BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
FACULTY OF MECHANICAL ENGINEERING
DEPARTMENT OF MANUFACTURING SCIENCE AND ENGINEERING

PhD Dissertation

MATERIAL REMOVAL SIMULATION AND CUTTING FORCE PREDICTION OF MULTI-AXIS MACHINING PROCESSES ON GENERAL-PURPOSE GRAPHICS PROCESSING UNITS

Author:

BALÁZS TUKORA

Supervisor:

TIBOR SZALAY, PhD

Budapest, 2012

Table of contents

Acknowledgements

I would like to thank my supervisor, Dr. Tibor Szalay for his guidance during my doctoral studies; Dr. Gyula Mátyási for his assistance in performing the required experiments; and the collective of the Department of Manufacturing Sciences and Technology for receiving me into the family of the mechanical engineers.

I express my thanks to my employer, the University of Pécs, for the financial support that enabled me to take part in high-level conferences and purchase essential equipments for the research work.

And last but not least I would like to thank my family for their unflagging love and support.

List of abbreviations and symbols

Abbreviations

API	Application programming interface
BRep	Boundary representation
CAPP	Computer-aided process planning
CAD	Computer-aided design
CAM	Computer-aided manufacturing
CC-data	Cutter contact data
CG	Computer graphics
CL-data	Cutter location data
CPU	Central processing unit
CSG	Constructive solid geometry
CUDA	Compute unified device architecture
EDM	electrical discharge machining
FB	frame buffer
FE	Finite element
FPS	frames per second
GFLOPS	Billion floating point operations per second
GPGPU	General-purpose graphics processing unit
GPU	Graphics processing unit
GUI	Graphical user interface
HSD	Hierarchical space decomposition
LDNI	Layered depth-normal image
MC	Marching cubes algorithm
MRR	Material removal rate
NURBS	Non-uniform rational B-spline
PLM	Product lifetime management
ROP	Raster operation
RPM	Revolutions per second
SIMD	Single instructions, multiple data
SISD	Single instruction, single data
SP	Stream processor
STL	File format for stereolitography
TF	Texture addressing and filtering unit
USD	Uniform space decomposition

Symbols

F_t , F_a , F_r	[N]	Tangential, axial and radial force components
F_x , F_y , F_z	[N]	Force components related to the tool coordinate system
K_{te}, K_{re}, K_{ae}	[N/mm]	Ploughing (edge) specific cutting force coefficients
K_{tc}, K_{rc}, K_{ac}	[N/mm^2]	Shear (cut) specific cutting force coefficients
dS	[mm]	Length of the elemental edge
t_n	[mm]	Undeformed chip thickness
db	[mm]	Chip width
θ	[rad]	Spindle rotation angle
Ψ, κ	[rad]	Horizontal and vertical positioning angles of the edge segment
f_z	[mm]	Feed per tooth
δ		Dexel grid interval
A	[mm^2]	Surface area
i_0	[rad]	Nominal helix angle
R_0	[mm]	Tool radius
r	[mm]	Local tool radius
N_f		Number of cutter teeth
P		Number of slices
Λ, Γ, Ω	[mm],[mm^2]	Scalars carrying geometrical information about the contact area
$\mathbf{G}$		Augmented matrix for the Gaussian elimination
$\mathbf{F}_{xyz,k}$	[N]	Calculated cutting force vector in the *k*th simulation step
$\mathbf{F}_{xyz,k\,m}$	[N]	Measured cutting force vector in the *k*th simulation step
S	[rpm]	Spindle speed
v_c	[m/min]	Cutting speed
f	[mm/min]	Feed rate
a_e	[mm]	Radial depth of cut
a_p	[mm]	Axial depth of cut

1 Introduction

The efficient planning of automated machining processes is unthinkable without the use of offline CAM systems. Though machining programs can be written and input manually, right at the machine controller, if the workpiece geometry is complex, or if the machined features are numerous, the help of CAM software is essential for generating the program both accurately and quickly. This is particularly true in the case of multi-axis machining, when complex, often sculpture-like shapes are machined in a single set-up. The determination of the optimal toolpath in accordance with the CAD part designs requires sophisticated computerized algorithms. On the other hand, it is indispensable to allow the user to verify and – if it's needed – manually modify the computer-generated operations in the course of the interactive planning process. The most evident mode of the verification is the ocular inspection of the planned machining operations on the screen of the CAM software. The visual representation of the machining processes has several levels, from the simple linear drawing of the toolpath until the detailed displaying of the altering workpiece geometry during the material removal process. While the first helps shifting out the rough errors of the automated toolpath generation, the latter can bring the unwanted geometrical errors of the machined part to light. The first part of the dissertation deals with this higher level of machining process representation.

The simulation of the material removal process presents a special challenge for the CAM software developers. To get the final shape of the workpiece the simulation of the whole material removal process has to be fulfilled. Considering that the resolution of the displayed objects must satisfy the requirements of the visual verification, a huge amount of data, which carries enough geometrical information for the detailed description, has to be processed within a short time. In the course of time several simulation methods have come to life to perform this task, with various levels of performance. One of the oldest, and still the most effective solution up to now exploited that the operations of the material removal can be performed in a highly parallelized thus effective manner by the use of graphics processing units (GPU). Unfortunately this method proved to be viable only in the simple geometrical circumstances of 3-axis machining. The complex geometry of a workpiece that is shaped during multi-axis machining cannot be represented with data structures used by the traditional GPUs, and the sequential execution by the CPU cannot even approach the performance of the GPU based solution.

The appearance of general-purpose graphics processing units (GPGPUs) has given a new tool for executing non-graphical (i.e. general-purpose), highly parallelized computations in a common personal computer environment. The primary aim of the doctoral work was the creation of a material removal simulation method for multi-axis machining, which is fully supported by this new generation of graphics hardware, in order to reach the performance of the 3-axis methods supported by the traditional GPUs. This means not only running a traditional algorithm on a modern device, but reshaping the data representation to satisfy the requirements of the unified hardware architecture of the

GPGPUs; developing a highly parallelized manner of executing cutting operations; and defining the appropriate way of object-reconstruction for visualization. The result is a high-resolution, real-time simulation, where millions of geometrical data elements take part in the computations, making the fast and detailed visualization of multi-axis material removal processes available.

Besides the simple visualization, there is another, more and more important reason of integrating material removal simulation modules in the CAM systems. The newest versions of the most widely applied CAM programs offer automated feed rate optimization, to reduce machining time when the use of constant machining parameters of technology handbooks is not practical owing to the big alternation of the cutting geometry during the machining process. In general this characterizes the multi-axis machining. The basis of the optimization is the prediction of the expectable cutting forces, for which the material removal simulation provides the required geometry information about the process.

Though the advertisements of CAM systems offer significant improvement of machining time thanks to the feed rate optimization functions, their usage hides risks as well: the fast but rough force predicting methods cannot guarantee the keeping of machining parameters recommended by the tool manufacturers. This can lead to increased tool wear, and surface quality deterioration. For that reason we have to investigate if we could find a more adequate cutting force prediction model for this type of optimization.

The *mechanistic cutting force model*, though it provides much more sophisticated results than the recently used, is not applied in CAM systems. The main reason is that the determination of the cutting force coefficients figuring in the force equations requires circuitous experiments, and their validity is limited. In the second part of the dissertation it will be examined, whether the equations of the mechanistic cutting force model could be reconstructed to get a more applicable model in practice. A new method will be introduced to efficiently solve the equations in the GPGPU environment, together with an algorithm to determine the cutting force coefficients without such geometrical restrictions, which characterized the former solutions. However, the applicability of the proposed methods raises more questions: is there enough information about the cutting process in the offline state to accurately predict the forthcoming forces, or should we do it online right at the machine? What are the conditions of doing so? The main issues of the realization of such a system will also be discussed.

2 Literature and technology review

2.1 Virtual machining

The modern computer aided manufacturing (CAM) software packages provide two kinds of machining simulation with different purposes. The first focuses on the machining process: it visualizes the motion of the machine tool parts, displays the movement of the cutter to help verifying the generated toolpath, and performs collision detection among the moving and non-moving objects in the workspace. The shape of the objects in the virtual space is static; the altering of the workpiece and the cutting process is not visualized, instead the *kinematic behavior* of the machine tool is simulated by displaying the result of successive spatial transformations on rigid objects that are created with CAD modeling techniques (Figure 1), while the workpiece is displayed in its final, machined shape.

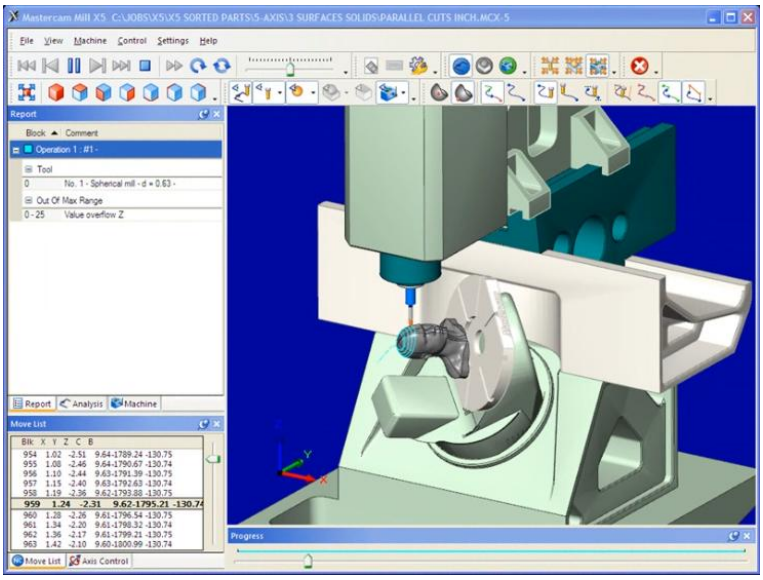

Figure 1. Kinematic simulation [1]

The second type of virtual machining provided by the CAM software is to simulate the material removal process (Figure 2). On the one hand it shows the altering shape of the blank while the cutter moves along the toolpath for the purpose of surface quality and shape verification. On the other hand it gives information about the continuously changing cutting geometry in the course of the cutting process, which is required for cutting force prediction and feed rate optimization. The basis of the simulation is the *solid object modeling* of the workpiece and the cutter in the virtual space. The cutter penetrates into the workpiece and removes material from it – this is represented by the Boolean subtraction of the cutter volume from the volume of the workpiece. The subtraction of the volume swept by the tool along the whole toolpath from the volume of the blank results the final shape of the workpiece at the end of the material removal process.

In the followings the material removal simulation techniques are enumerated, as this technology forms the background of the research work.

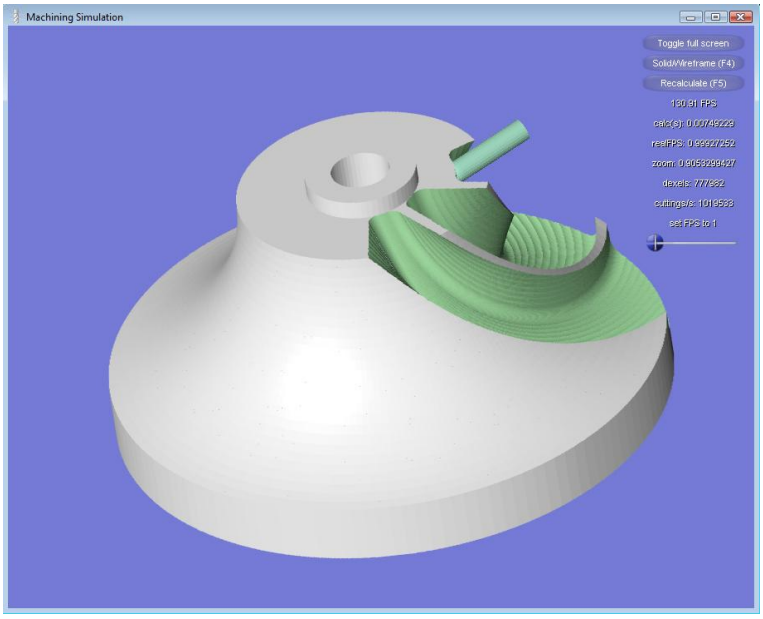

Figure 2. Material removal simulation

2.1.1 Vector clipping

The *vector clipping method* is to display the discrepancies between the desired shape of the workpiece and the shape that is effectively formed by the cutter that moves along the toolpath. In this way the correctness of the toolpath, which has been generated by the CAM software or drawn up by the user, can be verified.

The method, proposed by Chappel [2], starts from the surface representation of the finished workpiece, which is usually the result of the preceding computer-aided design and forms the source of the computer-aided manufacturing process. The surface is converted into a point cloud and the local normal vector of the surface is assigned to each point. The tool that is moving over the surface snips these vectors, just like a lawn mower cuts the blades of grass; the vectors are getting shorter as the tool is coming closer to the surface in the course of the cutting process (Figure 3). The final vectors show the remaining surface errors after the machining: vectors with positive length mark excess material, while vectors with negative length mean undercut at the given area. The errors are displayed by the coloring of the surface model.

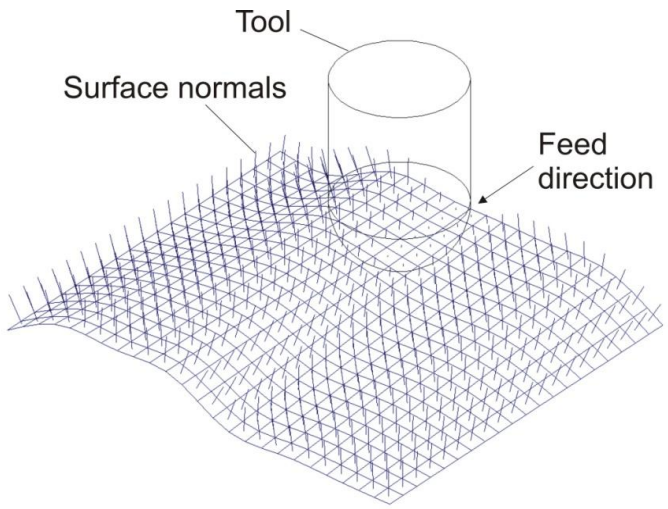

Figure 3. Vector clipping (on the basis of [3])

The vector clipping method is suitable for handling the machining of sculptured surfaces, but accuracy problems can occur when the directions of the normals change notably in a small surface area – at sharp edges, for example. Other disadvantages are the big memory demand and low computing speed. In addition it does not directly generate a solid model of the machined part, thus it cannot be called a fully efficient material removal simulation technique. Instead, CAM programs use a similar solution, developed by Oliver and Goodman [4], for surface verification. In this case the directions of the surface vectors are chosen to be the same in the whole inspected area. The advantage of this is that the intersection calculations are much faster on parallel vectors, as they can be efficiently executed by common computer graphics (CG) algorithms, such as the z-buffer method described in Section 2.1.2. However, parallel vectors can be assigned only to the surface points that can be seen from a given point of view. This prevents the inspection of hidden surface areas and all sides of the workpiece object in 5-axis machining – that's why the method is preferred in 3-axis machining simulation. Figure 4 shows the 3-axis surface verification module of VisualCAM by MecSoft, where blue areas show exceed material, and red areas indicate undercutting. Meanwhile Oliver and his colleagues [5][6] made efforts to improve the performance of their original method, while Jerard, Drysdale et al. [7][8] used an approach that shares the characteristics of the methods of Chappel and Oliver.

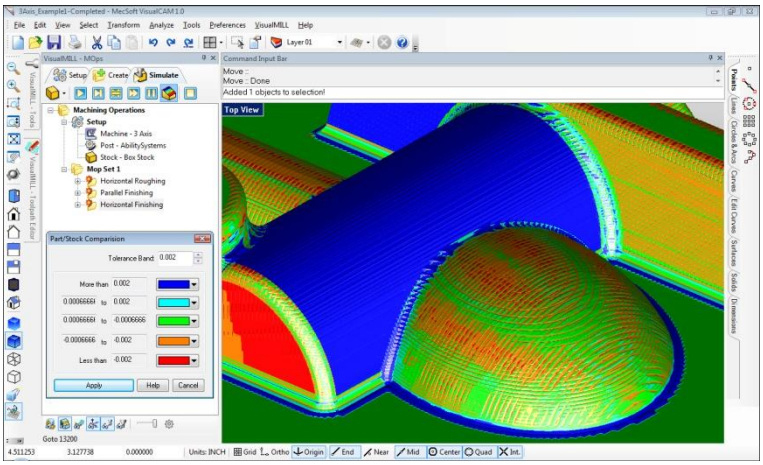

Figure 4. Colored surface model in VisualCAM

2.1.2 Image space based virtual machining

The most widespread 3-axis material removal simulation technique is the *z-map method*, first introduced by Anderson [9]. The blank is split into right prisms of equal cross-section along a plane perpendicular to the direction of the machining (Figure 5). The heights of the prisms are stored in a 2-dimensional data set to describe the top surface of the object. The degree of resolution is determined by the size of the cross-sections; Wang et al. [10] suggested that it should be equal to the resolution of the screen, as in such case the height values can be stored in a *depth image*, which is a common image format of the computer graphics (Figure 6). (A depth image does not contain color information but only one value per pixel that marks the distance of the screen-point from the camera plain. These images are hold in *z-buffers* in the memory of the graphics processing unit.) The swept volume of the cutter is also represented by a depth image with the same orientation and position as the former, where the depth values represent the minimum position of the cutter envelop at a given pixel up to the actual simulation step. Thereupon the volume cut from the blank by the cutter can be determined by the comparison of the related pixels of the two depth images: where the depth value regarded to the cutter is less than the depth value regarded to the blank, material is being removed; the pixels of the depth image of the blank must be updated to hold the new, smaller depth values. In formal description [11]:

$$F(x,y) := \min\left\{s(x,y), \min\left\{z \mid z := f(x-x', y-y', z'), (x',y',z') \in M\right\}\right\}, \tag{1}$$

where function *F(x,y)* represent the machined surface, *s(x,y)* represents the surface of the blank, $M \in \mathbb{R}^3$ is the set of toolpath points up to the actual simulation step, and the cutter envelop points are given by *f(x,y,z)*, e.g. for ball-end mill:

$$f(x,y,z) := z - \sqrt{R^2 - x^2 - y^2}, (x,y) \in D(f) := \left\{(x,y) \mid x^2 + y^2 \le R^2\right\} \tag{2}$$

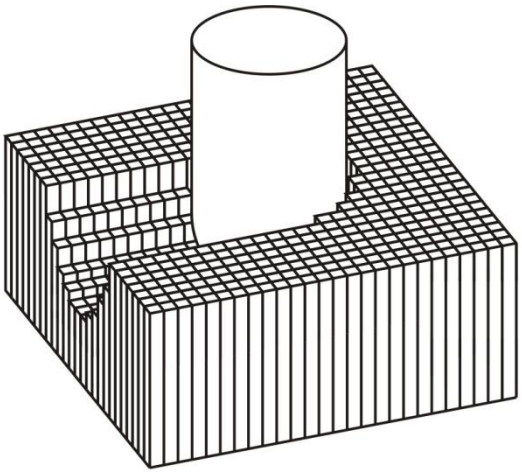

Figure 5. Z-map workpiece representation

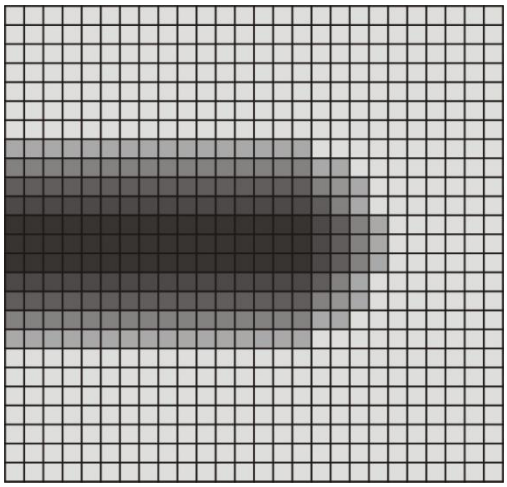

Figure 6. Depth image

As the operations on depth images are greatly supported by the graphics hardware, the z-map method is very fast and easily implementable. Numerous adaptations, which followed the development of the hardware technology, have made it to be the leader solution in 3-axis virtual machining [12]. On the other hand, the method can be applied only in such cases, when the direction of the machining is unchanging, as a single depth image cannot hold information about covered or back-faced surfaces, which characterize the sculptured surfaces. Several solutions came to life to this problem. Atherton et al. [13] stored the points of both the front and back faces of the cutter swept volume to perform intersection calculations with the workpiece surface. Hook [14] introduced the concept of *Dexels*. The name of dexels came from the abbreviation of *depth elements* that describe the whole cross-section of an object along a given line. A grid is laid onto a spatial plane, and a line is launched from each grid point perpendicular to the plane (Figure 7). Dexels are determined by the intersections of the lines and the surface of the object: where the line penetrates into the workpiece volume (at the front face) a dexel begins, and where the line leaves it (at the back face) the dexel ends. Dexels are described by the

coordinates of the entering and leaving, but other attributes can be attached to them too, like the normal vectors of the surface at the entering and leaving points, or several material properties. If the line intersects the workpiece more than once, a number of dexels are created, which are stringed onto a chain in the memory from the front up to the rearmost face.

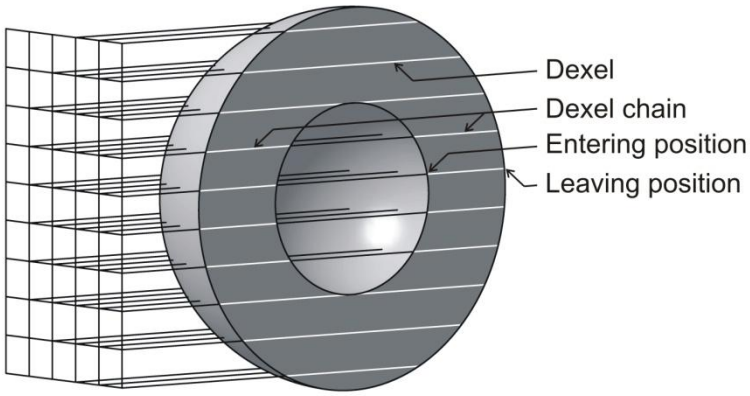

Figure 7. Dexel representation

The *multi-dexel* or *tri-dexel method* was proposed by Benouamer et al. [15] to improve the sampling quality of the dexel model at surface regions whose normal vectors are perpendicular or nearly perpendicular to the dexel lines. For that reason separate dexel sets are created in three orthogonal directions, along the three coordinate axes (Figure 8). The volume is entirely described by each of the three dexel sets, so three complete dexel descriptions are created. With this procedure significant increasing of sampling resolution can be reached with little growth of memory consumption [16].

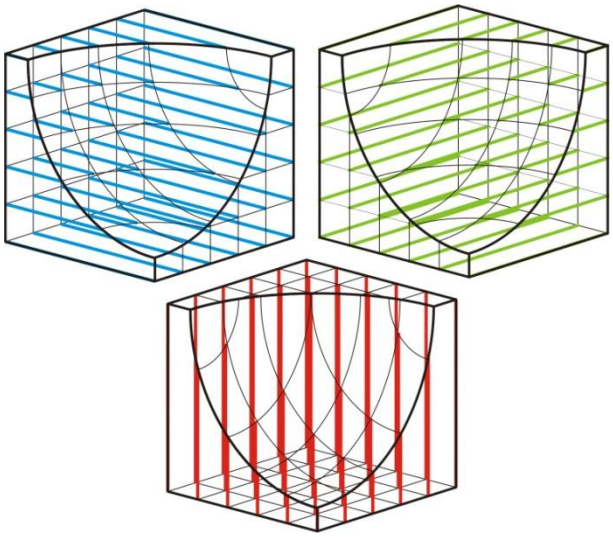

Figure 8. Multi-dexel object representation

2.1.3 Object space based virtual machining

Gossard et al. [17] started investigating material removal processes on the basis of theory of sets in the late seventies. Voelckler and Hunt [18] used the tools of solid modeling for virtual machining first; they applied the PADL *constructive solid geometry* (CSG) system to simulate cutting processes. When CSG is used for object modeling, the volume of the modeled object is the result of Boolean operations on solid primitives, such as cubes, spheres etc. (Figure 9). In the case of cutting simulations this means several Boolean subtractions while the tool moves along the toolpath. The main limitation of the CSG based virtual machining is the high computational expense. Though the data representation of the resulted volumes is simple and compact, the visualization of them requires numberless surface-surface intersection calculations. The cost of simulation is $O(n^4)$, where *n* is the number of tool movements; that's why we can find this method only in a few research project reports [19].

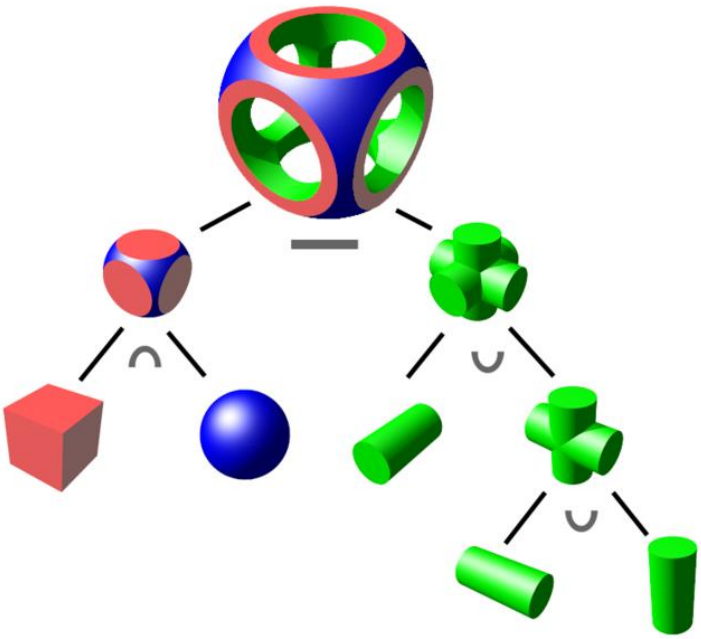

Figure 9. Object representation with CSG [20]

Spatial decomposition is another way of representing solid objects. The spatial decomposition can be uniform or hierarchical. *Uniform space decomposition* (USD) uses a set of identical cells, usually having the form of a cube of the same size, for representing objects (Figure 10.b) [21]. The cells are named *voxels* from the abbreviation of "volume elements". The *universe*, the space region that contains all objects of interest, is decomposed into empty and full cells. The object geometry is defined by set of the full voxels. In the case of USD the Boolean operations can be very fast and easily executed. However, the increasing of the model accuracy requires the reducing of the voxel size, which leads to large memory requirement to store the model. Voxels can be efficiently used for

representing objects with inhomogeneous components, such as geological formations or living organisms; the employment of the USD method is common and successful in these fields.

