s, all the other functions can also be represented as maps. That means that the value for every integer pair (x, y) is calculated and placed in 2D arrays, called maps. The first two steps of the algorithm are already discussed in the previous section. This section discusses the methods to build other maps. The first one is the tolerance map.

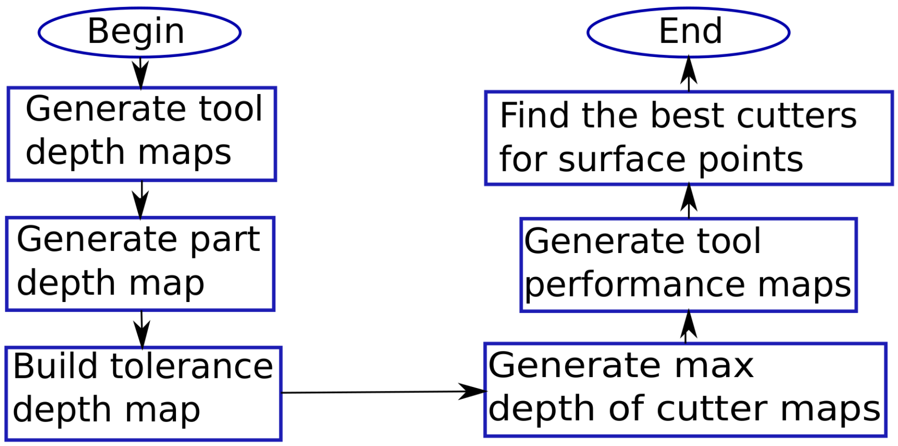

Figure 3.10: Defining the best cutter for surface points algorithm workflow

Recall that any function that satisfies Equation 3.1 is a valid tolerance surface function. The ideal tolerance surface can be built in such a way that the distance from any tolerance surface point to the closest point on the milling surface equals h. By building the sphere with the radius h around every surface point, the ideal tolerance surface can be formed from the points on the spheres with the highest z values. As was mentioned above, the part depth map can be represented as the columns of the part material. Therefore, if such spheres are built around every part point in the columns, the tolerance surface can be represented as the surface of a cylinder with radius h with a sphere with the same radius on the top of the cylinder. By building the intersected surfaces of this shape around every part column with coordinate (x, y) and taking the maximum z value for every point on the built surfaces, the discrete tolerance depth map can be built. Fig. 3.11 shows the discretized part surface and the primitives to build the tolerance surface. Considering the above, the tolerance surface depth map can be built using the equation:

$$\mathrm{dm}_t[x,y] = \max_{\substack{x_l=\lceil -h/s\rceil..\lceil h/s\rceil \\ y_l=\lceil -h/s\rceil..\lceil h/s\rceil}} \begin{cases} 0, & (h/s)^2 \le x_l^2 + y_l^2 \\ m_1 + m_2, & otherwise \end{cases}$$

where

m_1 $= \mathrm{dm}_p[x + x_l, y + y_l]$, part surface depth,

m_2 $= \lfloor\sqrt{(h/s)^2 - x_l^2 - y_l^2}\rfloor$, additional height from the tolerance surface sphere primitive.

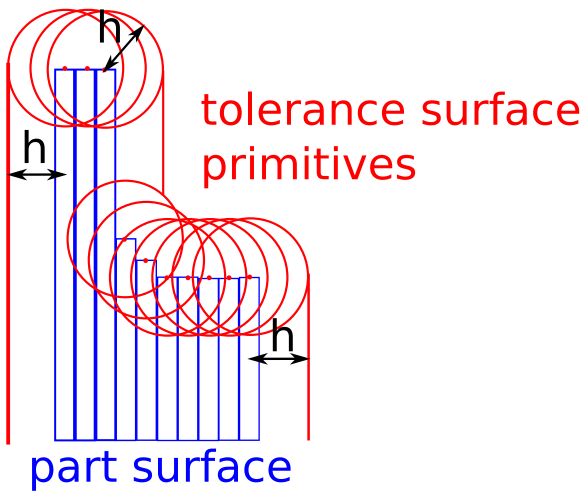

Figure 3.11: Discretized part surface and tolerance surface primitives. Note that the part depth map columns are shown in blue and tolerance surface primitives are shown in red.

The next step is to build the depth of the cutter map (dmc_i) for every ith cutter ($i = 1..N$). Recall, that this map shows the maximum tool depth (minimum z coordinate) for the ith cutter located in coordinates (x, y) without gouging. Using depth maps, the Equation 3.2 is slightly changed to the form:

$$\mathrm{dmc}_i[x,y] = \max_{\substack{x_l=-r_{vi}..r_{vi} \\ y_l=-r_{vi}..r_{vi}}} \mathrm{dm}_p[x + x_l, y + y_l] - \mathrm{dm}_i[x_l, y_l] \tag{3.7}$$

where $r_{vi} = \lceil \frac{r_i}{s} \rceil$ is ith tool radius in voxels. Fig. 3.12 shows the result of building the depth of cutter map for the part depth map and a flat-end cutter.

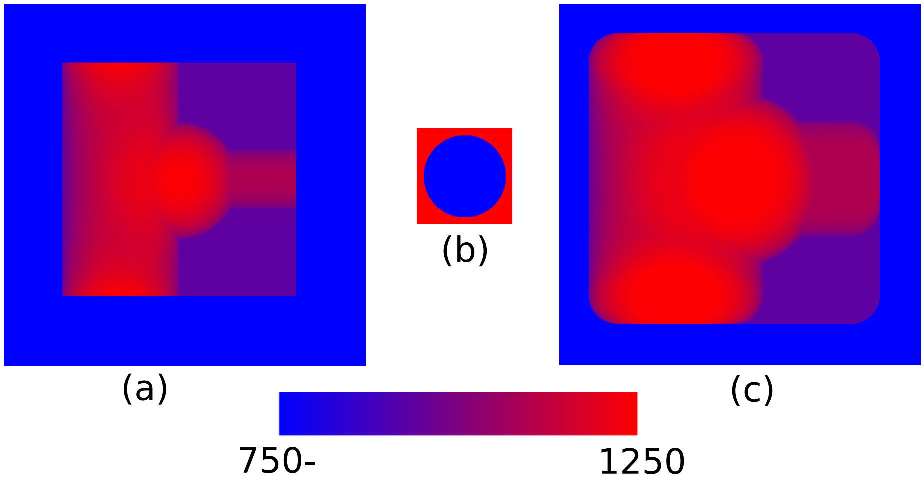

Figure 3.12: Part depth map (a), 10mm flat-end tool depth map (b) and the map of the maximum depth of cutter without gouging (c). Note that part depth map size is as before 1250x1250 and depth of cutter map is bigger than part depth map. The tool depth map size is 250x250. The range of depth is from less than 3cm (750*0.04mm) to 5cm (1250*0.04mm).

The discretized model reduces the integration in Equation 3.3 into summation. So, the finished area map (am_i) for ith tool can be expressed:

$$\mathrm{am}_i[x, y] = \sum_{x_l=-r_{vi}}^{r_{vi}} \sum_{y_l=-r_{vi}}^{r_{vi}} \begin{cases} 1, & m_1 + m_2 \leq m_3 \\ 0, & otherwise \end{cases} \tag{3.8}$$

where

$m_1 = \mathrm{dmc}_i[x, y]$, cutter depth in a cutter location point,

$m_2 = \mathrm{dm}_i[x_l, y_l]$, the depth of the tool surface point,

$m_3 = \mathrm{dm}_t[x + x_l, y + y_l]$, the depth of the tolerance surface.

Finished area maps for the part depth map provided above are shown in Fig. 3.13 for 10mm flat-end and ball-end tools, respectively. As expected, the zones of horizontal surfaces have bigger value for the flat-end tool, but free-form zones usually have bigger values for ball-end tools. Finishing performance maps (pm_i) can be easily calculated afterwards considering tool feed rates (f_i) with the discrete form of Equation 3.4:

$$\mathrm{pm}_i[x, y] = \mathrm{am}_i[x, y] * f_i \tag{3.9}$$

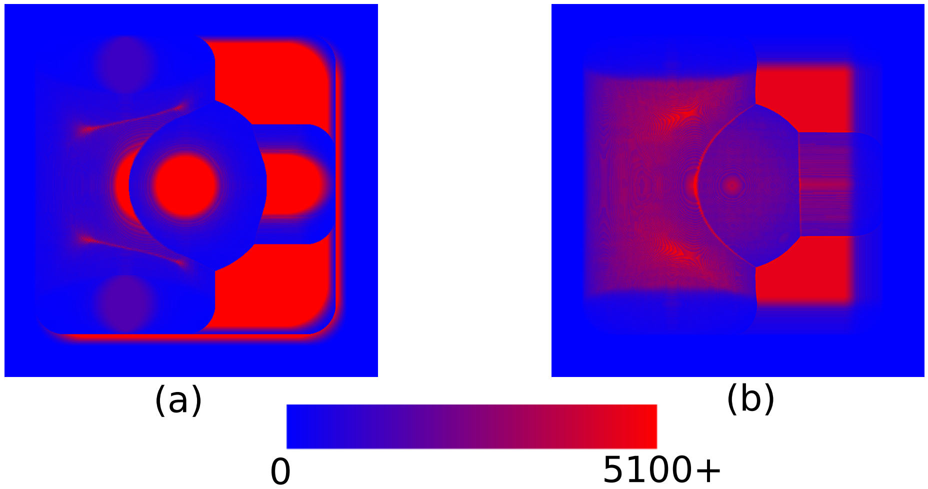

Figure 3.13: Finished area maps for 10mm flat-end tool (a) and for 10mm ball-end tool (b). Note that finished area is measured in voxel squares with the area 0.0016mm^2 (0.04mm*0.04mm) each. The area range is from 0 to more than 8mm^2 (5100*0.0016mm^2).

To build the map of the best tool indices, a discrete Ω function should be defined. However, in this work the Ω set is not calculated explicitly. The search of all the cutter location points that can "cover" the surface point and calculation of the maximum performance for the surface points are done simultaneously for one cutter. The equation to build such a map (psm_i) of surface finishing performance for the ith cutter can be obtained from Equations 3.5 and 3.6. For the discretized model, the equation is:

$$\mathrm{psm}_i[x, y] = \max_{\substack{x_l = -r_{vi}..r_{vi} \\ y_l = -rvi..r_{vi}}} \begin{cases} p, & m_1 + m_2 \le m_3 \\ 0, & otherwise \end{cases} \tag{3.10}$$

where

$p \ = \mathrm{pm}_i[x + x_l, y + y_l]$, finishing performance in a cutter location,

$m_1 \ = \mathrm{dcm}_i[x + x_l, y + y_l]$, cutter depth in a cutter location point,

m_2 $= \mathrm{dm}_i[-x_l, -y_l]$, the depth of the tool surface point,

m_3 $= \mathrm{dm}_t[x, y]$, the depth of the tolerance surface.

Surface finishing performance maps for the same 10mm flat-end and 10mm ball-end tools are shown in Fig. 3.14. The feed rates for both tools are considered the same and equal to 1mm/sec, so it does not affect finishing performance. It can be seen that such maps are close to feasibility maps with the exception that for the feasibility map only two values for a surface point are appropriate (one value if the tool can reach the surface point and another one if it cannot). For the surface finishing performance map, the value in every surface point represents the tool efficiency for this surface point.

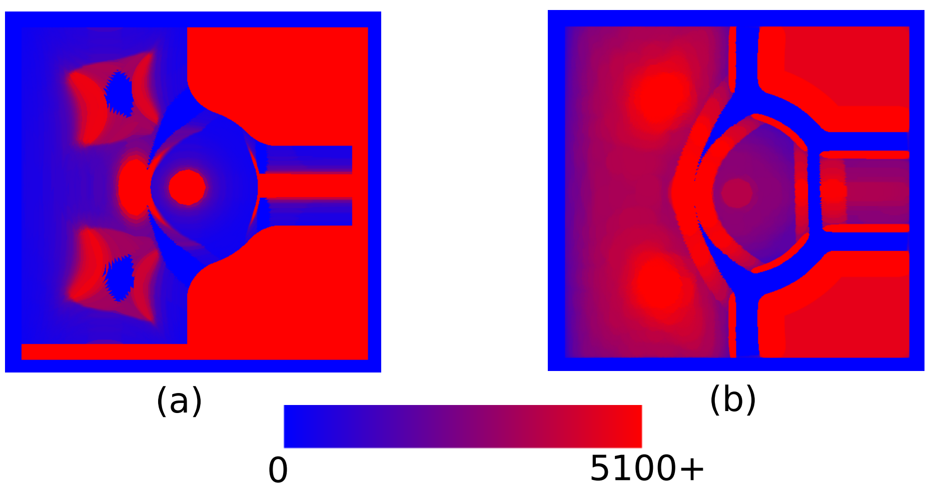

Figure 3.14: Surface finishing performance maps for 10mm flat-end tool (a) and for 10mm ball-end tool (b). Note that the surface finishing performance is measured in mm^3/sec. The finishing performance range is from 0 to more than $8mm^3$/sec.

Finally, the map of the best tool indices (tm) can be built. Having the maps of surface finishing performance, the tool with the best performance in a surface point is the best tool for this surface point. Thus, the equation is:

$$\mathrm{tm}[x, y] = \underset{i=1..N}{\arg\max}(\mathrm{psm}_i[x, y]) \tag{3.11}$$

Five ball-end cutters and five flat-end cutters were used in the simulation to build zones for the finishing stage of milling. Their shapes, sizes, depths of cut and feed rates are shown in Table 3.1. These values are considered given in this work, but they can be easily calculated for the known work piece material, tool shape and size. Note, that the same cutters with their parameters were used for the simulation of the whole algorithm to minimize the total machining time as stated in the next chapters. After the tool set optimization, some of these cutters were also used for the actual milling. The result of building a map of the best tool indices for the given part and for ten available cutters is shown in Fig. 3.15.

Table 3.1: List of cutters, used for simulation

Tool number	Diameter, mm	Cutter Shape	Depth of cut, mm	Feed rate, mm/min
T1	9.525	flat end	16.67	5376
T10	7.983	ball end	13.90	4163
T2	4.763	flat end	8.33	2080
T3	4.763	ball end	8.33	2080
T4	3.175	flat end	5.56	1440
T5	3.175	ball end	5.56	1200
T6	2.000	flat end	3.50	886
T7	1.984	ball end	3.18	818
T8	1.191	ball end	2.08	450
T9	1.000	flat end	1.75	378

The generated map seems intuitively correct. Most of the horizontal zones are finished with the biggest possible flat-end (white) cutter. On the other hand, for most of the non-horizontal zones, the biggest ball-end (red) cutter is selected. If bigger tools cannot be used to reach some surface zones, smaller flat-end (green, cyan, yellow, dark green) and ball-end (gray, blue, dark yellow, dark red) tools are used for horizontal and non-horizontal surfaces respectively as shown in fragment (b). Fragment(c) shows part of the surface that is flat enough to use a flat end cutter instead of ball-end. The boundary of the zone for every cutter is noisy, because the best tool index is defined independently for every surface point. That means the

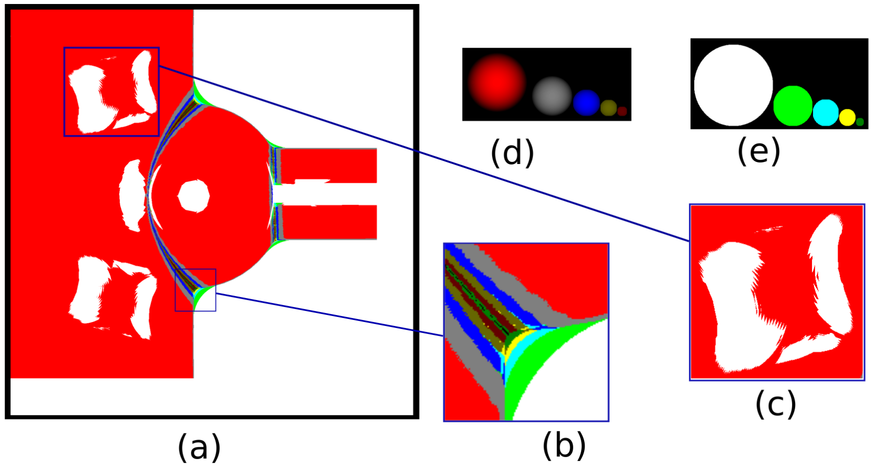

Figure 3.15: Best tool indices map (a) with its fragments (b, c) for five ball-end (d) and five flat-end (e) cutters

algorithm for tool path planning should be robust enough to build tool paths for such milling zones. However, by increasing the resolution, the boundaries of the milling zones can become less noisy.

Every available cutter was used in this map, that highlights the need of the tool set optimization. Without optimization, the tool must be changed ten times in CNC which inceases the total machining time dramatically. A robust algorithm to generate tool paths for an arbitrary milling zone and the way to optimize the tool set are shown in next chapters.

3.5 Algorithms complexity and GPU acceleration

The most time-consuming operations are building the depth of cutter maps, building "finished area" maps and building surface finishing performance maps. From Equations 3.7, 3.8 and 3.10, it can be seen that the asymptotic complexity of all the three algorithms is the same. The amount of work that needs to be done for every map element is proportional to the tool radius squared. So, the total work to build one map for a part with surface area (S) and a tool with radius (r_i) is proportional to S and r_i^2. The part surface square and tool radius are measured in "voxel units".

This means that by decreasing the discretization step (s) two times, the part surface is increased four times and the tool radius is increased two times in "voxel units". Thus, the asymptotic complexity of building one map operation ($f(S, s, r_i)$) can be expressed:

$$f(S, s, r_i) = O(\frac{1}{s^4} * r_i^2 * S)$$

By having N tools with the maximum tool radius equal to r_{max}, the complexity ($g(S, s, r_{max}, N)$) of these three algorithms to generate maps for all cutters is:

$$g(S, s, r_{max}, N) = O(\frac{1}{s^4} * r_{max}^2 * S * N)$$

It can be seen that increasing the resolution (decreasing the discretization step) is expensive. However, every element in a map can be calculated independently. This means that all three time-consuming algorithms can be easily parallelized for thousands of processors. Therefore, employing GPU computations seems a logical choice. The GPU executes parallel code by running parallel functions called kernels. Any function that can be run in parallel can therefore by run by thousands of threads processing the same kernel in parallel. These kernels can be run with the OpenCL framework. The OpenCL is a framework for writing programs that can be executed on CPUs, GPUs or other processors or hardware accelerators. OpenCL provides a standard interface for parallel computing.