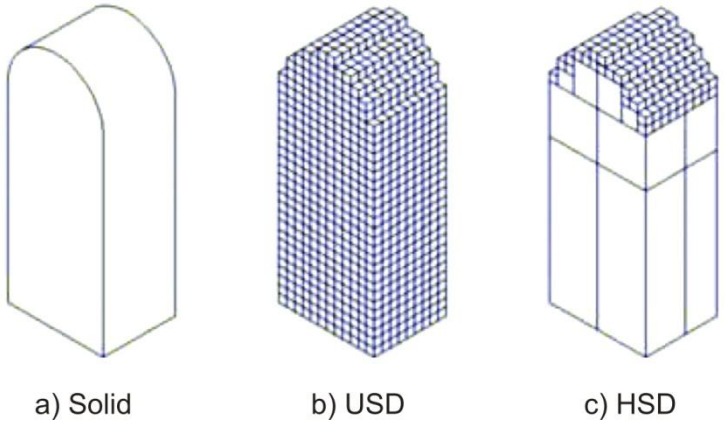

Figure 10. Spatial decomposition [22]

The *hierarchical space decomposition* (HSD) model reduces the memory requirements of USD considerably by dividing the space adaptively (Figure 10.c). *Octree* is a typical HSD method, where the universe is subdivided into eight parts recursively. Each part forms a cube that is one eighth of its parent cube in size. The cubes, which are named *octant*s, are represented as the nodes of a tree, in which every node has eight branches (Figure 11). This tree is called octree [23]. The obtainable resolution of octree is limited by the fact that the required memory storage increases exponentially as the tree-depth increases [24]. Kawashima et al. [25] combined the CSG and Octree methods by putting CSG primitives into the octree nodes, thus creating a so-called *Graftree*, for increasing the displaying quality at the same tree-depth. Though several octree based approaches have been reported in the literature [26][27][28], not any commercial HSD based CAM system is available, due to the spreading of simpler and more effective techniques.

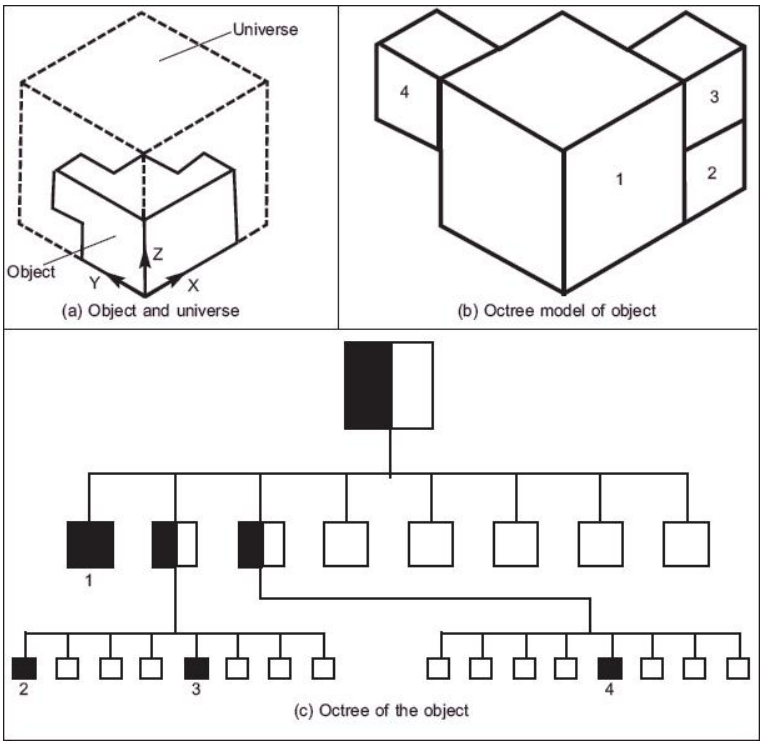

Figure 11. Octree representation [23]

Boundary representation (BRep) is the dominant modeling technique in CAD systems, and is also used for surface representation in CAM applications. The solid is defined analytically, by the topology and geometry of its surface; the main topological items are faces (bounded portion of a surface), edges (bounded piece of a curve) and vertices (spatial points). For virtual machining its simplified version, the triangle based boundary representation is also applied [29], where the surface is represented by a collection of connected, non-overlapping triangles, without topological information (Figure 12). The computational growth rate of BRep is $O(n^{1.5})$ for a total of n tool movements, which can cause problems at long tool paths. Fleisig and Spence [30] made efforts to improve the efficiency of the method with topology and algorithm optimization. The ACIS BRep based solid modeling kernel provided by Spatial Corp. is not only the core technology of CAM systems (e.g. Mastercam or the CAM modules of CATIA), but its academical version is preferred at research projects [31][32] where the determination of chip geometry constitutes the basis of cutting force computations.

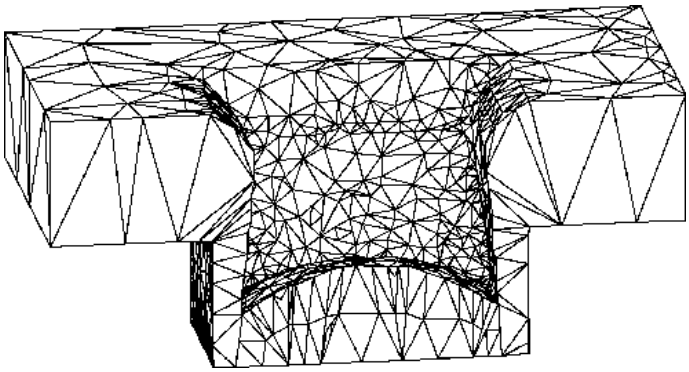

Figure 12. Boundary representation [33]

2.1.4 Data conversion steps in virtual machining

CAM applications usually import objects in their native, BRep based format from the CAD stage. This form of representation is used for tool path planning and kinematic simulation among others. As we could see, the object description formats applied in the material removal simulation differ from this, so a preliminary data conversion is needed before the simulation process begins. On the other hand, the graphics hardware can display only polyhedral surfaces, which means another conversion, whenever the altered shape of the workpiece is drawn in the course of the simulation. (The only exception is the triangle based BRep representation, which can be directly visualized.)

The process of producing triangle based BRep model from an analytical BRep model is called *tessellation* or *triangulation*. Tessellation results the approximation of the object surface, characterized by the *chordal error* which is the maximum shape difference between the tessellated model and the original model (Figure 13) [34]. The triangle based BRep is often used as an intermediate data format between the CAD and CAM systems, as it can be easily handled and processed with simple algorithms, thus speeding up the subsequent conversation steps. The triangulated surfaces can be stored in STL files, primary developed for stereolitography, a rapid prototyping method. An STL file simply enumerates the triangles that form the surface by the coordinates of their vertices, together with their normals. The format has become a de facto standard and is supported by every major CAM application.

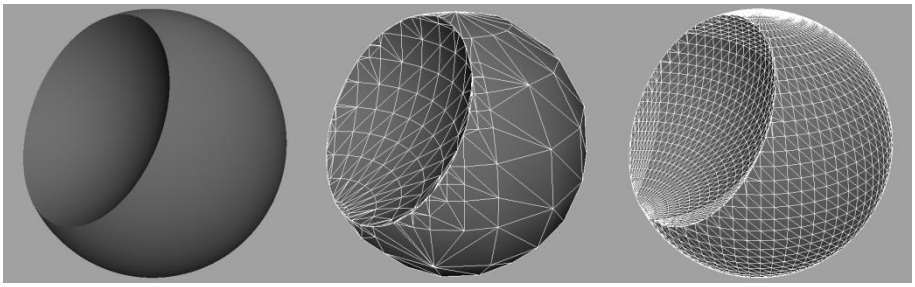

Figure 13. Surface tessellation at different levels

Layered depth-normal images (LDNIs) have been proposed by Wang and Chen [35][36][37] for representing very complex 3D objects and performing fast Boolean operations on them – but they can be used as an advantageous intermediate data representation in virtual machining too. LDNIs are the results of *depth peeling*, a common CG method, in which the object surface is disassembled into layers from a given point of view (Figure 14). The pixels of LDNIs hold the depth information (e.g. the distance from the camera plain) and the normal vector of the surface points; each LDNI belongs to one of the layers. Performing depth peeling on polyhedral objects is very fast due to the GPU support, while creating dexels from LDNIs is straightforward [38]. The entering and leaving dexel positions are obtained by transforming the screen space coordinates of the surface points on the odd and even LDNI layers to world space coordinates, and the vectors stored by the pixels represent the belonging normals (Figure 15). In the case of multi-dexel representation the dexelization process must be repeated in each of the three dexel directions.

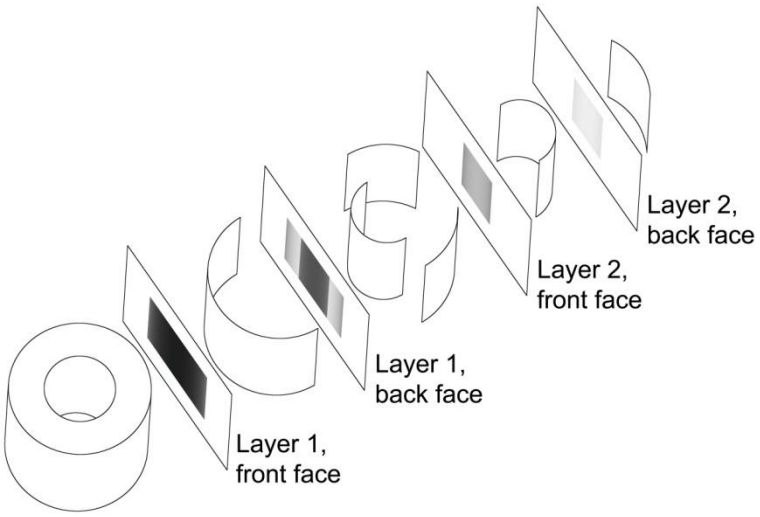

Figure 14. LDNI representation

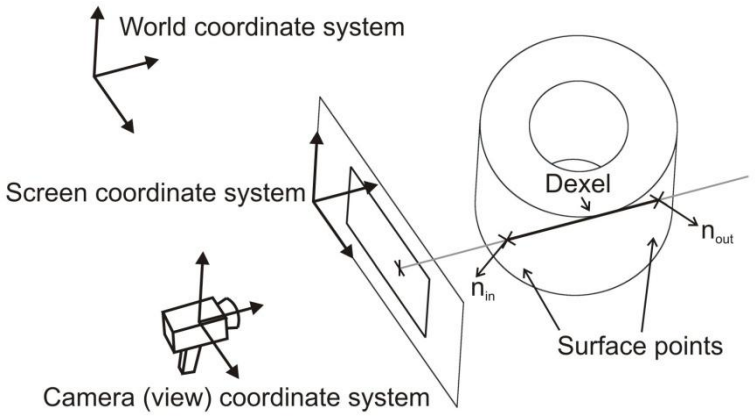

Figure 15. Dexelization

The USD, HSD, z-map and dexel-format data must be converted to triangle based BRep surface for the purpose of visualization. The simplest way is representing the data elements directly as cubes or right prisms (see Figure 10.b and c), but this technique gives satisfactory result only at very high modeling resolution, due to the jagged contours. A z-map can be tessellated as a *height map*, a common format of computer graphics [11]: the height values stored in neighbouring pixels of the height map mark the *z* coordinates of the triangle vertices that shape the BRep surface (Figure 16).

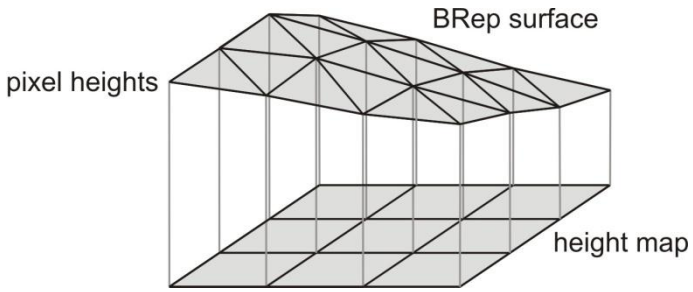

Figure 16. Height map tessellation

Marching cubes (MC) is a CG algorithm developed by Lorensen and Cline [39] to extract a polygonal mesh of an isosurface from a three dimensional scalar field. In the medical visualization the slices of CT and MRI images hold the gradient information of the field, where the neighboring pixels in the 3D space form a voxel. According to the local gradient vector, one of the 15 unique isosurface configurations, shown on Figure 17, can be assigned to each voxel. The polygonal faces within the voxels, according to the assigned configuration, shape the mesh.

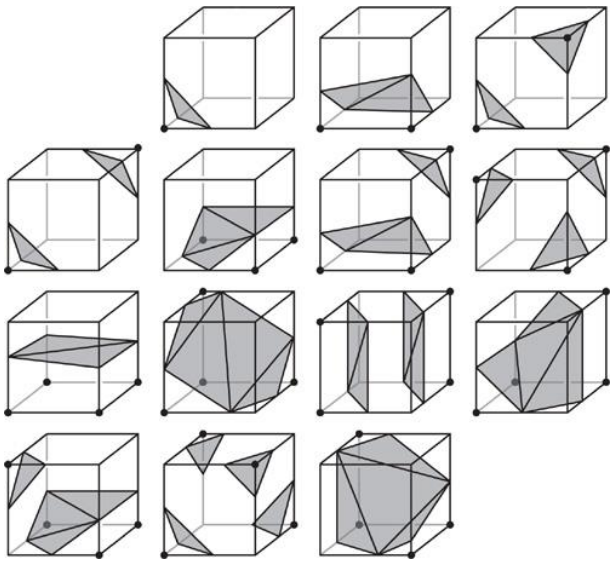

Figure 17. Unique surface configurations, MC algorithm [44]

The main problem of the marching cubes algorithm is that it cannot represent sharp features, such as edges and corners within the voxels. The *dual contouring method* [40] has been introduced by Ju et al. to eliminate this problem. In this case one representative vertex is assigned to each voxel. This vertex is placed at the sharp features of the surface, which allows the method to reproduce them in their original location (see Figure 18). The method is called *dual* since the surfaces produced by it are topologically dual to the surfaces produced by the marching cubes algorithm. The *extended marching cubes* (EMC) method [41] is a hybrid between the cube-based and the dual method. In the voxels which do not contain sharp features, the standard MC method is applied. For those voxels that do contain a feature, the method generates a vertex positioned at the edge or corner to represent it.

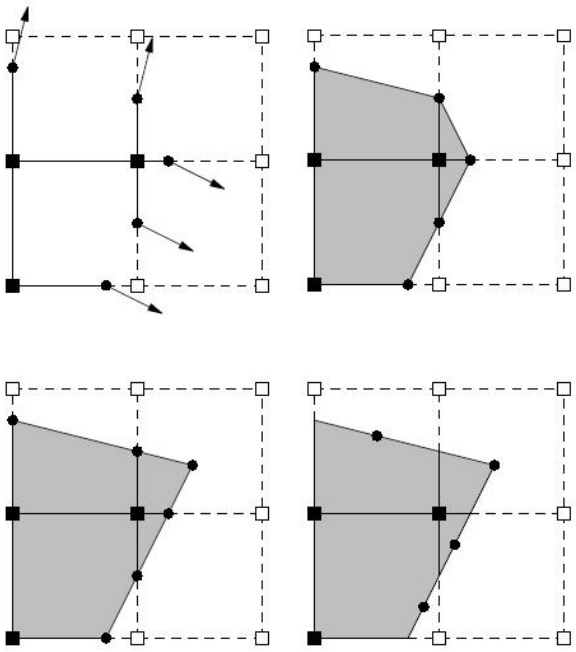

Figure 18. Surfaces by the MC (upper right), EMC (lower left) and dual contouring algorithms (lower right) [40]

The above enumerated contouring algorithms are also used to create the boundary surface of voxel-represented objects in virtual machining. The methods are suitable for triangulating HSD and dexel based data structures; in these cases the original data form must be first converted into voxel representation. The marching cubes algorithm has been further developed to display multi-dexel data sets as well [42]. Besides the MC based approaches, the work of Müller et al. [43] should also be mentioned, who used ray-casting for direct visualizing multi-dexel models with high image quality and moderated speed.

2.1.5 Cutter swept volume generation

As the tool moves across the space, it forms a *swept volume*. The basic principle of virtual machining is removing the swept volume, generated by cutter movements along the NC trajectory, from the model of the raw stock, and thereby obtaining the final machined surfaces or workpiece model [45]. The swept volume is often defined by the analytical or discrete description of its surface, the *swept envelope*. The simplest way of approximating the cutter swept volume is taking the union of the cutter volumes at a sufficient number of trajectory points of the tool path (Figure 19.a). The geometrical accuracy of the approximation depends on the density of trajectory points, the direction of displacement between the subsequent positions and the shape of the cutter. In practice, the surface quality resulted by this method is not only the function of the geometrical errors. For example, the generated surface looks much more rugged if flat-end mill is used in the simulation (Figure 20.b) instead of ball-end mill (Figure 20.a) apart from the corresponding errors (ε on the figure), because of the sudden change of the normals at sharp edges. We can improve the quality by increasing the number of trajectory points, but it must be considered that if we double the number of points, the simulation speed gets halved. In the case of the flat-end mill example the smoothing of the swept volume envelope would require extremely big density of trajectory points.

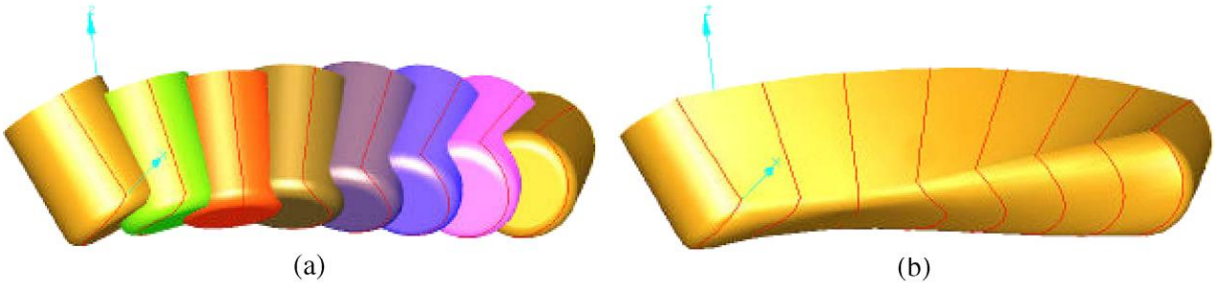

Figure 19. Ways of swept volume approximation [56]

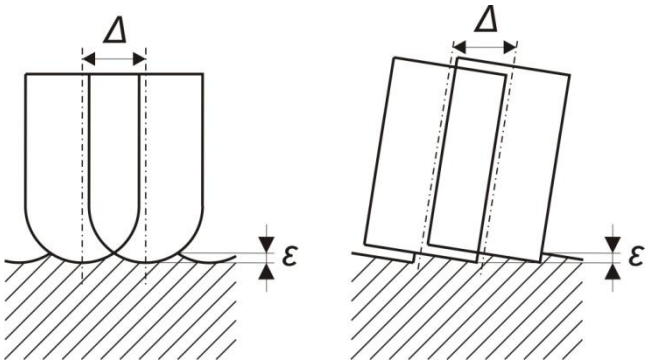

Figure 20. Geometric error on the resulted surface

In the past decades several analytical approaches have come to life in order to obtain more accurate swept volumes. Unfortunately there is no closed-form expression to describe the swept volume of a freely moving cutter [46]. One solution of this problem is to use the swept envelope for defining the swept volume [47]. Wang et al. [48] derived that the instantaneous velocity of a point on the envelope surface must be tangent to the envelope surface. Based on this tangency condition, trigonometric differential equations were solved to find the curves that spread the envelope (see the red curves on Figure 19.b). Abdel-Malek et al. [49] and Blackmore et al. [50] solved the system of the implicit equations numerically to obtain the swept surface. In the method of Chung et al. [51] a fourth order equation must be solved to get the swept profiles in the case of 3-axis milling. Sheltami et al. [52] found explicit equations of the swept profile curves. Later Roth et al. [53] developed a cross product method for a toroidal end mill in five-Axis machining. Mann et al. [54] extended this method to common cutter geometries. Chiou et al. [55] developed a methodology for generalized cutters in five-axis machining by analyzing the machine tool motions based on the machine configurations. Weinert et al. [56] introduced a BRep based approach which is partly derived from Wang's method. Aras [47] modeled the cutter by means of spheres to obtain the swept envelope of a generalized cutter that follows five-axis tool path during NC machining.

In spite of the promising research results, the use of analytically determined swept volumes has not fully replaced the simpler method. During 3, 4 and 3+2-axis machining, when the tool generally moves along straight lines and arcs, the swept volume can be easily divided into big pieces with well-definable shape. Edward and Lin [57] exploited this fact in their patent that constitutes the basis of a commercial graphics engine provided by MachineWorks Ltd. with remarkable simulation speed. But in the case of 5-axis machining, as the motion of the tool gets irregular when sculpting free-form surfaces, the analytical swept volume determination loses its effectiveness, and the traditional technique is used instead.

2.1.6 Material removal simulation in practice

The recent CAM applications guarantee high-quality visualization of the machining environment: they display the generated toolpath and other helper components to support the planning process, the CAD model of the raw stock and the final product, and the movement of the machine parts, tools and equipments – in a word, the kinematic simulation of the machining process. On the other hand, the material removal simulation modules greatly suffer from technical limitations, even in the products of the leader CAM software providers. In the case of 3-axis machining the z-map method and the dexel-based method is the mostly used, combined with the height map conversion technique before the displaying (VisualMill, SolidCam RapidVerify module). Some providers have tried to expand the z-buffer technique to 4 or 5-axis machining, involving pixel manipulating CG methods (SolidCam Verify Plus option, MasterCam standard simulation module); but they have only produced a rather

disappointing, view-dependent, disintegrating object representation. In Vericut dexel based simulation is used for both the three- and the multi-axis machining. Here the main problem arises at the conversation to BRep, before the visualization. As the marching cubes algorithm were probably too slow for this purpose, a special technique has been developed, where only the recently altered surface region is rebuilt and sent to the graphics hardware to be displayed. This means that the whole polyhedral surface must be re-created whenever the point of view has been changed – the result is a view-dependent displaying with very limited camera-rotating and zooming functions. The best surface quality has been achieved with the BRep based simulation (VisualCam, polygon module, SolidCam SolidVerify option, FeatureCam). The slowing of the Boolean calculations during long machining processes – which is the concomitant of the method – is compensated with reduced modeling or tool path accuracy. In FeatureCam the analytical swept volume determination of MachineWorks [57] is used for speeding up the 3-axis machining simulation (RapidCut module), but the method is not suitable when turned, indexed, or multiple fixture parts are used – namely in the case of 4 or 5-axis machining.

2.2 Parallel processing with general-purpose graphics processing units

The evolution of graphics processing units (GPUs) has been enormous in the past few years. Their calculation power has improved exponentially, while the range of the tasks that we can compute on GPUs has got significantly wider. Figure 21 shows the growth of calculation power of the high-end GPUs provided by the NVIDIA Corporation, versus the performance of the Intel central processing units (CPUs). The abbreviation GFLOPS means ’billion floating point operations per second’. (In February 2012 NVIDIA has announced its newest GPU, GeForce GTX 680, with 3090 GFLOPS calculation power. The flagship GPU of the rival manufacturer AMD, Radeon HD 7970, is able to reach 3789 GFLOPS [58].) The reason behind the discrepancy is that the GPU is specialized for compute-intensive, highly parallel computation – exactly what graphics rendering is about – and therefore it is devoted to data processing rather than data caching and flow control, which is the greatest asset of the CPU. More specifically, the GPU is especially well-suited to perform SIMD (single instructions, multiple data) operations, when the same program is executed on many data elements in parallel, and there is a lower requirement for sophisticated flow control; and because it is executed on many data elements, the memory access latency can be hidden with calculations instead of big data caches [59].

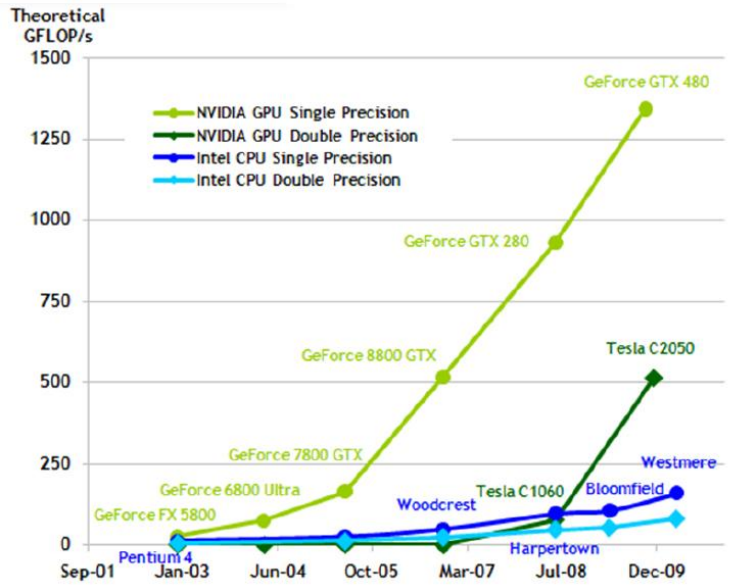

Figure 21. GPU vs. CPU performance [59]

A milestone of GPU development is the appearance of the *unified hardware architecture* in the recent years. Formerly the GPUs had utilized the traditional pipelined design shown on Figure 22. The functioning of the classic pipeline is the following. After the GPU has received vertex data from the host CPU, geometry transformations and lighting functions are executed at the *vertex stage*. In the 90's fixed-function hardware made these tasks, later this stage became programmable: from this time so-called *vertex shaders* (programs on the GPU) can be written for advanced vertex processing. In the next step the vertices are assembled into primitives such as triangles, lines or points. The primitives are then converted by the *rasterization stage* into pixel fragments. Fragments undergo many other operations such as shading and texturing (also programmable with *pixel shaders*), z-testing, possible frame buffer blending, and antialiasing. Fragments are finally considered pixels after they have been written into the *frame buffer*, the contents of which appear at last on the screen.

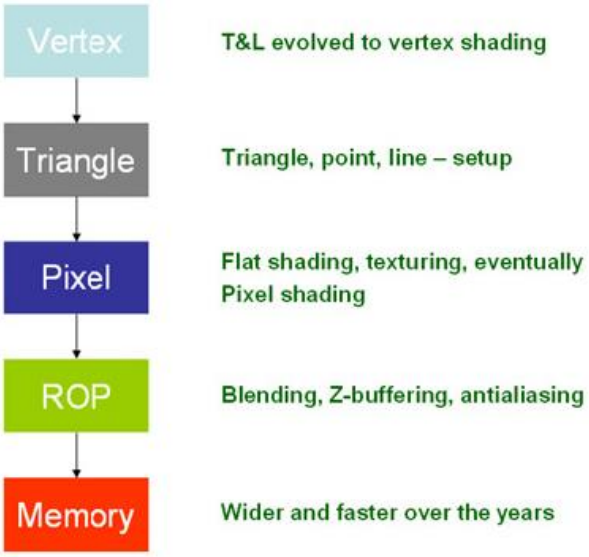

Figure 22. Classic GPU pipeline [60]

In the case of the unified architecture the various shaders run on the same hardware components (Figure 23). The results provided by the shaders are sent back for further processing until all shader operations are performed and the pixel fragment is passed on to the ROP subsystem. This solution allows defining new types of shaders as well – shaders, which are not strictly dedicated to perform only traditional graphical tasks. The *geometry shader* is an example, with which new geometry data can be generated on-the-fly, but it can execute physics calculations too. The unified hardware architecture has brought the possibility of general-purpose computing on the GPUs. The *general-purpose graphics processing unit* (GPGPU) is a graphics processing unit, which can be programmed to execute non-graphical (i.e. general-purpose), massively parallelized computations besides the conventional graphical tasks.

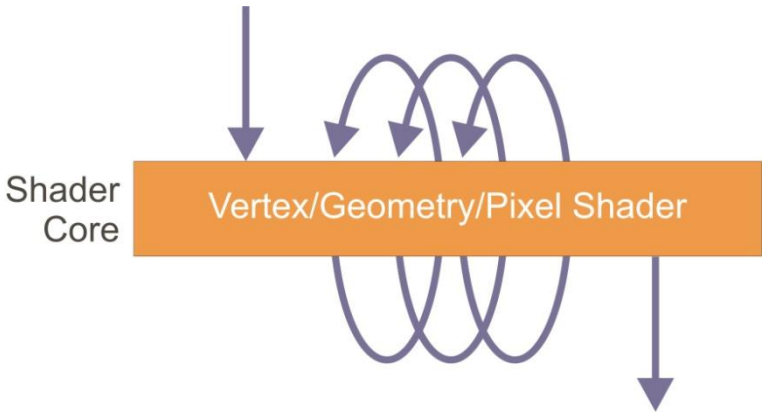

Figure 23. Unified hardware architecture

Shortly after the first GPGPUs had been put on the market, high-level programming languages appeared for implementing general-purpose tasks on the graphics hardware. The practicability and performance of CUDA (compute unified device architecture), the product of NVIDIA Corporation, and OpenCL, an open-source standard of GPGPU programming, made these tools very popular among scientists who searched for cheap solutions to solve highly parallel problems. The first academic paper in the field was published in 2007, and the number of publications rapidly grows since. Liu et al. [61] were among the first, who presented a new approach to execute high performance molecular dynamics simulations on graphics processing units. Manavski [62] introduced an efficient implementation of the advanced encryption standard (AES) algorithm in the CUDA platform. The results showed that the GPU can perform as an efficient cryptographic accelerator for the first time. Various Nbody (e.g. [63]) and molecular dynamics simulations (e.g. [64]) also showed impressive speedups. From day to day several new applications are announced on the websites dedicated to GPGPU programming [65].

The row of scientific applications using GPGPUs is long. In the field of the computer-aided manufacturing the GPGPU based accelerating of the finite element computations shows good results; most of the recent FE solvers support this kind of parallelized program execution. Lysenko et al. [66] performed interactive machinability analysis of free-form surfaces using GPGPU. Several [67][68] adapted the marching cubes algorithm to traditional GPUs with the use of common CG functions, and later to GPGPUs [44]. Nevertheless, not a comprehensive solution has still come to life, where the whole machining simulation process is supported by GPGPU based parallel computations.

2.3 Cutting force prediction in simulation environment

The material removal simulation is not only a help to visually verify the planned machining process, but it provides necessary geometrical information about the cutting for the purpose of cutting force prediction and feed rate optimization. The right choice of the cutting forces affects the safety of the cutting process, the machining time and the quality of the surface. The forces must be adjusted to the maximum horsepower of the spindle and to the permissible bending load on the tool to avoid its breaking and to keep a hand on the tool leaning, which results surface quality deterioration and shape errors. The knowledge of the cutting forces is also important to predict tool wearing and chatter.

Traditionally the cutting parameters are chosen from handbooks (e.g. [69]) that contain proposed values for several workpiece – tool couples. This manner works well if the cutting geometry, thus the machining parameters does not alter significantly during the machining process. This can be reached in 2½ axis milling, when the axial and radial depth of cut is hold in a tight range; but the shaping of free-form surfaces naturally implies the continuously changing of these values. Without the preliminary knowledge of the cutting geometry, the worst case must be considered at process planning. The conservative selection of the parameters often leads to additional machining time, wide

fluctuations of the cutting forces and premature tool wear due to light chip load, which reduces productivity [70].

The CAM software providers have recognized this problem and included functions into their applications in order to keep the cutting forces near to the optimal all the time during the machining. There are two stages of the process planning for doing so: the *tool path optimization* and the *feed rate optimization* steps. In the tool path optimization, in general, alternative trajectories are examined to get the shortest path [71][72] without unnecessary idle running of the tool [73], considering the kinematic structure and behavior of the machine tool [74] and the possible collisions in the working area [75]. Though efforts have been made [76][77] to involve the mechanics of the cutting processes into the computations, the problem is so complex, that the paths available in CAM systems are generated only on the basis of geometric computations.

The off-line feed rate optimization is the subsequent step after the geometric tool path optimization. Here the trajectory of the tool path, generated by the preceding, is no longer manipulated. Only one parameter, the feed rate, is controlled to achieve the optimal balance of cutting forces – thus the computational complexity is kept manageable. The common approach used in feed rate scheduling is the *material removal rate* (MRR) model. In the MRR based approach, feed rate is inversely proportional to either the average or the instantaneous volumetric removal rate. It is considered that the power (P_c in kW) required to cut the material is proportional to the volumetric removal rate (Q in cubic centimeters per minute) [78][79].

$$P_c = kQ \tag{3}$$

$$F_t = \frac{kQ}{v} \tag{4}$$

$$F_a = k_a F_t \tag{5}$$

$$F = \sqrt{F_t^2 + F_a^2} \tag{6}$$

where k and k_a are constants belonging to the actual workpiece material – tool couple, v is the cutting speed, F_t, F_a and F are the tangential, axial and resultant forces.

Several CAM packages contain feed rate optimization module (Optipath module in Vericut, Hifeed in Mastercam, Optifeed in Powermill, FeatureMILL) based on the MRR model with short calculation time, promising longer tool life and shorter cutting times. However, the main problem of the MRR model is, it is incapable of determining instantaneous cutting force magnitude and direction, thus the MRR based feed rate scheduling neither has an effect on surface texture and roughness nor has an effect on increased accuracy [80]. Unreliability is another issue of the MRR method: Kurt et al. [81] showed that the same material removal rate can hide various milling forces under different cutting conditions. For that very reason it is a highlighted area of recent research works to involve such

mechanistic cutting force models into the feed rate optimization, which concern the physics of the machining process too. Yazar et al. [82] performed feed rate optimization on the basis of cutting force calculations in three-axis milling of dies and molds. Ko et al. [83] presented similar feed rate scheduling models for flat end milling. Lazoglu et al. [84] work on a feed rate scheduling system for sculptured surface machining based on the cutting force model.

In the study of Erdim et al. [85] the MRR and force-based feed rate scheduling strategies were compared theoretically and experimentally in the case of 3D ball-end milling of free-form surfaces. It was shown that the MRR based strategy outputs higher feed rate, thus higher force values compared to the force based strategy, which can increase the tool wear and deteriorate the final surface. On the other hand, the complexity and validity problems of the mechanistic force model make it hardly applicable in practice. As Kurt and Bagci [81] sum in their comprehensive review: "It is necessary to use MRR based feed rate optimization for rough machining due to the more simple and short calculation time, but it is necessary to use more precise and correct force based feed rate scheduling approaches, which require more calculation time and at the moment they are not adapted in commercial software packages for finish machining." In the followings the mechanistic cutting force model will be detailed, together with its advantages and limits, with especial regard to the feasibility issues.

2.3.1 Mechanistic cutting force model

The *mechanistic cutting force model* extends the fundamental cutting force equations – which describe the effect of shearing and ploughing along an elemental cutting edge – to the complex geometry of the milling tool. The traditional theory of the mechanistic modeling [86][87] has been further developed to the conditions of milling in several ways. Yang and Park [88] represented the ball-end mill geometry by segments of orthogonal cutting planes. Tai and Fuh [89] described the cutting edge geometry as intersections between skew planes and spherical ball-end mill surface. The currently accepted model was published by Lee and Altintas [90] on the basis of the semi-mechanistic model of Yucesan and Altintas [91]; they divided the cutting edge into small oblique cutting edges and applied the traditional force model of oblique cutting on an elemental cutting edge to each segment. They described the method for ball-end mills, but it is applicable for general end mills without significant modifications [92].

In most of the earlier models the effect of the ploughing forces was not taken into consideration, or the ploughing was taken into account only in the "specific cutting force" constants [93]. Lee and Altintas use the *dual mechanistic model*, where both the ploughing and shearing effects are included and they appear separately in the force equations. For each segment the tangential, radial and axial force components (N) are calculated as shown in Eq. (7).

$$\begin{cases} dF_t(\theta,z) = K_{te}dS + K_{tc}\cdot t_n(\Psi,\theta,\kappa)db \\ dF_r(\theta,z) = K_{re}dS + K_{rc}\cdot t_n(\Psi,\theta,\kappa)db \\ dF_a(\theta,z) = K_{ae}dS + K_{ac}\cdot t_n(\Psi,\theta,\kappa)db \end{cases} \tag{7}$$

where K_{te}, K_{re}, K_{ae} (N/mm) are ploughing (edge) specific coefficients, K_{tc}, K_{rc}, K_{ac} (N/mm^2) are shear (cut) specific coefficients, dS (mm) is the length of the elemental edge, t_n (mm) is the undeformed chip thickness, and db (mm) is the projected length of the edge segment in the direction along the cutting velocity, or in other words the chip width. θ is the spindle rotation angle, Ψ and κ are the horizontal and vertical positioning angles of the segment, according to Figure 24.

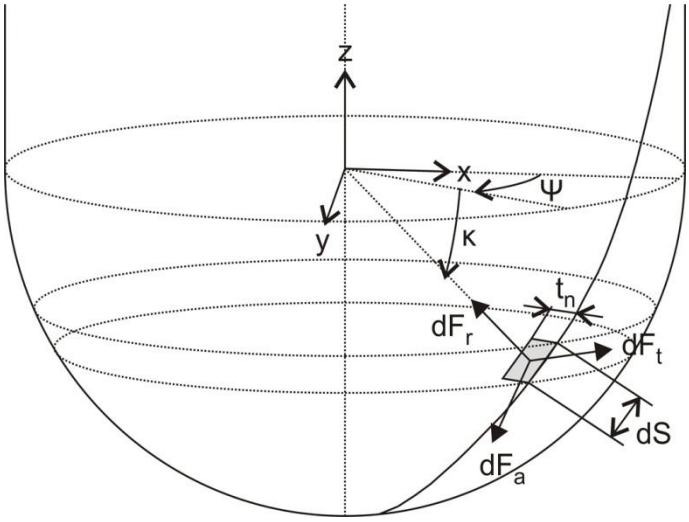

Figure 24. Cutting edge segment

The classical chip thickness model assumes that the tooth path is circular, thus the chip thickness can be approximated by Eq. (8), where f_z is the feed per tooth. Some researchers [94]-[97] focused on the more accurate description of the chip geometry considering the cycloidal motion of the teeth, while others [98][99] involved the effect of cutter runout caused by the misalignment of the tool as well – but in these cases only a limited set of machining operations were investigated, such as plain cutting or upward and downward ramping. The knowledge of the instantaneous cutting force vectors allows the estimation and consideration of cutter deflection [100]-[104] and forms the basis of the chatter and stability investigations, which is beyond the scope of the dissertation.

$$t_n = f_z \sin\psi \sin\kappa \tag{8}$$

To get the total cutting force along one entire edge, the force components at the *active segments*, which are taking part in the cutting at the moment, are transformed into a common coordinate system and integrated. The force components related to the tool coordinate system can be obtained by Eq. (10) resulted by the transformation in Eq. (9).

$$\begin{bmatrix} dF_x \\ dF_y \\ dF_z \end{bmatrix} = \begin{bmatrix} -\cos(\kappa)\cos(\Psi) & \sin(\Psi) & -\sin(\kappa)\cos(\Psi) \\ -\cos(\kappa)\sin(\Psi) & -\cos(\Psi) & -\sin(\kappa)\sin(\Psi) \\ \sin(\kappa) & 0 & -\cos(\kappa) \end{bmatrix} \begin{bmatrix} dF_r \\ dF_t \\ dF_a \end{bmatrix} \tag{9}$$

$$\begin{cases} dF_x = -dF_r \cos(\kappa)\cos(\Psi) + dF_t \sin(\Psi) - dF_a \sin(\kappa)\cos(\Psi) \\ dF_y = -dF_r \cos(\kappa)\sin(\Psi) - dF_t \cos(\Psi) - dF_a \sin(\kappa)\sin(\Psi) \\ dF_z = dF_r \sin(\kappa) - dF_a \cos(\kappa) \end{cases} \tag{10}$$

The most common way of dividing the cutters into small segments – after the proposal of Kline and DeVor [105] – is slicing the cutter into a series of axial discs (Figure 25). The active range of each disc, in which the corresponding segment is engaged with the workpiece, is marked out by the *contact area* between the cutter envelope and the workpiece. In the beginning analytical calculations were used to determine the contact area, assuming simple and constant cutting geometry during the machining process. In the course of multi-axis machining, when the geometrical conditions are continuously altering, this method does not provide sufficient results. For that reason, with the improving of the simulation technology, the use of analytical calculations has been exchanged by discrete methods. The most widely applied is the z-map technique, proposed by Kim et al. [106]. The contact area is obtained from the comparison of the z-map data of the cutter envelop and the workpiece, where the plains of the z-maps are perpendicular to the direction of the machining. The projected image of the contact area appears on this plane (see the bottom of Figure 25), to which the axial discs are also projected. By comparing each projected disc with the projected contact area, the entry and exit cutter rotation angles that indicate the active range on the disc are determined.

Unfortunately the methodology of the z-map technique calls forth that the lateral surface parts, the normal of which are perpendicular to the z-axis, cannot be represented on the projection plane, and the surface patches with almost-perpendicular normals become strongly distorted after the projection. This means that the z-map method cannot be applied for flat-end mills and for machining processes when the cylindrical part of the tool takes part in the cutting. Though further techniques [107] have been proposed to determine the contact area in a more complex way, their calculation demand is so big that they can be used only for short machining movements. It can be said that no general solution has come to life until now to solve this problem.

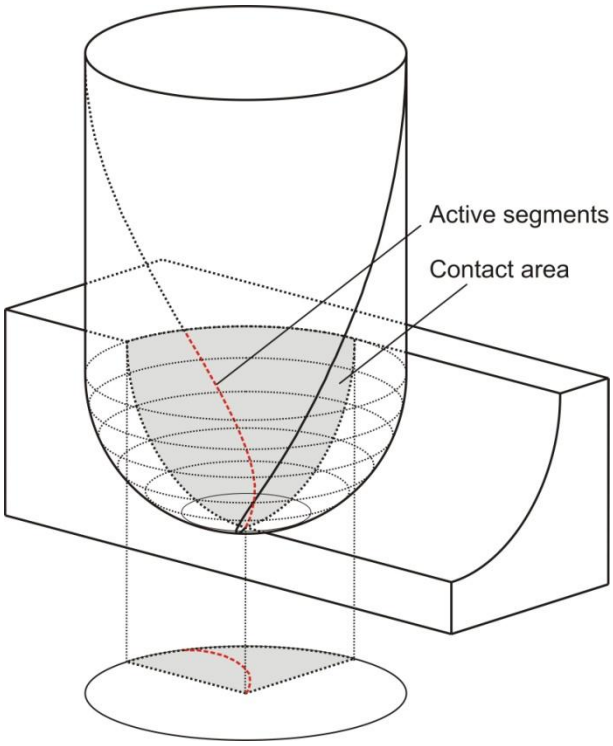

Figure 25. Active segments and contact area

2.3.2 Determination of the cutting force coefficients

In the case of cylindrical end-mills the inclination angle of the edge coincides with the helix angle, since the tangent plane to the edge is always parallel to the tool axis. However, there is no coincidence between the inclination and helix angles at the ball-end and general end mills. The tangential, radial, and axial components of the cutting force depend on the cutting edge inclination angle. Therefore, the contribution of each cutting edge element to the resulting cutting force is varying and it will depend on the inclination angle of the edge. The inclination of the ball-end mill cutting edge varies along the edge and depends directly on position *z* of each element; thus using coefficients that depend on *z*, the effect of this variation is included [108]. Different functions can be used for this purpose, but the most usual approach is the use of polynomial functions (see Eq. (11)). Though some authors apply quadratic [92] or cubic [109][110] polynomial shear coefficients, Lamikiz et al. [108] showed that linear approximation is enough for cutting force estimations; the improvements obtained from using coefficients of higher order is below 2%, while the calculation demand is much higher (with 33% and 67% when considering quadratic and cubic coefficients). On the other hand, most of the authors consider the ploughing coefficients as constant. Tests prove that the total resultant force hardly varies when polynomial approximation is applied [111].

$$\begin{cases} K_{tc} = K_{tc0} + K_{tc1} \cdot z + K_{tc2} \cdot z^2 + \cdots + K_{tcn} \cdot z^n \\ K_{rc} = K_{rc0} + K_{rc1} \cdot z + K_{rc2} \cdot z^2 + \cdots + K_{rcn} \cdot z^n \\ K_{ac} = K_{ac0} + K_{ac1} \cdot z + K_{ac2} \cdot z^2 + \cdots + K_{acn} \cdot z^n \end{cases} \quad (11)$$

Besides the dual mechanistic model the *lumped model* is also widespread (see the works of Kim [106][112] and others [113][114]), where the shearing and ploughing effects are described by a single coefficient. Based on Kline and DeVor's assumption [115] Feng and Menq [109] expressed the elemental cutting forces acting on a cutting edge element as

$$\begin{cases} dF_t(\theta, z) = K_t(z)dz[\mathrm{t}(\theta)]^{m_T} \\ dF_r(\theta, z) = K_r(z)dz[\mathrm{t}(\theta)]^{m_R} \end{cases} \tag{12}$$

where the K_t and K_r coefficients are approximated as polynomials. m_T and m_R explicitly characterize the *size effect* that refers to the increase of the cutting force for small values of the uncut chip thickness, caused by the plowing forces acting on the rounded edge on the flank face of the cutter [116][117]. In other forms of the lumped model [32][118] the size effect is not taken into consideration but the force equations are widened with the axial force component. The lumped model is popular because of the less computational demand, but experimental results indicate [119] that the dual mechanistic model predicts the forces with better accuracy than the models that use lumped cutting coefficients.

In the case of the mechanistic model the determination of the cutting force coefficients is a critical point. For getting coefficients which are independent from the cutting conditions, most authors propose the evaluation of a set of experiments with well-defined technological parameters; the coefficients are then obtained from these experiments via numerical or analytical calculations. Two different types of calibration approaches have been reported in the literature. The *unified cutting mechanics approach* relies on an experimentally established orthogonal cutting database, incorporating the tool geometric variables, and the cutting force models are developed on the basis of a generic oblique cutting analysis. Armarego and Deshpande [120] proposed a flat-end milling model where the force coefficients were obtained by transforming the orthogonal cutting parameters to oblique milling. Yang and Park [88] developed a ball-end milling model using orthogonal cutting data obtained from end turning tests. Tai and Fuh [89] used a similar model, but derived from the oblique cutting theory. Yucesan and Altintas [121] evaluated the varying rake face friction and pressure distribution and the chip flow angle in peripheral milling in order to provide accurate cutting force predictions. Later Altintas et al. [122][90] determined the milling force coefficients from orthogonal cutting tests with oblique cutting analysis and transformation; they demonstrated that this model could predict total machining forces over a wide range of cutting conditions.

According to the *direct calibration approach*, if we look for the coefficients that are valid to a given cutting tool, we should use the tool itself during the characterization tests. The cutting forces are measured and, along with the cutting conditions of each test, are used as inputs for obtaining the coefficients. Kline et al. [115] applied this approach first to develop a mechanistic cutting force model for flat-end milling. Feng and Menq [109][110] employed the same approach when they worked out their lumped mechanistic cutting force model for the ball-end milling process. For the dual

mechanistic model Wang and Zheng [119] developed a method, where constant shearing and plowing force coefficients were expressed in terms of the measured cutting forces as well as the axial depth of cut and cutter radius. Lamikiz et al. [108][111] determined polynomial shear and constant ploughing coefficients from a set of horizontal slot milling tests with different cutting conditions. They applied an inverse calculation method for the dual mechanistic force equations and obtained the coefficients by least square adjustment.

The disadvantage of all the referenced methods is the need of a large number of calibration test cuts to determine the coefficients. Different works have been focused on developing calculation methods in order to get the coefficients from minimum experimentation. Yun and Cho [123] determined the cutting force coefficients for flat-end milling on the basis of the measured force signals of one reference test cut – but the cutting mechanics parameters were kept constant and the size effect was not considered. Azeem et al. [113] proposed a solution for ball-end mills to simplify the calibration tests with an effort to create a model which is valid over a wide range of cutting conditions. They showed that the information required for the coefficient determination could be gained from a single half-slot milling experiment with the use of instantaneous measured cutting forces instead of average forces that was generally applied in the previous studies. On the other hand – besides they used the less accurate lumped force model instead of the dual mechanistic model – their model needs manual intervention to select the utilizable samples from the measured force values, which prevents the effective execution of the calibration. In addition, the main problems related to the coefficients have not been solved yet. First of all the coefficients depend on the part material, the material and coating of the tool and the geometry of the cutting edges. This requires calculating them for each tool-material couple by experimental tests [108]. Secondly, the value of the coefficients are changing with the wearing of the cutter during the machining, thus they lose their validity soon after the start of the milling process [124]. These issues make the application of the mechanical cutting force model, in spite of being more accurate and providing more information than the MRR model, very difficult to implement in practice.

3 Material removal simulation and cutting force prediction with GPGPU

3.1 Discussion of the extant technological shortcomings – aims and scopes of the research

The modern CAM systems provide high-level graphical support for the process planning and preliminary verification of machining. The visual appearance of these applications is fascinating due to the high-resolution polygon-based kinematical simulation and numerous visual helpers on the screen. On the other hand, it cannot be clearly declared that the same holds to the available material removal simulation modules as well. The z-map method has proved to be very effective in the case of 3-axis machining simulation by exploiting the intense support of the traditional graphics hardware; while the analytical swept volume generation can speed up the three- four- and 3+2-axis machining simulations due to the well-definable movement of the tool. However, when free-form surfaces are shaped with 5-axis machining, these techniques cannot be used any more. The adaptation of the original z-map method to 5-axis simulation brought poor result; the extended z-map methods and the object space based techniques loosed the support of the graphics hardware, thus slowing down the process; the advanced swept volume generation solutions did not proved viable.

The primary aim of the research was to create a material removal simulation technique for 5-axis machining that enjoys the benefits of the highly parallel execution on general-purpose graphics processing units, the newest generation of graphics hardware. This means the minimizing of the CPU intervention in data conversion – from BRep CAD model to solid model and further to displayable polyhedral model –, data representation and processing, and visualization. The method must provide the followings:

- Calculation speed of the same order of magnitude as the speed that the z-map method can reach on traditional graphics hardware;
- Highly parallelized execution of Boolean operations by GPU shaders, supporting swept volumes and complex cutter geometries besides the analytical calculations;
- Fully GPU based data conversion from solid to polyhedral model to avoid the CPU-GPU data transfer bottleneck;
- Visualization of the process with the quality of the BRep based CAD models.

The other, more and more important purpose of performing material removal simulation besides the simple visualization is to provide geometrical information about the cutting process for cutting force estimation and feed rate optimization. By the feed rate optimization modules, which are included in the recent CAM applications, the MRR method is applied to keep the machining parameters optimal – notwithstanding the application of the mechanistic cutting force model, which concern the physics of the machining process, is proved to give more accurate and comprehensive results. The main reasons are related to the complicated determination of the coefficients acting in the force equations, together

with the limited validity of them, which makes the creating and using of a general coefficient database impossible.

The second part of the research focuses on the applicability and feasibility questions of the mechanistic cutting force model. It has been assumed that two main objectives must be fulfilled for making the model more effective in practice. First of all, the way of force-calculation must be reconstructed to reach the requirements of the real-time multi-axis simulation. This means the need of direct access and parallelized consuming of the geometrical data gained during the simulation, to avoid the unreliable projection steps and the mainly sequential execution related to the existing z-map technique. Secondly, the method of the coefficient determination must be fastened and simplified. It was supposed and shown afterwards, that the reducing of the geometrical requirements belonging to the existing calculation methods, together with the real-time execution of the coefficient determination can lead to the solving of the validity problem.

3.2 Material removal simulation

Figure 26 shows the flowchart of the traditional multi-axis material removal simulation process. The chart concentrates on the virtual machining and visualization of the workpiece and does not contains the displaying of the machine parts, the cutter and other visual components of the process planning and verification. The boxes in the diagram represent the main steps of the program code execution; the connecting arrows display the data flow between the consecutive steps. The thickness of the arrows indicates the amount of the transferred data – the thin lines mean a few moving data, while intensive data flow occurs along the thick ones. The manner of the data is indicated above the arrows. The independent thick gray arrows show the three main phases of the execution, such as the *initialization*, the *CPU* and the *GPU threads*. The two boxes on the left side mark the steps of the initialization, as it can be read on the top edge of the figure. The initialization is executed only once at the beginning of the simulation; it causes a one-time data generation and transfer. The initialization is followed by the cyclically executed CPU thread, which forms the backbone cycle of the program. The steps belonging to the CPU thread are executed by the central processing unit (CPU). The CPU starts a new GPU thread in each simulation cycle after the data required for the visualization has been transferred to the graphics hardware. The moment of the CPU-GPU data transfer and the synchronized starting of the GPU thread are marked with a hatched box on the diagram. The GPU thread runs on the graphics hardware in parallel to the CPU thread on the central processing unit; the beginning of the next CPU cycle is subsequent to the termination of the current GPU thread.

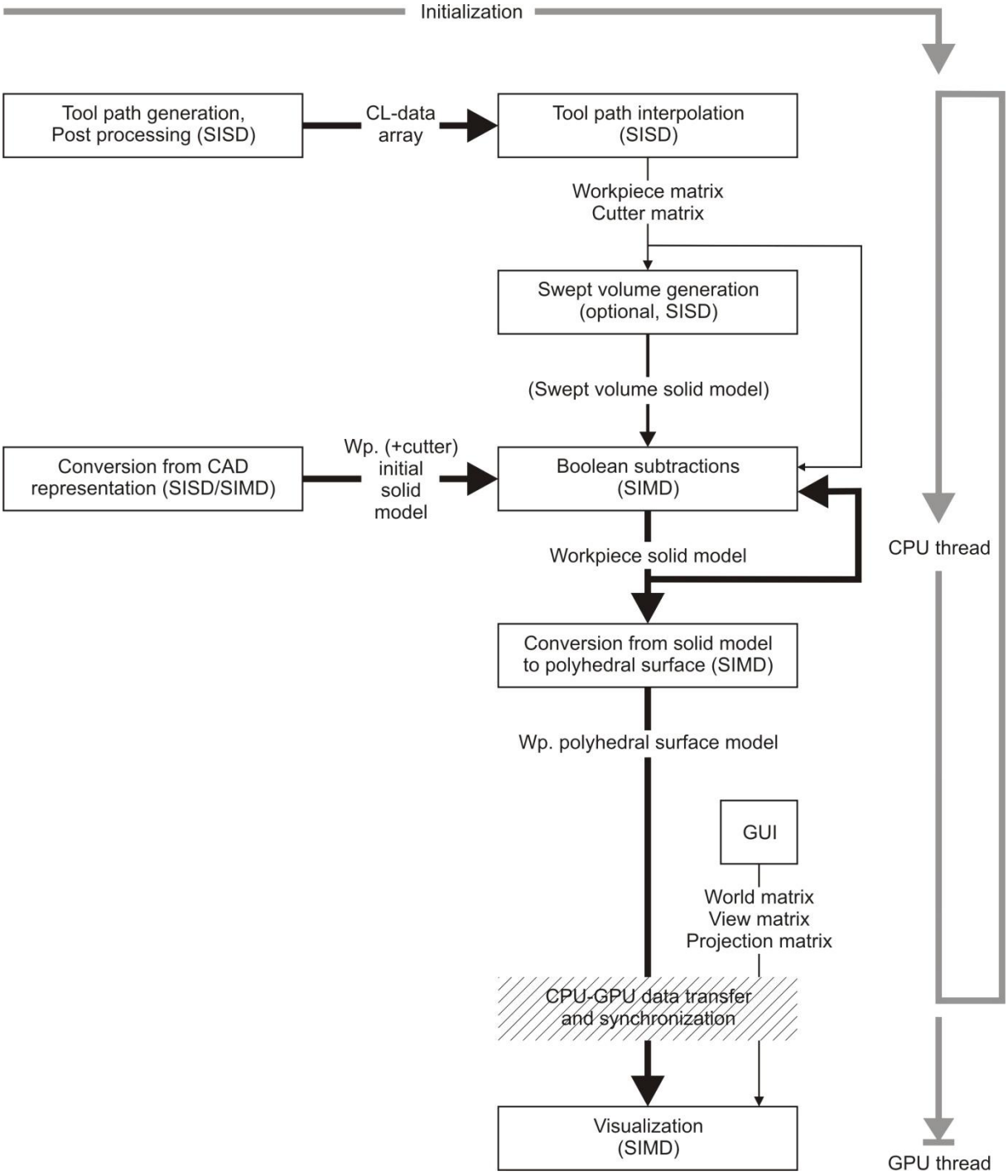

Figure 26. Traditional material removal simulation flowchart

Every step of the flowchart is characterized by the nature of the operations that generally form the belonging program code. *SISD* means "single instruction, single data" – and originally refers to a computer architecture in which a single processor executes a single instruction stream, to operate on data stored in a single memory. This corresponds to a single core CPU with Neumann architecture. A program code characterized by SISD manner "feels good" in this environment: it includes subsequent operations to be executed in one thread, while each operation has its own element of the data set to

manipulate. The inputs of the operations are often the results of the preceding ones, and the program contains conditional jumps depending on the previous results, thus the code cannot be effectively parallelized. The other appearing abbreviation on the flow diagram, *SIMD* means "single instruction, multiple data". Originally it describes computers with multiple processing elements that perform the same operation on multiple data simultaneously, exploiting data level parallelism. A typical example is the graphics hardware, where the same transformations and manipulations are applied to many vertices and pixels in parallel. Program codes with SIMD manner can be effectively executed on this architecture: they can be unfolded to many parallel threads that execute the same instructions on independent data elements. When a program code with SISD manner runs on SIMD architecture or vice versa, it loses its efficiency. If we don't use all of the available processing elements, or at least many of them; or we try to run a many-threaded code on a single processor core, we don't exploit the advantages of the architecture and we don't get acceptable result.