The code in this work was developed in C++ and OpenCL. The code was run on an Intel i7-4770 CPU and on two AMD Radeon RX 480 GPUs. For this work, the OpenCL kernels are the code to update one map value. The performances for building maps for the (5cm x 5cm) part surface and the biggest 9.525mm cutter, with different discretization steps are shown in Table 3.2. One more algorithm with the same complexity is shown in the table. This algorithm builds maps of rouging depth of the cutter. The need for this algorithm is provided in Chapter 5.

It can be seen that the theoretical complexity is close to the experimental complexity. By increasing the resolution two times, the map-building time is increased

Table 3.2: GPU performance for different map building operations

GPUs number	Discretization step, mm	Time of building map, seconds			
		Finishing depth of cutter	Roughing depth of cutter	Finished area	Surface finishing performance
2	0.04	1.6	3.0	3.1	1.2
	0.02	27.1	46.4	47.14	20.8
	0.01	541.1	720.4	742.3	335.2

by 15 to 20 times (the theoretical value is $2^4 = 16$). To apply this algorithm without optimization for resolution less than 10 microns can be time-consuming. However, because the algorithm can be run on parallel on any number of nodes with little overhead, employing more GPU or FPGA devices can significantly reduce computation time.

After these three maps for every cutter are generated, building the map of the best tool indices as shown in Equation 3.11 is cheap. The complexity of this operation is $O(\frac{1}{s^2} * S * N)$. Therefore, with changing the set of used tools, new maps of the best tool indices can be built fast. This means that the performance of the whole optimization algorithm depends on two major factors:

- building four time-consuming maps for every cutter
- building the tool paths for the generated map of the best tool indices to calculate milling time

These algorithms with the analysis if their performances is provided in next chapters.

Chapter 4

Tool path planning for finishing

It is important to have a robust algorithm that can build tool paths for an arbitrary milling zone on a free-form surface and a generalized cutter. As was shown in Chapter 3, milling zones can be build for cutters based on their performance. For the algorithm to build tool paths, it is assumed that the cutter geometry and the milling zone for this cutter are given.

Often, the milling zone for one cutter can be defined in a trivial manner. It might be the whole milling surface or the surface for one feature, but it can be defined algebraically [38]. In this work the milling zone is defined by an arbitrary boolean map. Every cell of this map represents one surface point and contains "True" if this surface point should be milled with the cutter and "False" otherwise. It is also assumed that for every map cell containing "True", the material from the surface point can be milled with the given tool within the tolerance distance. Because of the fact that such an arbitrary defined milling zone can be discontinued and its boundary can be noisy, it is difficult to select an optimal milling direction in the direction-parallel approach. Therefore, the contour-parallel approach to build tool paths was selected in this research.

Among approaches to define tool path parameters, such as iso-parametric, iso-planar and iso-scallop height, the iso-scallop method is known to generate the shortest overall tool path. That is why, the iso-scallop approach was selected to be used in this work. The traditional approach in building iso-scallop tool paths is to calculate the CL point based on the previous CL point, the surface curvature, and the cutter shape. However, for a free-form milling surface and a generalized cutter model, it is a difficult problem. Another approach to find the next CL point by testing different CL points and select the one with the maximum performance is shown in the present research. This approach is more expensive, but, using a digital model, it avoids the computational complexity associated with the analytical approaches. Because of simpler and uniform calculations, GPUs can be employed to accelerate the most time consuming operations.

The output of the algorithm from this chapter is generated tool paths. These tool paths are built in order to mill the material from every surface point from the milling zone within predefined tolerance size. After connecting all the generated tool paths together, milling time can be calculated. This milling time is used to minimize the total machining time as described in Chapter 6. The generated tool paths from different milling layers are also used to build the final tool path, after defining the optimal tool set.

4.1 Main tool path planning algorithm

The input data for the algorithm was partially mentioned above. Here is the complete list of all the input parameters. All the maps represent some values in surface points or CL points with a small discretization step (s):

dm_i - tool surface depth map,

dm_p - part surface depth map,

dm_t - tolerance surface depth map,

dmc_i - maximum depth of cutter map,

mzm - boolean map of the milling zone,

F_{max} - the maximum feed rate for the tool and the workpiece material,

s - discretization step,

a - CNC machine tool acceleration and deceleration.

The same part, 7.983 mm ball-end tool, and the milling zone, in which this tool shows the best performance (red area from Fig. 3.15) are used as a representative input for the tool path planning algorithm. Fig. 4.1 shows all the essential input data together.

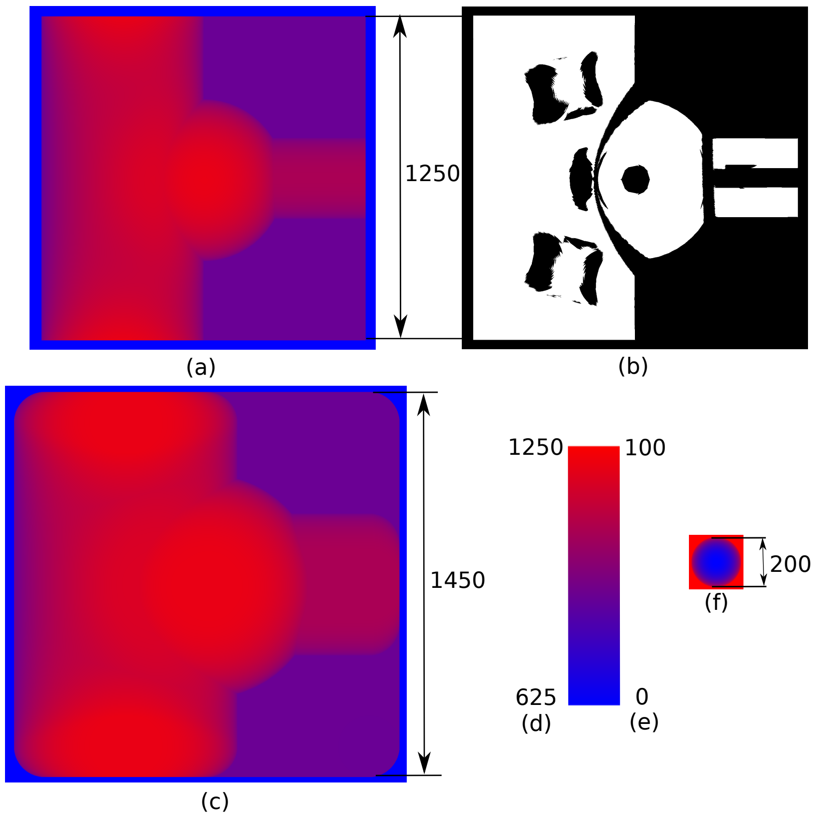

Figure 4.1: Input data for tool path planning algorithm: 5mm x 5mm part depth map (a), milling zone map (b), 7.983 mm ball-end tool depth map (f), maximum depth of the cutter map (c) and the legends for part depth map (d) and tool depth map (e). Note, that part depth map and the milling zone map have the same sizes and the depth range for part depth map and the maximum depth of the cutter map is the same. The discretization step (s) it the same for all the map and equals to 0.04mm.

The requirements for the algorithm of tool path building are derived from its characteristics. Because the resultant tool path is contour-parallel, the tool path must “follow” the milling zone boundary. “Following” the boundary means that the cutter on the way to the next CL point should remove the material from the boundary points of the milling area within the tolerance. Also, the tool paths must be iso-scallop. Because the constant scallop height guarantees the maximum tool "performance" while the cutter is moving from one CL point to the next one, the "performance" of the tool path segment to the next CL point should be maximized. Here, the "performance" of the tool path segment is the fraction of the "covered" surface area and the milling time. Using these two requirements, the essential steps of the algorithm for the provided discretized model are shown in Algorithm 4.1.

Algorithm 4.1 Build tool paths

```
Input: mzm is a boolean map of milling zone
TPs ← {}                                          ▷ TPs is a set of resultant tool paths
while There are non-milled pixels in mzm do
    ▷ bpm is a map of boundary pixels of the milling zone in mzm
    bpm ← Find boundary pixels in mzm
    while There are boundary pixels in bpm do
        tp ← {}                                   ▷ tp is a tool path to build
        pcl ← null                                ▷ pcl is a previous CL position
        p ← Get any boundary pixel from bpm
        BPs ← {p}                                 ▷ BPs is a set of boundary pixels to "cover"
        while There are elements in BPs do
            ▷ CLs is a next CL candidates set
            CLs ← Find CLs that "covers" max pixel number from BPs
            ▷ Ss is a line segments set of possible tool motions
            Ss ← Build line segment (pcl, cl_i), for ∀cl_i ∈ CLs
            Remove segments that produce gouges from Ss
            Ts ← Calculate milling times for ∀s_i ∈ Ss
            As ← Calculate milled areas for ∀s_i ∈ Ss
            ▷ Ps is a performance set for tool swipes along segments
            Ps ← {a_i / t_i}, for ∀a_i ∈ As, t_i ∈ Ts
            ▷ Find the CL from segment with the maximum performance
            best_cl ← CLs[arg max_{p_i ∈ Ps} p_i]
            Add best_cl to tp
            Remove "covered" pixels from mzm and bpm on (pcl, best_cl)
            BPs ← Find boundary pixels next to removed pixels
            pcl ← best_cl
        end while
        TPs.add TP
    end while
end while
return TPs
```

The algorithm builds one tool path on a point-by-point basis, while it can find the next appropriate CL point, given the previous one. If there are no appropriate tool path points, the tool path is added to the set of tool paths and the algorithm starts building the next tool path while there are non-milled pixels in the milling zone. The next appropriate CL point is the one that can "cover" any pixels from the set, made of the boundary pixels, located next to removed milling zone pixels from the previous CL. If there are multiple CL points that meet this criterion, the ones that cover the maximum number of boundary pixels from the set are selected. Then, the CL points that produce gouges with tool path segments from the previous CL point are removed. Finally, the CL with the maximum performance is considered the next CL point. Selecting CL points that "cover" the maximum number of boundary pixels allows to work efficiently with the noisy boundary of the milling area.

This approach might lead to non-optimal tool paths, because the global optimality is not considered while locally optimal CL points are selected. Thus, the whole milling direction can be badly chosen. This problem is similar to choosing the direction of the milling in the direction-parallel approach to tool path building. Overall, the problem of tool path building is similar to the maximum coverage problem that is NP-hard, and using the greedy approach to select the next CL point seems reasonable.

Algorithm 4.1 has many sub-algorithms. While some of them are trivial, the others needs to have the detailed explanation. Here is the short list of these algorithms with the corresponding line number:

1. Detecting the existence of non-milled pixels in milling zone map (line 2).
2. Finding boundary pixels of the milling zone (line 4).
3. Detecting the existence of non-milled boundary pixels in the map of boundary pixels (line 5).
4. Finding any boundary pixel in the map of boundary pixels (line 8).

5. Searching CL positions that can "cover" a max number of pixels from the provided list (line 12).
6. Detecting if a line segment produces gouges (line 15).
7. Calculating the milling time of line segment tool motion (line 16).
8. Calculating the milled area, produced by line segment tool motions (line 17).
9. Removing pixels, that are "covered" by a line segment tool motion from the milling zone and boundary pixel maps (line 23).
10. Detecting the new set of boundary pixels, next to removed pixels in the milling zone map (line 24).

Detecting the existence of non-milled pixels from boolean maps (items 1 and 2) is as simple as searching the 2D binary array for the existence of "True" values. The algorithm to find any boundary pixel in a boolean map (item 4) (the 2D coordinate of the first "True" value needs to be returned) is also trivial. The other algorithms detailed description is provided in the next sections.

4.2 Milling zone boundary pixels detection

This algorithm finds all the boundary pixels in the provided milling zone map. The approach of the algorithm is similar to ones that are used for edge detection in image processing [33]. Recall, that the milling zone map is a boolean map, with "True" values for pixels that need to be milled and "False", otherwise. The output is a boolean map of the same size with "True" values for boundary pixels and "False", otherwise. The boundary pixel is a milling zone pixel that has at least one non-milling zone pixel in its neighborhood. In other words, the pixels with "True" value that have at least one neighboring "False" pixel need to be found. The size of the neighborhood is 1, so for every pixel, eight adjacent pixels need to be checked. There is one more criteria, how a pixel is considered a boundary one. If a distance in depth between to adjacent milling zone pixels is more than some threshold value, they are both considered a boundary pixels. It is done this way to distinguish milling zones at different depths, even if they need to be milled with the same tool. Fig. 4.2 shows the milling zone with its boundary and its fragment. The Algorithm 4.2 provides the details of the implementation.

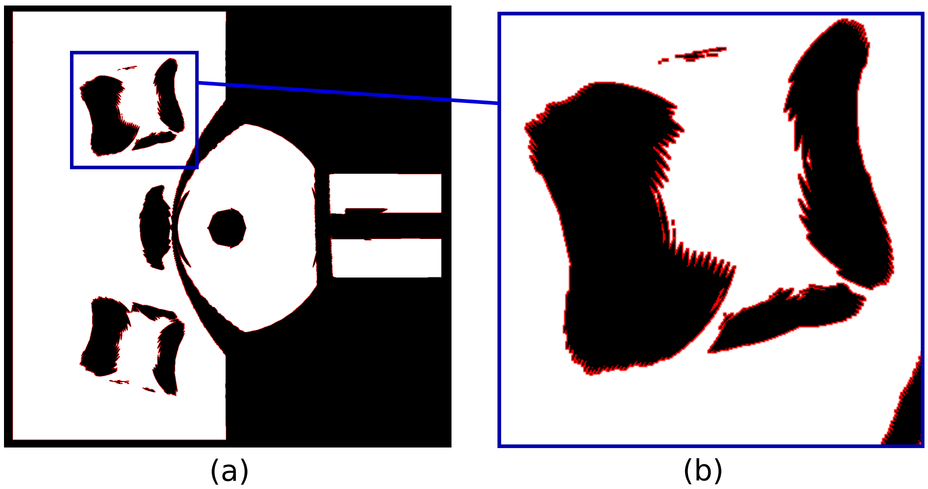

Figure 4.2: Detected boundary if the milling zone (a) and its fragment (b). The milling area is white, while boundary pixels are red. Note, that the boundary in the fragment is very noisy.

Algorithm 4.2 Detect milling zone boundary

Input: mzm is a boolean map of milling zone with width W and height H
Input: t is a maximum depth threshold
Input: dm_t is a tolerance surface depth map

```
▷ Create a map of boundary pixels with WxH size
bpm ← [1..W, 1..H]
for y = 1..H, x = 1..W do
    bpm[x, y] = False
    for dy = −1..1, dx = −1..1 do
        if mzm[x, y] and not mzm[x + dx, y + dy] then
            bpm[x, y] = True
        end if
        if mzm[x, y] and |dm_t[x + dx, y + dy] − dm_t[x, y]| > t then
            bpm[x, y] = True
        end if
    end for
end for
return bpm
```

4.3 The search of CL positions that "cover" the maximum number of the provided boundary pixels

This algorithm searches the CL positions that "cover" the maximum number of provided points. Recall, that the tool "covers" a surface point from a CL location if it is able to remove the material from the surface point within the tolerance surface. To check that a CL can "cover" the given surface point, the difference in depths of CL position and the corresponding point in tool depth map should be less or equal than the corresponding depth in the tolerance surface map. A similar approach is used in Equation 3.3 to calculate the finished area from one CL point. The algorithm is cheap, because, the test on "coverage" is made only for the given boundary points not for all of them.

The algorithm is straight forward. All the possible CL points need to be checked. For every CL point the number of "covered" pixels from the provided list of boundary pixels is counted. Then, the maximum number of "covered" boundary pixels is defined. Finally, all the CL positions with the number of "covered" pixels less than the maximum is removed from the list. The maximum boundary pixels need to be covered to work with the noisy boundary of the milling zone. Otherwise, single pixels from the milling zone might be missed and later they have to be addressed which increases the number of tool paths. The output of the algorithm is a set of CL with the maximum number of "covered" boundary points.

The result of this algorithm for two consecutive tool path generation steps is shown in Fig. 4.3. Note, that while the boundary of the milling zone is noisy, the algorithm is able to follow the boundary, while no single pixels of the milling zone are left. The boundary of the milling zone left for the next stages is also noisy, but it is not an issue since the algorithm can generate the tool path for the noisy milling zone.

Algorithm 4.3 Find CL positions to "cover" the maximum number of boundary pixels

Input: BPs is a set boundary pixels to "cover"
Input: dm_i is a tool surface depth map
Input: dm_t is a tolerance surface depth map
Input: dmc_i is a maximum depth of cutter map
Input: W and H are the maximum x and y coordinates of CL positions respectively

$CLs \leftarrow \{\}$ ▷ The resultant set of CL positions
▷ Find all positions CL that can "cover" pixels from BPs
for $cl.y = 1..H, cl.x = 1..W$ **do**
    $c \leftarrow 0$ ▷ c is a counter of "covered" boundary pixels
    $z \leftarrow dmc_i[cl]$ ▷ z is the depth of cutter in cl
    for $\forall bp \in BPs$ **do** ▷ check all the boundary points
        ▷ Check if cl "covers" bp
        if $z + dm_i[bp - cl] \leq dm_t[bp]$ **then**
            $c \leftarrow c + 1$
        end if
    end for
    if $c > 0$ **then**
        ▷ Save 3D coordinates of found cl point with the counter value
        Add $\{cl.x, cl.y, z, c\}$ to CLs
    end if
end for
▷ Filter CLs values if not max number of pixels is "covered"
$max\_c \leftarrow \max_{\forall cl \in CLs} cl.c$
for $\forall cl \in CLs$ **do**
    if $cl.c < max\_c$ **then**
        Remove cl from CLs
    end if
end for
return CLs

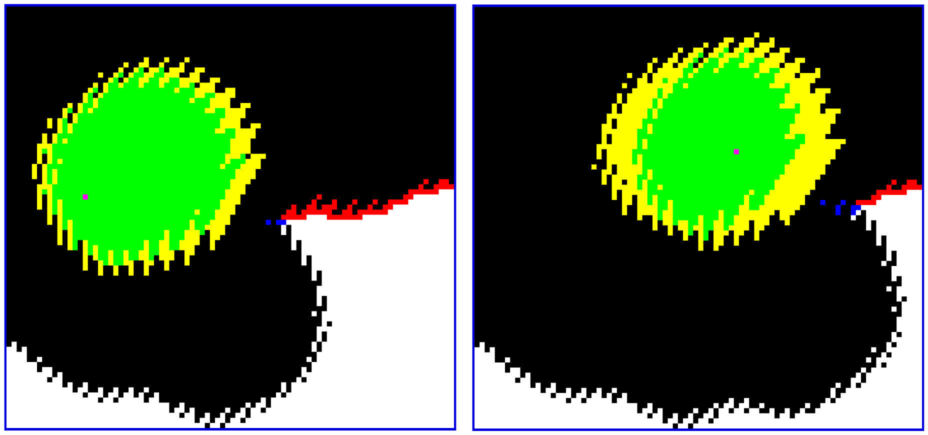

Figure 4.3: CL positions to "cover" the provided boundary pixels set. White pixels represent milling area, red ones represent all the boundary pixels, while the blue ones represent boundary pixels to "cover" with the tool. Yellow pixels show all CL positions that can cover at least one boundary point, and the green ones shows CL positions that cover the maximum number of the provided boundary points (blue ones). The purple pixel is the best CL position, defined by following sub-algorithms.