Let's go through the steps of the material removal simulation process before discussing the problems related to the method and their reasons. The initial data required for the starting of the material removal simulation is created in the initialization phase in the appropriate format. *CL-data* consists of the set of spatial points along the tool path, defined by tool reference position ***p*** and tool axis vector ***u*** (Figure 27). CL-data generation is the task of the CAPP (computer-aided process planning) subsystem of CAM, and includes the following steps (shown on Figure 28):

- Tool path planning, which starts at the CAD surface design and results the set of *cutter contact (CC) points* along the tool path, defined by the cutter contact position ***c*** and unit normal vector ***n***;
- CL-data calculation from CC-data, on the basis of kinematic modeling, interference and collision check and CL-data optimization;
- In addition, for correct timing and later cutting force calculations, the local feed vector ***f*** can be attached to each CL point. The feed vectors are calculated in the post processing phase considering the machine controller parameters [125].

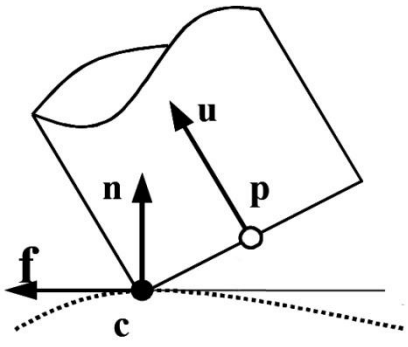

Figure 27. CL-data [125]

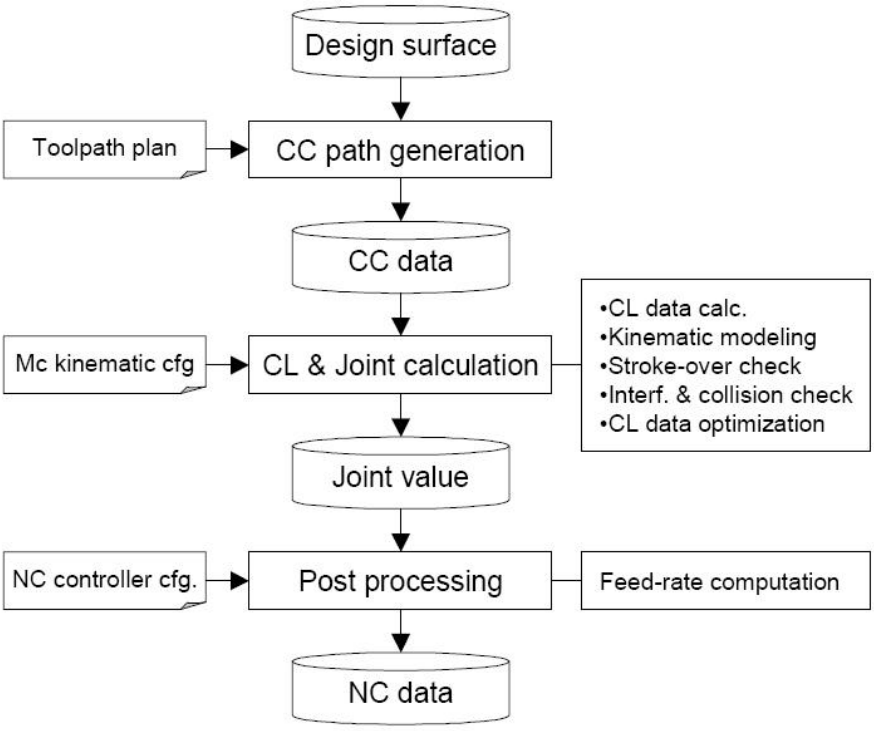

Figure 28. General five-axis NC data generation procedure [125]

The other initial data required for the material removal simulation is the solid model of the blank. The model must be converted into the inner object representation format of the simulation, such as dexels, BRep triangles etc. from the original CAD representation (which is analytical BRep in most cases). Usually the object is exported in STL format from the CAD program to be further converted by the use of the CPU but with involving the GPU as well – as it was described in the technical review part of the dissertation. The triangle based surface representation of the cutter is essential for its visualization and in certain cases it is employed at the Boolean operations too.

In the first step of the backbone (CPU) cycle the advance of the cutter is controlled. The tool moves along the tool path shaped by the CL points; the position and orientation of the advancing tool is interpolated along the curve in each simulation cycle. If no additional swept volume generation is applied, the subsequent tool positions must follow densely each other to result a smooth swept surface. The position and orientation of the tool – similarly to the workpiece – is encapsulated in a homogenous transformation matrix

$$\mathbf{T}_t = \begin{bmatrix} \mathbf{x}_t^T & 0 \\ \mathbf{y}_t^T & 0 \\ \mathbf{z}_t^T & 0 \\ \mathbf{p}_t^T & 1 \end{bmatrix}, \tag{13}$$

where $\mathbf{x}_t$, $\mathbf{y}_t$ and $\mathbf{z}_t$ are the standard basis vectors of the tool coordinate system and $\mathbf{p}_t$ is the vector to its origin (tool reference position) in the workpiece coordinate system. Note that the matrix is valid in a

left-handed system. In contrast to the engineering, graphical APIs (application programming interfaces) often apply left-handed representation instead of right-handed, thus following the logic of the image space: origin at the left-bottom of the screen, horizontal *x*-axle, vertical *y*-axle and the *z* values mark the depth – the distance from the screen-plane – of the pixels. In the followings, for avoiding confusion and being in compliance with the programmed graphical algorithms, matrices and vector-equations that are valid in left-handed coordinate systems will be presented in the text.

It could be read in the previous part that the swept volume generation is not widely held in 5-axis simulation. However if tools with sharp-edged envelope, such as a flat-end mill, take part in the simulation, an additional swept volume generation step can improve the quality of the visualization. This step usually provides the triangular BRep of the swept surface, which does not mean a big amount of data to be created and transferred, just a couple of tens or at most hundreds of triangles. The solutions published up to now use analytical computations with SISD manner.

The main part of the material simulation is the cyclical subtraction of the cutter volume or the cutter swept volume from the volume of the workpiece. The basis of these Boolean operations is constituted by the set of data elements that forms the solid representation of the workpiece and the polygonal or analytical description of the cutter (swept) envelope. In usual the same operations are performed on each data element (SIMD), demanding big computational capacity from the hardware. The result – the whole of the modified and newly created data elements – is fed back to the inputs of the Boolean operation step.

The data elements of the solid model are not suitable for direct visualization – if only they are not surface triangles of a polygonal BRep model. This requires a conversion step before the displaying. The conversation can be even more complex and more time-consuming than the step of Boolean operations; it often includes the rearranging of the data besides the SIMD-like transformations and results a big number of polygons. Nevertheless the conversation process does not necessary affect every data element; only the visible, changing parts must be redrawn if we want to speed up the simulation. Furthermore, at the cost of the smooth visualization, the conversation and redraw must not be performed in every simulation step.

The visualization is the only step of the traditional material removal simulation process, which is executed by the graphics hardware. The CPU transfers the polyhedral (triangle based) surface model of the workpiece to the GPU and instructs it to display. The GPU sets the point of view in compliance with the World ($\mathbf{T}_W$) and View ($\mathbf{T}_V$) transformation matrices and projects the scene onto the screen by performing a perspective or orthogonal projection ($\mathbf{T}_P$):

$$\left[\mathbf{p}_{screen} \quad 1\right] = \left[\mathbf{p}_{scene} \quad 1\right] \cdot \mathbf{T}_W \cdot \mathbf{T}_V \cdot \mathbf{T}_P , \tag{14}$$

for each vertex of the surface model with $\mathbf{p}_{scene}$ coordinates relative to the coordinate system assigned to the scene, which includes the machine tool, workpiece and tool. The transformation matrices are set by the user through the graphical user interface (GUI). The transformations are fulfilled by the *vertex*

shader, a short program in the graphics hardware dedicated to vertex based transformations and lighting. In the next step the vertices are assembled into triangles and then converted into fragments of the screen space. Fragments undergo many other operations such as shading and texturing (also programmable with *fragment* or *pixel shaders*), depth testing, possible frame buffer blending, and anti-aliasing. Fragments are finally considered pixels when they have been written into the *frame buffer*, the contents of which appear at last on the screen (Figure 29).

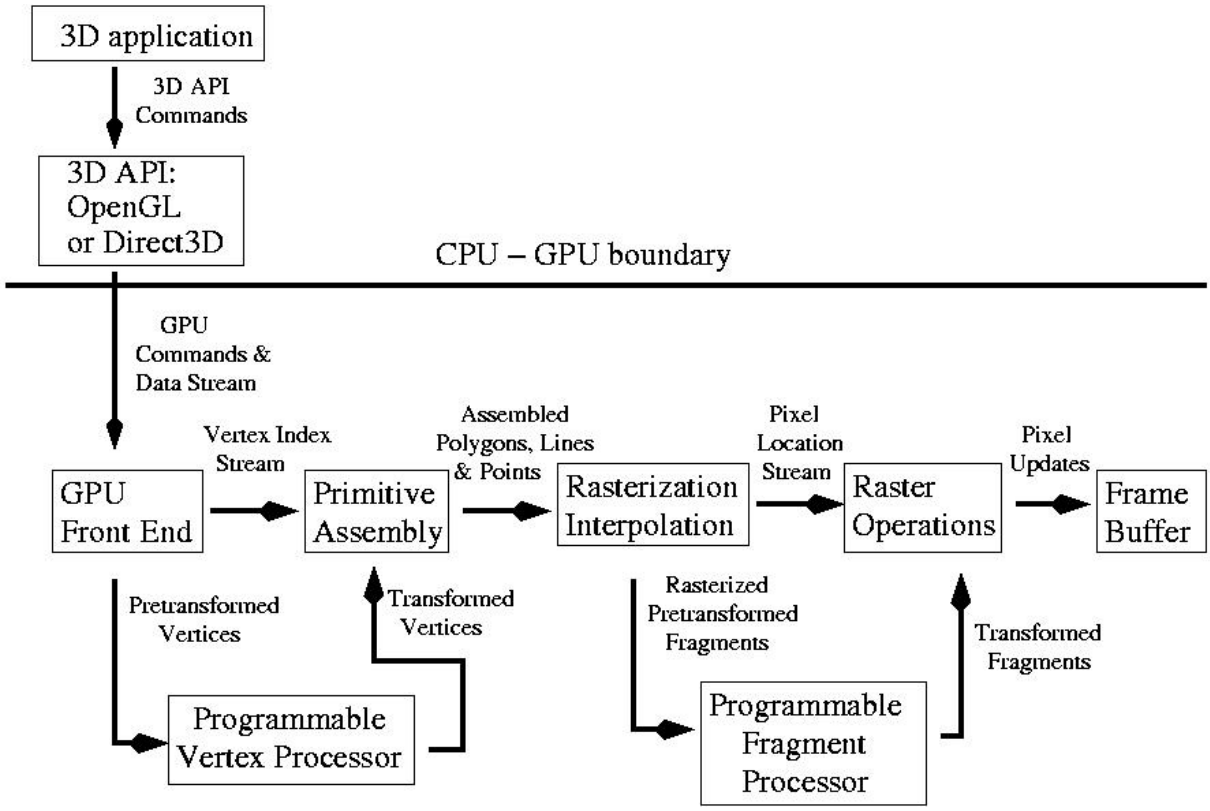

Figure 29. Classic GPU pipeline [126]

Let's see what the disadvantages of the traditional process are. First of all, the CPU is highly overloaded with tasks that would require SIMD hardware environment instead of the greatly inefficient sequential execution on the CPU. The high parallelism of the Boolean subtraction and conversion calculations makes even the CPUs with four or eight cores unsuitable for effectively performing these tasks. On the other hand, the GPU is not properly utilized: in most of the time it just waits for the results of the CPU thread instead of taking over some of the calculations which would exactly require its parallelized architecture.

Another problem is the big amount of transferred data between the CPU and the GPU. The CPU-GPU data transmission bandwidth is limited as real physical data flow occurs from hardware to hardware. The action when the polygons of whole objects or scenes have to be transferred to the GPU and have to be displayed is called *context switching* in the computer graphics, and the programmers try to avoid

its frequent use. Unfortunately the described material removal simulation process includes context switching in almost every simulation cycle, which extremely slows down the program execution.

Nevertheless, this kind of process execution is still the only applied in CAM systems. The main reason is the rigid hardware architecture and programming limitations of the traditional GPUs that do not allow us to significantly change it. The one-way GPU pipeline works well when a number of polygons provided by the CPU must be displayed, but it's not suitable to execute cyclical computations: it does not ensure the possibility of data feed-back within the GPU, which would be required for the Boolean operations. The duty of creating objects of the virtual space is conventionally belongs to the CPU, while the GPU is only for visualization: that's why it was primarily developed to transform existing graphical elements instead of creating them. There is no way of breaking dexels or cutting BRep triangles into pieces by the GPU in the traditional simulation process but it's also impossible to convert the elements of the solid object representation into displayable polygons with the use of the graphics hardware.

The appearance of graphics processing units with unified hardware architecture has changed this state. In this architecture, as it was mentioned earlier, various shaders run on the same hardware components, and the results computed by them are sent back for further processing until all shader operations are performed and the fragments are passed on to the subsystem of raster operations. Microsoft's DirectX 10 graphical API introduced *Shader Model 4*, which supported this architecture for the first time. Shader Model 4 includes a new type of shader, *geometry shader*, with which new geometry data can be generated by the GPU on-the-fly and general-purpose (non-graphical) calculations can be executed (Figure 30). These developments have made the way of executing the Boolean subtraction and conversion operations on the graphics hardware and gave me the idea of creating a GPU based material removal simulation by exploiting the possibilities of this brand new technology.

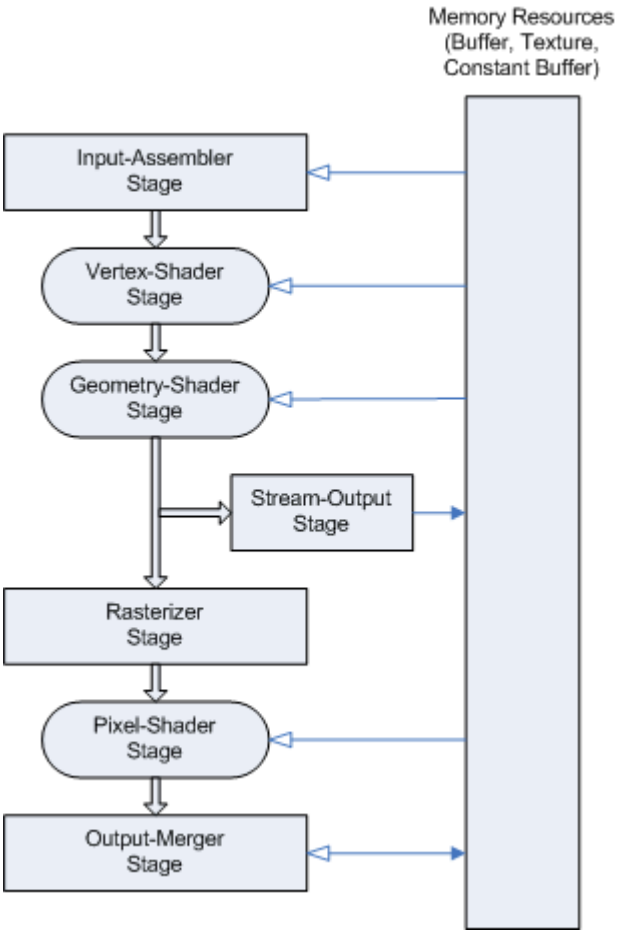

Figure 30. DirectX 10 pipeline [127]

Figure 31 shows the flowchart of the proposed, GPU based simulation method. It can be seen on the diagram that the steps of the process remain the same; the main change is the redistribution of the tasks between the CPU and GPU. The CPU thread has got shorter: the central processing unit is now charged only with the calculations with SISD manner, such as the initialization steps and tool path interpolation besides the managing of the graphical user interface. All the steps with SIMD nature are taken over by the GPU. The advantage of this is obvious: exploiting the high degree of parallelism of the GPU architecture and freeing the CPU from the inappropriate tasks. If we look closer, we can detect a changing in one of the diagram boxes: the character of the swept volume generation step is now marked with SIMD instead of SISD that stated in the flowchart of the traditional process. The reason is that the simulation software realized during the research work includes a newly developed swept volume generation module with parallelized execution on the graphics hardware. Nevertheless, the common swept volume generation methods can also be used since the CPU still possesses enough computational capacity for fulfilling them besides the tool path interpolation task.

The proposed way of simulation offers a solution to the problem of the high data transfer rate between the hardware units as well. Apart from a one-time intensive data transfer in the initialization phase, only the few bytes of the transformation matrices move constantly between the CPU and GPU. In the

case of a possible CPU based swept volume generation the polygons of the swept surface cannot be called a big amount of data to be transferred either – this gives the possibility of choosing the swept volume generation method at will.

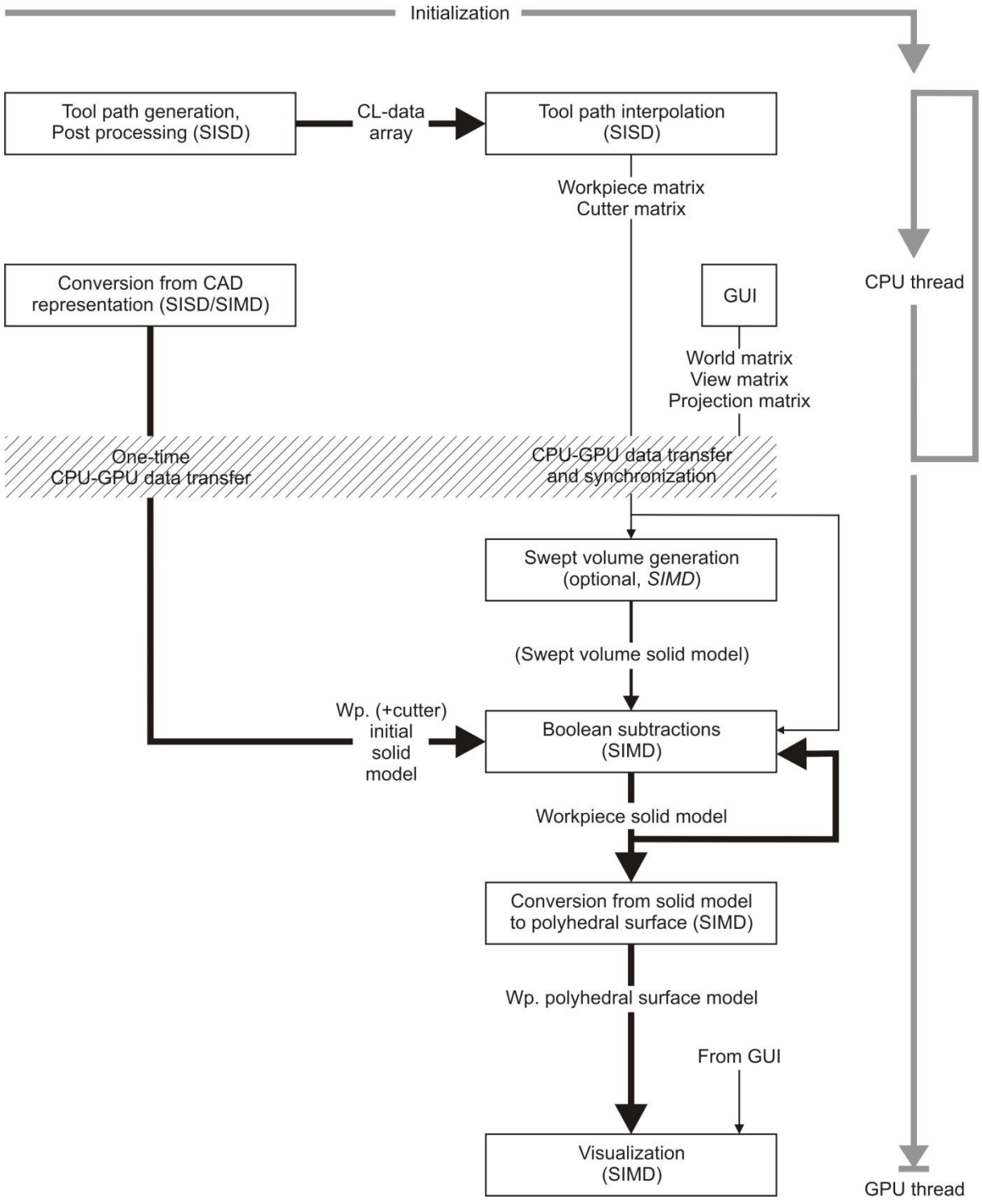

Figure 31. Flowchart of the GPU based material removal simulation

The fact, that general-purpose calculations can be executed on the GPU does not mean that we could directly adapt the conventional virtual machining algorithms to the new architecture. The main reason is just the strict SIMD hardware architecture of the GPGPU. For understanding the problem we must get acquainted more closely with the unified hardware layout. The key to the architecture is the use of numerous scalar *stream processors* (SP) to perform shader operations (Figure 32). Each SP is fully generalized, fully decoupled, scalar, can dual-issue a MAD (multiplication and addition) and a MUL (multiplication) operation, and supports IEEE 754 (single, 32 bit) floating-point precision. The stream processors are grouped into *clusters* (also called *multiprocessors* or *processor cores*); each cluster contains 16 SPs. The SPs in the same cluster share some registers, memory and first level cache in order to fasten their communication, and always process the same instruction but on multiple data (SIMD). In addition, the SPs are supplemented with texture addressing and filtering units (TF) for graphical purposes. The system memory (seen on the figure as a collection of frame buffers, FB) can be reached via the level 2 cache. The data distribution and management is the duty of the thread processor.

This arrangement results a massively parallel design, providing immense processing power for graphical and general-purpose tasks as well, but it allows only a limited interconnection among the program threads running on different parts of the structure. In the context of the GPU programming every thread on the GPU – that means a series of operations to be executed sequentially - consist of the same instructions, but work on different data. Each thread is executed by one SP. The threads are bundled into blocks and all threads of a block are expected to reside on the same multiprocessor. Only the threads in the same block can be synchronized and they can only exchange data with each other. The number of interconnecting threads is therefore restricted by the limited memory resources of a multiprocessor. This prevents attaching adjacency information to the data elements or storing them in such complex structures, like the dexel chains in the case of the extended z-map technique.

The threads reach the global memory in a random order. This means that the concurrent writing onto the same memory location must be avoided, unless we get unpredictable result. To solve this problem, each thread must have a dedicated block in the global memory to write to, with pre-defined location and size. The fixed size does not allow the arbitrary multiplication of data elements in a single computing cycle: we have to say preliminary the number of elements that we want to create from a single one before each calculation. If this number is unknown, the algorithm can hardly be adapted to GPGPUs: though we could decompose it into an unknown number of sequential algorithms, the required data reorganizing steps and the multiple GPU calls would greatly reduce its efficiency. Another problem emerges when the highest possible number of originating elements is big, but the *average* number of them is low, thus resulting long, but in most cases empty memory structures. Making these memory holes disappear needs additional reorganization steps which slow down the simulation. These issues cause that the techniques, where the primitives of the solid object representation can break into numerous pieces – such as the triangles of the BRep surface after a

possible Boolean subtraction –, cannot be taken into account in the case of a GPGPU based material removal simulation. They also make it impossible – or at least extremely complicated and slow – to rearrange independent dexels into common boundary cells, as it's used to do in the marching cubes and dual contouring algorithms. This can be the reason, why the method of Wang et al. [35] is not applied for virtual machining: they did not omit or rethink the use of such traditional steps like the rearranging, which led to performance decrease, notwithstanding they offered a fully GPU based execution of Boolean operations on solids.

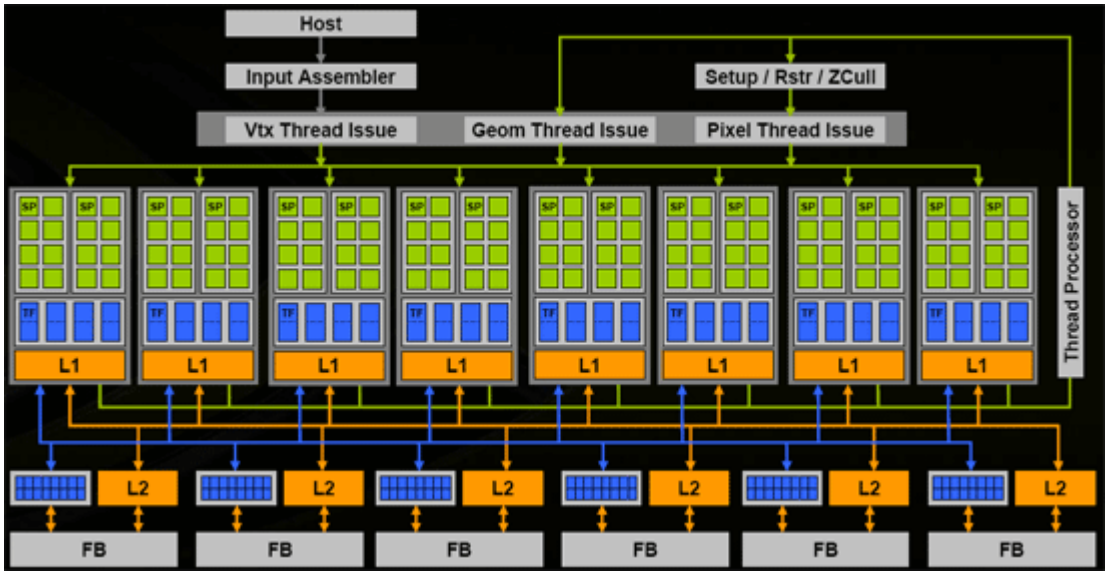

Figure 32. NVIDIA GeForce 8800 GTX GPGPU architecture [128]

Some further questions also arise when choosing the appropriate technique for the GPU based simulation. There are two levels of the general-purpose programming of the GPGPUs. At the first one we use the graphics pipeline as if we programmed a simple graphics task, but with exploiting the non-graphical programmability of the geometry shader. The second means the use of an application programming interface (API), such as CUDA from NVIDIA or OpenCL from the open community, dedicated to compile highly parallel general-purpose algorithms to be executed on a GPGPU or a GPGPU cluster. The advantage of the first one is the use of tried and tested graphics APIs (DirectX or OpenGL) that ensure easy portability and widespread compatibility – at the prize of reduced flexibility and freedom in programming general-purpose tasks. Applying CUDA or OpenCL gives sophisticated tools to create more efficient parallel algorithms with considering the architecture and features of the graphics hardware – but these solutions often require special graphics cards (such as NVIDIA cards for CUDA), complicated installation and the use of non-reliable, just-developed drivers and software components.