4.4 Test if a line segment produce gouges

This algorithm checks if a straight line tool path segment produces gouges. The line segment is defined by the coordinates of its ends. The one end is a previous CL position and another end is a candidate for new CL positions. If a line segment fails the test, it is removed from the list of candidates and it cannot be a new tool path segment. Because of the fact that all the algorithms work in a discrete world, the most obvious solution is to test every CL position on the segment for gouging. Thus, the problem can be divided into two parts. The first one is to build every point on the line segment. And another one is to test each point from the list for gouging, by placing the cutter in this point.

Recall, from Chapter 3 that all the coordinates used in this work are integers. They can be converted into real world coordinates by multiplying them by the discretization step (s). However, for all the algorithms integer coordinates are enough, so all CL positions are converted into real world coordinates at the very end, when they are post-processed into G-code. For the purpose of this algorithm, it means that the coordinates of both of the segment ends are integers and all coordinates of the intermediate points are also should be integers. The algorithm to build intermediate points can be stated as: find all the points in integer coordinates to have a close approximation to a straight segment between two points.

This problem is exactly the line rasterization problem. Usually, rasterization algorithms are used to show geometric primitives on the screen, i.e. rasterize them. Despite the fact that most of them are designed to rasterize primitives in 2D, some can be easily adapted for 3D rasterization. For the line segment rasterization, the simplest algorithm is "Bresenham's line algorithm" [43]. In the algorithm, first the slope of the line is calculated, then, by moving the coordinate with the longest distance by one, the other coordinates are calculated and rounded for every point. The output of the algorithm the list of points with integer coordinates. Algorithm 4.4 shows the details. Note, that x and y coordinates are rounded before adding a point into the resultant list, while for z coordinate is rounded down. It is done to check z coordinates that are lower than the actual coordinate on the segment. If it is rounded, it can lead to small (less than discretization step (s)) over-cut. This can be avoided by rounding down, so the lower z position is checked on gouging.

Algorithm 4.4 Rasterize 3D line segment

Input: $p1, p2$ are the ends of the segment with integer coordinates

$result \leftarrow \{\}$
$dv \leftarrow p2 - p1$ ▷ Get the difference per coordinate
$N \leftarrow \max_{\forall dv_i \in dv} dv_i$ ▷ Find the maximum coordinate distance
$s \leftarrow dv/N$ ▷ Find the delta step per one longest distance pixel
for $d = 0..N$ **do**
    $p \leftarrow p1 + s * d$ ▷ p is a point to add in float coordinates
    Add $\{[p.x], [p.y], \lfloor p.z \rfloor\}$ to $result$ ▷ p is rounded before adding
end for
return $result$

In the second part of the algorithm every rasterized point should be checked for gouging. It can be accomplished by using the maximum depth of the cutter the map (dmc_i). Recall, that this map is showing the maximum depth of the cutter without gouging in different CL points. Thus, if z coordinate of every rasterized point is more or equal than the corresponding value from this map, the segment does not produce gouges. It is shown in Algorithm 4.5.

Algorithm 4.5 Is tool path rasterized segment produces gouges

Input: ps the set of rasterized points
Input: dmc_i is a maximum depth of cutter map
for $\forall p \in ps$ **do**
  if $p.z < \mathrm{dmc}_i[p.x, p.y]$ **then**
    return True
  end if
end for
return False

4.5 Milling time of the segment calculation

After generating all the possible tool path segments and removing ones that produce gouges, the one with the maximum performance should be selected. Recall, that performance is calculated as a ratio of the "covered" area to the milling time. Thus, both of these parameters for every segment should be calculated.

To calculate the milling time for the straight line segment motion the trapezoid motion interpolation model is selected. Despite the liner interpolation, the trapezoid model favors longer segments. It is considered that the cutter has zero feed rate at the first and the last segment points. The tool is accelerated and decelerated with a constant acceleration value. Fig. 4.4 shows two possible cases, if the maximum feed rate can be reached and if it cannot be reached due to the segment length. In both cases only milling time is a point of interest. Thus, the problem can be stated as: with a given segment length, acceleration/deceleration value and the maximum feed rate, what is the motion time with using trapezoid motion interpolation. The maximum feed rate is defined based on cutter shape, size and workpiece material. For the purpose of this research it is considered given. The feed rate values for all used cutters are shown in Table 3.1. Acceleration and deceleration value is a characteristic of the exact CNC machine and is also considered given. In this research the value for acceleration/deceleration is $3000mm/sec^2$. Algorithm 4.6 shows the details of the milling time calculation. Note, that the distance should be converted into real distance units before time calculation.

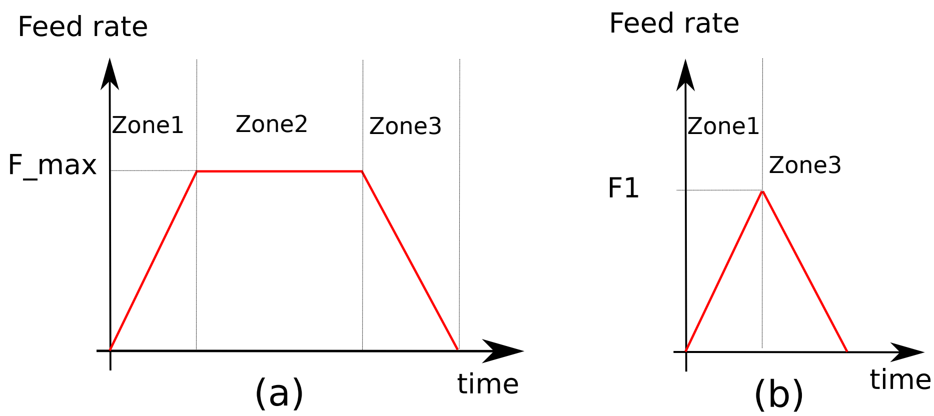

Figure 4.4: Feed rate dependence on time in trapezoid motion interpolation model. If a maximum feed rate (F_max) can be reached (a): there are three zones, acceleration zone (Zone1), constant feed rate zone (Zone2) and deceleration zone (Zone3). If the maximum feed rate cannot be reached (b), there are only two zones: acceleration zone (Zone1) and deceleration zone (Zone3).

4.6 The calculation of the area, "covered" by a tool path segment

The second component of the segment performance is the area, "covered" by a tool path segment. The easiest way to find this area is to check the coverage of every surface pixel by all CL position on the segment. The number of all "covered" pixels can be easily converted into the "covered" surface area. For the purpose of comparing different segment performances, however, the area is not converted into mm^2. It is enough just to know the number of "covered" pixels by a tool motion segment. By having the rasterized segment of the tool motion, the problem can be stated as:

Algorithm 4.6 Calculate milling time of the tool path segment

Input: $p1, p2$ are the ends of the segment with integer coordinates
Input: F_{max} is the maximum feed rate
Input: a is an acceleration
Input: s is a discretization step

$\triangleright$ Calculate the length of the segment in mm
$l \leftarrow s * \sqrt{(p1.x - p2.x)^2 + (p1.y - p2.y)^2 + (p1.y - p2.y)^2}$
$t_{acc} \leftarrow F_{max}/a$ $\triangleright$ Calculate the time for acceleration to F_{max}
$l_{acc} \leftarrow (a * t_{acc}^2)/2$ $\triangleright$ Calculate length for acceleration
if $2 * l_{acc} < l$ **then**
    return $2 * \sqrt{l/a}$ $\triangleright$ It is a triangular case, no time to reach F_{max}
else
    return $2 * t_{acc} + (l - 2 * l_{acc})/F_{max}$ $\triangleright$ It is a trapezoid case
end if

find the number of "covered" surface pixels by any of the CL from the list. Note that this algorithm is close to the search of CL positions to cover boundary pixels. Algorithm 4.7 shows the details.

Algorithm 4.7 Calculate the number of "covered" pixels by a tool path segment

Input: mzm is a boolean map of milling zone with width W and height H
Input: CLs is a set of rasterized CL point of the tool path segment
Input: dm_i is a tool surface depth map
Input: dm_t is a tolerance surface depth map

$A \leftarrow 0$ $\triangleright$ The resultant "covered" area
$\triangleright$ Go through all the possible milling zone points
for $y = 1..H, x = 1..W$ **do**
    for $\forall cl \in CLs$ **do**
        $\triangleright$ Check if cl "covers" (x, y) point
        if $mzm[x, y]$ **and** $cl.z + dm_i[x - cl.x, y - cl.y] \leq dm_t[x, y]$ **then**
            $A \leftarrow A + 1$
        end if
    end for
end for
return A

After defining the milling time for every segment and the "covered" area, the performance of every segment can be calculated and the segment with the best performance can be selected for the next tool path segment. After that, the "covered" pixels with this tool path segment should be removed from the milling zone map and the boundary pixels map. The set of the next boundary pixels to "cover" should also be defined as stated in the next section.

The algorithms to calculate the number of "covered" pixels and to calculate the milling time should be run for every tool path segment candidate. While calculating the milling time is a cheap operation (its asymptotic complexity is $O(1)$), calculating "covered" area is very expensive. It is the most expensive one among other algorithms from this chapter. It checks every surface point with every CL from every rasterized segment. With increasing the resolution (decreasing s) two times, the number of surface points to check is increased four times. The number of CL candidates is also increased four times and the number of rasterized points in a segment increased two times. Thus, the complexity of this algorithm is $O(1/s^5)$, which becomes very expensive with increasing the resolution. However, the "covered" area is calculated independently for every tool path segment candidate, and the contribution of every surface point can also be calculated independently. Therefore, this algorithm can be easily implemented by using OpenCL to work on GPU, as was done for this research. The OpenCL kernel run for every surface point pixel and for every tool path segment candidate as shown in Algorithm 4.8. This algorithm is highly parallelizable and can be written to run on other platforms. However, the atomic increasing operation (Line 6 in the algorithm) needs to be supported by a platform and hardware to have full advantage of its parallelizability. An atomic operation means that the same value can be safely changed from different threads.

Despite the fact, that this algorithm is highly parallelizable, it still can be a bottle neck for the whole algorithm to build the tool paths, so further optimizations can be applied. If an adequate tool path segment is selected rather than the best one, not all the tool path segment candidates have to be tested. Some predefined number of random samples can be selected from the set of all tool path segment candidates. Moreover, every Kth rasterized point in a rasterized segment candidate

Algorithm 4.8 Calculate the number of "covered" pixels by a tool path segment in parallel

Input: Ss is an array with size N of all rasterized tool path segment candidates
Input: As is an array of "covered" areas by segments with size N filled with zeros
Input: mzm is a boolean map of milling zone with width W and height H
Input: dm_i is a tool surface depth map
Input: dm_t is a tolerance surface depth map

```
▷ Go through all the possible milling zone points and all the segments in parallel
for y = 1..H, x = 1..W, i = 1..N in parallel do
    for ∀cl ∈ Ss[i] do
        ▷ Check if cl "covers" (x, y) point
        if mzm[x, y] and cl.z + dm_i[x − cl.x, y − cl.y] ≤ dm_t[x, y] then
            Increase As[i] by 1 atomically
        end if
    end for
end for
```

can be checked, rather than every one. Here, K is some predefined value that can be increased with increasing the resolution, to compensate complexity. In the same way, not every surface point has to be checked on "coverage". All these optimizations affect the precision of calculating "covered" areas, but can reduce the complexity of the algorithm to $O(1)$. If all of them are applied, the calculation of the "covered area" for a constant number of tool path segment candidates need to be done. Also, the test on "coverage" is done for the constant number of rasterized CL points for every segment and for the constant number of surface pixels. In this work, none of this optimization is done in order to build the best possible tool paths and the detailed investigation of the different optimization influences on the tool path length is not performed either.

4.7 Removing "covered" pixels with a tool path segment

After selecting the tool path segment with the maximum performance, the "covered" pixels from the milling zone and the boundary pixel maps need to be removed, so that the new CL points can be searched. The approach is similar to the approach of calculating the milling area. Every surface pixel is checked on coverage from any of the CL positions from the rasterized tool path segment. If a pixel is "covered", its value should be set in "False" in both maps. The implementation is provided in Algorithm 4.9. Note, that this algorithm is close to Algorithm 4.7, with the different action when the "covered" pixels are found (Lines 6, 7).

Algorithm 4.9 Remove pixels, "covered" by a tool path segment, from milling zone and boundary pixels maps

Input: mzm is a boolean map of milling zone with width W and height H
Input: bpm is a boolean map of boundary pixels width W and height H
Input: CLs is a set of rasterized CL point of the tool path segment
Input: dm_i is a tool surface depth map
Input: dm_t is a tolerance surface depth map

1: ▷ Go through all the possible milling zone points
2: **for** $y = 1..H, x = 1..W$ **do**
3:     **for** $\forall cl \in CLs$ **do**
4:         ▷ Check if cl "covers" (x, y) point
5:         **if** $cl.z + dm_i[x - cl.x, y - cl.y] \leq dm_t[x, y]$ **then**
6:             $mzm[x, y] \leftarrow False$
7:             $bpm[x, y] \leftarrow False$
8:         **end if**
9:     **end for**
10: **end for**

4.8 Search of the new boundary pixels to "cover"

The last step, before the search of the next CL points is to find the new boundary pixels to "cover" with the next CL position. This step is essential if the resultant tool path should follow the boundary. However, only the closest boundary pixels should be considered for the next search. If many distant pixels are in the set of boundary pixels to "cover", the next CL point might miss some of the boundary pixels that might lead to non-removed single milling zone pixels. In this case, such pixels have to be addressed later, which might increase the resultant tool path length dramatically. Selecting only close boundary pixels to "cover" does not guarantee the absence of single pixels, but should decrease their number. Therefore, only non-removed pixels from milling zone that are located next to removed pixels are addressed. Thus, the algorithm to build a new set of boundary pixels to “cover” can be stated as: build a set of boundary pixels, located next to removed pixels in the previous step. The main issue here is to find the milling zone pixels that were removed in the previous step (in Algorithm 4.7). The easiest way to do it is to save the milling zone map before removing pixels and compare it with the map after. The pixels that have “True” value in the old map and “False” value in the new map are the removed ones. The rest is trivial, all the boundary pixels that are next to removed ones are need to be added to the resultant set. The implementation is shown in Algorithm 4.10. Note, that this algorithm uses Algorithm 4.9 in Line 3.

Algorithm 4.10 Find new boundary pixels to “cover”

Input: mzm is a boolean map of milling zone with width W and height H
Input: bpm is a boolean map of boundary pixels width W and height H
1: ▷ Save the milling zone map before removing pixels
2: $mzm_{old} \leftarrow mzm$
3: Remove pixels, “covered” by the last tool path segment, from mzm and bpm
4: $BPs \leftarrow \{\}$ ▷ BPs is a resultant set of boundary pixels to “cover”
5: ▷ Go through all the possible milling zone points
6: **for** $y = 1..H, x = 1..W$ **do**
7:     **for** $dy = -1..1, dx = -1..1$ **do**
8:         ▷ Find a new boundary pixel to “cover”
9:         **if not** $mzm[x, y]$ **and** $mzm_{old}[x, y]$ **and** $bpm[x + dx, y + dy]$ **then**
10:             Add $(x + dx, y + dy)$ to BPs
11:         **end if**
12:     **end for**
13: **end for**
14: **return** BPs

4.9 Simulation result and discussion

In the previous sections, the implementation details of the Algorithms to build tool paths to “cover” the provided milling zone are shown. The resultant tool paths are shown and discussed in this section. First of all, the generated tool paths can be seen in Fig. 4.5. The generated tool paths are not perfect, but they match two main criteria: following the boundary and being iso-scallop. Fragment (d) shows how tool path are built for almost circular boundary line of the milling zone. The generated tool path are close to concentric circles. In fragment (c) it can be seen that the step over distance between adjacent tool paths is different and depends on the part geometry. Unfortunately, short tool paths are inevitable, because of the noisy boundary and the arbitrary milling zone. Such short tool paths are high-lighted with green circles in both Fragments (c) and (d).

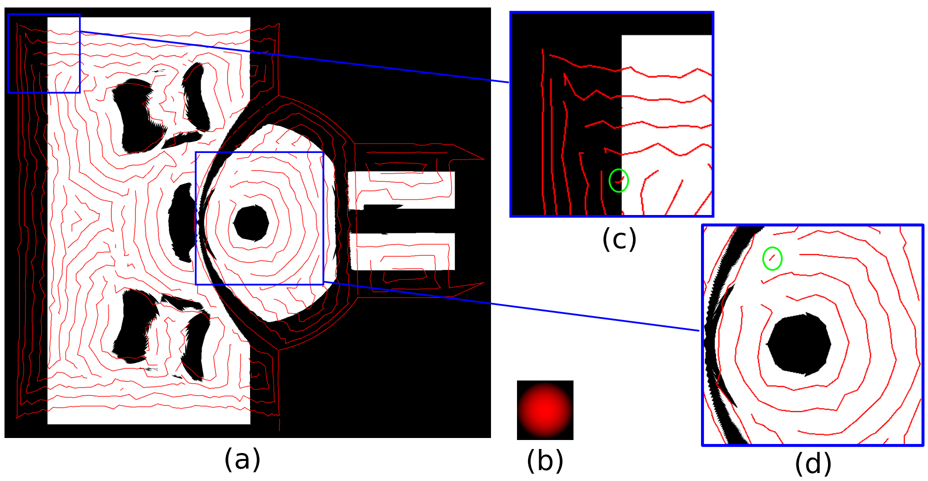

Figure 4.5: Generated tool paths (a) with its fragments (c, d) for 7.983mm ball-end tool (b). White pixels show the original milling zone, red pixels represents the generated tool paths, and green circles shows very short generated tool paths.