During the development of the GPU based simulation several techniques were probed and then thrown out. Due to the abovementioned reasons the BRep based simulation, as well as the other object space

methods with complex data structure, could not be applied. The dexel based method implies the use of the marching cubes algorithm, which is not fully adaptable to the GPGPUs – this would break the GPU thread with a CPU intervention, significantly increasing the dependency of the two hardware threads and causing vast data transfer between the CPU and GPU. Otherwise the dexel based representation could be a good choice, as the graphics hardware can effectively perform Boolean operations on dexels – but only if the dexels were stored independently from each other in the device memory and an appropriate technique were found to visualize the set of independent dexels. There is a common way for performing this task: displaying dexels as numerous parallel right prisms, using the brute force of the graphics hardware. Unfortunately this method did not proved viable: the big number of dexels made the simulation very slow, while the surface areas that are represented only by the sides of the dexels did not possess information about the surface normals, which would be indispensable for the adequate shading.

Choosing the multi-dexel based object representation and simulation seemed a good solution to the problems. The method inherits the good applicability of the dexel based method, while the three orthogonal dexel sets describe the whole surface of the workpiece. The fact that the drawing of only the dexel tips can be enough for displaying the whole object surface offers the possibility of creating a fast technique to display the workpiece, composed by a big number of independent dexels, without the need of data rearranging. Performing Boolean operations on dexels is quite easy and the amount of newly created dexels is predictable: each dexel breaks at most into two new dexels after the subtraction. This allows the simplest and more robust way of GPGPU programming, when the geometry shader deals with the general-purpose tasks. In the followings the details of a multi-dexel based material removal simulation method will be presented, programmed with the DirectX 10 graphics application programming interface. The more complex general-purpose programming technique of GPGPUs, using CUDA for creating clearly non-graphical algorithms to be executed on GPUs, have been applied in the second part of the research work and will be introduced later, together with the proposed cutting force prediction method.

3.2.1 Initial conversion from native CAD format to multi-dexel based description

The conversion step is basically the same as the method applied by Ren et al. [42], introduced in Section 2.1.4, thought meanwhile some problems have arisen which had to be solved. The generation of multi-dexel based description of the blank is made by converting BRep data stored in an STL file. The object vertices and normals are read from the file and loaded into a vertex buffer to be rendered to layered depth-normal images (LDNIs) by depth peeling, a common CG method, in which the object surface is disassembled into layers from a given point of view. The entering and leaving positions of the dexels are computed by transforming the screen space coordinates of the surface points of the front

and back faces to world space coordinates, while the belonging normals are the vectors previously stored in the LDNI pixels. All of the three multi-dexel directions must be taken into account.

While reading the STL file and filling the vertex buffer it must be considered that the CAD programs use right-handed coordinate systems but DirectX use left-handed ones. On that account two of the three coordinates of the vertices has to be exchanged with each other, and the order of the triangle vertices has to be reversed, otherwise the faces would turn to the opposite direction (they would be visible from the back) at rendering:

$$\begin{bmatrix} p'_{1x} & p'_{1y} & p'_{1z} & 1 \\ p'_{2x} & p'_{2y} & p'_{2z} & 1 \\ p'_{3x} & p'_{3y} & p'_{3z} & 1 \end{bmatrix} = \begin{bmatrix} p_{1x} & p_{1z} & p_{1y} & 1 \\ p_{3x} & p_{3z} & p_{3y} & 1 \\ p_{2x} & p_{2z} & p_{2y} & 1 \end{bmatrix} \tag{15}$$

where $p'_{1..3,x..z}$ are the coordinates of the three surface triangle vertices according to the graphics API, and $p_{1..3,x..z}$ mark the vertex coordinates of the original surface triangles.

During the depth peeling algorithm the front and back faces (surface polyhedral) of the workpiece are projected onto images according to the followings. The workpiece object is situated in front of the camera after the camera transformation owning the center coordinates

$$\mathbf{p}_{wpc} = \begin{bmatrix} p_{wpcx} & p_{wpcy} & p_{wpcz} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \dfrac{d_{wp\max}}{2} \end{bmatrix} \tag{16}$$

where the distance of the two farthest object vertices is:

$$\begin{aligned} d_{wp\max} &= \sqrt{\Delta x_{\max}^2 + \Delta y_{\max}^2 + \Delta z_{\max}^2} = \\ &= \sqrt{(x_{wp\max} - x_{wp\min})^2 + (y_{wp\max} - y_{wp\min})^2 + (z_{wp\max} - z_{wp\min})^2} \end{aligned} \tag{17}$$

The situation of the workpiece center in relation to the object vertices is

$$\begin{bmatrix} p_{wpcx} & p_{wpcy} & p_{wpcz} \end{bmatrix} = \begin{bmatrix} \dfrac{x_{wp\max} - x_{wp\min}}{2} & \dfrac{y_{wp\max} - y_{wp\min}}{2} & \dfrac{z_{wp\max} - z_{wp\min}}{2} \end{bmatrix} \tag{18}$$

The workpiece vertices are transformed into the $\begin{bmatrix} -1..1 & -1..1 & 0..1 \end{bmatrix}$ image space volume using the projection matrix

$$\mathbf{T}_p = \begin{bmatrix} \dfrac{2}{x_{wp\max} - x_{wp\min}} & 0 & 0 & 0 \\ 0 & \dfrac{2}{y_{wp\max} - y_{wp\min}} & 0 & 0 \\ 0 & 0 & \dfrac{1}{z_{wp\max} - z_{wp\min}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{19}$$

The surface faces appear on the LDNIs after the *rasterization* process. The triangles are filled and projected orthogonally onto the $\begin{bmatrix} -1..1 & -1..1 & 0 \end{bmatrix}$ 2D camera plane; the view on the plane is then stored in 2D pixel arrays – namely images. In contrast to the common rendering processes, the pixels of the LDNIs hold the normal vector (as three 32bit single precision floating point values) and the depth value (as one single precision floating point value) of the surface point instead of coloring information. The first LDNI contains the front side of the object that is visible from the camera. The second contains the points of the back-looking faces that are the closest to the camera plane (see Figure 14). During the creation of the following LDNIs the same steps have to be fulfilled, but the surface points which have been already rasterized have to be ignored. In this way the object surface can be decomposed into layers. Owing to the common CG algorithm, the process of LDNI creation can be entirely performed by the GPU.

The next step is the dexelization, when we create dexels on the basis of the LDNIs. We lay a grid on the pictures and the pixels beneath the grid points, one after the other, are investigated: if the pixels hide subsequent layers, a dexel is started from the front face up to the next back face. During the rasterization the projected view on the camera plane is automatically stretched onto the LDNIs, characterized by a w_{LDNI} and h_{LDNI} image width and height and a [0..1] depth range stored by the pixels. This means that the inverse transformation, with which the coordinates of the surface points are converted from image space into object space, a little differs from the direct transformation defined by Eq. (19):

$$T_{p'}^{-1} = \begin{bmatrix} \dfrac{x_{wp\max} - x_{wp\min}}{w_{LDNI}} & 0 & 0 & 0 \\ 0 & \dfrac{y_{wp\max} - y_{wp\min}}{h_{LDNI}} & 0 & 0 \\ 0 & 0 & z_{wp\max} - z_{wp\min} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

The result of dexelization is three orthogonal dexel sets stored in vector structures, which are then further converted and loaded into vertex buffers in the graphics device memory for the purpose of GPU based processing. The grid interval determines the resolution of the representation. The best resolution can be reached if each LDNI pixel gets a grid point. It is worth choosing the size of the LDNIs to be the same as the size of the screen used for displaying the workpiece: smaller images would cause reduced visualization quality, while the plus information of bigger images could not be displayed due to the limited screen resolution. However, the image sizes and dexel resolution that we really can use in the simulation depend on the GPU performance and the available device memory – so I will revert to this question later, after presenting the Boolean subtraction algorithm and the exact dexel data structure.

The conversion process detailed above allows the dexelization of a portion of the workpiece instead of the whole object too, with keeping the number of dexels thus increasing the resolution. This means that the portion that we zoom to appears in increased detail, by representing it with the same number of dexels that we used for representing the whole object earlier. For this purpose the x_{wpmin}, x_{wpmax}, y_{wpmin}, y_{wpmax}, z_{wpmin}, z_{wpmax} values in Eqs. (16) - (20) must be exchanged with the limits of the volume that we dexelize.

As sequential vector structures are filled up with dexels during the dexelization step, it cannot be evaluated by the GPU. The LDNIs have to be transferred into the system memory, and the CPU has to perform the conversion, after which the dexels are transferred again, into the GPU memory. Fortunately these steps must be fulfilled only once, at the beginning of the simulation. On the other hand, performance tests have shown that the creation of even a huge number of dexels takes up only tenth of seconds, which is practically unnoticeable. (Exact test results are enumerated later.)

3.2.2 Cutting operations

The original dexel model [14] describes virtual objects by a set of depth elements with the same direction. The model is characterized by a constant grid interval of δ. At the grid location [i,j] relative to the origin point O a set of $\{D_{i,j,k}\}$ dexels are situated in succession, started at increasing k's in the z direction. The multi-dexel model is represented by three independent orthogonal dexel sets along the x, y and z axes. In our case – as opposed to the original model identification – the processing of the dexel data by the GPU shaders does not allow the linked representation of the dexels. Each dexel $D_{i,j,k}$ consists of two dexel nodes ξ_l and ξ_u as the lower and upper end of the segment, represented by an extended vertex structure N in the vertex buffer V with the following properties: start position $\mathbf{p} = [i \quad j \quad k]^T$, length l, direction vector $\mathbf{d} = [d_x \quad d_y \quad d_z]^T$, and the surface normals $\mathbf{n}_l, \mathbf{n}_u$ at the lower and upper end (Figure 33).

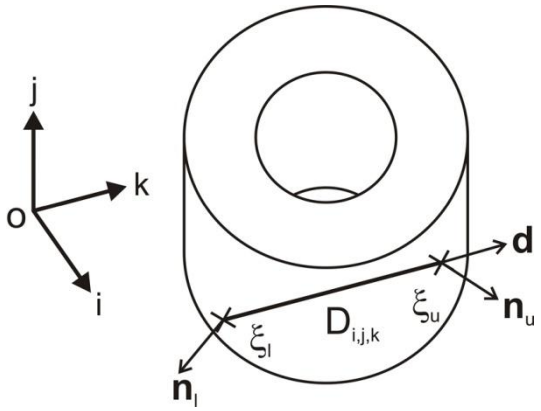

Figure 33. Definition of dexels

In every single simulation step a Boolean subtraction operation is executed between each dexel $D_{i,j,k}$ in vertex buffer V_1 and the tool, resulting one unchanged or shortened dexel $D'_{i,j,k}$ or two shortened, independent dexels $D^1_{i,j,k}$ and $D^2_{i,j,k}$ as an element of vertex buffer V_2 if only the dexel is not discontinued (Figure 34). The uncut ends of the new dexel or dexels inherit the normals of the original one ($\mathbf{n}_l^1 = \mathbf{n}_l$ and $\mathbf{n}_u^2 = \mathbf{n}_u$ on the figure); while the normals of the cut ends (ξ_u^1 and ξ_l^2) become the inverse normals of the tool surface at the surface-dexel intersection points. Vertex buffers V_1 and V_2 are exchanged cyclically as source and target buffers.

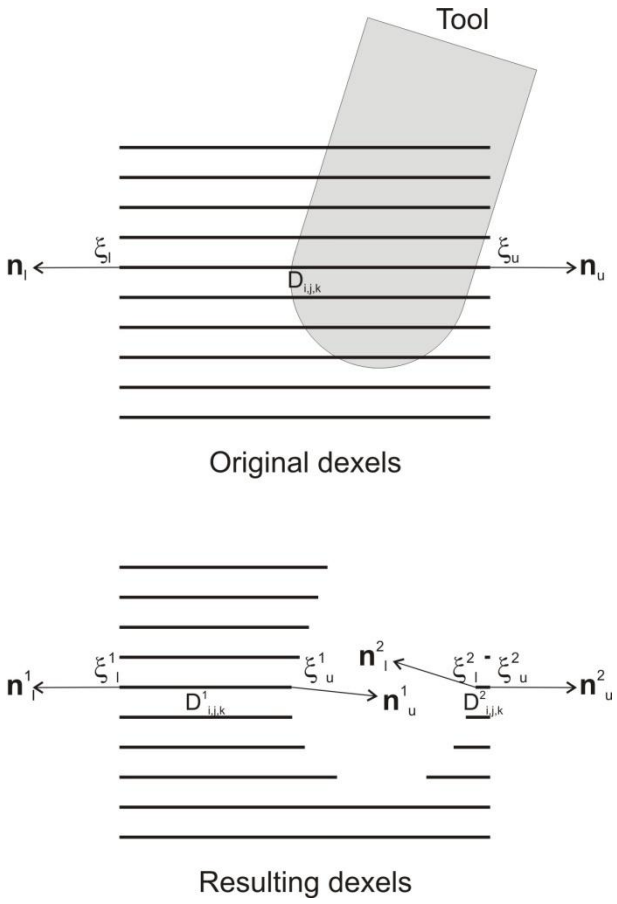

Figure 34. Boolean subtraction

The basis of the operations is the determination of the intersection points between each dexel represented by a spatial line and the tool represented by spatial primitives, like spheres or cylinders. For ball-end cutters the position and direction vectors **p** and **d** are transformed from the workpiece coordinate system into the tool coordinate system:

$$\left[\mathbf{p}_t^T \quad 1\right] = \left[\mathbf{p}^T \quad 1\right] \cdot \mathbf{T}_{wp} \cdot \mathbf{T}_t^{-1} \tag{21}$$

$$\mathbf{d}_t^T = \mathbf{d}^T \cdot \mathbf{R}_{wp} \cdot \mathbf{R}_t^{-1} \tag{22}$$

where $\mathbf{T}_{wp}$ is the homogenous transformation matrix belonging to the workpiece; $\mathbf{T}_t$ is the homogenous transformation matrix belonging to the tool, related to the left-handed world coordinate system defined by the graphics programming interface; while $\mathbf{R}_{wp}$ and $\mathbf{R}_t$ are the rotation submatrices:

$$\mathbf{R}_{wp} = \mathbf{T}_{wp}\left[1,2,3;1,2,3\right] \tag{23}$$

$$\mathbf{R}_{t} = \mathbf{T}_{t}\left[1,2,3;1,2,3\right] \tag{24}$$

For the half-sphere part of the tool, the intersection points $\mathbf{p}_1, \mathbf{p}_2$ are gained by the solution of the following matrix equation (Figure 35).

$$|\mathbf{p}_t - \mathbf{d}_t t|^2 = |\mathbf{p}_{1,2}|^2 = r^2 \tag{25}$$

thus

$$(p_{tx} - d_{tx}t)^2 + (p_{ty} - d_{ty}t)^2 + (p_{tz} - d_{tz}t)^2 = r^2 \tag{26}$$

while for the cylindrical part the *z*-component is simply discarded:

$$(p_{tx} - d_{tx}t)^2 + (p_{ty} - d_{ty}t)^2 = r^2 \tag{27}$$

where *r* is the radius of the tool. For other tool shapes the equations change accordingly.

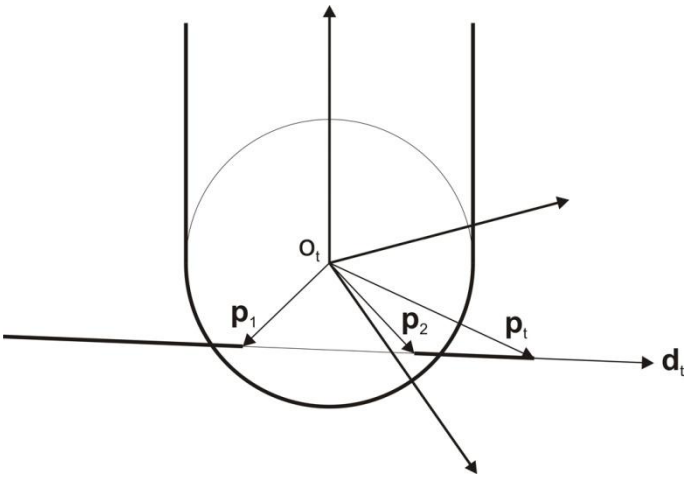

Figure 35. Intersection points

In practice each of the three orthogonal dexel sets are stored in independent vertex buffers. Another vertex buffer has been formed to be filled up with dexels which are newly created during the subtraction operations. The four buffers are cyclically rotated in the following order: $v_1 \rightarrow v_4$; $v_2 \rightarrow v_1$; $v_3 \rightarrow v_2$; $v_4 \rightarrow v_1$; where the arrows mark the dexel transfer during the volume subtraction. In this way some device memory can be saved as opposed to the use of one bigger buffer for all of the

dexels and another for the filling up. According to the earlier, each dexel is composed of the following data:

- Surface normals at the two ends of the dexel, stored as six single precision floating point numbers;
- Three coordinates of one of the dexel tips and the length of the dexel, as four single precision floating point numbers;
- One dexel direction marker in a one-byte integer, which holds color and material information too.

This means 41 bytes in all. Even most of the low-cost GPGPUs possess 512MB device memory, while a 512x512 screen part for displaying the workpiece can be said average in the recent CAM programs. Counting with 41-byte dexel size, if 512x512 LDNIs are applied with maximum dexel density, the four vertex buffers, full with about 262 thousand dexels in each direction, take about 123 Mbytes, which is far less than the available. The possible maximum number of dexels is more than 3 millions, which is much higher than the usual (the published projects, though concrete values were not revealed, used a couple of hundreds or at most thousands of dexels). Of course, the number of dexels that we can use in the simulation will depend on the performance of the GPU based simulation too.

3.2.3 Image space based Boolean subtraction

The above introduced method used analytical surfaces to describe the cutter envelope. For the case when the tool is so complex that it cannot or hardly can be defined analytically, or the swept volume of the cutter has to be involved into the computation, another method has been developed. It starts out from the "watertight" (hole-free) polyhedral description of the cutter envelope or swept volume surface. The basic conception is using LDNIs for describing the subtrahend volume and applying these LDNIs directly at the Boolean subtraction. The process is the following:

1. The cutter object is created in a CAD program with some of the available surface or solid modeling techniques and converted into polyhedral surface model (STL file). If swept volume is used, its polyhedral surface can be created by the common swept volume generation methods in each simulation step.
2. In every simulation step the cutter or swept volume surface is placed to its actual spatial position and rendered into LDNIs with exactly the same depth peeling method and projection parameters that we used for the dexelization.
3. During the dexelization we do not convert the dexel coordinates into object space (with Eq. (20)), instead we leave them in image space, thus they can be directly compared with the cutter or swept volume coordinates stored in the LDNIs. In this case the dexels (which are perpendicular to the LDNI planes) are represented by an (x,y) screen coordinate couple and two depth values that mark the distance of the dexel tips from the image plain.

4. We remove the cutter or swept volume from the dexels layer by layer, with comparing the depth values of the dexel tips with the depth values of the front and back faces of the current layer at the same (x,y) screen coordinates.
5. Step 4 has to be repeated for each layer.
6. Step 2-5 has to be repeated for each of the three dexel directions.

The image space based volume subtraction, similarly to the analytical method, is executed solely by the GPGPU; the depth peeling algorithm is a common CG method, and the LDNIs as image resources can be processed in a parallelized manner by the geometry shader. Speed comparison tests between the two methods will be presented in a later section.

3.2.4 GPU based swept volume generation

The developed simulation method includes a swept volume generation function as well. Its advantage is – as compared with the existing techniques – that it is completely GPU based and does not involve the CPU into the computations. On the other hand, the technique introduced below does not provide a general solution to the problem of swept volume generation – it can be applied just for the above described image space based volume subtraction. It cannot create the swept volume along complex multi-axis toolpaths yet – it is still an unsolved problem – but gives a good result for the linearized movement of the tool between the cutter location points.

It could be seen in Section 2.1.5 that the biggest difficulty in swept volume generation is creating an error-free (manifold) swept surface analytically. However, creating the LDNI description of the swept volume is not so difficult. Figure 36 shows, from several points of view, the swept surface of a ball-end mill during the linearized movement between two cutter location points. We can recognize an important characteristic of the surface on the figure: there is not any surface point, which is not visible from at least one of the three orthogonal points of view, from the front or the back (the back sides cannot be seen). This means that these six depth images (three from the front and three from the back) hold full geometrical information about the surface. The figure also suggests that we cannot find any hidden surface parts – namely such surface parts that look towards us but they are hidden from our view – so we can use these images at the Boolean subtraction instead of LDNIs. That's not true: the swept surfaces generally possess concave sides that can contain such areas from a certain point of view (see the yellow area on the figure). That's why we need layered depth images instead of simple (single layered) depth images.

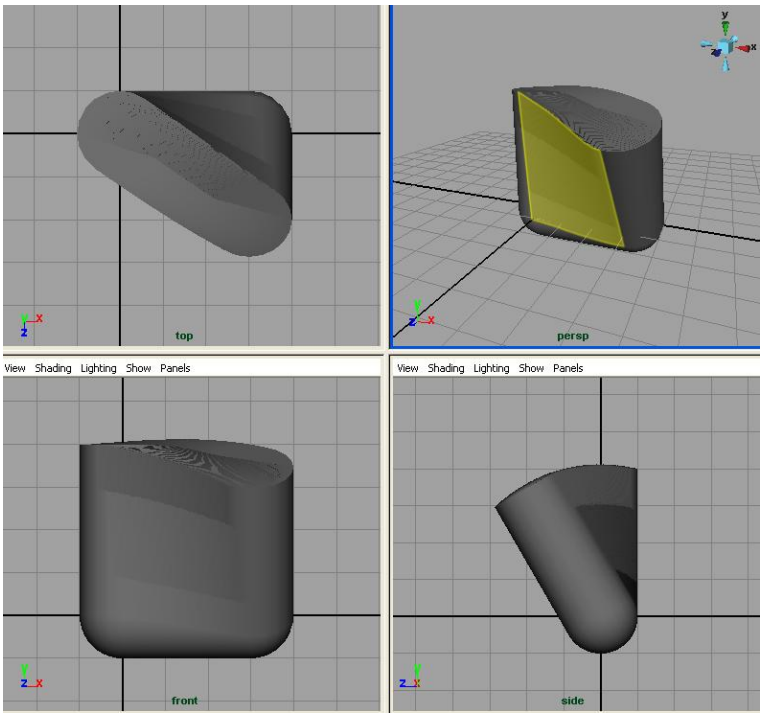

Figure 36. Swept volume of a moving ball-end mill

Another fact, which can be exploited at the image space based swept volume generation, is that the creation of the six images that describe the surface requires hole-free but not necessarily manifold polyhedral surface. This gives us more freedom while generating the swept surface. Let's see Figure 37, which shows three consecutive spatial positions of a simplified cutter. In each position the edges that form the swept profiles – marked with thick red and green lines – can be selected according to the theory of Wang et al. [48] (Section 2.1.5) in the following way. The common edge of two neighboring polygons, characterized by a common vertex with position $\mathbf{p}_i$ in the ith step and $\mathbf{p}_{i+1}$ in the (i+1)th step and normal vectors $\mathbf{n}_i^1$ and $\mathbf{n}_i^2$, is the part of the swept profile in the ith position, if one of the normals tends towards the moving of the common point and the other tends to the opposite direction:

$$\operatorname{sgn}(\mathbf{n}_i^1 \cdot (\mathbf{p}_{i+1} - \mathbf{p}_i)) = -\operatorname{sgn}(\mathbf{n}_i^2 \cdot (\mathbf{p}_{i+1} - \mathbf{p}_i)) \tag{28}$$

or one of the normals is perpendicular to the movement vector and the other tends towards the opposite direction:

$$\begin{aligned} &((\mathbf{n}_i^1 \cdot (\mathbf{p}_{i+1} - \mathbf{p}_i) = 0) \wedge (\operatorname{sgn}(\mathbf{n}_i^2 \cdot (\mathbf{p}_{i+1} - \mathbf{p}_i)) = -1)) \vee \\ &((\mathbf{n}_i^2 \cdot (\mathbf{p}_{i+1} - \mathbf{p}_i) = 0) \wedge (\operatorname{sgn}(\mathbf{n}_i^1 \cdot (\mathbf{p}_{i+1} - \mathbf{p}_i)) = -1)) \end{aligned} \tag{29}$$

By creating tetragonal polygons with the use of the same swept profile edges in the sequential positions as the opposite edges of the tetragon (marked with red and green filling on the figure), the swept volume between the two positions can be well approximated. The final shape of the swept surface consists of these polygons (marked with gray on the figure) considering every position that take part in the computation; together with the original faces of the cutter object that close the surface,

e.g. in the starting and end positions (the remaining, not filled polygons on the last figure which can be selected similarly to the above).

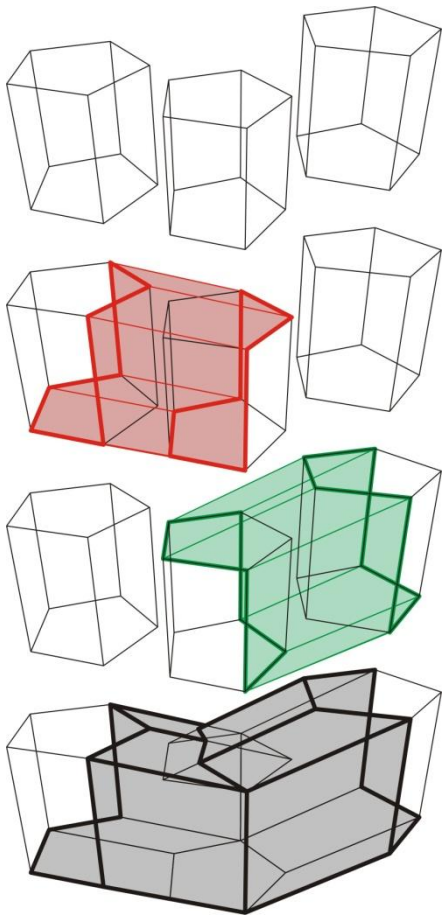

Figure 37. Swept envelope approximation

The union of the polygons of the sequential swept surface components (the union of the red and green polygons on the figure) does not form a manifold surface. For the creation of a manifold swept volume (contoured with thick lines on the last figure) we would have to investigate the intersections of the polygons and truncate them along the intersection lines. That would mean a very complicated task with involving the CPU that prevents its effective application in practice. Instead a simpler method has been developed which can be executed by the GPU in a highly parallelized way by exploiting the characteristics of the six orthogonal depth images that can be used for describing the whole swept volume surface. The process steps are the followings:

1. After the creation of the swept volume between two sequential cutter positions, it is rendered to all of the six orthogonal depth images by the common z-map technique. The image plains are characterized by three homogeneous transformation matrices $\mathbf{T}_{c1},\mathbf{T}_{c1},\mathbf{T}_{c3}$. Onto three of the images the front faces are rendered, onto the others the back faces.