Using the same hardware (Intel i7-4770 CPU two AMD Radeon RX 480 GPUs) for the simulation and performing the "covered" areas calculation on GPUs, the time to build the shown tool paths for the provided milling zone is 312*sec*. About half of this time (143*sec*) is for the "covered" areas calculations. Tool path generation time can be improved by using optimization techniques from Section 4.6 and other optimizations can be done for other parts of the algorithm. E.g. some of the other operations can be implemented to run in parallel.

In this Chapter it was shown that using maps, as representative models, iso-scallop, contour parallel tool paths for an arbitrary milling zone on the free-form surface for a generalized cutter can be generated. The most time consuming algorithm was implemented to run in parallel on GPUs to reduce the simulation time. Despite the fact that generated tool paths are not perfect, they can be used for milling and they can be used to calculate the milling time for different zones and different tools as shown in the next chapters. This milling time calculation is an important operation to perform the global optimization of the tool set.

Chapter 5

Tool selection and path planning for roughing

In Chapters 3 and 4 methods to select the best cutter for surface points and to build tool paths for an arbitrary milling zone are provided. However, these methods can be used for finishing milling only. If the attempt to minimize the total machining time is made, it is essential to understand the criteria to select tool, assign zones and build tool paths for the selected tools in rough milling.

We assume here, that we have the milling zone and the tool paths for every used tool from the finishing stage. The set of used tools can be optimized as shown later in Chapter 6; this set might contain all the available tools or even one tool to traverse the whole milling surface. The provided tool paths do not need to be connected, but it should be guaranteed that the tool paths for every tool "cover" every pixel in the milling zone provided for the tool. In this work, the tool paths generated in Chapter 4 are used, but other tool paths can be used in the same manner. For simplicity, all the available tools, their milling zones and their tool paths are used to illustrate the approaches in this Chapter. However, in Chapter 6 we show how the methods from this Chapter can be used in the total machining time minimization.

5.1 Milling layers

Usually, the rough milling tool paths are generated before finishing tool paths. In this research the opposite approach is used. The rough milling tool paths and tool set can be generated based on the information from the milling layer below. Thus, starting from the finishing layer, the information for the first rough milling layer can be generated, followed by the second one and for all outer milling layers, consecutively. The definition of one milling layer is shown below by using the tolerance surface. However, using different milling layers does not mean that all of these layers are milled consecutively. That would be extremely inefficient, considering that every milling layer might use several tools, so the same tool might have to be selected several times for different milling layers. The intuition behind different milling layers is rather the order of generating tool paths that can be efficiently connected after all the tool paths for all the milling layers and for all the tools are generated. In this approach the criterion to stop generating more milling layers should be stated. This criterion is simple: if the next generated milling layer has the whole work piece inside, there is no need for more milling layers and the generation should be stopped. Fig. 5.1 shows this approach schematically.

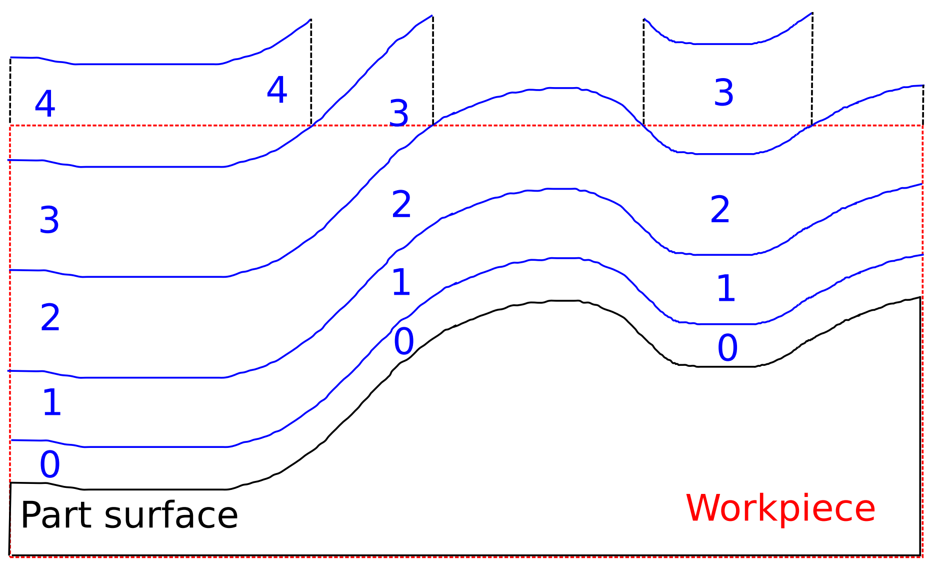

Figure 5.1: Different milling layers in order of generation. The original workpiece is shown with the red dash line, while the milling surface is black. All the milling layers are blue and are numbered in order of generation, starting with 0 for finishing and ending with 4 for the outermost milling layer. Note, that if the previous milling layer is outside of the workpiece even partially, the generation of outer milling layers in this place is stopped.

5.2 Next milling layer generation

The main approach to generate the milling layers is stated. However, the definition of the milling layer is not done yet. Recall that to build the zones of the best tools indices for finishing we assumed that we have small enough material left to support the given feed rates for all the tools. That means that the rough surface left for the finishing stage must be less or equal to the tool depth of cut in every CL point for every used tool. Therefore, after building finishing tool paths, the rough surface is defined by all CL positions and depth of cut of the used tools. In the same way, the rough surface, left by the outer rough milling layer, is defined by all CL positions of the current milling layer and depth of cut of all the used tools. This rough surface can be reconstructed by building the cylinders on every CL position, such that the center of its bottom is located in CL position, the radius of the cylinder equals the radius of the tool, the height equals the depth of cut and the axis is parallel to the milling direction ($z - axis$). Then, z coordinate for every surface point (x, y) can be defined as the maximum of z coordinate of all (x, y) points, placed on the top of all the built cylinders. The example of the reconstructed rough surface in 2D is shown in Fig. 5.2.

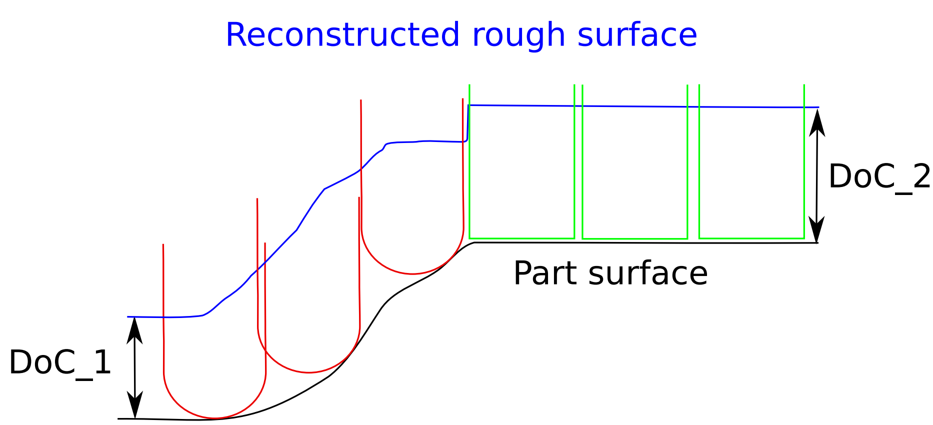

Figure 5.2: Reconstructed rough surface (blue) for two different tools (red, green) with different depth of cutters (DoC_1, DoC_2) respectively.

Such a rough surface can be easily built using the discretized model. For every used tool, the rough surface can be built independently and then, the maximum value for every surface point can be defined among the rough surfaces from every tool as shown in Algorithm 5.1.

Algorithm 5.1 Build rough surface for all tools

Input: dm_t is a tolerance surface depth map with width W and height H
Input: $Tools$ is a list of tools with their tool paths

$\triangleright$ rdm is a resultant rough surface depth map filled with $0s$
$rdm \leftarrow [1..W, 1..H]\{0\}$
$\triangleright$ Go through all the tool with their tool paths
for $\forall t \in Tools$ **do**
    $rdm_i \leftarrow$ Generate new rough surface for tool t
    $\triangleright$ Go through all the surface points
    for $y = 1..H, x = 1..W$ **do**
        $rdm[x, y] \leftarrow \max(rdm[x, y], rdm_i[x, y])$
    end for
end for

The actual algorithm to generate the rough surface for one tool is more complicated than it was described above: z position should be calculated not for every point on the top of the cylinder, but only for the points that were "covered" with the tool in this position. Otherwise, the situations like the ones shown in Fig. 5.3 might happen and the rough surface can be reconstructed the wrong way.

Algorithm 5.2 shows the implementation for the proper rough surface reconstruction for one given tool, its characteristics and tool paths. Note, that Algorithm 4.4 is used in Line 3 to rasterize the tool paths.

Recall that the meaning of the reconstructed rough surface is that the material left for the finishing milling layer (or the previous rough milling layer) should not exceed the volume bounded by this rough surface. Otherwise, for some of the CL points on the tool path the depth of cut is bigger than required and the tool might be damaged or broken or the milled part might have gouges.

It can be noted that the definition of the reconstructed rough surface is the exact

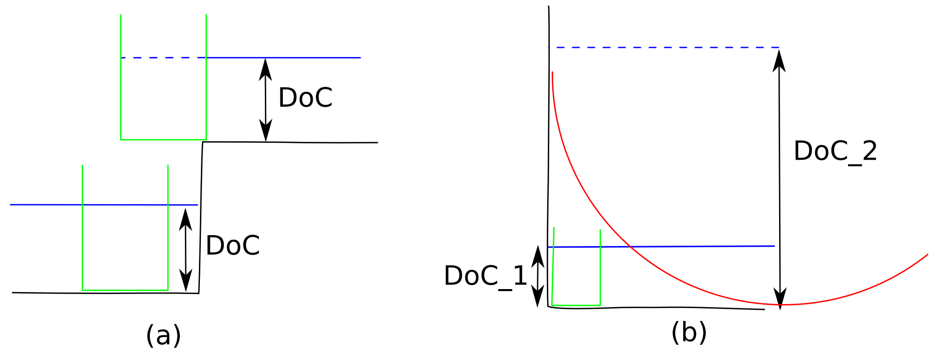

Figure 5.3: The wrong (dashed blue line) and the proper (blue line) reconstructed rough surfaces. In the first case (a) it can be reconstructed the wrong way because of the different depth of the milling surface. In the second case (b), if a big (red) tool cannot "cover" surface points, the rough milling surface should be reconstructed based on the tool that can "cover" surface points (green one). Note, that different tools have different depths of cut (DoC).

definition of the tolerance surface that was done in Section 3.1. For both of them all the material outside of the surface must be machined, gouges should not be produced during milling and the material inside the surfaces can either be machined or not machined. Therefore, the milling layer can be defined as the set of tool paths for different tools to mill the material that is inside some tolerance surface. After building the tool path, the new rough tolerance surface can be reconstructed to define the new milling layer as shown above. When, the tolerance surface bounds the whole initial workpiece, the generation of outer milling layers can be stopped. Fig. 5.4 shows the milling zones and the finishing tool paths, used for rough tolerance surface reconstructing, while Fig. 5.5 shows the reconstructed rough tolerance surface and the original finishing tolerance surface depth maps. The depth of cut values for the tools, used for the simulation is provided in the Table 3.1

Despite the fact that the new tolerance surface can be reconstructed for the next rough milling layer, the approaches from Chapters 3 and 4 cannot be used directly to define milling zones for cutters and to build rough tool paths, respectively. The modifications of the approaches, that need to be done to build the valid milling zones

Algorithm 5.2 Build rough surface for one tool

Input: dm_t is the tolerance surface depth map with width W and height H
Input: DoC_i is the depth of cut for the tool
Input: s is a discretization step
Input: dm_i is the tool surface depth map with width D_i and height D_i
Input: TPs is the set of tool paths for the tool

$\triangleright$ rdm_i is a resultant rough surface depth map for the tool filled with $0s$
$rdm_i \leftarrow [1..W, 1..H]\{0\}$
$\triangleright$ CLs is a list of rasterized CL positions
$CLs \leftarrow$ Rasterize all the tool paths from TPs
$\triangleright$ Go through all the CL positions
for $\forall cl \in CLs$ **do**
  $\triangleright$ Go through all the tool surface points
  for $dy = -\lfloor D_i/2 \rfloor .. \lfloor D_i/2 \rfloor, dx = -\lfloor D_i/2 \rfloor .. \lfloor D_i/2 \rfloor$ **do**
    $x \leftarrow cl.x + dx, y \leftarrow cl.y + dy$
    $\triangleright$ Check if cl "covers" (x, y) point
    if $cl.z + dm_i[dx, dy] \leq dm_t[x, y]$ **then**
      $rdm_i[x, y] \leftarrow \max(rdm_i[x, y]_i, cl.z + \lfloor DoC_i/s \rfloor)$
    end if
  end for
end for
return rdm_i

and the valid tool paths for rough milling are is provided in next sections.

5.3 Roughing depth of the cutter

First, the depth of the cutter in CL (x, y) position needs to be revised. For the finishing stage it was defined as the maximum cutter depth without gouging. This approach cannot be used for rough milling. The cutter depth in CL point should be moved up in some distance. Because the rough milling layer is defined by a rough tolerance surface, as was stated before, this distance should be defined by a cutter and part geometries and by a rough tolerance surface. The amount of work, done by a cutter in a (x, y) CL position is defined by a volume outside of the tolerance surface, removed by the cutter. It is limited with the depth of cut. In the used model, all the volume inside the tolerance surface is milled during the finishing stage (or inner rough milling stages), so removing too much volume inside the tolerance surface is a

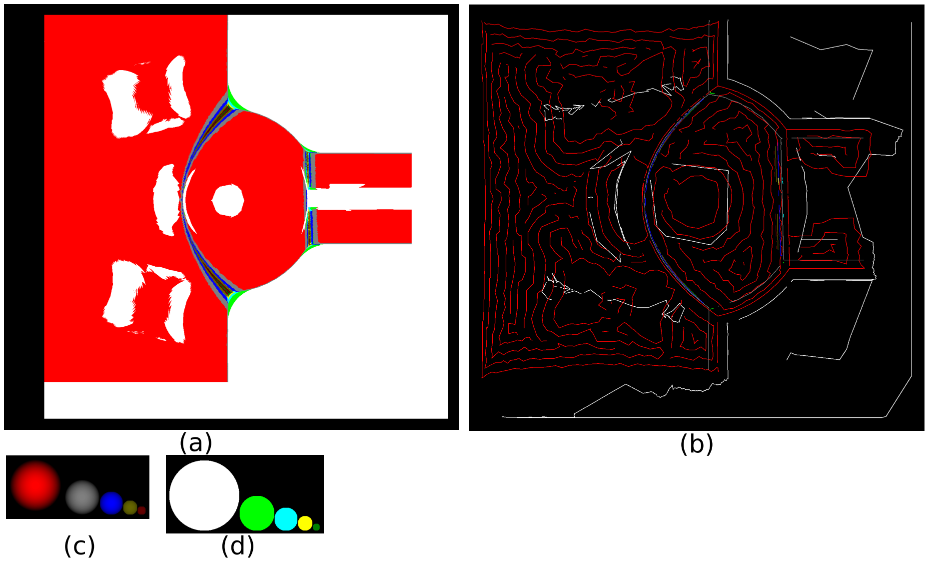

Figure 5.4: Milling zones (a) and generated tool paths for these zones (b) using five ball-end cutters (c) and five flat-end cutters (d).

waste of milling time. That means that z coordinate of the cutter should be as high as possible. However, if its z coordinate is too high, the cutter can leave too much material outside of the tolerance surface that can lead to a smaller step over distance and increase the whole tool path length, because the goal of the rough milling layer is the same: "cover" all the milling surface with the cutter. Such situations are shown in Fig. 5.6.

All the cutter depth values between (a) and (b) are a waste of milling time for rough milling. All the depth values higher than (d) do not "cover" any surface points, because the tool is completely outside of the tolerance surface. However, all the depth values between (b) and (d), do different amounts of work, by "covering" different amount of surface points. The depth of the cutter can be selected, using different strategies. For the purpose of this research, the highest tool position that can "cover" all the surface points was selected. It is (b) position in Fig. 5.6. In other words, z coordinate for an (x, y) CL point should be as high (as far from milling surface) as possible without decreasing the "finished area".

Recall that finishing depth is calculated only based on tool and part geometries.

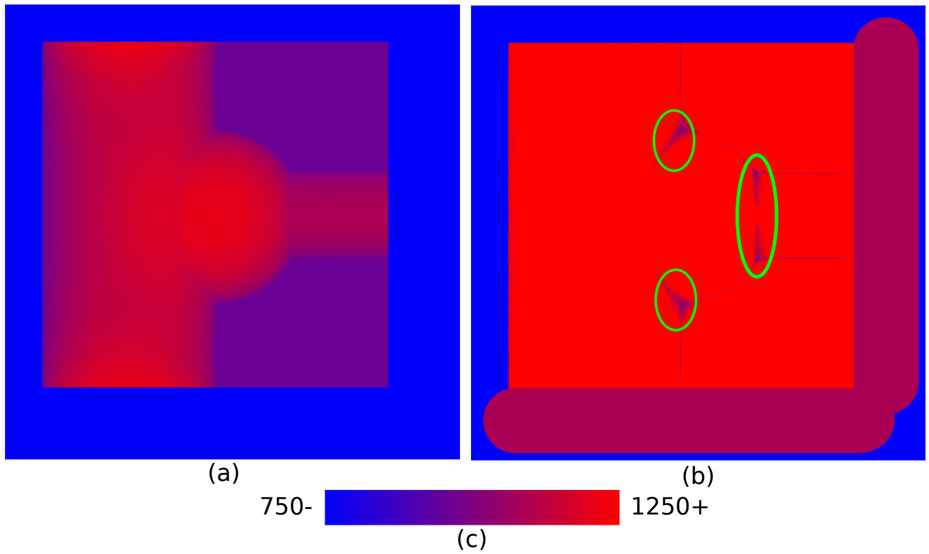

Figure 5.5: Finishing tolerance (a) and reconstructed rough tolerance (b) surfaces depth maps. Note, that the finishing tolerance surface is very close to the part depth map, because the finishing tolerance distance was selected to be 0.12mm that is three times bigger that the discretization step ($s = 0.04mm$). For big cutters, which were used in most of the areas, the reconstructed rough tolerance surface is much higher than the finishing tolerance surface. For the narrow zones, where smaller tools were used, the rough tolerance surface is not as high. Some such zones are highlighted with green circles. The range of depth (c) for both maps is from less than 3mm (750 * 0.04mm) to 5mm (1250 * 0.04mm).

The roughing depth has to be calculated using a rough tolerance surface. This is done by calculating the difference between finishing depth and roughing depth for every (x, y) CL point and then increase the finishing depth on this difference value. Therefore, for every surface point that is "covered" with a tool, in its finishing depth position, the value to lift the tool up, while keeping the surface point "covered", can be calculated. The minimum among that value for all "covered" points is the resultant difference between finishing and rough milling depths. The continuous form of the equations for the roughing depth of the cutter (d_{ri}) is:

$$d_{ri}(x, y) = h_1 + \min_{\substack{x_l \in [-r_i, r_i] \\ y_l \in [-r_i, r_i]}} \begin{cases} h_3 - (h_1 + h_2), & h_1 + h_2 \leq h_3 \\ \infty, & otherwise \end{cases} \tag{5.1}$$

where

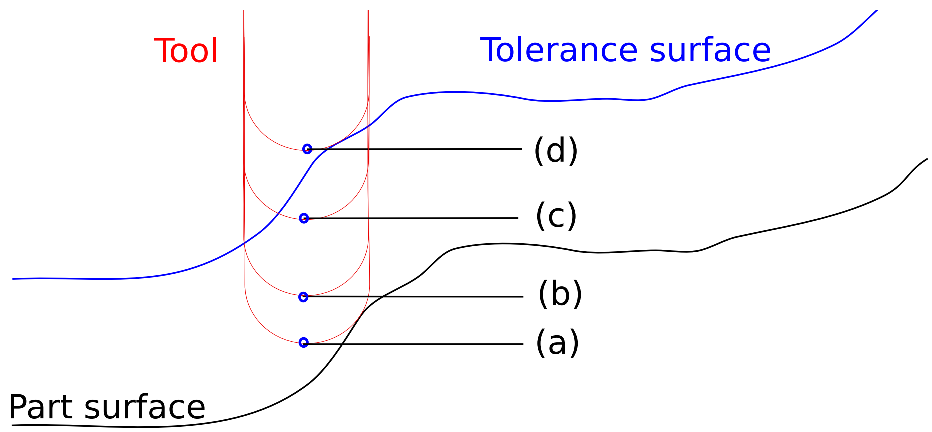

Figure 5.6: Different cutter depth values: finishing cutter depth (a), the maximum cutter depth to "cover" all the surface points under the cutter (b), the minimum cutter depth to "cover" any surface points (d) and the intermediate depth value, where the cutter "covers" some surface points (c).

$h_1 = d_i(x, y)$, finishing cutter depth in a cutter location point,

$h_2 = z_i(x_l, y_l)$, the depth of the tool surface point,

$h_3 = z_t(x + x_l, y + y_l)$, the depth of the tolerance surface

Note, that the same notation as in Chapter 3 is used. For the discretized model, the corresponding map of roughing depth of the cutter (drmc_i) is:

$$\mathrm{drmc}_i[x, y] = m_1 + \min_{\substack{x_l = -r_{vi}..r_{vi} \\ y_l = -r_{vi}..r_{vi}}} \begin{cases} m_3 - (m_1 + m_2), & m_1 + m_2 \leq m_3 \\ \infty, & otherwise \end{cases} \tag{5.2}$$

where

$m_1 = \mathrm{dmc}_i[x, y]$, cutter depth in a cutter location point,

$m_2 = \mathrm{dm}_i[x_l, y_l]$, the depth of the tool surface point,

$m_3 = \mathrm{dm}_t[x + x_l, y + y_l]$, the depth of the tolerance surface.

This algorithm of calculating the rough depth of cutter values has the same complexity as the hardest algorithms from Chapter 3 as described in Section 3.5 and the performance of it on the used hardware is shown in Table 3.2. Fig. 5.7 shows the finishing and roughing depth of cutter maps for the biggest 9.525mm flat-end cutter. The roughing depth of the cutter map is built for the tolerance surface shown in Fig. 5.5.

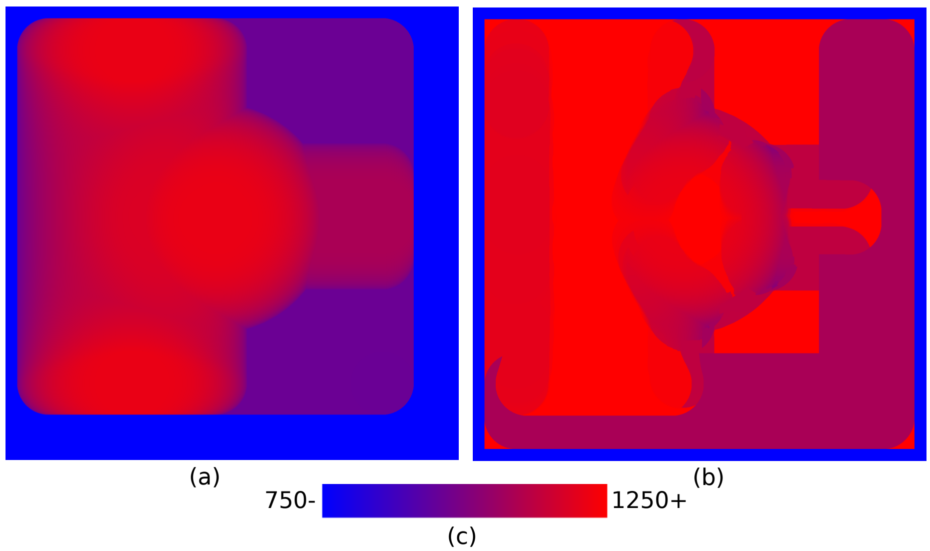

Figure 5.7: Finishing (a) and roughing (b) depth of the cutter maps with the legend (c) for 9.525mm flat-end cutter. The value for every CL position in the roughing map is more or equal than the value for the same CL position in the finishing map.

5.4 Performance criterion for rough milling

The goal for the rough milling is different from the finishing. For finishing, the goal is to traverse the whole milling surface as fast as possible, but for the rough milling, the goal is to remove as much material as possible. Therefore, the second thing to revise is a tool performance in a CL position. Recall that for the finishing milling stage the performance criteria was the "finished area", i.e the area, milled within the tolerance surface. Based on the goal of the rough milling, the performance criterion should be defined with the milled volume. Because it is guaranteed that the volume inside the tolerance surface is milled with other milling stages, only the volume outside of the tolerance surface matters. In addition, such a volume should only be calculated for the surface that is "covered" by the cutter in the CL position. Recall from the previous section, that the "covered" surface for finishing tool depth and roughing tool depth is the same, so the "covered" surface is not reduced by placing the cutter in the roughing depth. For the volume, built on the "covered" surface outside of the tolerance map, the term "roughed volume" is used. It can be calculated by summing the part of the material columns volume, built on covered surface points. The considered part of a material volume should be outside of the tolerance surface but inside the cutter's depth of cut. Such a "roughed volume" is schematically shown in Fig. 5.8.

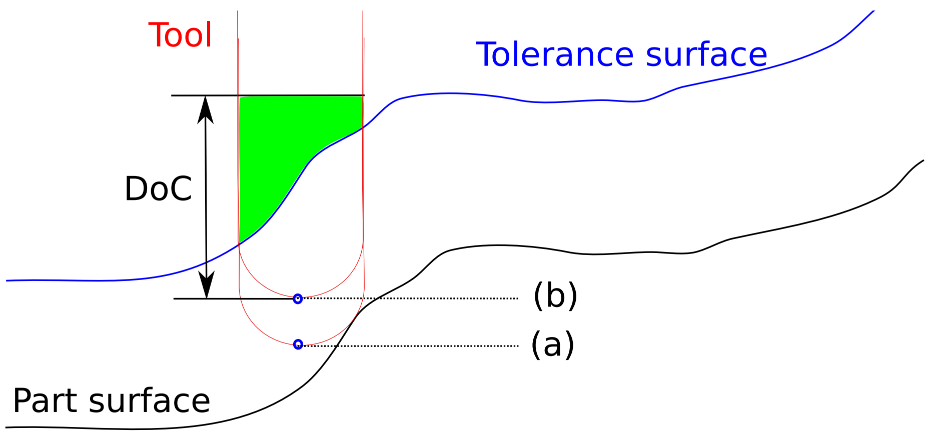

Figure 5.8: "Roughed volume" for a CL position. Two depths of the cutter are shown: the finishing one (a) and the roughing one (b). The volume, outside of the tolerance surface but within depth of cut is the "roughed volume" (green).

The equation to calculate "roughed volume" (V_i) for the (x, y) CL position is:

$$V_i(x,y) = \iint_{(x_l,y_l)\in S_i} \begin{cases} dx_l dy_l * (h_1 + \mathrm{DoC}_i - h_3), & h_1 + h_2 \leq h_3 \\ 0, & otherwise \end{cases} \tag{5.3}$$

where

S_i , the circle with the center at the origin and the radius r_i,

h_1 $= d_r i(x, y)$, roughing cutter depth in a cutter location point,

h_2 $= z_i(x_l, y_l)$, the height of the tool surface point,

h_3 $= z_t(x + x_l, y + y_l)$, the height of the tolerance surface.

DoC_i , the depth of cut for ith tool.

To build the map, the integration is reduced into summation and the finished volume map (vm_i) for ith tool can expressed:

$$\mathrm{vm}_i[x,y] = \sum_{x_l=-r_{vi}}^{r_{vi}} \sum_{y_l=-r_{vi}}^{r_{vi}} \begin{cases} m_1 + \lfloor \frac{\mathrm{DoC}_i}{s} \rfloor - m_3, & m_1 + m_2 \leq m_3 \\ 0, & otherwise \end{cases} \tag{5.4}$$

where

$m_1 = \mathrm{dmc}_i[x,y]$, the roughing cutter depth in a CL point,

$m_2 = \mathrm{dm}_i[x_l,y_l]$, the depth of the tool surface point,

$m_3 = \mathrm{dm}_t[x+x_l,y+y_l]$, the depth of the tolerance surface.

5.5 Algorithms adaptation for rough milling

Having the methods to calculate roughing depth of the cutter and “roughed volume”, two main algorithms can be adapted for rough milling. The first algorithm to modify is building milling zones for the used tools. There are two modifications that need to be done: the roughing depth of the cutter map needs to be built and the “roughed volume” as a performance criterion needs to be used instead of the “finished area”. In addition, one more condition needs to be checked: in building the surface performance maps, if the workpiece depth in the surface point is already inside of the tolerance surface, there is not need to mill any material from this point. The rest of the algorithm to is the same as for finishing.

For the tool path building only one change needs to be made. Every time when z coordinate for CL (x,y) positions is defined, the roughing depth of the cutter map needs to be used. However, when checking if a CL line segment produces gouges, the finishing depth of the cutter map needs to be used as before.

The result of the two main algorithms for the first rough milling layer can be seen in Fig 5.9. Note that most of the milling surface is black, which means the workpiece is located outside the tolerance surface in black points. The biggest flat-end cutter (white) is used for the most rough milling zones. However, if it cannot "cover" some surface points, smaller ball-ends and flat-end tools are used as can be seen in fragments.

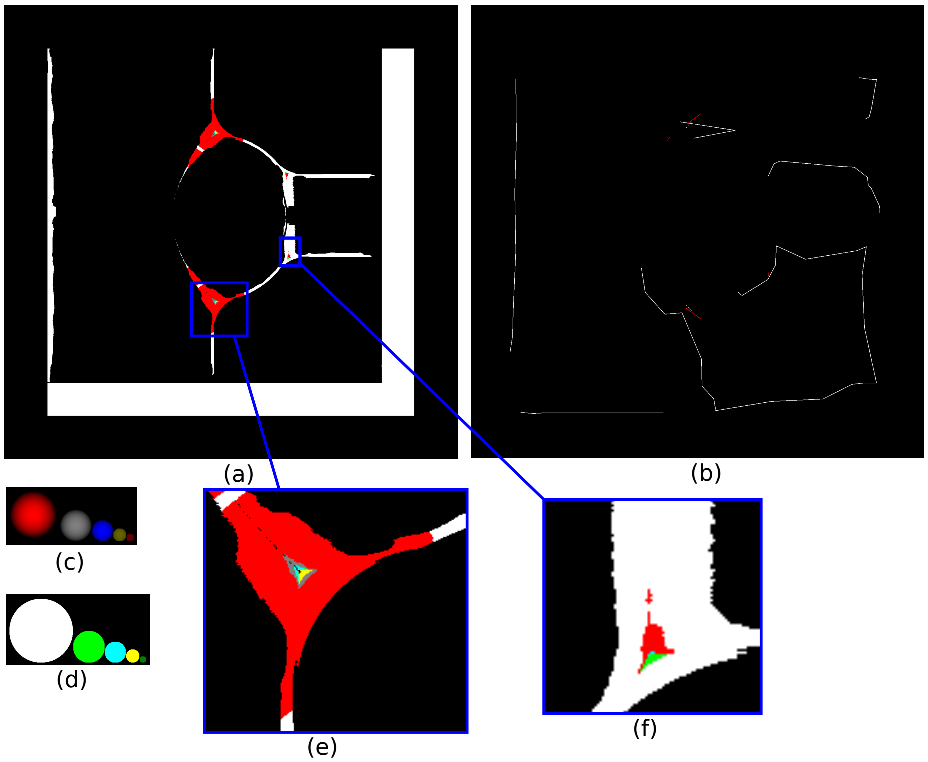

Figure 5.9: Roughing zones for cutting (a) with its fragments (e), (f) and built tool paths (b) for the available ball-end (c) and flat-end (d) cutters.

In this Chapter, the approach to build outer milling layers, based on the information from inner milling layers was shown. The adaptations for the main algorithms from previous chapters were also described. Now the best tools for surface points can be found and the tool paths can be built for all milling layers. The problem of the total machining time minimization is considered in the next chapter.

Chapter 6

Tool set optimization to minimize the total machining time

In the previous chapters, methods to assign milling zones for cutters and to build tool paths for all stages of the milling were provided. However, it was assumed that the set of cutters to use is given and equals to the set of all cutters for the previous applications. Because the algorithm to assign zones for cutters calculates the best cutter for every surface point independently, it is likely that all the cutters are used in milling zones map. It can be seen from Fig. 3.15 that for the representative part, all the available cutters are used.

Recall, that milling zones are built in such a way as to minimize the total milling time for all the tools. However, using more tools is expensive. For every additional tool used, the tool changing time is added to the total machining time. But if a tool is removed from the list of tools, the milling time might be increased. The balance needs to be found between the milling time and number of tools to use. The goal of this chapter is to show the approach to find the optimal tool set to have the minimum total machining time, with reasonable assumptions.

6.1 Total machining time calculation

First of all, the equation to calculate the total machining time should be defined. The total machining time comprises the milling time for every used tool, the time of rapid motions and the time to change the tool. Using N tools, the equation for the total machining time is:

$$T = \sum_{i=1}^{N} T_{mi} + \sum_{i=1}^{N} T_{ri} + NT_c$$

where

T , total machining time,

T_{mi} , total milling time for ith tool,

T_{ri} , rapid motions time for ith tool,

T_c , average tool changing time for CNC machine

Assuming that the tool path for one tool is composed of straight line segments, the time for every segment can be calculated by using the trapezoid motion interpolation model as explained in Section 4.5. The same model can be used for both milling motions and rapid tool motions, using the different feed rate values: the milling feed rate and the rapid CNC machine feed rate, respectively.

In Chapters 4 and 5 the methods to generate tool paths to "cover" the whole milling zone for different milling layers were shown. To calculate the total machining time for one cutter, tool paths from different milling layers should be effectively connected together with either milling or rapid motions.

6.2 Defining the order of tool selection

Having generated tool paths for different tools and for different milling layers the problem of connecting them should be addressed. It is known that all the tool paths together do all the necessary milling work and the global goal is to minimize the total milling time. Thus, the tool paths for one tool should be connected in such a way as to minimize the time of motions between tool paths. It can be seen that this problem is similar to the travelling salesman problem (TSP). The TSP asks the following question: given the list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the original city [36]. There are many varieties of this problem. E.g. the requirement to return to the original city may be omitted or additional constraints may be imposed on the order of path between cities. However, most of the varieties do not affect the complexity of the problem. The problem is NP-hard which means that it is considered impossible to find the exact solution to the problem in polynomial time. Many heuristic methods can be used to solve it, though.

The problem of connecting tool paths is also NP-hard, because it is at least as hard as the TSP problem. Due to the fact that finding the optimal solution for a NP-hard problem is not the goal of this research, only the formulation of the problem and the simple greedy algorithm as a solution are provided. The provided algorithm cannot find the optimal solution for all the cases. However, it can find a good solution fast. Other heuristic algorithms can be used to archive better result.

Considering the above, the input data for the algorithm of connecting all the tool paths are the tool paths for different tools on different milling layers. The output is the tool order to mill and the single tool path for every tool made of straight line segments. For every segment it is known if it is a milling segment or a rapid motion segment. Such information can be easily post-processed into G-code for the used CNC machine. Milling straight line segment motions are converted into "G1" code, while rapid motions are converted into "G0" code. The code to change the tool is also trivial to build. With the output data it is also easy to calculate the total machining time as stated in Section 6.1. It is worth noting that this algorithm is used during tool set optimization. If the machining time need to be calculated for one milling layer, the special case of the algorithm is used: all the tool paths are independent. In general case, different milling layers are considered, which means that there are dependencies between tool paths. The algorithm of connecting tool paths from different layers has two parts: tool order building and connecting tool path segments for every tool in order.

First, the proper tool order should be built. It is assumed that every tool can be used only once, i.e. after using one tool a CNC machine never switch to this tool again. If the tool order is wrong, e.g. if smaller tools used before the bigger ones, it can lead to tool or part damaging. In traditional approaches to tool path planning, operators are responsible to select the tools in the right order, but in this research, the tool order must be built automatically. The best approach to build the proper tool order is to find all the dependencies between tool paths from different tools on different layers and build the dependencies graph. Then, using this graph it is possible to build the proper tool order, starting from the tool that does not depend on any other tools and finishing with the one that depends on all the other tools. It is assumed here, that it is always possible to build such an order, i.e. the dependencies graph has no cycles. A dependence between two tool paths on different milling layers means that without milling the one from the outer layer, the one from the inner layer cannot be milled. All the tool paths on the same milling layer are considered independent. In addition, only a tool path from the inner layer can depend on the tool path from outer layer, not vice versa. Below, the exact algorithm for how dependencies are detected is shown.