2. Step 1 is repeated at every cutter position which is included into the computation, by using the same depth images to render the swept volume components onto. After the process the images represent the outer skin of the final swept volume.
3. The swept volume polygons are rendered again, now onto LDNIs, but only the surface points are considered that lay on the outer skin. This can be performed, similar to the *shadow mapping* CG technique, by transforming the point coordinates into the image space of the six images (Eq. (30)) and checking if any of the images contains a skin point on the same coordinates or not. If the answer is negative, the point must be dropped, otherwise it can take part in the creation of LDNIs.

$$\left[\mathbf{p}_{c1..3} \quad 1\right] = \left[\mathbf{p}_{LDNI} \quad 1\right] \cdot (\mathbf{T}_{LDNI})^{-1} \cdot \mathbf{T}_{c1..3} \tag{30}$$

4. The subtraction of the cutter swept volume from the workpiece volume is performed by using the received LDNIs as inputs of the image space based Boolean subtraction.

The introduced method, as compared to the image space based subtraction without swept volume generation, increases the number of rendering calls and contains additional transformations but decreases the number of operations on dexels. After all, with the use of it, the improvement of the resulted surface quality and the moderate growing of the simulation speed can be achieved.

3.2.5 Visualization

Since the marching cubes algorithm cannot be applied in the GPGPU environment and the brute-force dexel displaying did not provide acceptable image quality and speed, a new visualization method had to be created to convert the data structure, composed by three-directional independent dexels, to a displayable polyhedral surface. The method exploits the fact that the dexel tips hold all the important information that is needed for displaying the whole surface from any point of view.

Figure 38 helps understanding the visualization process, showing only two dexel directions for reasons of clarity. The intersections of the dexels perpendicular to each other point out a set of $\{\varepsilon_{l,m,n}\}$ grid points at the locations [*l*,*m*,*n*] relative to the O_{wp} origin of the workpiece coordinate system. Furthermore each dexel node $\xi_{l,m,n}$ points out two neighboring grid points: $\varepsilon_{l1,m1,n1}$ and $\varepsilon_{l2,m2,n2}$, to which $l1 \leq l < l2$, $m1 \leq m < m2$ and $n1 \leq n < n2$. If one $\rho_{l',m',n'}$ square face is assigned to each $\xi_{l,m,n}$ dexel node with $l' = (l1 + l2)/2$, $m' = (m1 + m2)/2$ and $n' = (n1 + n2)/2$ midpoint coordinates, $\mathbf{n}_\rho = \overrightarrow{\xi^l \xi^u} / \left\| \overrightarrow{\xi^l \xi^u} \right\|$ normal if it belongs to the ξ^u upper dexel node or $\mathbf{n}_\rho = \overrightarrow{\xi^u \xi^l} / \left\| \overrightarrow{\xi^l \xi^u} \right\|$ if it belongs to the ξ^l lower dexel node, one additional normal $\mathbf{n}_\rho^{sh} = \mathbf{n}_l$ or $\mathbf{n}_\rho^{sh} = \mathbf{n}_u$ accordingly for shading purposes, and a side length equal with the grid interval δ, the set of $\{\rho_{l',m',n'}\}$ square faces figures out the watertight polyhedral surface model of the workpiece. For visualizing the

model from a certain point of view given by the camera transformation matrix $\mathbf{T}_c$ relative to the world coordinate system, the ρ square faces with $n_z^c > 0$ and $n_z^{sh,c} > 0$ are displayed, where

$$\left[\left(\mathbf{n}_\rho^c\right)^T \quad 1\right] = \left[n_x^c \quad n_y^c \quad n_z^c \quad 1\right] = \left[\mathbf{n}_\rho^T \quad 1\right] \cdot \mathbf{T}_{wp} \cdot \mathbf{T}_c^{-1} \tag{31}$$

and

$$\left[\left(\mathbf{n}_\rho^{sh,c}\right)^T \quad 1\right] = \left[n_x^{sh,c} \quad n_y^{sh,c} \quad n_z^{sh,c} \quad 1\right] = \left[\left(\mathbf{n}_\rho^{sh}\right)^T \quad 1\right] \cdot \mathbf{T}_{wp} \cdot \mathbf{T}_c^{-1} \tag{32}$$

In other words, for the purpose of fast rendering, only those faces are created and displayed, whose normal vector together with the normal vector of the belonging dexel-tip point towards the camera. It should be mentioned that this method is not concerned about the data inconsistency among the three independent dexel sets, which results gaps on the surface, but as the image quality tests show, these errors do not depreciate noticeable the level of visualization (see next Section).

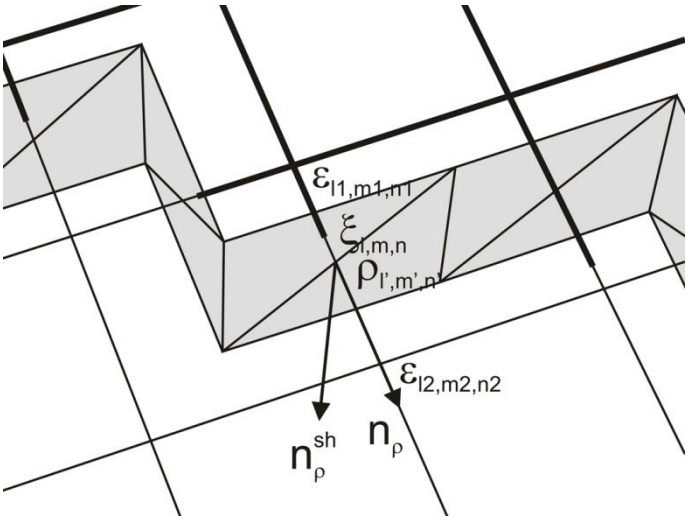

Figure 38. Conversion to polyhedral surface model

The process is managed by the geometry shader that converts the point stream of dexel objects into a displayable triangle stream, assigning four triangles, as the constituents of the two square faces, to the two dexel tips. At last the GPU displays the polyhedral object according to the predefined light conditions. The surface is shaded by the pixel shader with the use of any conventional shading techniques, such as flat or Phong shadings, etc.

3.2.6 Model verification

3.2.6.1 Visualization quality and performance tests

For demonstrating the visual level of the developed method, several material removal simulations have been made. The simulations are performed by a GeForce 8800 GTS graphics card with 640MB

memory, appearing in 2008 on the market as the flagship of the first generation DirectX 10 compliant graphics processing units of NVIDIA. The first presentation demonstrates a five-axis material removal simulation: the rough milling of an impeller, with the use of a ball-end tool. Figure 39 shows the workpiece at the beginning of the manufacturing. The model consists of about 700 000 dexels. The geometry shader performed analytical cutting computations, considering the cutter as a combination of a sphere and a cylinder. The final form of the workpiece, which can be seen on Figure 40, took shape after about 210 000 steps. By the end of the simulation, the number of dexels became about 1 million. This means less than 1.5-time growth in size, and the same growth in memory demand.

The number of dexels determines the speed of the simulation as well. For testing the performance of the method a series of tests was fulfilled. During the tests the duration of the computations and the displaying was measured in each simulation cycle, as a function of the number of dexels. As it can be seen on Figure 41 the functions are nearly linear, resulting about 0.025-0.03s displaying duration and 0,01s computing duration per cycle per one million dexels (Figure 42). The results allow real-time simulation of the manufacturing and smooth visualization during the whole process.

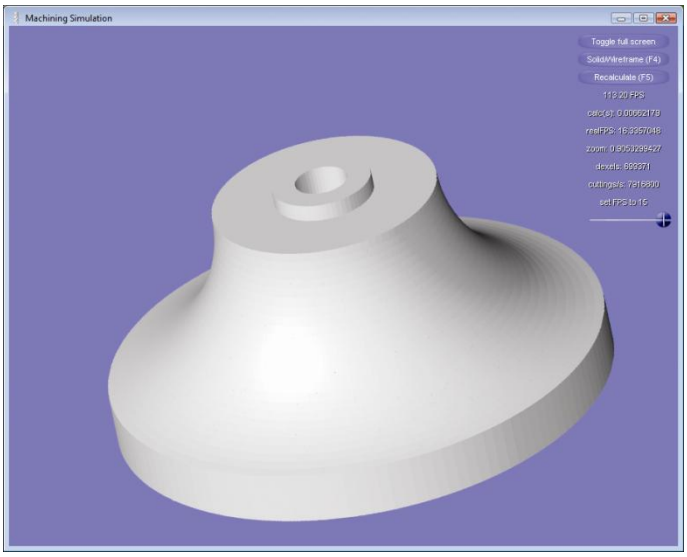

Figure 39. The blank workpiece of the impeller

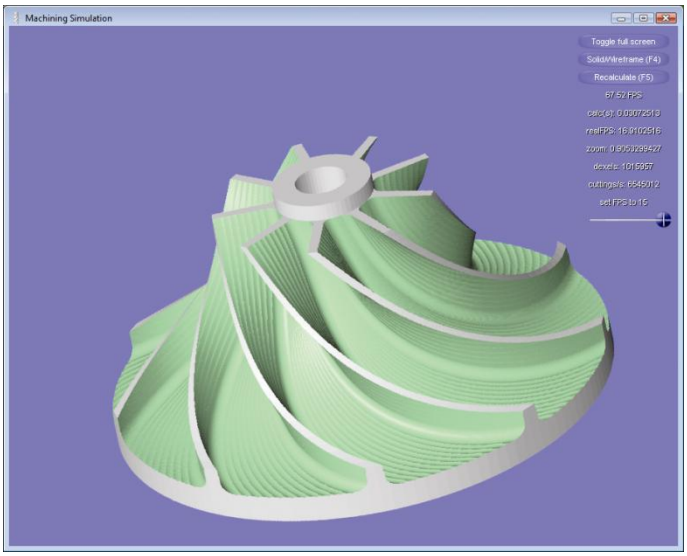

Figure 40. The final form of the impeller

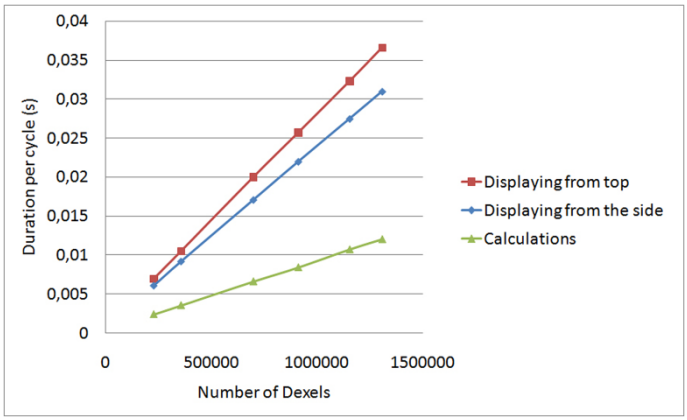

Figure 41. Duration per cycle of calculations and displaying

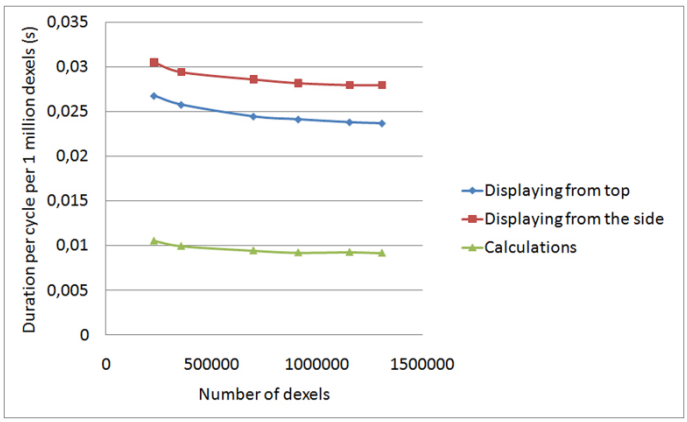

Figure 42. Duration per cycle per 1 million dexels

The GPU-supported data conversion allows fast rebuilding of a smaller portion of the work piece with the resolution of the whole original object. With the recalculation of the simulation steps relating to this portion, unlimited zooming feature can be realized without the deterioration of quality. Figure 43 shows an intermediate state of the simulation with 50-time zooming. The number of dexels is more than 1 million. The rebuilding needed 0.65s. It is to be noted, that the cutter (right above) is displayed with polygons, but it doesn't have an effect on the quality, since the calculations are analytical.

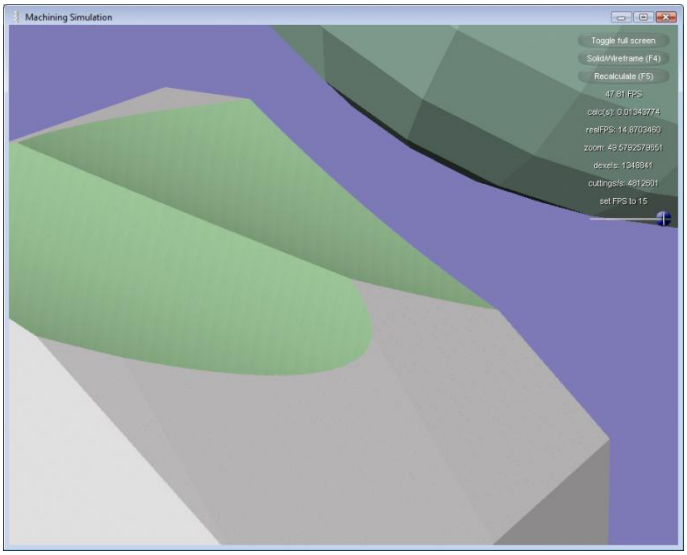

Figure 43. Fifty-time zooming of the cutting process

During the following simulations, the geometry shader executes image space based volume subtraction operations. This makes it possible to perform material removal simulation even with complex cutters or swept volumes. For testing the potentials of the method, a very complex cutter was chosen for visualizing a gear cutting process (Figure 44). As a matter of fact, the simulation would not require such a complex tool model; a cutter with a simplified surface envelope would be also enough – it serves only performance testing and demonstrating purposes.

During the process each dexel is compared directly with the LDNIs of the tool in every simulation cycles. The images are created by depth peeling method. The test results show that the speed of the calculations is mainly determined by the speed of the depth peeling: it hardly depends on the tool complexity (not measurable difference between tools with 100 and 500 polygons), but it depends rather on the size of the LDNIs (see Figure 45). Doubling the number of LDNI layers causes the doubling of calculation time. The relation between the calculation time and the number of dexels is shown in Figure 46. With about 240 000 dexels of the blank work piece, the calculation cycle time remained within 0.02 seconds, resulting convenient real-time simulation.

On the basis of the diagrams it can be calculated that 0.005-0.0075 second is required for the cutting calculations depending on the size of LDNIs, in the case of 700 thousand dexels and one LDNI layer. This means that the speed of the analytical and the image space based methods is about the same.

0.01 second per one million dexels for cutting calculations and 0.03 second for displaying allows examining 100 cutter positions per second without displaying the results or 60 cutter positions with 10 FPS (frames per second) displaying. These results guarantee real-time simulation of manufacturing processes with very high resolution and fluent visualization, even by using medium-performance graphics hardware.

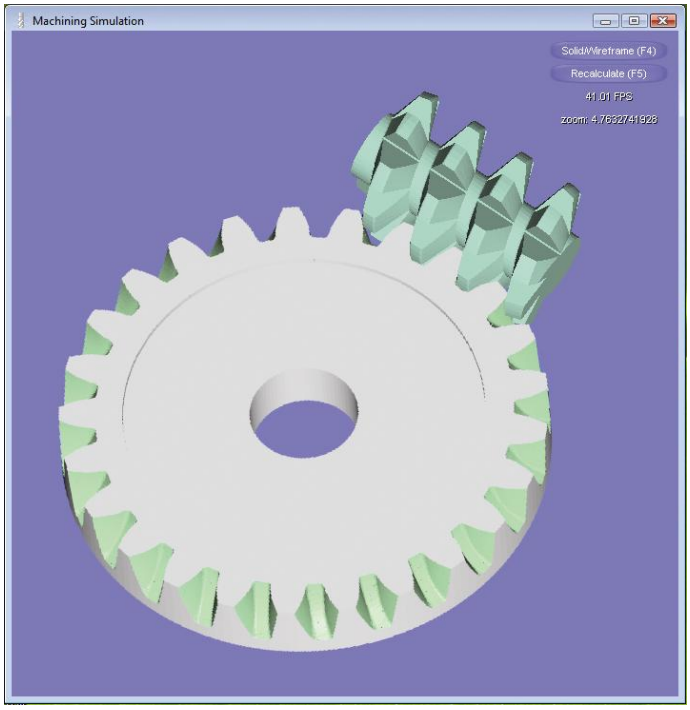

Figure 44. Gear cutting simulation with complex tool

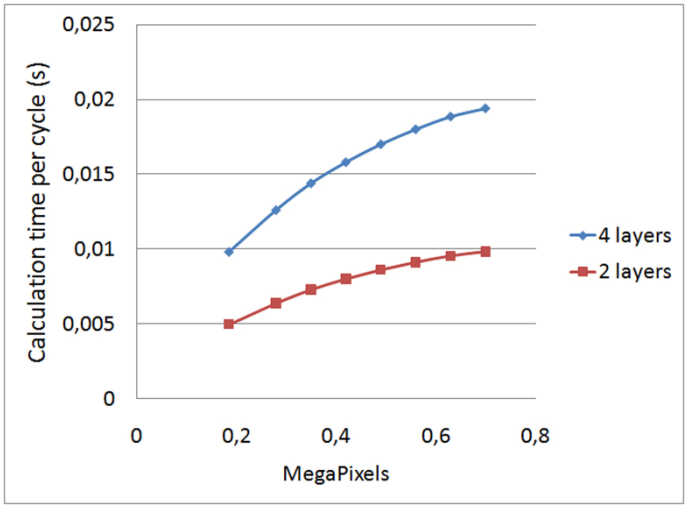

Figure 45. Calculation time per cycle against the size of LDNIs, in case of 242016 dexels

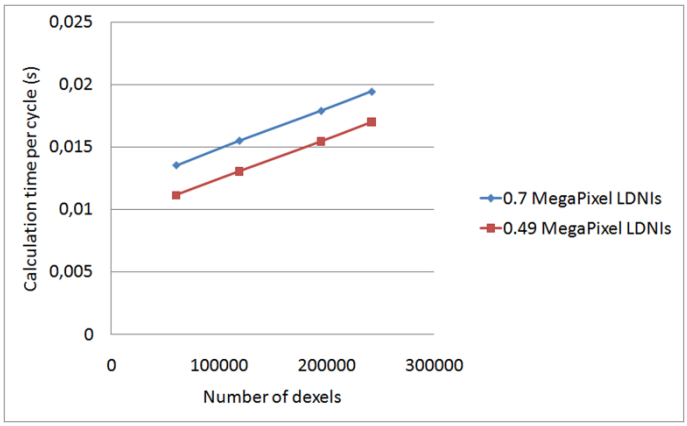

Figure 46. Relation between the calculation time and the number of dexels (4 layers used)

Figure 47 shows the simulation of an electrical discharge machining (EDM) process. The electrode is a free-form polygonal object and it approaches the workpiece from four several directions. The figure shows the process in progress, with slowly moving electrode, but as the Boolean subtraction of the workpiece and the electrode could be in one computing cycle executed, the complete result of the EDM process could be in 0.1 second produced. Figure 48 shows an extreme 200-time zooming of the simulation. The area between the eye and the upper part of the nose can be seen on the figure.

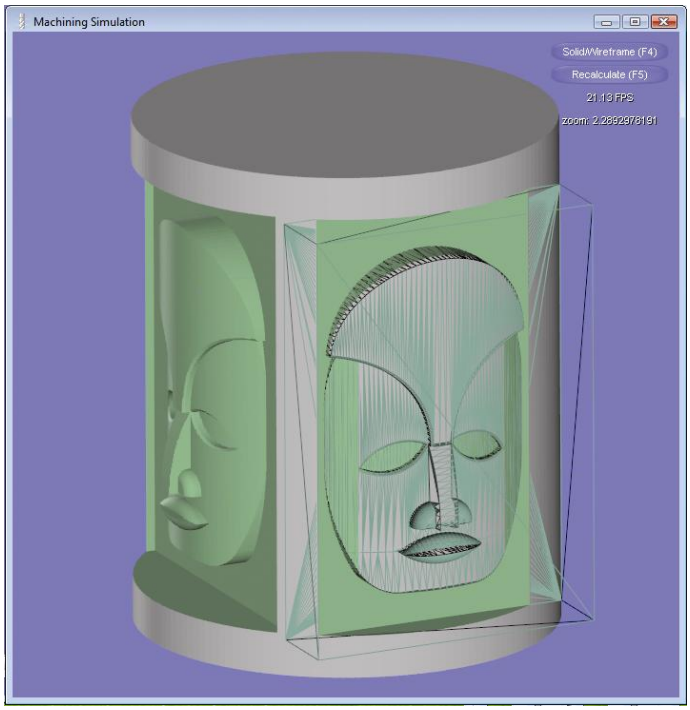

Figure 47. EDM simulation

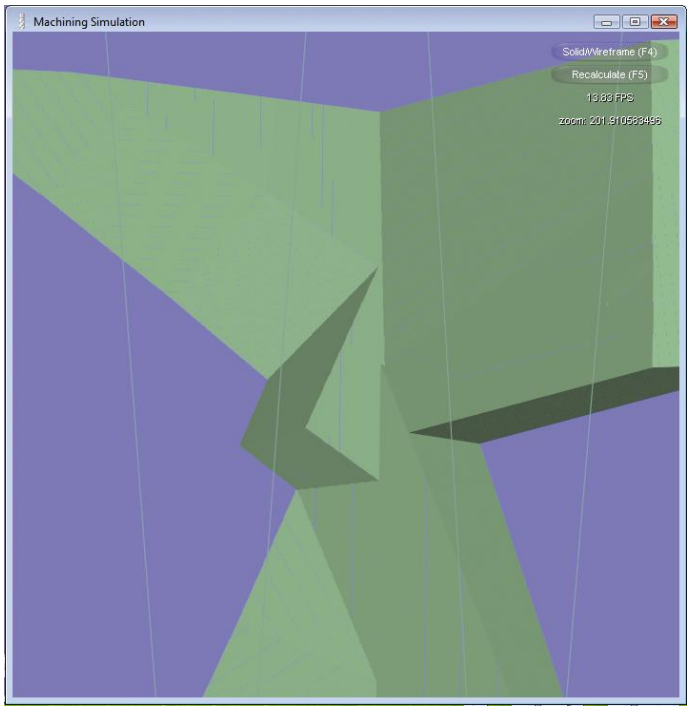

Figure 48. Two hundred-time zooming of the EDM process

3.2.6.2 Comparison with existing techniques

In Section 2.1 numerous virtual machining techniques were enumerated. The z-map method proved to be the most successful technique ever, but we could see that the original image space method is not suitable for the simulation of multi-axis machining. The extended z-map method uses dexels, the USD technique a set of cubes of the same size, while the HSD based solutions apply cubes of varying sizes to describe object geometry. Some proposed constructive solid geometry (CSG) for simulating milling processes. Boundary representation (BRep) is the dominant modeling technique in CAD systems; in the case of virtual machining its simplified version, the triangle-based boundary representation is preferred. However, it can be said of the multi-axis simulation methods that they cannot even approach the performance of the image space technique due to their limited support by the graphics hardware. Nehéz demonstrated [11] that the GPU based z-map method, even with the use of the cheapest graphics cards, is at least 10 times faster than the extended z-map method executed solely by the CPU. The same investigation shows that the HSD method is slightly slower than the latter because of its more complicated algorithm. The high computational expense of CSG prevents its use in the CAM systems. The speed of the BRep technique can excel the speed of the dexel based method in the course of short simulations, although its $O(n^{1.5})$ growth rate greatly slackens it after a few thousand steps. After all, though many of the listed methods can be found in the literature as the basis of recent virtual machining projects (see the solution of Nakamoto et al. [129], which uses voxel structure, or the HSD (Octree) based simulation system of Karunakaran et al. [23]), only the variants of dexel based and BRep techniques proved viable in the commercial CAM applications.

With taking these facts into consideration, the newly developed GPU based simulation method has been compared with two existing techniques. The first is the BRep based material removal simulation method as the most widespread in CAD/CAM systems; the second is the multi-dexel based method with traditional CPU execution. The z-map and simple dexel methods were not examined due to their limited applicability; as well as the other simulation methods, which show poor performance in practice. For testing purposes I set up a simple end-milling process simulation with zigzag tool path, and programmed exactly the same process on my own in three different ways. For the BRep simulation I coded the latest, R21 version of ACIS/HOOPS commercial application programming interface (API); for executing the multi-dexel based method by the CPU I simply altered the GPU based program. Two typical computer configurations were tested: a mid-cost laptop with entry-level GPU, and a high-performance workstation with top-level GPU. The main configuration parameters are listed in Table 1.

Table 1. Test configuration parameters

	Laptop	Workstation
Processor	Intel Core2 Duo T5800 2GHz	Intel i5 3.2GHz
Processor cores	2	4
System memory	3 GB	8 GB
Operation system	Windows Vista 32bit	Windows 7 64bit
Graphics hardware	NVIDIA GeForce 9600M GT 512MB	NVIDIA GeForce GTX480 1536MB
GPU processor cores	32	480

Figure 49 and Figure 50 show the results of the speed comparison test between the proposed method and the ACIS/HOOPS solution. As the displaying of the simulation process is not needed for the volume subtraction computations, the time consuming of the Boolean operations and the visualization were examined separately. Apart from the fact that the performance of the GPU surpasses the performance of the CPU, the most important characteristic of the multi-dexel based method is the steadily growing curve as opposed to the increasing rise in the case of the BRep representation: the constant cycle time makes the real-time cutting force computing possible for long tool paths too.

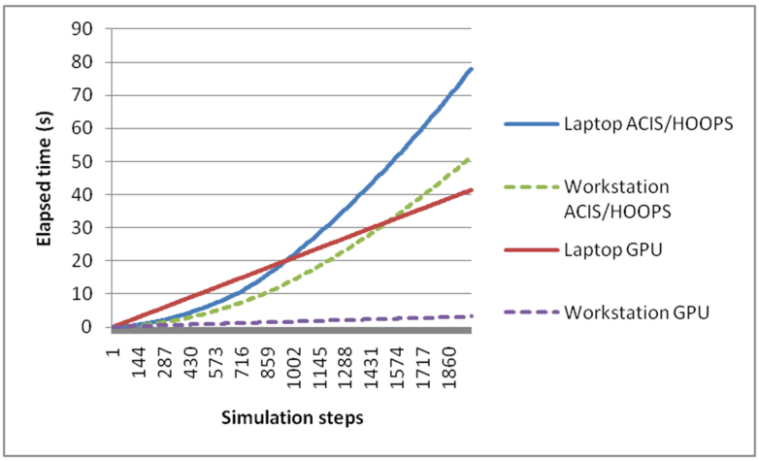

Figure 49. Speed comparison test – Boolean operations

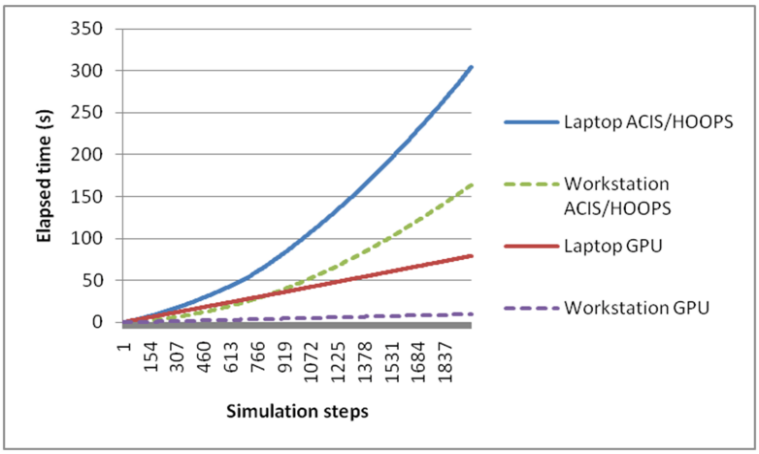

Figure 50. Speed comparison test – Boolean operations plus visualization

Figure 51 shows the speed ratio between the GPU and CPU execution of the multi-dexel based calculations. The highly parallelized execution results remarkable growing of computing performance. The factual cycle times are enumerated in Table 2. The initial number of dexels is 699371, which is far greater than the usually applied hundreds of depth elements; the values indicate very high resolution and real-time execution of the simulation process.