In this work, however, the dependencies graph for different tools is not built. Recall, that tools to mill every layer are not random, for every layer, the optimal tool set is created and the milling zones are defined based on tool performances. That means that if a tool is already used for some point, on some layer, the same or bigger tools will be used for outer layers. And the tool order can be simply built from the bigger tools to smaller ones. The exact proof that the bigger tool must always be used before the smaller one is not addressed in this work, but seems quite reasonable. If there are any doubts, the proper tool dependencies graph should be built to define the proper tool order. If two tools have the same size but different shape it is trickier to select the one that should be used first. In this work, the tool with the highest depth of cut is selected first. Thus, the algorithm to build the tool order can be stated as: build the set of all the used tools by uniting tool sets from every milling layer and sort the resultant tool set from the biggest depth of cut to smallest. The order of tools in the sorted set is the order to change the tools in CNC Machine and the order to build a single tool path for every used tool. The next step is to connect all the tool paths from every milling layer for every used tool.

6.3 Dependencies between tool paths building

To connect all the tool paths for one tool, first the dependencies should be defined for all the tool paths from every two consecutive milling layers as shown in Algorithm 6.1.

Algorithm 6.1 Build all dependencies for milling layers

Input: Ls is a milling layers with their tool paths set of size N for one tool

```
1:  Ds ← {}                                   ▷ The resultant set of tool paths dependencies
2:  ▷ Go through all the milling layers, except the first one
3:  for i = N..2 do
4:      ▷ Go through all the tool paths on a milling layer and the next inner one
5:      for ∀tp1 ∈ Ls[i], ∀tp2 ∈ Ls[i − 1] do
6:          if tp2 depends on tp1 then
7:              Add the dependence (tp1, tp2) into Ds
8:          end if
9:      end for
10: end for
```

Note, that all the tool paths on the same milling layer are considered independent, so only all the possible pairs of tool paths from two consecutive milling layers should be addressed. In addition, the dependencies are not checked if the distance between milling layers is more than one. It is considered that if the same tool is used on several milling layers, the dependencies can be formed only from neighboring milling layers.

The code in the algorithm is trivial except for the one sub-algorithm to define the dependence between two tool paths on different layers on line 6. Recall, that two tool paths are considered dependent if before milling one, another one must be milled, i.e. one is blocking another one. The dependence can be easily checked if both tool paths, drawn with the circular brush with the tool radius on the (x, y) plane, have intersections. z coordinate can be ignored in drawing. Fig 6.1 shows a simple example of dependent and independent tool paths.

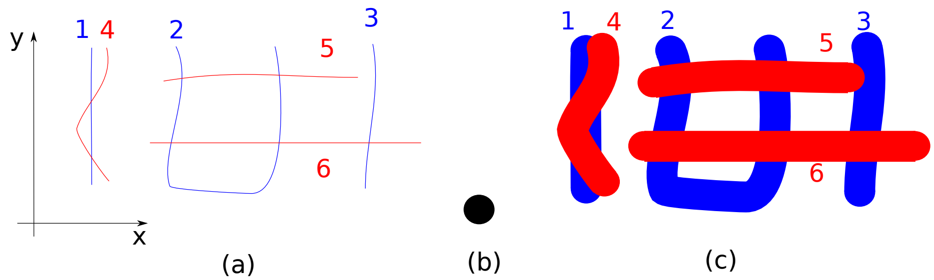

Figure 6.1: Tool paths on inner (blue) and outer (red) consecutive milling layers (a), tool size (b) and drawn with a circular brush tool paths. The inner tool paths are 1, 2 and 3, while the outer tool paths are 4, 5 and 6. The full list of dependencies is: $[\{1 \rightarrow [4]\}, \{2 \rightarrow [5, 6]\}, \{3 \rightarrow [5, 6]\}]$. Note, even if the tool paths 3 and 5 are not intersected, they are dependent.

With the used discretized model, this algorithm to check two tool path dependencies can be implemented with calculating the distances on (x, y) plane from every surface point to all the CL positions from both tool paths. If both distances from a surface point to any two CL positions belonging to different tool paths are less than the tool radius, the tool paths are dependent and this surface point is the point of tool paths intersection. All segments in both tool paths should be rasterized with the Algorithm 4.4; z coordinate of every rasterized CL position is ignored. Pseudo code for the algorithm is shown in Algorithm 6.2. This algorithm is also implemented on GPU, because every surface point can be checked independently. Note, that the distances in lines 5 and 6 are simple Euclidean distances.

After building all the dependencies for all the tool paths on different milling layers for the given tool, everything is ready to build one single connected tool path out of all tool paths for the tool. Recall, that the optimal solution is not provided here, due to the NP-hardness of the problem. A simple greedy algorithm is provided, but

Algorithm 6.2 Check if two tool paths dependent

Input: CLs_1 and CLs_2 are two rasterized tool paths, respectively
Input: width W and height H is the size of milling zone
Input: r is a tool radius

```
▷ Go through all the surface points except the first one
for y = 1..H, x = 1..W in parallel do
    ▷ Go through all CL positions in both tool paths
    for ∀cl1 ∈ CLs1, ∀cl2 ∈ CLs2 do
        ▷ Calculate distances from the surface point to both CL positions
        d1 ← distance between (x, y) and (cl1.x, cl1.y)
        d2 ← distance between (x, y) and (cl2.x, cl2.y)
        if d1 ≤ r and d2 ≤ r then
            return True
        end if
    end for
end for
return False
```

better heuristic algorithms can be used.

6.4 Greedy approach to connect all tool paths for one tool

The problem of connecting tool paths for one tool can be stated as: having N tool paths with dependencies, find the fastest tool motions to connect all the tool paths in one single tool path, having all the dependencies satisfied. For simplicity, in the algorithm of connecting tool path, the Euclidean distance is considered an equivalent of the time for tool motions in connecting tool paths. That means that if two points are closer two each other, we consider that they can be connected with faster tool motions. It is not always true, because tool motions should be gouge-free, but it is a reasonable assumption to make. In better algorithms the time to connect two CL positions with gouge-free tool motions should be considered instead of the distance between these positions.

A greedy approach is good to solve hard problems like this, but it cannot guarantee the optimal solution. The main idea of all greedy algorithms is to select the local optimal case for the next iteration, without considering the global optimality. It might not lead to the optimal final solution of the problem, but the solution can be found extremely fast and it can be good for most cases.

For the current problem, the main steps of the greedy approach are described here. First, any tool path without dependencies is selected as the final tool path. Then the next best tool path to connect to the final tool path is sought. The best tool path should not have dependencies and the distance from any of its ends should be closer to the end of the final tool path than any other ends of other tool paths. The procedure is repeated iteratively until all the tool paths are connected to the final tool path. The tool path is considered to be without dependencies if all dependent tool paths are already connected to the final tool path. The pseudo code is shown in Algorithm 6.3.

The only non-trivial part of the algorithm is connecting two tool paths together in line 31, because they should be connected with tool motions that do not produce gouges. This sub-algorithm is addressed in the next Section. Note: the distances in lines 16 and 17 are the Euclidean distances. If the single tool path is built out of tool paths from one milling layer, the set of dependencies is empty, but the whole approach is the same.

6.5 Connect two tool paths in one

Connecting two tool paths would be a trivial problem if not for a part geometry. The connecting tool motions should be gouge-free and take the minimum time. Because both ends of the tool paths to connect are known, the problem can be stated as the connection of two CL points with the fastest gouge-free tool path. Recall, that no milling work needs to be done here, so the motions can be either milling or rapid, only the fastest total motions time is required. In this research, the best solution for this problem is not provided. Three connection tool paths are tested instead. For each one the motions time is calculated and the minimum motions time is selected.

Algorithm 6.3 Connect tool paths

Input: TPs is a set of tool paths
Input: Ds is a set of tool path dependencies

▷ Find a tool path without dependencies
for $\forall tp \in TPs$ **do**
    if $tp \notin Ds$ **then**
        $f \leftarrow tp$ ▷ f is a resultant tool path
    end if
end for
▷ Do until all the tool path from TPs connected to f
while $\exists tp \in TPs, tp \notin f$ **do**
    $d_{min} \leftarrow \infty$ ▷ d_{min} is the shortest distance
    $tp_{best} \leftarrow \{\}$ ▷ tp_{best} is the closest segment
    ▷ Find the next tool path to connect
    for $\forall tp \in TPs$ **do**
        ▷ ignore tp if it is already connected to f or has dependencies
        if $tp \notin Ds, tp \notin f$ **then**
            ▷ Calculate distances from the end if f to both ends of tp
            $d1 \leftarrow$ distance between end of f and the beginning of tp
            $d2 \leftarrow$ distance between end of f and the end of tp
            ▷ If the end of segment is closer than the beginning, reverse it
            if $d2 < d1$ **then**
                $d1 \leftarrow d2$
                Reverse tp
            end if
            ▷ if tp is closer than other ones checked, save it
            if $d1 < d_{min}$ **then**
                $d_{min} \leftarrow d1$
                $tp_{best} \leftarrow tp$
            end if
        end if
    end for
    ▷ Connect the best tool path to the final tool path
    Connect tp_{best} to the end of f
    Remove tp_{best} from Ds
end while
return f

The first connection motion is the straight line segment milling motion. If this motion does not produce gouges, the milling time is calculated and compared with the times of other motions. The problem of checking if straight segment tool motions produce gouges was addressed in Section 4.4.

The second connection motions are the rapid motion vertically up, so that the tool is outside of the workpiece, then the rapid straight motions to the (x, y) coordinates of the second CL positions and the milling motion vertically down to reach the z coordinate of the second CL position. Such a motion can be accomplished in every conditions and no gouge checking needs to be done.

The last connection motion is the straight motion in (x, y) plane, where the depth for every tool position is calculated based on the maximum tool depth in the point without gouges. The tool "follows" the maximum cutter depth map. Such a motion is gouge-free by the definition and also can be accomplished in every conditions. All three connection motions are shown in Fig. 6.2 in 2D,

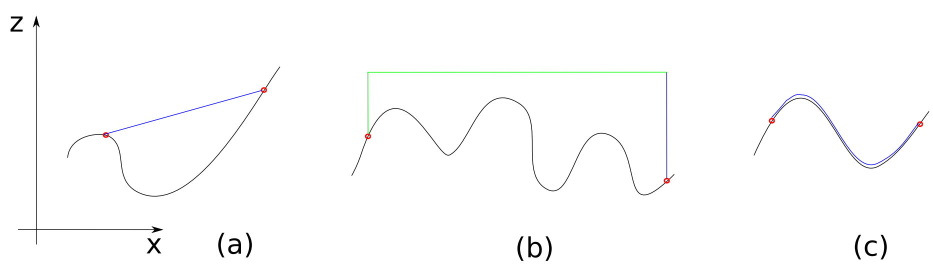

Figure 6.2: Straight line segment milling connection (a), Outside of the workpiece connection (b) and "following" the part connection (c). Note, that tool part surface is shown in black, milling motions are shown in blue and the rapid motions are shown in green.

After building all three connection motions, the time for every one is calculated and the fastest connection motions are selected. The connection of all the tool paths, generated for the specified milling zone and for 7.983mm ball-end tool, for the finishing milling layer as described in Section 4.9 is shown in Fig 6.3. It can be seen, that sometimes long connection tool paths are generated, as might happen with greedy algorithms.

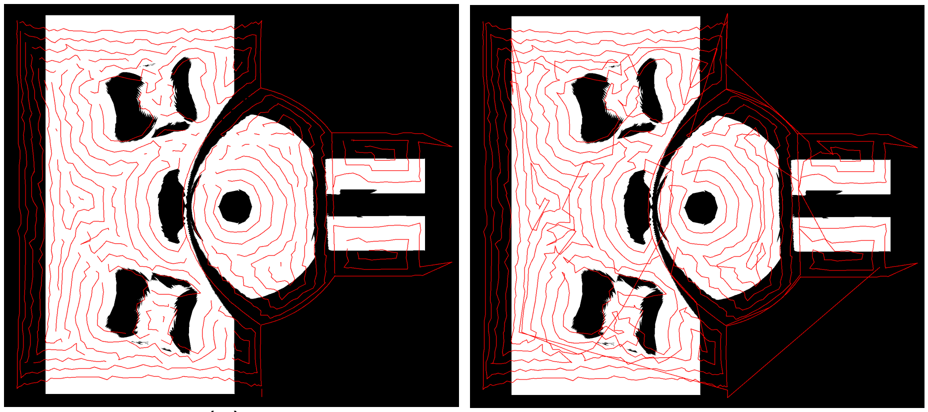

Figure 6.3: Generated tool paths for the milling zone (a) and connections between them with the greedy approach (b). White pixels show the original milling zone, red pixels represents the generated tool paths.

6.6 Greedy approach to tool set optimization problem

As was shown in previous chapters, the milling zones can be defined for every cutter from the set of used cutters and tool paths can be built for different milling layers. After this, the tool order can be defined and all the tool paths can be connected in a single tool path for every tool. Finally, the total machining time can be calculated, considering tool paths for every used cutter and the time of changing tools. With the different set of cutters to use the whole process can be repeated and the new total machining time can be calculated. Repeating this process for all the possible tool sets, the minimum total machining time can be found. The main assumption in the tool set optimization process is that the total machining time depends much more on tool set than on tool path for every tool. It is very expensive to find the truly optimal tool path for every tool and its milling zone, and it is considered that the milling time of the generated tool path is a good evaluation of the truly optimal (minimum) milling time.

The brute force approach of trying all possible tool set would work but it is very expensive. It is known that for the set of size N the number of subsets is 2^N, because every element in the original set can be either included in a subset or not. That means that if a set of all available tools contains N tools, the whole algorithm to define milling zones for tools, build tool paths, connect them and calculate the total machining time should be repeated 2^N times to find the optimal tool set with the minimum machining time. With increasing the number of available tools, the time complexity increases exponentially which is unacceptable with the big number of available tools. Better approaches can be used instead. There are no guarantees that they can find the optimal solution in all the cases, but they can find a good solution much faster.

The main idea of many optimization algorithms can be stated as: starting with an arbitrary solution, search of other solutions and if a better solution is found, repeat the process, starting with the better solution, until no further improvement is possible. If only closer solutions are searched, this approach might get stuck in the local optimum. To avoid this, different techniques to randomly change the solution can be used.

In this research, only a limited set of solutions are searched in order to find the better solution. The closer solution is a tool set that can be obtained by removing one tool from the previous tool set. The starting solution is the set of all the available tools. Thus, after checking all the tool sets without every tool from the starting set, the one that generates less total machining time is selected as the new starting set. If it is not possible to find better total machining time by removing one tool from the starting set, the algorithm stops. Because of the fact that the best local improvement of the tool set is selected for every iteration, this algorithm is also greedy. It has all the same advantages and disadvantages as were discussed above, but it can find good solutions quickly. Algorithm 6.4 shows the pseudo code for the tool set optimization algorithm.

In other word, the less efficient tools are removed from the list of all tools iter-

Algorithm 6.4 Optimize tool set

Input: T_{all} is a set of all available tools
1: $M_f \leftarrow$ Build feasibility map with T_{all} tool set
2: $t_{min} \leftarrow$ Calculate total machining time with T_{all} tool set
3: $T_s \leftarrow T_{all}$ $\triangleright$ Put all tools into the starting tool set.
4: $T_p \leftarrow \{\}$ $\triangleright$ T_p is the starting tool set on the previous iteration
5: $\triangleright$ Do until no better tool set is found
6: **while** $T_s \neq T_p$ **do**
7:     $T_p \leftarrow T_s$ $\triangleright$ Save the best tool set
8:     $\triangleright$ Try to remove every tool (c) from the T_p
9:     **for** $\forall c \in T_p$ **do**
10:         $T_c \leftarrow T_p \setminus \{c\}$ $\triangleright$ Build the tool set candidate by removing c from T_p
11:         $M_{fc} \leftarrow$ Build feasibility map with T_c tool set
12:         $\triangleright$ Make sure that tools from T_c can mill the same surface as all tools
13:         **if** $M_{cf} == M_{fc}$ **then**
14:             $t_c \leftarrow$ Calculate total machining time with T_c tool set
15:             **if** $t_c < t_{min}$ **then** $\triangleright$ Save better tool set and machining time
16:                 $t_{min} \leftarrow t_c$
17:                 $T_s \leftarrow T_c$
18:             **end if**
19:         **end if**
20:     **end for**
21: **end while**
22: **return** T_s $\triangleright$ Return the best found tool set

atively, while it leads to decreasing the total machining time. The feasibility map that is built in Lines 1 and 11 is the boolean map that shows which surface points can be removed by the given tool set on the finishing stage of milling. It is essential to compare the feasibility maps (Line 13) to make sure that the new tool set can "cover" all the same surface points as all available tools. If it is not checked, all the smallest tools are likely to be removed first, but it is important to have them in the final tool set to reach the required precision. This feasibility map can be easily build by combining all the surface performance maps together. If at least one tool does not show zero performance in a surface point, the map value for this point is "True", otherwise it is "False". Comparing two boolean maps is also a trivial operation. Fig. 6.4 shows the fragments from the original feasibility map for all the available tools and from the feasibility map for all the tools except the smallest 1.75mm flat-end cutter.

Because the feasibility map for all the tools without the smallest flat-end cutter is smaller than the one for all the tools, this cutter cannot be removed from the optimal tool set.

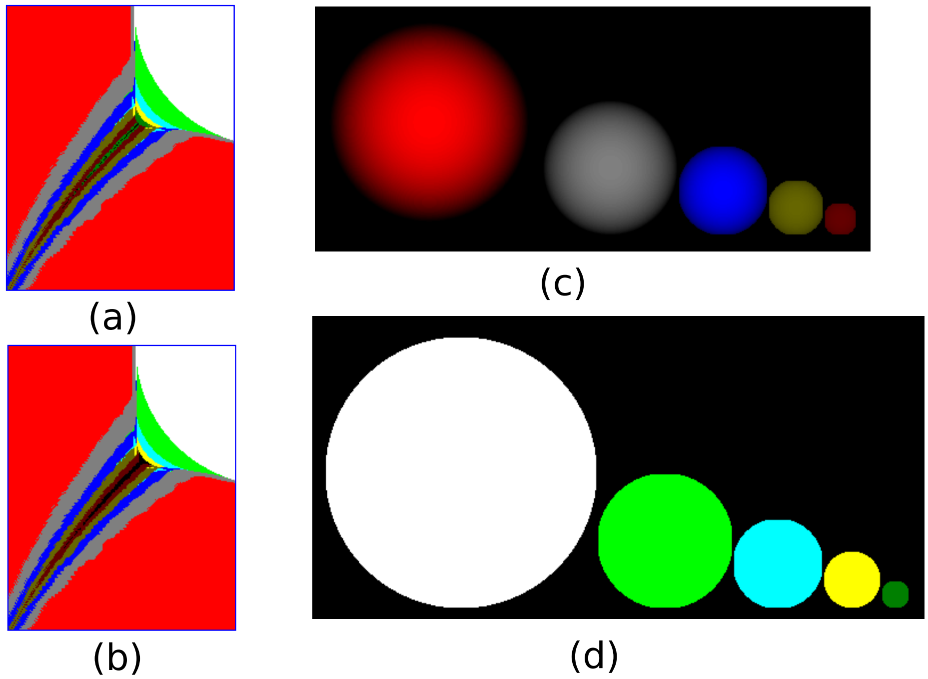

Figure 6.4: The fragment of the feasibility map for the set of all the tools (a), the fragment of the feasibility for all the tools except the smallest 1.75mm flat-end cutter (dark green) (b) and all the ball-end tools (c) and flat-end tools (d). Note, that feasibility map (b) does not have the dark green color and some surface points (black) cannot be "covered" with other cutters.

Note, that calculating the total machining time in Lines 2 and 14 is the whole cycle of building milling zones for the tools, generating tool paths, connecting them and, finally, calculating the machining time. The complexity of the whole optimization algorithms is $O(N^2)$, where N is the number of all available tools, considering that the time of calculating the total machining time for the given set of tools is constant. It is much better than $O(2^N)$ as in the brute force approach, described above. The intermediate steps and the result of the optimization algorithm using all the cutters from Table 3.1 is shown in Table 6.1. Note that the value for the time to change the tool in CNC is assumed to be 40 seconds.

Table 6.1: Tool set optimization steps

Step 1										
Starting tool set	T1, T2, T3, T4, T5, T6, T7, T8, T9, T10									
Machining time, sec	588									
Tool to remove	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Machining time, sec	563	555	542	552	548	554	550	N/A	N/A	644
Step 2										
Starting tool set	T1, T2, T4, T5, T6, T7, T8, T9, T10									
Machining time, sec	542									
Tool to remove	T1	T2		T4	T5	T6	T7	T8	T9	T10
Machining time, sec	516	505		501	501	510	499	N/A	N/A	583
Step 3										
Starting tool set	T1, T2, T4, T5, T6, T8, T9, T10									
Machining time, sec	499									
Tool to remove	T1	T2		T4	T5	T6		T8	T9	T10
Machining time, sec	475	460		455	478	458		N/A	N/A	541
Step 4										
Starting tool set	T1, T2, T5, T6, T8, T9, T10									
Machining time, sec	455									
Tool to remove	T1	T2			T5	T6		T8	T9	T10
Machining time, sec	439	424			424	415		N/A	N/A	495
Step 5										
Starting tool set	T1, T2, T5, T8, T9, T10									
Machining time, sec	415									
Tool to remove	T1	T2			T5			T8	T9	T10
Machining time, sec	396	378			382			N/A	N/A	456
Step 6										
Starting tool set	T1, T5, T8, T9, T10									
Machining time, sec	378									
Tool to remove	T1				T5			T8	T9	T10
Machining time, sec	459				353			N/A	N/A	412
Step 7										
Starting tool set	T1, T8, T9, T10									
Machining time, sec	353									
Tool to remove	T1							T8	T9	T10
Machining time, sec	508							N/A	N/A	430

The table shows the starting tool set on every optimization step, and its total machining time. Then, it shows the total machining time for all the sets, after removing one tool. Finally, the tool set with the minimum total machining time is selected as the new starting tool set. The optimization algorithm stops on step

7, because there are no tools in the tool set that can be removed to decrease the machining time. Note, if a tool cannot be removed from the tool set, because of the feasibility map difference, the total machining time is not calculated and is represented as "N/A" in the Table. This happens for the smallest ball-end and flat-end tools (T8, T9). The resultant set, after optimization, has the smallest flat-end (T9) and ball-end (T8) cutters and the biggest flat-end (T1) and ball-end (T10) ones.

On the given hardware, the average time for calculating the total machining time for one tool set is 7 minutes. For the whole optimization process the total machining time was calculated 35 times, which takes about 4 hours for the whole optimization process. It is slow, so, more optimizations can be done to accelerate the algorithm.

First, it should be noted that the building tool paths for the finishing milling layer is the most time consuming operation. However, for many tool sets, the tool paths should be built for the same tool and the same milling area as shown in Fig. 6.5.

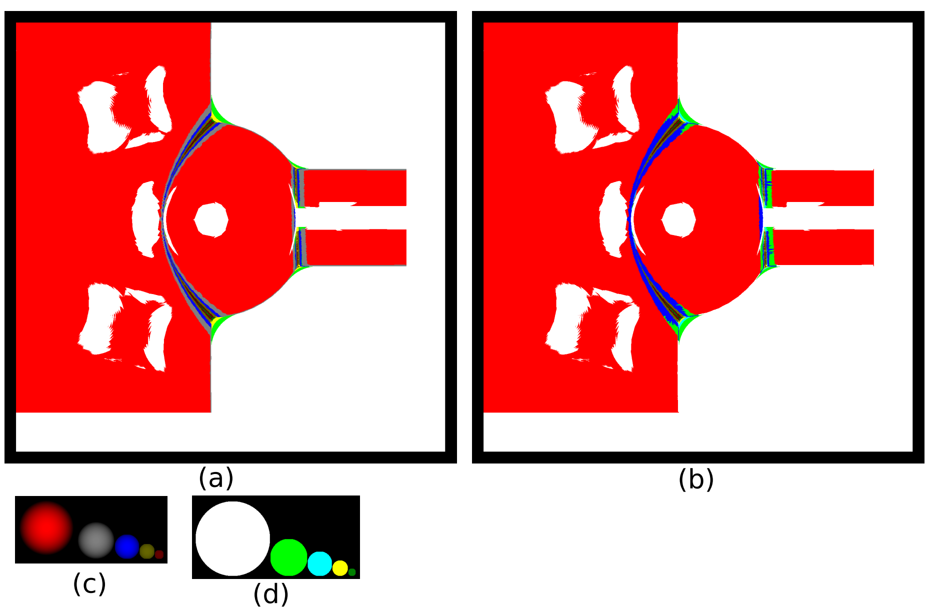

Figure 6.5: All the tools (c), (d) and finishing milling zones for two different tool sets (a), (b). All the tools except the tool T4 (cyan) and all the tools except the tool T3 (gray) are used for milling zones (a) and (b) respectively. The milling zones for the biggest tools T1 and T10 are the same for both tool sets (a) and (b). So, the tool paths for them need to be generated only once.

Therefore, the tool paths for different tools and different milling zones can be cached and the cached values can be used on request if they are calculated before.

The second optimization that can be applied to the current algorithm is saving bad tool sets. If removing one tool from the tool set leads to more machining time than it was with this tool, such a tool set is considered a bad one. That means that in the given tool set, the removed tool is very efficient, and for any subset of the bad configuration tool set, trying to remove the same tool does not leads to shorter machining time. From the Table 6.1 it can be seen that with every attempt to remove tool T10 from the set of available tools, the total machining time increases. Therefore, after the first try, tool T10 does not need to be checked again, it is very efficient and it should be in the optimal tool set.

The milling zones and generated tool paths for the optimal tool set (T1, T10, T8, T9) in the finishing milling layer is shown in Fig. 6.6.

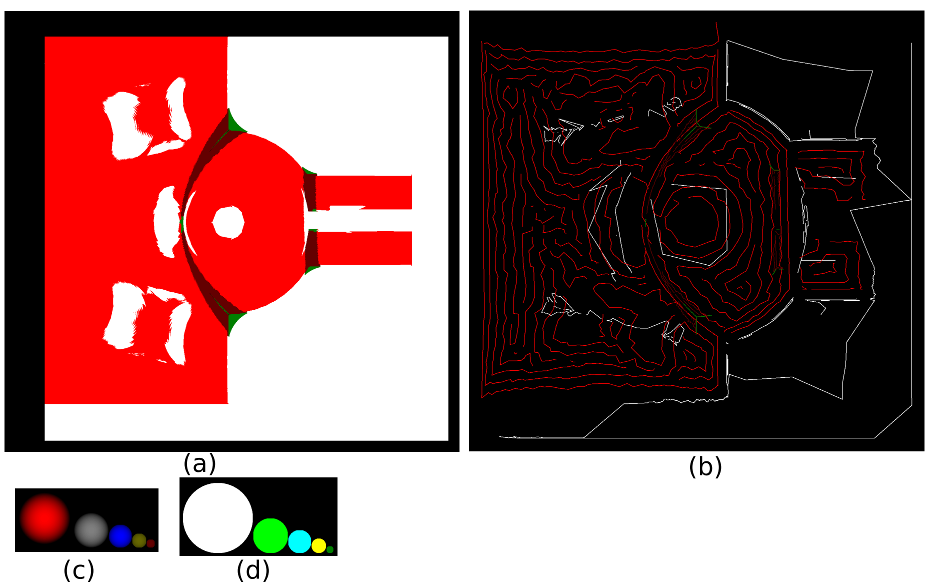

Figure 6.6: Milling zones (a) and generated tool paths (b) for the optimal tool set: T1 (white), T10 (red), T8 (dark red) and T9 (dark green).

After defining the optimal set of cutters, building tool paths for every cutter from the set, selecting the order of cutters to mill and connecting the tool paths for every cutter in a single tool path, G-code can be generated to mill a part.

To generate G-code, all the tool motions should be postprocessed in G-code. Also, the code for safe cutter changing should be generated for a used CNC machine. Becuse this G-code is machine specific and is trivail to generate, after having all the tool motions, the process of postprocessing is not shown here. In the next chapter, the simulaton results and the milling results are shown.

Chapter 7

Experimental results

In the previous chapters, the methodology to optimize the tool set and build tool paths for every tool from the set was introduced. This chapter presents experimental results of the methodology that is used to machine the part model shown in Fig. 7.1. The part is a representative free-form surface part to highlight the advantages and disadvantages of the methodology.

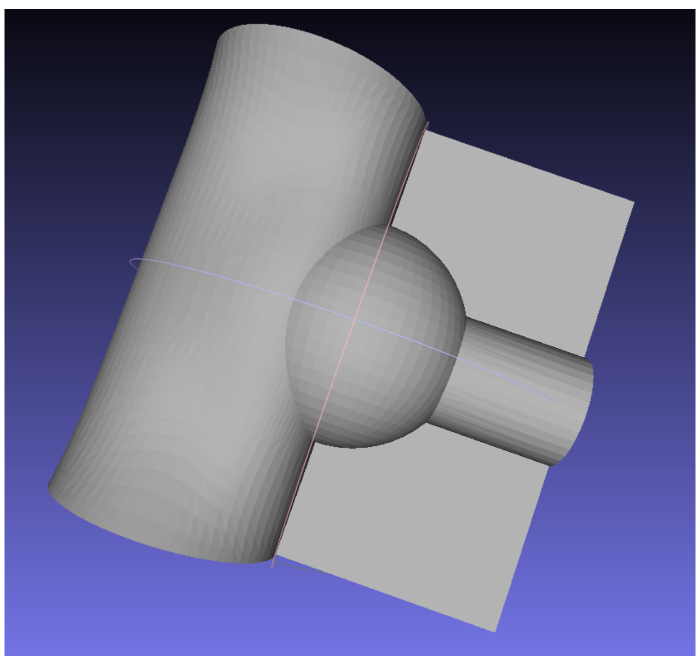

Figure 7.1: The part mesh for milling

The tool paths generated by the methodology are post-processed to translate the Euclidean CL positions to the G-code. Also the code was generated to start milling, change a tool and stop milling. In addition, the traditional tool path was generated using CAM software for the same part. Both G-codes were simulated, using open source software for 3-axis CNC milling simulating CAMotics [11].

7.1 Equipment

A HAAS VF-5/50 5-axis vertical machining center was used to machine the part shown in Figure 7.1, it is shown in Figure 7.2. After optimizing the tool set, four cutters (T1, T10, T8 and T9) were used for the simulation and for the milling process, while cutter T11 was used for the traditional tool path milling. All the cutters are shown in the Table 7.1. The material used to machine the parts is made of dense polyurethane foam called tooling board. It takes very little force to cut and yet retains surface accuracy. It is commonly used as a prototype material.

Figure 7.2: HAAS VF-5/50 CNC Machine

7.2 Simulated and milled results of the representative part

The simulated tool paths for the developed algorithm from this research and the traditional CAM approach are shown in Fig. 7.3. The traditional tool path was obtained using Inventor HSM 2016 software. The finishing tool path was build using

Table 7.1: List of cutters, used for milling

Tool number	Diameter, mm	Cutter Shape	Depth of cut, mm	Feed rate, mm/min
T1	9.525	flat end	16.67	5376
T10	7.983	ball end	13.90	4163
T8	1.191	ball end	2.08	450
T9	1.000	flat end	1.75	378
T11	6.35	flat end	11.36	2838

3D parallel strategy with the step over equals to 159 microns. In this strategy, the tool paths are parallel to x-axis and the depth is defined as the maximum depth without gouging.

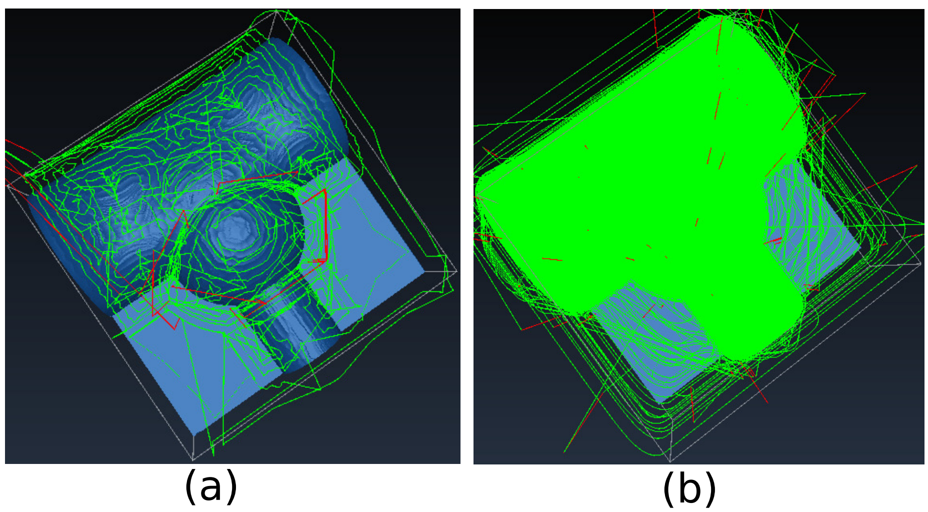

Figure 7.3: Simulation of the developed (a) and the traditional (b) tool paths for the representative part.

It can be seen that the traditional tool path is longer than the developed one, mainly because the same tool is used the whole milling surface. However, the surface quality is better for the traditional tool path, so the tool paths cannot be compared directly. The advantages and disadvantages of the developed method are discussed below. The simulated milled part for both approaches is shown in Fig. 7.4 and the milled parts for both tool paths are shown in Fig. 7.5. The discussion of the milled parts is provided in the next section.

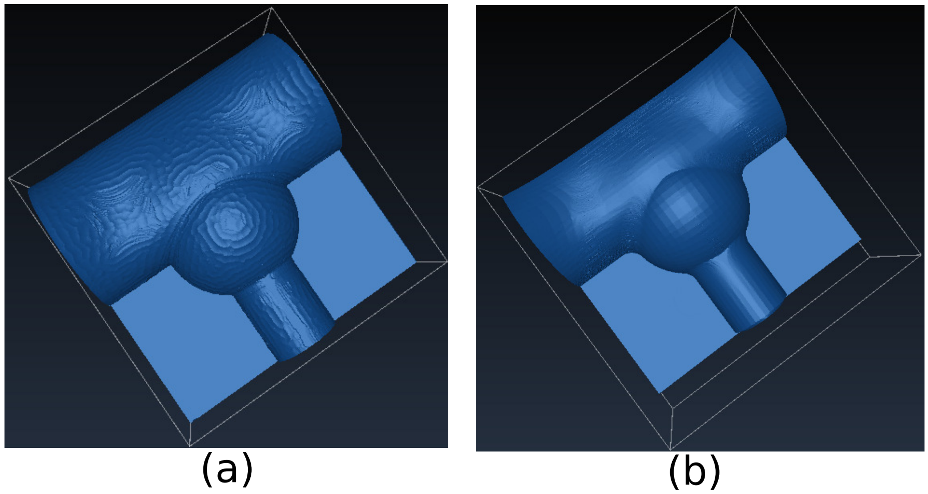

Figure 7.4: Milling simulation of the developed (a) and the traditional (b) tool paths for the representative part.

7.3 Simulated results of other parts

To show that the developed algorithms is robust and can be used with any parts, more parts were simulated. One of them is a sculptured part "Yoda" with some geometric errors in its polygon model, so traditional CAM/CAD software might have troubles to build tool paths for it. Another one "Tubes" does not have any free form surfaces, only horizontal ones. Even though CAM/CAD systems can easily build a tool path for this part, the developed approach has an advantage in using cutters of different size. The triangular meshes, and the simulated milled surfaces for these two parts are shown in Fig. 7.6.

The discretization step and the tolerance distance are considered the same for these parts as for the representative part and equal 40 microns and 120 microns respectively.