Table 2. Multi-dexel based method with CPU vs. GPU

	Laptop		Workstation	
	CPU	GPU	CPU	GPU
Simulation cycle time (s)	1,365	0,0314	0,3083	0,0034159

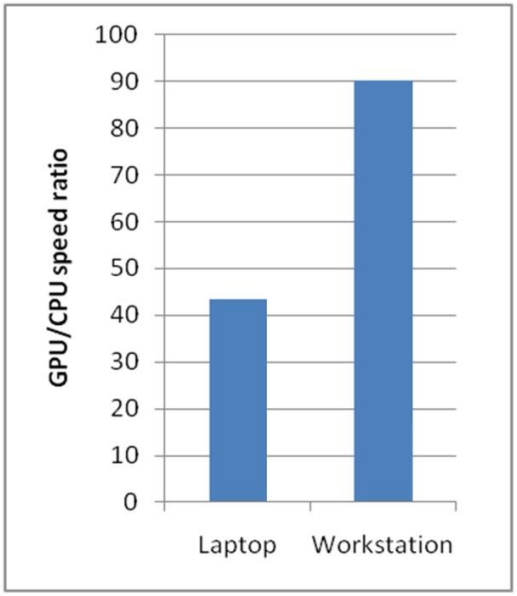

Figure 51. Speed comparison test – multi-dexel based simulation

3.2.7 Conclusions

In this part of the dissertation the adaptation of multi-dexel based material removal simulation to GPGPUs has been introduced. A complete process has been developed, where all the simulation tasks – besides the initial data creation, which is shared between the CPU and GPU – are maintained by the graphics hardware, including data representation and manipulation, together with the visualization of the involved objects. After the convenient volumetric method had been chosen, its data structure was adapted to be stored in vertex buffers and to be processed by GPU shaders, for avoiding CPU intervention. Two kinds of cutting calculation methods have been applied: the traditional analytical algorithm and an image space based one, both adapted in accordance with the requirements of the GPGPU based parallelized execution. For serving the latter a GPU based swept volume generation method has also been proposed. At last a fast and robust visualization technique has been worked out considering the special data structure and the high dexel resolution.

The fulfilled tests demonstrated that the highly parallelized execution of the Boolean calculations and data conversion steps, together with the elimination of the CPU-GPU data transfer bottleneck, lead to considerable speed and visual quality improvement. By adapting the multi-dexel based method to the graphics hardware in its entirely, it could be managed for the first time that a multi-axis simulation technique reached the performance of the 3-axis simulation methods. The fast data conversion and the massively parallelized process of Boolean operations have made an effective recalculation and zooming procedure possible, improving the simulation quality. Using LDNIs not only at the initial data conversion but for representing the cutter volume at the Boolean subtraction operations allows involving complex cutting tools and polygonal swept volumes into the simulation process.

The results show that the use of the new technology brings new possibilities into the field of machining simulations, especially if we have a look at the enormous growing of the graphics hardware segment. As GPGPUs become common parts of the personal computers, software technologies have to be changed as well. The effectiveness of the new hardware architecture in massively parallel computations cannot be disregarded any more in the development of engineering software systems.

3.3 GPGPU based cutting force prediction

3.3.1 The mechanistic cutting force model in parallel computing environment

As it was introduced in Section 2.3.1, the cutting edge geometry is described as a sequence of linear segments in the mechanistic force model. The tangential, radial and axial force components are calculated at each segment by the traditional force equations of oblique cutting. For getting the total instantaneous cutting force along one entire edge, the force components at the active segments, which are taking part in the cutting at the moment, are summed up. The active segments are selected by the determination of the contact area between the cutter envelope and the workpiece in every simulation step. For this purpose the contact area, together with the segments, is projected onto a plane which is perpendicular to the z-axis of the cutter and the inspection is carried out in the image space: the segments outside the contact area are disregarded. We could also see in the technology review that this way of contact area determination cannot be applied in many cases. In the followings such a method of cutting force prediction will be introduced, which provides a general solution to the contact area problem and assures fast calculations by parallelized program execution and direct accessing to the geometrical data gained in the course of the multi-dexel based material removal simulation. While detailing the calculations the use of a ball-end mill will be assumed, since the method has been programmed and verified in the same way during the research work. The mode of its adaptation to general end mills will be mentioned in the relating sections.

When the workpiece object is described with multi-dexel structure, the path of the cutter is followed by cut dexel tips. Each newly cut dexel tip marks out a little patch of the contact area at the intersection point of the dexel and the cutter envelope. In the case of high dexel density the patches can be considered as planes, thus we can approximate the surface area of the patches by the following formula:

$$A = \left| \frac{1}{\mathbf{n}_{ce} \cdot \mathbf{d}} \right| \cdot \delta^2 \tag{33}$$

where $\mathbf{n}_{ce}$ is the normal vector of the cutter envelope at the intersection point, $\mathbf{d}$ is the direction vector of the dexel and δ is the dexel grid interval, according to the notations of Section 3.2.2 and Figure 52.

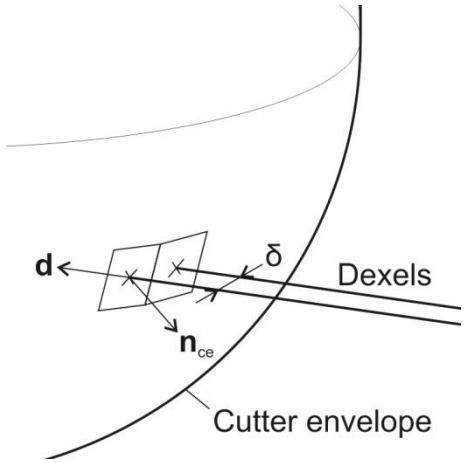

Figure 52. Patches of the contact area

The whole of the contact area is shaped by all the newly cut dexel tips at the current simulation step. However we must keep in mind that we have three independent dexel sets that fully describe the workpiece volume independently from each other, and this causes that three overlapped surface parts come to life during the contact area creation. On the other hand, these surface parts do not provide three independent, complete description of the contact area, as there can be surface areas which are "invisible" from a given point of view but they can be seen from the other two directions. The regions whose surface normal vector is perpendicular to the dexel lines cannot be represented by unidirectional dexels; while the regions with normals *nearly* perpendicular to the dexel lines cannot be described but with stretched, distorted patches with low accuracy. For the purpose of getting the most exact contact area we must select the set of dexels from the three independent sets, which represent the contact area with the best accuracy and without redundancy. It can be done by keeping only the patches where the straight of the direction vector of the intersecting dexel encloses the smallest angle with the straight of the surface normal of the cutter envelope as compared to the other two perpendicular dexel direction vectors. In the workpiece coordinate system, where the three dexel direction vectors point toward the coordinate axes: $\mathbf{p}_1 = [1\ 0\ 0]$, $\mathbf{p}_2 = [0\ 1\ 0]$ and $\mathbf{p}_3 = [0\ 0\ 1]$, the surface normal component with the biggest absolute value, $|n_{cex}|$, $|n_{cey}|$ or $|n_{cez}|$ from $\mathbf{n}_{ce} = [n_{cex} \quad n_{cey} \quad n_{cez}]$ points out the requested direction. Figure 53 shows the contact area between a ball-end mill and a prismatic workpiece during slot milling. The red, green and blue parts distinguish the contact area regions where different dexel directions were taken into consideration.

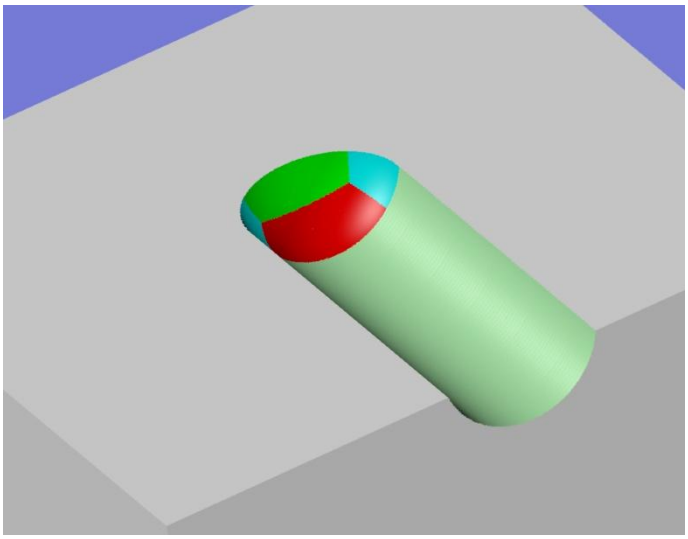

Figure 53. Various contact area components

Thus the contact area is composed by a set of surface patches, which are pointed by the intersections of the selected workpiece dexels and the cutter envelope. The contact area is the characteristic of the cutting process in a certain moment, but as the simulation is discrete in time, the contact area is considered to be valid for the whole duration of the simulation step. The spindle rotation angle and the position of the tool are known in every moment of the cycle period, thus for every intersection point a spindle rotation angle can be assigned, at which the first from the N_f cutting edges touches the point (Figure 54). According to the accepted geometrical description of ball-end mills [111] the line of the cutting edge evolves by the projection of a helix onto the tool envelope (Figure 55), so the rotation angle assigned to the *j*th surface patch is, considering that the *x* basis vector of the tool coordinate system marks out the zero spindle rotation angle:

$$\theta_j = \Psi_j + \psi_j = \Psi_j + \frac{z_j}{\tan(\varphi) R_0} \tag{34}$$

where z_j is the z-coordinate of the intersection point, R_0 is the tool radius, and $\varphi = 90^0 - i_0$, if i_0 is the nominal helix angle.

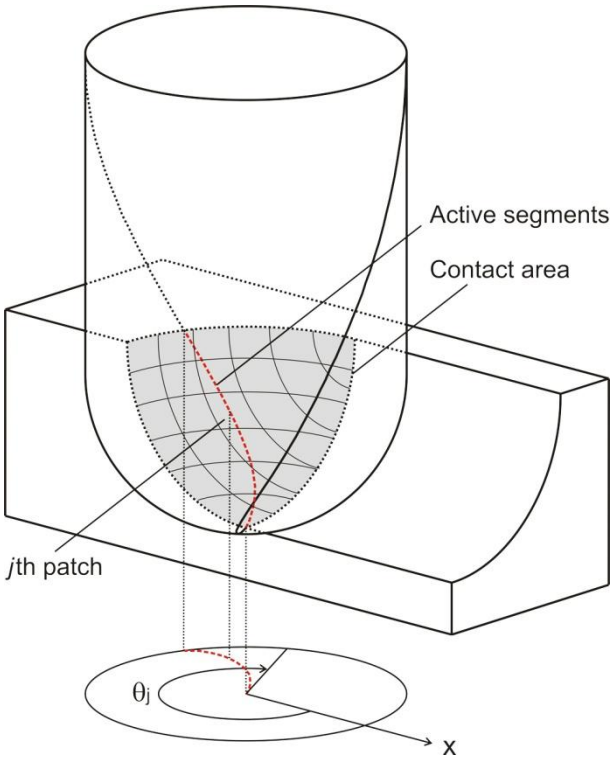

Figure 54. Spindle rotation angle at the *j*th patch

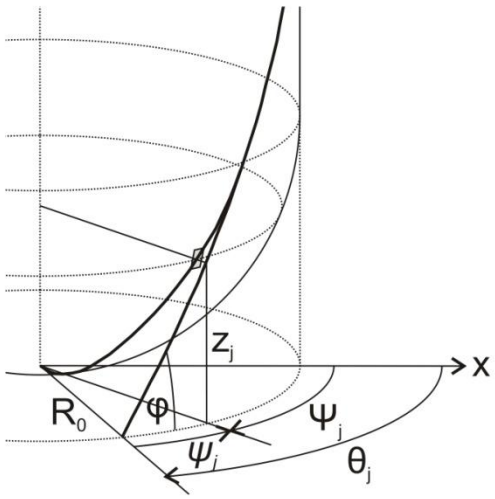

Figure 55. Edge segment localization

Let $\Delta \boldsymbol{F}_{xyz,j}$ represent the cutting force that affects to an elemental cutting edge at the *j*th intersection point, presuming this edge segment produces unit chip width ($db = 1$). From Eqs. (7) and (10):

$$\begin{cases} \Delta F_{t,j} = K_{te} dS_j + K_{tc} t_{n,j} \\ \Delta F_{r,j} = K_{re} dS_j + K_{rc} t_{n,j} \\ \Delta F_{a,j} = K_{ae} dS_j + K_{ac} t_{n,j} \end{cases} \tag{35}$$

$$\begin{cases} \Delta F_{x,j} = -\Delta F_{r,j}\cos\left(\kappa_j\right)\cos\left(\Psi_j\right) + \Delta F_{t,j}\sin\left(\Psi_j\right) - \Delta F_{a,j}\sin\left(\kappa_j\right)\cos(\Psi_j) \\ \Delta F_{y,j} = -\Delta F_{r,j}\cos\left(\kappa_j\right)\sin\left(\Psi_j\right) - \Delta F_{t,j}\cos\left(\Psi_j\right) - \Delta F_{a,j}\sin\left(\kappa_j\right)\sin(\Psi_j) \\ \Delta F_{z,j} = \Delta F_{r,j}\sin\left(\kappa_j\right) - \Delta F_{a,j}\cos\left(\kappa_j\right) \end{cases} \tag{36}$$

where K_{te}, K_{re}, K_{ae} (N/mm) are the edge specific coefficients, K_{tc}, K_{rc}, K_{ac} (N/mm^2) are the shear specific coefficients, dS_j (mm) is the length of the elemental edge, and $t_{n,j}$ (mm) is the undeformed chip thickness at the patch. Ψ_j and κ_j are the horizontal and vertical positioning angles of the segment, according to Figure 56. On the cylindrical cutter part we calculate with $\kappa_j = 0$.

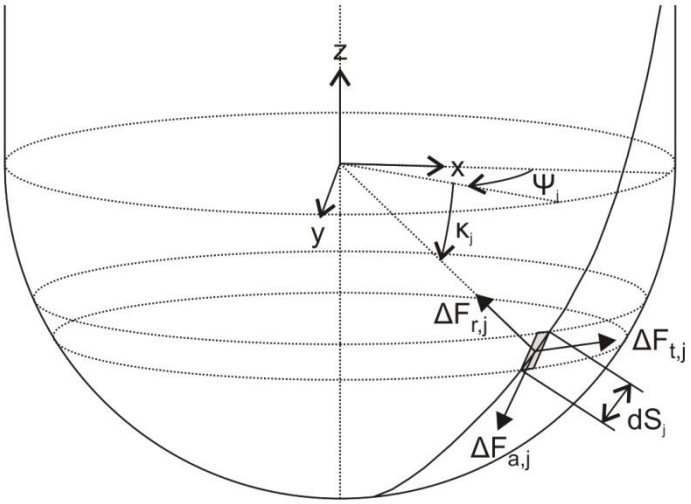

Figure 56. Elemental force components at the *j*th patch

The projection of the helix to the cutter envelope causes varying helix angle on the spherical surface area. The smaller radius means bigger steepness during the same turning and rise:

$$r_j \cdot \tan(\varphi_r) = R_0 \cdot \tan(\varphi) \tag{37}$$

From Figure 57 we can read that

$$\cos(\kappa_j) = dz \tag{38}$$

and

$$\tan(\varphi_r) = \frac{dt}{dz} \tag{39}$$

and furthermore

$$dS_j = \sqrt{1+dt} = \sqrt{1+\frac{dz^2}{\tan^2(\varphi_r)}} = \sqrt{1+\frac{\cos^2(\kappa_j)}{\tan^2(\varphi_r)}} \tag{40}$$

Considering that

$$\cos(\kappa_j) = \frac{r_j}{R_0} \tag{41}$$

we can get the length of the elemental edge by substituting φ_r from Eq. (37):

$$dS_j = \sqrt{1 + \frac{\cos^2(\kappa_j)}{\left(\frac{R_0}{r_j}\tan(\varphi_r)\right)^2}} = \sqrt{1 + \frac{\cos^4(\kappa_j)}{\tan^2(\varphi)}} \tag{42}$$

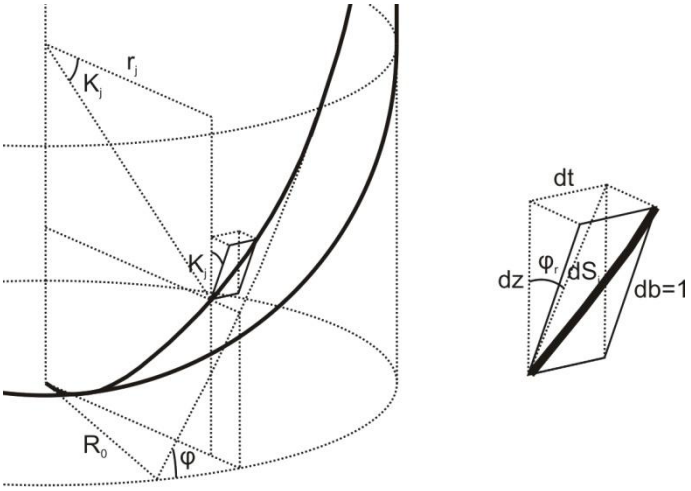

Figure 57. Calculation of dS_j

According to the generalized modeling of milling cutters by Altintas and Engin [92], Eqs. (7)-(10) thus Eqs. (35) and (36) can be applied not only to ball end but to general end mills, depending on the calculation method of dS_j.

The $t_{n,j}$ local chip thickness is measured along the line of the local surface normal, and its extent is the length of the line segment between the surfaces of the cutter in the (*k*-1)th and *k*th sequent simulation steps (Figure 58). With marking the intersection points of the line and the (*k*-1)th surface with $\mathbf{p}_a$ and $\mathbf{p}_b$, the $t_{n,j}$ local chip thickness at $\mathbf{p}_j$, situated at the half-sphere part of a ball-end cutter, along with the belonging normal $\mathbf{n}_j$ can be gained by Eq. (43):

$$|\mathbf{p}_j - \mathbf{n}_j t_{n,j}|^2 = |\mathbf{p}_{a,b}|^2 = R_0^2 \tag{43}$$

where the position and normal vectors are relative to the tool coordinate system at the (*k*-1)th position. In other form:

$$(p_{jx} - n_{jx}t_{n,j})^2 + (p_{jy} - n_{jy}t_{n,j})^2 + (p_{jz} - n_{jz}t_{n,j})^2 = R_0^2 \tag{44}$$

while in the case of the cylindrical part the *z*-component has to be discarded:

$$(p_{jx} - n_{jx}t_{n,j})^2 + (p_{jy} - n_{jy}t_{n,j})^2 = R_0^2 \tag{45}$$

Note that the form of Eqs. (44) and (45) is just the same as the form of Eqs. (26) and (27). This means that the functions of the intersection calculation between the workpiece dexels and the tool, described in Section 3.2.2, can be directly used for chip thickness calculation as well, for any type of tool shape.

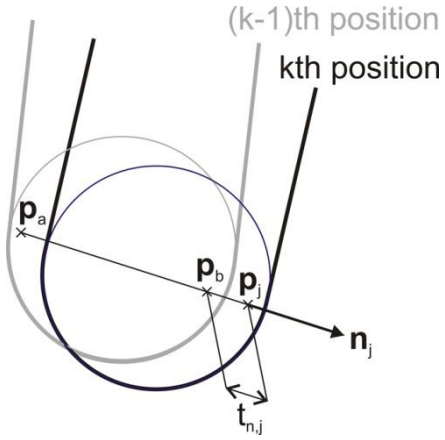

Figure 58. Chip thickness calculation

With the above described calculation we get the right value of local chip thickness only if the turning of the tool is $\frac{1}{N_f}$ in each simulation step, where N_f is the number of cutter teeth – or in other words the moving of the tool equals the feed per tooth (f_z). This is a too strict restriction, which would greatly slow down the simulation process. To avoid this we have to allow bigger moving and calculate with a transitional cutter position between the two consecutive simulation steps, as the destination of a moving with $\frac{1}{N_f}$ spindle rotation angle increment.

Let $\mathbf{T}_{c(k-1)}$ and $\mathbf{T}_{ck}$ mark the homogenous transformation matrices that define the positions of the cutter in the (*k*-1)th and *k*th simulation steps in relation to the workpiece coordinate system. The displacement between the two locations can be defined as a series of three transformations (Figure 59 a,b,c):

- A translation form $\mathbf{p}_{c(k-1)}$ to $\mathbf{p}_{ck}$ cutter matrix origins:

$$\mathbf{p}_{c(k-1)\to ck} = \mathbf{p}_{ck} - \mathbf{p}_{c(k-1)} \tag{46}$$

- a rotation that transfers the $\mathbf{z}_{c(k-1)}$ tool axis to its new $\mathbf{z}_{ck}$ orientation, practically by using a quaternion for this operation:

$$q_{c(k-1)\to ck} = (\alpha, \mathbf{v}) = \left(\arccos(\mathbf{z}_{c(k-1)} \cdot \mathbf{z}_{ck}), \mathbf{z}_{c(k-1)} \times \mathbf{z}_{ck}\right) \tag{47}$$

- and another rotation around the $\mathbf{z}_{ck}$ axis that transfers the *x* and *y* tool coordinate axes to their new orientation:

$$\mathbf{R}_{c(k-1)\to ck} = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{48}$$

where β is the spindle rotation increment between the two positions, which is, at S (1/min) spindle speed and t_{cycle} (s) simulation cycle time:

$$\beta = 2\pi \frac{S}{60} t_{cycle} \tag{49}$$

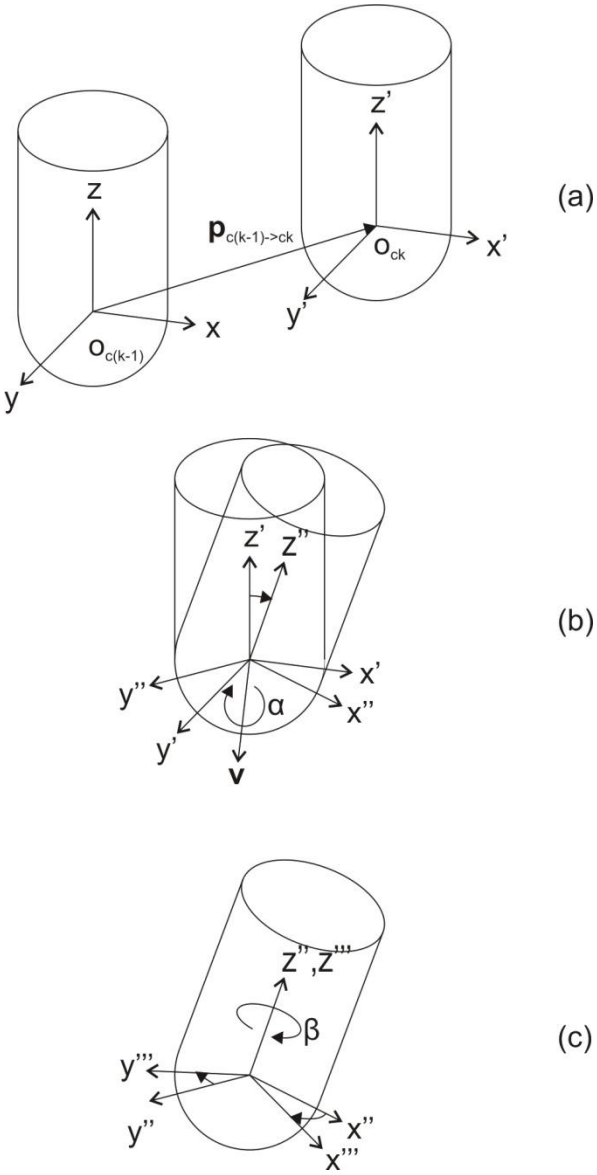

Figure 59. Tool displacement between two simulation steps

If we approximate the displacement between the two positions with linear movement, the $\mathbf{T}_{ctr}$ matrix that defines the transitional position of the tool in relation to the tool coordinate system at the (*k*-1)th step is resulted by the following transformations:

- A translation form $\mathbf{p}_{c(k-1)}$ to $\mathbf{p}_{ctr}$ cutter matrix origins:

$$\mathbf{p}_{c(k-1)\to ctr} = \frac{2\pi}{N_f \beta}\mathbf{p}_{c(k-1)\to ck} \tag{50}$$

- a rotation that transfers the $\mathbf{z}_{c(k-1)}$ tool axis to its new $\mathbf{z}_{ctr}$ orientation, by using the quaternion:

$$q_{c(k-1)\to ctr} = \left(\frac{2\pi}{N_f \beta}\alpha, \mathbf{v} \right) \tag{51}$$

- and a rotation around the $\mathbf{z}_{ctr}$ axis that transfers the *x* and *y* tool coordinate axes to their new orientation:

$$\mathbf{R}_{c(k-1)\to ctr} = \begin{bmatrix} \cos(\frac{2\pi}{N_f}) & -\sin(\frac{2\pi}{N_f}) & 0 \\ \sin(\frac{2\pi}{N_f}) & \cos(\frac{2\pi}{N_f}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{52}$$