7.4 Discussion

First of all it can be seen, that the generated tool path can mill the given part, so the methodology from this research can be used to mill real parts. The total machining

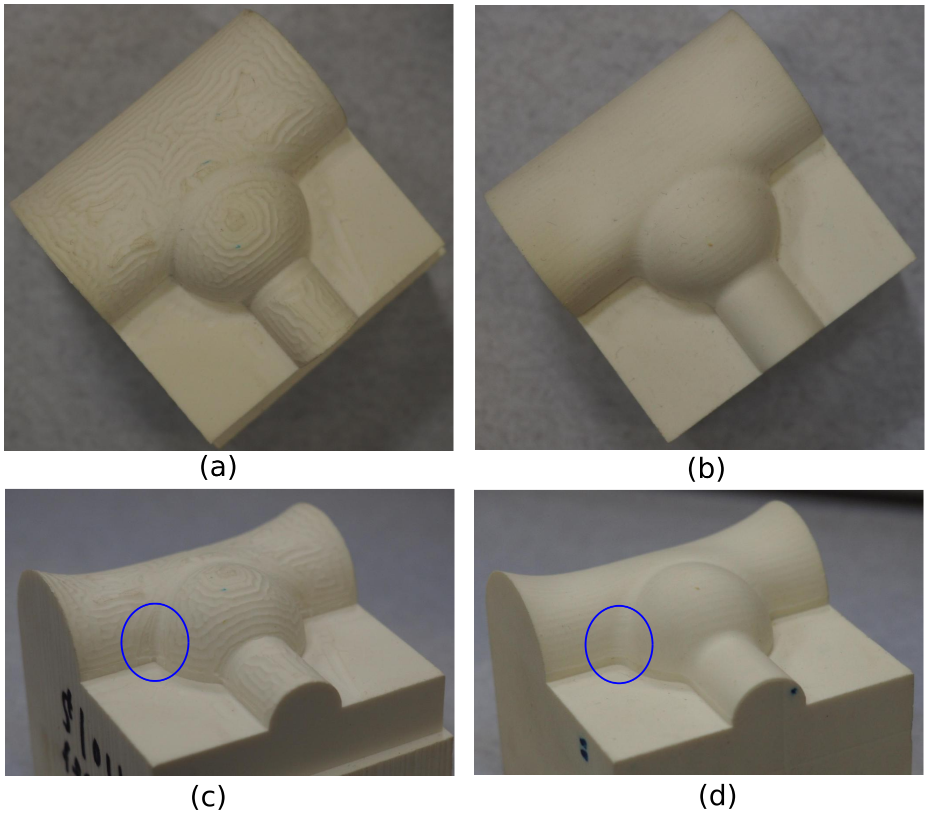

Figure 7.5: Milled representative part with using the developed (a, c) and the traditional (b, d) tool paths from different angle. The interesting features are highlighted with blue circles.

time for the developed tool path is 353 sec (6 min), with 193 sec of milling time and 160 sec to change four tools. The total machining time of the traditional tool path is about 18 min which is much slower. The traditional tool path is also longer.

It can be seen that the surface for the traditional tool path is smoother, than for the developed tool path. The main reason for that, it that the goal of the developed method is to maintain the same tolerance for all the milled surface. Based on the tolerance surface, different tools are selected for different milling zones on the milling surface. For the given part, it is guaranteed that the tolerance of the milled surface is maximum 217 microns, considering error. For the traditional approach the maximum tolerance is 1.315mm that is much higher. It can be seen in the highlighted regions in Fig. 7.5. The main reason of the bigger maximum tolerance for the traditional

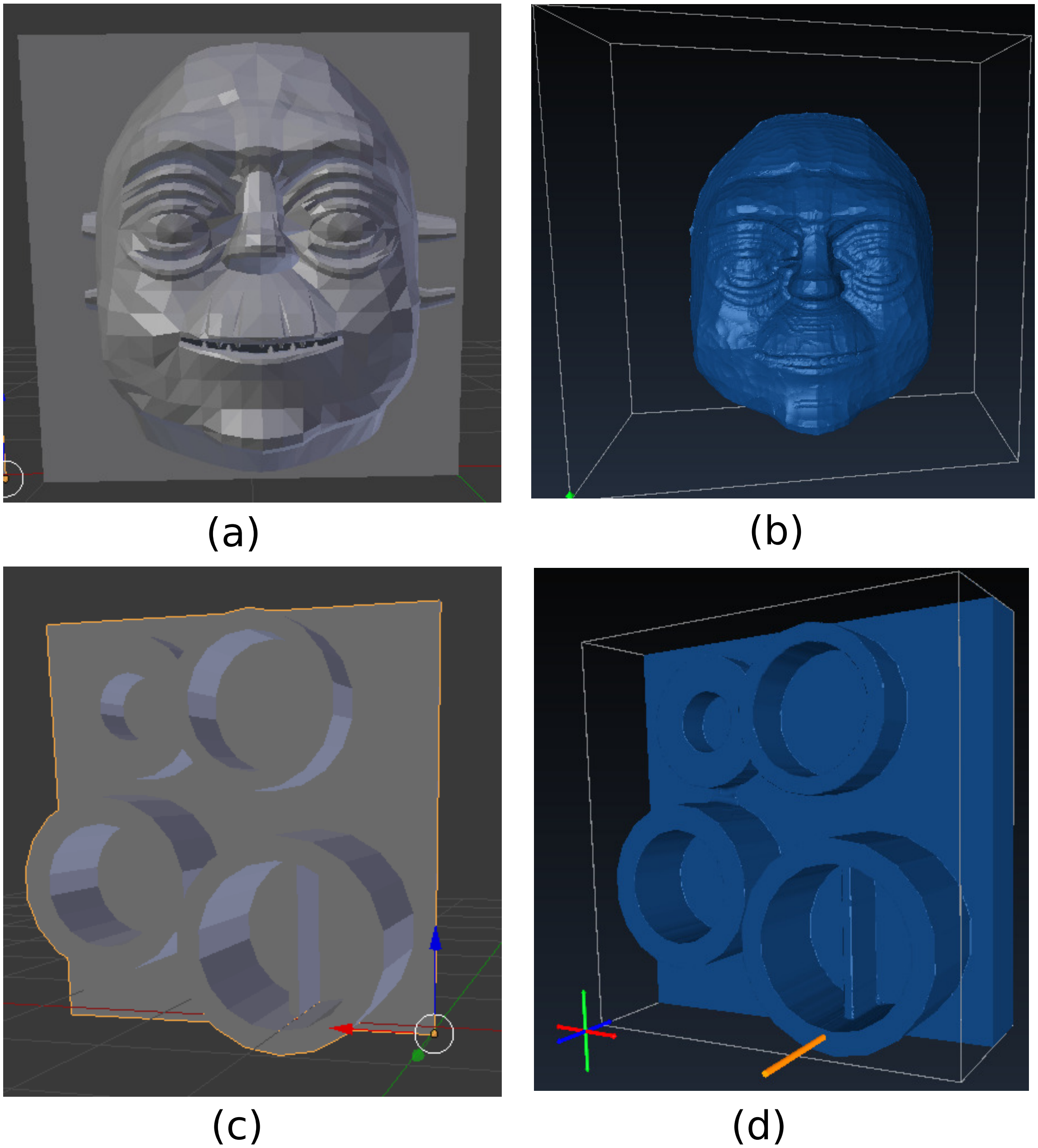

Figure 7.6: Triangular meshes(a, c) and milling simulation (b, d) for parts “Yoda” and “Tubes” respectively.

tool path is that the selected tool was too big to maintain smaller tolerance in the selected regions. If the smaller tool is selected, the milling time for the traditional approach will be even bigger. It is the usual trade-off between the surface quality and the machining time. The developed approach allows avoiding this trade-off, it selects the best tools to machine the selected part with the given tolerance for the minimum machining time.

Unfortunately, the developed approach also has some disadvantages, which are discussed in the next chapter.

Chapter 8

Conclusion and Discussion

In this research, the methodology to automatically select tools, generate milling zones and build tool paths for them in order to minimize the total machining time in free-form 3-axis CNC milling is presented. It was also shown that the total machining time for the generated tool path is less than the total machining time for the traditional tool path, generated with a CAM package with better maximum tolerance of the milled surface.

However, there are disadvantages of the developed method which needs to be discussed. The main disadvantage is that the proposed algorithm is resolution dependent. The smaller the discretization step (the bigger resolution) is, the less error in milling. However, the discretization step significantly affects the performance. To deal with this issue, several GPUs were used to accelerate the calculations. Some algorithms can be easily parallelized on more computation kernels from more GPUs, clusters, etc, but some cannot. The most time consuming algorithm is the generation of tool paths, which is done consecutively. To have a significant boost for this algorithm, tool path CL positions should be generated in parallel, so this algorithm needs to be revised. One way to do this is to generate CL positions independently, using other performance criteria.

Other algorithms can also be improved as was described in dedicated chapters, but the main bottle neck is the one of generating tool paths. The other bottle neck is the tool set optimization process, however, it can be accelerated by using more CPUs in calculating the machining time for different tool sets or caching some tool path building results. The third disadvantage of the developed approach is that the resolution can be limited with the maximum memory available on the computation device, such as GPU, for big parts and/or small resolutions.

The are also possible improvements and the future work that can be done. First, in the tool performance calculation process, the direction of the swipe can be considered to better estimate the tool performance in CL points. It is expensive, but because the performance for every CL position is calculated independently, it can be done in parallel.

The second possible improvement is that the machining time can be decreased even more by using feed rate scheduling, based on cutting forces. Fortunately, the models from this research can be easily used for cutting force calculation.

Finally, the same approach can be used for 5-axis milling. Efficient data structures should be found to store an arbitrary 3D surface in the memory instead of maps. Efficient algorithms should be used as well to reproduce the work from this research for 5-axis milling. The complexity is also increased significantly, because of having more degrees of freedom in tool motions. However, all the approaches from this research should work with the new surfaces. The tolerance surface can be built, all the cutter locations and rotations can be checked to find the best cutters for tolerance surface points and the milling zones and tool paths can also be built for the new tolerance surface for both roughing and finishing. The tool set optimization approach can also be performed to find the best tool set.

Bibliography

[1] Tomas Akenine-Möller. “Fast 3D triangle-box overlap testing”. In: *ACM SIGGRAPH 2005 Courses*. ACM. 2005, p. 8.

[2] Esther M Arkin, Martin Held, and Christopher L Smith. “Optimization problems related to zigzag pocket machining”. In: *Algorithmica* 26.2 (2000), pp. 197–236.

[3] James E Bobrow. “NC machine tool path generation from CSG part representations”. In: *Computer-aided design* 17.2 (1985), pp. 69–76.

[4] Gerardo Salas Bolaños, Sanjeev Bedi, and Stephen Mann. “A topological-free method for three-axis tool path planning for generalized radiused end milled cutting of a triangular mesh surface”. In: *The International Journal of Advanced Manufacturing Technology* 70.9-12 (2014), pp. 1813–1825.

[5] Richard J Campbell and Patrick J Flynn. “A survey of free-form object representation and recognition techniques”. In: *Computer Vision and Image Understanding* 81.2 (2001), pp. 166–210.

[6] Ahmet Can and Ali Ünüvar. “A novel iso-scallop tool-path generation for efficient five-axis machining of free-form surfaces”. In: *The International Journal of Advanced Manufacturing Technology* 51.9-12 (2010), pp. 1083–1098.

[7] Tao Chen and Zhiliang Shi. “A tool path generation strategy for three-axis ball-end milling of free-form surfaces”. In: *journal of materials processing technology* 208.1 (2008), pp. 259–263.

[8] James J Childs. *Numerical control part programming*. Industrial Press, Inc., 1973.

[9] Chuang-Jang Chiou and Yuan-Shin Lee. “A machining potential field approach to tool path generation for multi-axis sculptured surface machining”. In: *Computer-Aided Design* 34.5 (2002), pp. 357–371.

[10] Young-Keun Choi and A Banerjee. “Tool path generation and tolerance analysis for free-form surfaces”. In: *International Journal of machine Tools and manufacture* 47.3 (2007), pp. 689–696.

[11] Joseph Coffland. *CAMotics is an Open-Source software which simulates 3-axis CNC milling or engraving*. [Online; accessed 3-October-2016]. 2016. URL: `http://camotics.org`.

[12] Cyril Crassin and Simon Green. “Octree-based sparse voxelization using the GPU hardware rasterizer”. In: *OpenGL Insights* (2012), pp. 303–318.

[13] XM Ding et al. “Optimal cutter selection for complex three-axis NC mould machining”. In: *International journal of production research* 42.22 (2004), pp. 4785–4801.

[14] Ravinder Kumar Duvedi et al. “A multipoint method for 5-axis machining of triangulated surface models”. In: *Computer-Aided Design* 52 (2014), pp. 17–26.

[15] Gershon Elber and Elaine Cohen. “Tool path generation for freeform surface models”. In: *Proceedings on the second ACM symposium on Solid modeling and applications*. ACM. 1993, pp. 419–428.

[16] S Engin and Y Altintas. “Mechanics and dynamics of general milling cutters.: Part I: helical end mills”. In: *International Journal of Machine Tools and Manufacture* 41.15 (2001), pp. 2195–2212.

[17] Wen-Feng Gan et al. “Five-axis tool path generation in CNC machining of T-spline surfaces”. In: *Computer-Aided Design* 52 (2014), pp. 51–63.

[18] FY Han et al. “Optimal CNC plunge cutter selection and tool path generation for multi-axis roughing free-form surface impeller channel”. In: *The International Journal of Advanced Manufacturing Technology* 71.9-12 (2014), pp. 1801–1810.

[19] Martin Held, Gábor Lukács, and László Andor. “Pocket machining based on contour-parallel tool paths generated by means of proximity maps”. In: *Computer-Aided Design* 26.3 (1994), pp. 189–203.

[20] Bo H Kim and Byoung K Choi. “Machining efficiency comparison direction-parallel tool path with contour-parallel tool path”. In: *Computer-Aided Design* 34.2 (2002), pp. 89–95.

[21] Hyun-Chul Kim, Sung-Gun Lee, and Min-Yang Yang. “An optimized contour parallel tool path for 2D milling with flat endmill”. In: *The International Journal of Advanced Manufacturing Technology* 31.5-6 (2006), pp. 567–573.

[22] Su-Jin Kim and Min-Yang Yang. "A CL surface deformation approach for constant scallop height tool path generation from triangular mesh". In: *The International Journal of Advanced Manufacturing Technology* 28.3-4 (2006), pp. 314–320.

[23] Guillermo H Kumazawa, Hsi-Yung Feng, and M Javad Barakchi Fard. "Preferred feed direction field: A new tool path generation method for efficient sculptured surface machining". In: *Computer-Aided Design* 67 (2015), pp. 1–12.

[24] Kunwoo Lee, Tae Ju Kim, and Sung Eui Hong. "Generation of toolpath with selection of proper tools for rough cutting process". In: *Computer-Aided Design* 26.11 (1994), pp. 822–831.

[25] Sung-Gun Lee, Hyun-Chul Kim, and Min-Yang Yang. "Mesh-based tool path generation for constant scallop-height machining". In: *The International Journal of Advanced Manufacturing Technology* 37.1-2 (2008), pp. 15–22.

[26] Yuan-Shin Lee, BK Choi, and TC Chang. "Cut distribution and cutter selection for sculptured surface cavity machining". In: *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH* 30.6 (1992), pp. 1447–1470.

[27] Wei Liu, Lai-shui Zhou, and Lu-ling An. "Constant scallop-height tool path generation for three-axis discrete data points machining". In: *The International Journal of Advanced Manufacturing Technology* 63.1-4 (2012), pp. 137–146.

[28] Xu Liu et al. "A tool path generation method for freeform surface machining by introducing the tensor property of machining strip width". In: *Computer-Aided Design* 66 (2015), pp. 1–13.

[29] S Marshall and John G Griffiths. "A new cutter-path topology for milling machines". In: *Computer-Aided Design* 26.3 (1994), pp. 204–214.

[30] Mario Mejia-Ugalde et al. "Directional morphological approaches from image processing applied to automatic tool selection in computer numerical control milling machine". In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* (2013), p. 0954405413491402.

[31] Tawfik T El-Midany, Ahmed Elkeran, and Hamdy Tawfik. "Toolpath pattern comparison: Contour-parallel with direction-parallel". In: *Geometric Modeling and Imaging–New Trends, 2006*. IEEE. 1993, pp. 77–82.

[32] TE Mwinuka and MI Mgwatu. "Tool selection for rough and finish CNC milling operations based on tool-path generation and machining optimisation". In: *Advances in Production Engineering & Management* 10.1 (2015), p. 18.

[33] Giuseppe Papari and Nicolai Petkov. "Edge and line oriented contour detection: State of the art". In: *Image and Vision Computing* 29.2 (2011), pp. 79–103.

[34] Sang C Park. "Sculptured surface machining using triangular mesh slicing". In: *Computer-Aided Design* 36.3 (2004), pp. 279–288.

[35] Kandarp Patel et al. "Optimal tool shape selection based on surface geometry for three-axis CNC machining". In: *The International Journal of Advanced Manufacturing Technology* 57.5-8 (2011), pp. 655–670.

[36] Gerhard Reinelt. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, 1994.

[37] Yongfu Ren, Hong Tzong Yau, and Yuan-Shin Lee. "Clean-up tool path generation by contraction tool method for machining complex polyhedral models". In: *Computers in Industry* 54.1 (2004), pp. 17–33.

[38] Subhajit Sarkar and Partha Pratim Dey. "Tool path generation for algebraically parameterized surface". In: *Journal of Intelligent Manufacturing* 26.2 (2015), pp. 415–421.

[39] Sanjay E Sarma. "The crossing function and its application to zig-zag tool paths". In: *Computer-Aided Design* 31.14 (1999), pp. 881–890.

[40] Joshua Tarbutton et al. "Gouge-free voxel-based machining for parallel processors". In: *The International Journal of Advanced Manufacturing Technology* 69.9-12 (2013), pp. 1941–1953.

[41] Christophe Tournier and Emmanuel Duc. "Iso-scallop tool path generation in 5-axis milling". In: *The International Journal of Advanced Manufacturing Technology* 25.9-10 (2005), pp. 867–875.

[42] Yu Wang et al. "A computer aided tool selection system for 3D die/mould-cavity NC machining using both a heuristic and analytical approach". In: *International Journal of Computer Integrated Manufacturing* 18.8 (2005), pp. 686–701.

[43] William E Wright. "Parallelization of Bresenham's line and circle algorithms". In: *IEEE Computer Graphics and Applications* 10.5 (1990), pp. 60–67.

[44] Daniel CH Yang and Zhonglin Han. "Interference detection and optimal tool selection in 3-axis NC machining of free-form surfaces". In: *Computer-Aided Design* 31.5 (1999), pp. 303–315.

[45] H-T Yau, C-M Chuang, and Y-S Lee. "Numerical control machining of triangulated sculptured surfaces in a stereo lithography format with a generalized

cutter". In: *International journal of production research* 42.13 (2004), pp. 2573–2598.

[46] Sun Yuwen, Guo Dongming, Wang Haixia, et al. "Iso-parametric tool path generation from triangular meshes for free-form surface machining". In: *The International Journal of Advanced Manufacturing Technology* 28.7-8 (2006), pp. 721–726.

[47] Bo Zhou, Jibin Zhao, and Lun Li. "CNC double spiral tool-path generation based on parametric surface mapping". In: *Computer-Aided Design* 67 (2015), pp. 87–106.

[48] Min Zhou, Guolei Zheng, and Zezhong Chevy Chen. "An automated CNC programming approach to machining pocket with complex islands and boundaries by using multiple cutters in hybrid tool path patterns". In: *The International Journal of Advanced Manufacturing Technology* 83.1-4 (2016), pp. 407–420.