Now let us divide the whole revolution of the cutter into *P* slices, so that the turning of the tool will be $2\pi/P$ revolution at each slice (Figure 60), and instead of calculating instantaneous cutting forces that was done in the original model, let us calculate average forces at the slices. In this case each patch that is situated in the region of a slice takes part in forming the force affecting on the region, with the weight of its surface area. For getting the average values, we must divide these elemental forces with the local width of the slice and sum them up. Therefore the average force components affecting to the first cutting edge while sweeping through the *k*th slice can be determined as:

$$\mathbf{F}_{xyz,k} = \sum_j \left(\Delta\mathbf{F}_{xyz,j} \cdot \frac{A_j P}{2\pi r_j} \right) \tag{53}$$

for every *j* to which

$$k\frac{2\pi}{P} \le \theta_j < (k+1)\frac{2\pi}{P} \tag{54}$$

where A_j is the surface area of the *j*th patch and $2\pi r_j/P$ is the length of the arch, on which the point of the edge that touches the actual intersection point sweeps during the slice. In this way the force components at the patches that are situated within the swept area of the *k*th slice are averaged. If *P* is equal with 1, then $\boldsymbol{F}_{xyz,0}$ gives the average force vector which affects to the edge during a whole revolution. With greater number of slices the force values approach the instantaneous forces. The

experience shows that in most cases 32 slices are enough for getting accurate instantaneous values. The forces on the additional edges can be calculated in the same way, considering that the related spindle rotation angle must be increased with the angle between the first and *i*th blade:

$$\theta_{j,i} = \theta_j + \frac{2\pi}{N_f} i \tag{55}$$

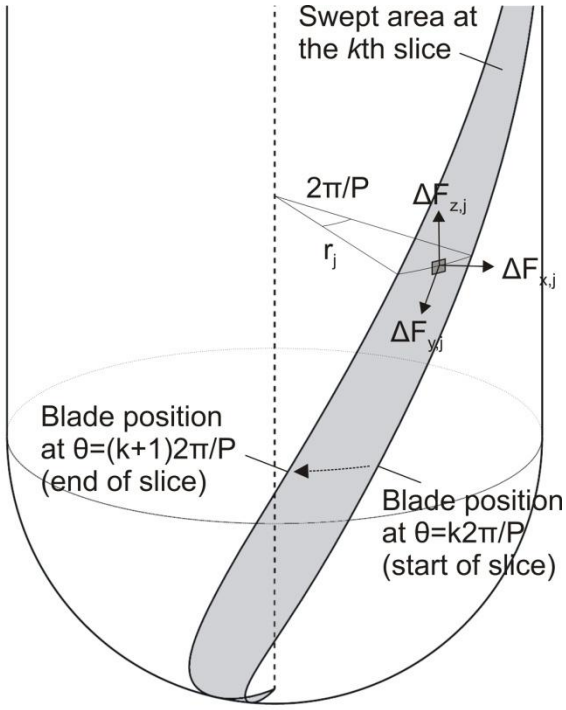

Figure 60. A slice of the whole revolution

The form of Eq. (53) lets the force components belonging to different surface patches to be calculated in parallel to a whole revolution, with the use of the GPGPU. Figure 61 shows the implementation of the algorithm in CUDA parallel programming environment. In the first GPU call each elemental force vector belonging to a surface area patch is calculated by an independent GPU thread. The rate of parallelization equals the number of stream processors, which varies from a couple of dozens to more than a thousand, depending on the graphics card model. In the second call the elemental forces belonging to the same slice are grouped and summed up. Each GPU thread processes a subset of the input vectors for the purpose of parallelization, thus provides only partial results. The number of threads depends on the choice of the programmer: too few threads decrease the efficiency, while too many threads slow down the third computing step, which sum up the matching subtotals from the several input data blocks. Each thread in the third GPU call deals with one of the slices of the revolution, thus the number of them must equal the number of slices (*P*). In the second call *n* adding operation is performed in $\frac{n}{m}$ computing cycles if *n* is the number of patches and *m* is the number of

threads. The third call means $m \cdot P$ adding operations in $\frac{m \cdot P}{P} = m$ cycles, thus the number of computing cycles during the two GPU calls is

$$c(m) = \frac{n}{m} + m \tag{56}$$

The minimum of $c(m)$ is at

$$\frac{dc(m)}{dm} = \frac{d}{dm}\left(\frac{n}{m} + m\right) = 1 - \frac{n}{m^2} = 0 \tag{57}$$

from which the optimal number of threads in the second GPU call is

$$m_{opt} = \sqrt{n} \tag{58}$$

where the number of computing cycles is

$$c(m)\Big|_{m_{opt}} = 2\sqrt{n} \tag{59}$$

This means that the rate of parallelization in the second and third steps is:

$$rp = \frac{n}{2\sqrt{n}} = \frac{\sqrt{n}}{2} \tag{60}$$

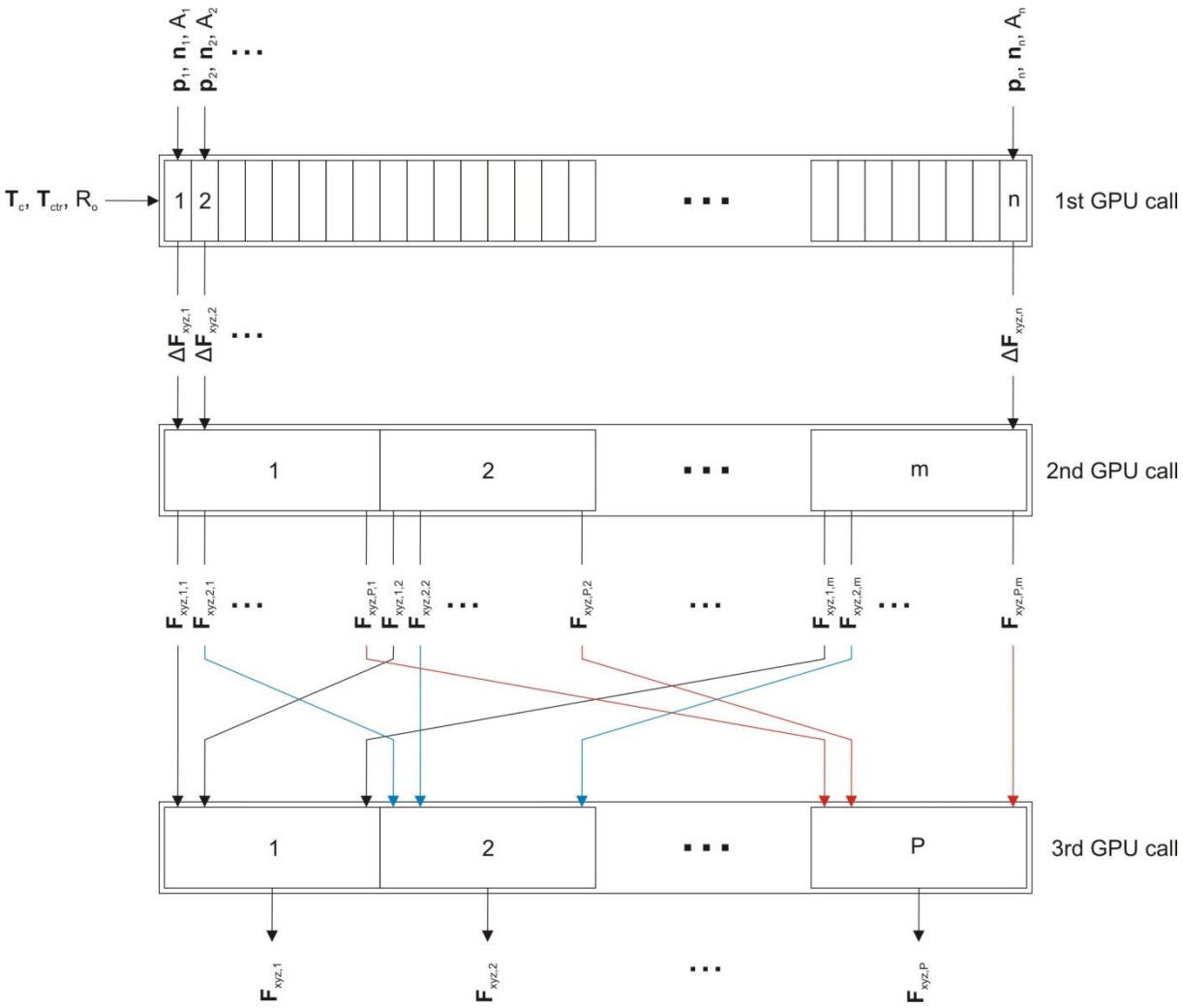

Figure 61. Parallelized force computing

3.3.2 Determination of the cutting force coefficients

As it turned out in the technical review part, the complicated methods of the cutting force coefficient determination – together with the limited validity of their results – prevent the application of the mechanistic cutting force model in practice. In the followings I will present that the above described form of the mechanistic model – besides the advantages that arise from the direct processing of the cutting geometry and the parallelized computing – offers a straightforward way of cutting force coefficient determination, which decreases the geometrical constraints of the determination process, thus allowing us a wider use of the mechanical cutting force model.

In accordance with the referenced papers, constant K_{te}, K_{re} and K_{ae} ploughing coefficients and polynomial shearing coefficients, which depend on the z-position, are considered in the force equations for getting the most accurate results:

$$\begin{cases} K_{tc} = K_{tc0} + K_{tc1} \cdot z + K_{tc2} \cdot z^2 + \cdots + K_{tcn} \cdot z^n \\ K_{rc} = K_{rc0} + K_{rc1} \cdot z + K_{rc2} \cdot z^2 + \cdots + K_{rcn} \cdot z^n \\ K_{ac} = K_{ac0} + K_{ac1} \cdot z + K_{ac2} \cdot z^2 + \cdots + K_{acn} \cdot z^n \end{cases} \tag{61}$$

Substituting Eqs. (35), (36) and Eq. (61) into Eq. (53) the x-component of the cutting force is:

$$\begin{aligned} F_{x,k} = & \\ & \sum_j \left(\left(K_{re} dS_j + \left(K_{rc0} + K_{rc1} \cdot z_j + K_{rc2} \cdot z_j^2 + \cdots + K_{rcn} \cdot z_j^n \right) t_{n,j} \right) \left(-\cos\left(\kappa_j\right) \cos\left(\Psi_j\right) \right) \frac{S_j P}{2\pi r_j} \right) \\ & + \sum_j \left(\left(K_{te} dS_j + \left(K_{tc0} + K_{tc1} \cdot z_j + K_{tc2} \cdot z_j^2 + \cdots + K_{tcn} \cdot z_j^n \right) t_{n,j} \right) \sin\left(\Psi_j\right) \frac{S_j P}{2\pi r_j} \right) \\ & + \sum_j \left(\left(K_{ae} dS_j + \left(K_{ac0} + K_{ac1} \cdot z_j + K_{ac2} \cdot z_j^2 + \cdots + K_{acn} \cdot z_j^n \right) t_{n,j} \right) \left(-\sin\left(\kappa_j\right) \cos\left(\Psi_j\right) \right) \frac{S_j P}{2\pi r_j} \right) \end{aligned} \tag{62}$$

Let us use the following substitutions:

$$\Lambda_{0,k} = \sum_j \left(dS_j \left(-\cos\left(\kappa_j\right) \cos\left(\Psi_j\right) \right) \frac{S_j P}{2\pi r_j} \right) \tag{63}$$

$$\Lambda_{p,k} = \sum_j \left(z_j^{p-1} t_{n,j} \left(-\cos\left(\kappa_j\right) \cos\left(\Psi_j\right) \right) \frac{S_j P}{2\pi r_j} \right), 1 \le p \le n \tag{64}$$

$$\Lambda_{n+1,k} = \sum_j \left(dS_j \sin\left(\Psi_j\right) \frac{S_j P}{2\pi r_j} \right) \tag{65}$$

$$\Lambda_{n+1+q,k} = \sum_j \left(t_j^{q-1} \sin\left(\Psi_j\right) \frac{S_j P}{2\pi r_j} \right), 1 \le q \le n \tag{66}$$

$$\Lambda_{2n+3,k} = \sum_j \left(dS_j \left(-\sin\left(\kappa_j\right) \cos\left(\Psi_j\right) \right) \frac{S_j P}{2\pi r_j} \right) \tag{67}$$

$$\Lambda_{2n+3+r,k} = \sum_j \left(z_j^{r-1} t_{n,j} \left(-\sin\left(\kappa_j\right) \cos\left(\Psi_j\right) \right) \frac{S_j P}{2\pi r_j} \right), 1 \le r \le n \tag{68}$$

$$\mathrm{K}_0 = K_{re} \tag{69}$$

$$\mathrm{K}_p = K_{rcp}, 1 \le p \le n \tag{70}$$

$$\mathrm{K}_{n+1} = K_{te} \tag{71}$$

$$\mathrm{K}_{n+1+q} = K_{tc(q-1)}, 1 \le q \le n \tag{72}$$

$$\mathrm{K}_{2n+3} = K_{ae} \tag{73}$$

$$\mathrm{K}_{2n+3+r} = K_{tc(r-1)}, 1 \le r \le n \tag{74}$$

In this way Eq. (62) looks as follows:

$$F_{x,k} = \sum_s \Lambda_{s,k} \mathrm{K}_s, 0 \le s \le 3(n+2) \tag{75}$$

And similarly to the *y* component:

$$\begin{aligned} F_{y,k} = & \\ & \sum_j \left(\left(K_{re} dS_j + \left(K_{rc0} + K_{rc1} \cdot z_j + K_{rc2} \cdot z_j^2 + \cdots + K_{rcn} \cdot z_j^n \right) t_{n,j} \right) \left(-\cos\left(\kappa_j\right) \sin\left(\Psi_{\mathrm{j}}\right) \right) \frac{S_j P}{2\pi r_j} \right) \\ & + \sum_j \left(\left(K_{te} dS_j + \left(K_{tc0} + K_{tc1} \cdot z_j + K_{tc2} \cdot z_j^2 + \cdots + K_{tcn} \cdot z_j^n \right) t_{n,j} \right) \left(-\cos\left(\Psi_{\mathrm{j}}\right) \right) \frac{S_j P}{2\pi r_j} \right) \\ & + \sum_j \left(\left(K_{ae} dS_j + \left(K_{ac0} + K_{ac1} \cdot z_j + K_{ac2} \cdot z_j^2 + \cdots + K_{acn} \cdot z_j^n \right) t_{n,j} \right) \left(-\sin\left(\kappa_j\right) \sin\left(\Psi_{\mathrm{j}}\right) \right) \frac{S_j P}{2\pi r_j} \right) \end{aligned} \tag{76}$$

The substitutions:

$$\Gamma_{0,k} = \sum_j \left(dS_j \left(-\cos\left(\kappa_j\right) \sin\left(\Psi_{\mathrm{j}}\right) \right) \frac{S_j P}{2\pi r_j} \right) \tag{77}$$

$$\Gamma_{p,k} = \sum_j \left(z_j^{p-1} t_{n,j} \left(-\cos\left(\kappa_j\right) \sin\left(\Psi_{\mathrm{j}}\right) \right) \frac{S_j P}{2\pi r_j} \right), 1 \le p \le n \tag{78}$$

$$\Gamma_{n+1,k} = \sum_j \left(dS_j \left(-\cos\left(\Psi_{\mathrm{j}}\right) \right) \frac{S_j P}{2\pi r_j} \right) \tag{79}$$

$$\Gamma_{n+1+q,k} = \sum_j \left(t_j^{q-1} \left(-\cos\left(\Psi_{\mathrm{j}}\right) \right) \frac{S_j P}{2\pi r_j} \right), 1 \le q \le n \tag{80}$$

$$\Gamma_{2n+3,k} = \sum_j \left(dS_j \left(-\sin\left(\kappa_j\right) \sin\left(\Psi_{\mathrm{j}}\right) \right) \frac{S_j P}{2\pi r_j} \right) \tag{81}$$

$$\Gamma_{2n+3+r,k} = \sum_j \left(z_j^{r-1} t_{n,j} \left(-\sin\left(\kappa_j\right) \sin\left(\Psi_{\mathrm{j}}\right) \right) \frac{S_j P}{2\pi r_j} \right), 1 \le r \le n \tag{82}$$

Thus from Eq. (76):

$$F_{y,k} = \sum_s \Gamma_{s,k} \mathrm{K}_s, 0 \le s \le 3(n+2) \tag{83}$$

And at last to $F_{z,k}$:

$$F_{z,k} =$$

$$\sum_j \left(\left(K_{re} dS_j + \left(K_{rc0} + K_{rc1} \cdot z_j + K_{rc2} \cdot z_j^2 + \cdots + K_{rcn} \cdot z_j^n \right) t_{n,j} \right) \sin\left(\kappa_j\right) \frac{S_j P}{2\pi r_j} \right)$$

$$+0 \tag{84}$$

$$+\sum_j \left(\left(K_{ae} dS_j + \left(K_{ac0} + K_{ac1} \cdot z_j + K_{ac2} \cdot z_j^2 + \cdots + K_{acn} \cdot z_j^n \right) t_{n,j} \right) \left(-\cos\left(\kappa_j\right) \right) \frac{S_j P}{2\pi r_j} \right)$$

The substitutions:

$$\Omega_{0,k} = \sum_j \left(dS_j \sin\left(\kappa_j\right) \frac{S_j P}{2\pi r_j} \right) \tag{85}$$

$$\Omega_{p,k} = \sum_j \left(z_j^{p-1} t_{n,j} \sin\left(\kappa_j\right) \frac{S_j P}{2\pi r_j} \right), 1 \le p \le n \tag{86}$$

$$\Omega_{n+1,k} = 0 \tag{87}$$

$$\Omega_{n+1+q,k} = 0, 1 \le q \le n \tag{88}$$

$$\Omega_{2n+3,k} = \sum_j \left(dS_j \left(-\cos\left(\kappa_j\right) \right) \frac{S_j P}{2\pi r_j} \right) \tag{89}$$

$$\Omega_{2n+3+r,k} = \sum_j \left(z_j^{r-1} t_{n,j} \left(-\cos\left(\kappa_j\right) \right) \frac{S_j P}{2\pi r_j} \right), 1 \le r \le n \tag{90}$$

Thus form Eq. (84):

$$F_{z,k} = \sum_s \Omega_{s,k} \mathrm{K}_s, 0 \le s \le 3(n+2) \tag{91}$$

Eqs. (75), (83) and (91) consist of two well distinguishable parts: The first is composed of the $\mathrm{K_s}$ cutting force coefficient components, while the $\Lambda_{s,k}$, $\Gamma_{s,k}$, $\Omega_{s,k}$ values carry clearly geometrical information belonging to the active patches within the *k*th slice area. According to the equations the multiplication operations between $\Lambda_{s,k}$, $\Gamma_{s,k}$, $\Omega_{s,k}$ and $\mathrm{K_s}$ can be taken up from the first GPGPU call and placed to the end of the calculations, thus they have to be executed only once instead of *n* times in the parallel threads (Figure 62). This means one calculation cycle versus *n/SP* cycles, where *SP* marks the number of GPU stream processors. Depending on the number of stream processors and the number of patches – which is typically a couple of hundred –, this causes slight speed up. The main advantage of the formulas will appear at the calculation of the coefficients.

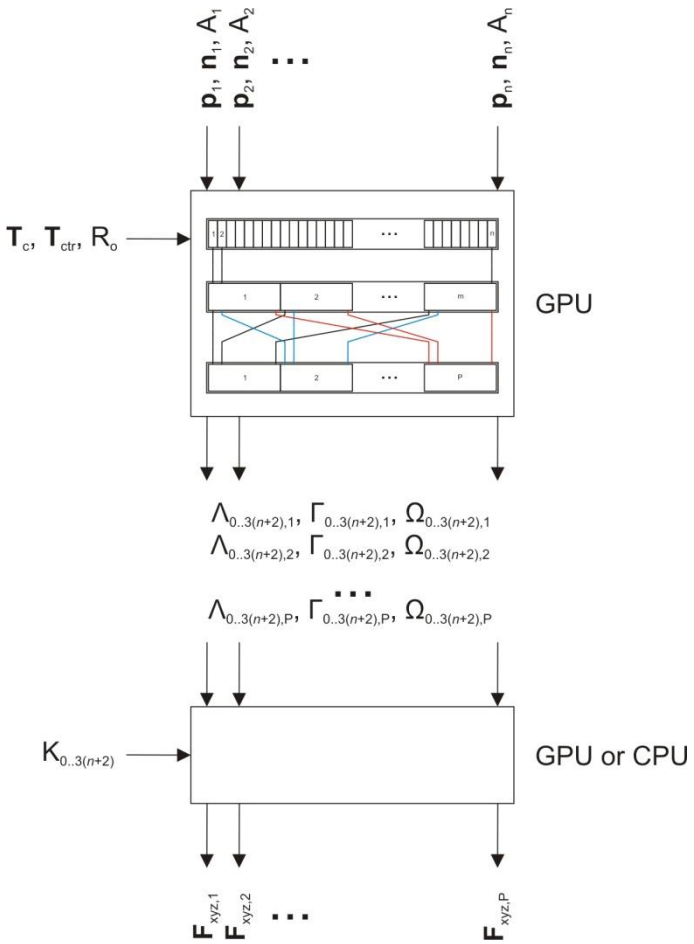

Figure 62. Force computing with GPU-created geometry data

The form of the equations lets us calculate the cutting force coefficients in the following way. During a milling experiment, the *x*-component of the cutting force ($F_{x,k\,m}$) is being measured, while the $\Lambda_{\mathrm{s,k}}$ values are being produced by the simulation of the same process; *k* marks the *k*th slice swept by the blades after the start of the experiment. Let Q_x in Eq. (92) indicate the difference between the calculated ($F_{x,k}$) and measured ($F_{x,k\,m}$) forces during the measuring. In accordance with the least-square method, trusting in the relevance of the equations of the mechanistic cutting force model, Q_x has to be minimized to get the best-fitting coefficients.

$$Q_x = \sum_k (F_{x,k} - F_{x,k\,m})^2 \tag{92}$$

Thus, from Eq. (75) and Eq. (92) for each K_s:

$$\frac{\partial Q_x}{\partial K_s} = \sum_k \frac{\partial \left(F_{x,k} - F_{x,k\,m}\right)^2}{\partial K_s} = \sum_k \left(\frac{\partial \left(F_{x,k} - F_{x,k\,m}\right)}{\partial K_s} \cdot 2\left(F_{x,k} - F_{x,k\,m}\right) \right) =$$

$$= \sum_k \left(\frac{\partial \left(\Lambda_{0,k} K_0 + \ldots + \Lambda_{t,k} K_t - F_{x,k\,m}\right)}{\partial K_s} \cdot 2\left(\Lambda_{0,k} K_0 + \ldots + \Lambda_{t,k} K_t - F_{x,k\,m}\right) \right) = \tag{93}$$

$$= \sum_k \left(2\Lambda_{s,k}\Lambda_{0,k} K_0 + 2\Lambda_{s,k}\Lambda_{1,k} K_1 + \ldots + 2\Lambda_{s,k}\Lambda_{t,k} K_t - 2\Lambda_{s,k} F_{x,k\,m}\right) = 0$$

Eq. (93) describes a linear equation system of $t = 3(n+2)$ equations, which can be solved with Gaussian elimination for every K_s:

$$\begin{cases} K_0 \sum_k 2\Lambda_{0,k}\Lambda_{0,k} + K_1 \sum_k 2\Lambda_{0,k}\Lambda_{1,k} + \ldots + K_t \sum_k 2\Lambda_{0,k}\Lambda_{t,k} = \sum_k 2\Lambda_{0,k} F_{x,k\,m} \\ \vdots \\ K_0 \sum_k 2\Lambda_{t,k}\Lambda_{0,k} + K_1 \sum_k 2\Lambda_{t,k}\Lambda_{1,k} + \ldots + K_t \sum_k 2\Lambda_{t,k}\Lambda_{t,k} = \sum_k 2\Lambda_{t,k} F_{x,k\,m} \end{cases} \tag{94}$$

Note that the coefficients can be determined from the measuring of the *x* force-component only. Nevertheless, for the *y* and *z* cutting force components similar equations can be created in the same way, and should be involved into the computation for improving the accuracy of the method:

$$\frac{\partial Q_y}{\partial K_s} = \sum_k \left(2\Gamma_{s,k}\Gamma_{0,k} K_0 + 2\Gamma_{s,k}\Gamma_{1,k} K_1 + \ldots + 2\Gamma_{s,k}\Gamma_{t,k} K_t - 2\Gamma_{s,k} F_{y,k\,m}\right) = 0 \tag{95}$$

$$\frac{\partial Q_z}{\partial K_s} = \sum_k \left(2\Omega_{s,k}\Omega_{0,k} K_0 + 2\Omega_{s,k}\Omega_{1,k} K_1 + \ldots + 2\Omega_{s,k}\Omega_{t,k} K_t - 2\Omega_{s,k} F_{z,k\,m}\right) = 0 \tag{96}$$

thus the linear equation system to be solved:

$$
\begin{cases}
\mathrm{K}_0 2\sum_k \left(\Lambda_{0,k}\Lambda_{0,k}+\Gamma_{0,k}\Gamma_{0,k}+\Omega_{0,k}\Omega_{0,k}\right)+ \\
+\mathrm{K}_1 2\sum_k \left(\Lambda_{0,k}\Lambda_{1,k}+\Gamma_{0,k}\Gamma_{1,k}+\Omega_{0,k}\Omega_{1,k}\right)+ \\
\cdots \\
+\mathrm{K}_t 2\sum_k \left(\Lambda_{0,k}\Lambda_{t,k}+\Gamma_{0,k}\Gamma_{t,k}+\Omega_{0,k}\Omega_{t,k}\right)= \\
=2\sum_k \left(\Lambda_{0,k}F_{x,k\,m}+\Gamma_{0,k}F_{y,k\,m}+\Omega_{0,k}F_{z,k\,m}\right) \\
\vdots \\
\mathrm{K}_0 2\sum_k \left(\Lambda_{t,k}\Lambda_{0,k}+\Gamma_{t,k}\Gamma_{0,k}+\Omega_{t,k}\Omega_{0,k}\right)+ \\
+\mathrm{K}_1 2\sum_k \left(\Lambda_{t,k}\Lambda_{1,k}+\Gamma_{t,k}\Gamma_{1,k}+\Omega_{t,k}\Omega_{1,k}\right)+ \\
\cdots \\
+\mathrm{K}_t 2\sum_k \left(\Lambda_{t,k}\Lambda_{t,k}+\Gamma_{t,k}\Gamma_{t,k}+\Omega_{t,k}\Omega_{t,k}\right)= \\
=2\sum_k \left(\Lambda_{t,k}F_{x,k\,m}+\Gamma_{t,k}F_{y,k\,m}+\Omega_{t,k}F_{z,k\,m}\right)
\end{cases}
\tag{97}
$$

If we write the equation system in matrix form:

$$
\begin{bmatrix}\mathrm{K}_0 & \mathrm{K}_1 & \ldots & \mathrm{K}_t\end{bmatrix}\mathbf{A}=\mathbf{B} \tag{98}
$$

the basis of the Gaussian elimination is the **G** augmented matrix:

$$
\mathbf{G}=\left(\mathbf{A}\,|\,\mathbf{B}\right) \tag{99}
$$

Since the Λ, Γ and Ω values are calculated by the GPU in each simulation step, the complementing of the **G** matrix with the new values and the solving of the linear equation system – which is favorably executed by the CPU that still has free computing capacity – can be instantly fulfilled after the GPU calls, without considerably slowing down the simulation process. This means that we can determine the coefficients in real-time, at any moment during the experiment. On the other hand, it can be seen that the introduced method does not require restrictions in regard to the cutting geometry in the course of the machining process, as opposed to the existing techniques that were based on a series of half-slot or slot milling tests. The equations do not prescribe the length of the experiment for the accurate cutting coefficient determination. The cutting geometry changes from slice to slice as the blades penetrate into the workpiece; this changing appears in the equation with the gradually altering Λ, Γ and Ω values, and just this variety provides the required amount of information for the least square method – even during a single revolution of the tool. Nevertheless, the noisiness and accuracy of the measuring, the agreement of the simulated and real cutting geometry: these are the parameters which truly determine the required steps of the experiment, as the validating tests will show it in Section 3.3.4.

3.3.3 Instant cutting force prediction with a look-forward algorithm

The fact that the above described method does not require special cutting conditions during the experiment makes it *theoretically* possible to determine the cutting force coefficients during *any* machining process, including multi-axis milling, and to apply the just-determined coefficients immediately for force prediction during the same process. Figure 63 shows the manner of this. The simulation goes ahead of the manufacturing, namely the current time of the simulation is ahead of the current time of the manufacturing. The geometrical data is put into a memory array, as well as the measured force values just after they have been obtained during the machining. The interdependent values are added to the **G** matrix, which is instantly solved by Gaussian elimination to determine the coefficients. With the freshly calculated coefficients and the currently simulated geometrical data we can predict the cutting forces which belong to a later ensuing moment of the machining process.

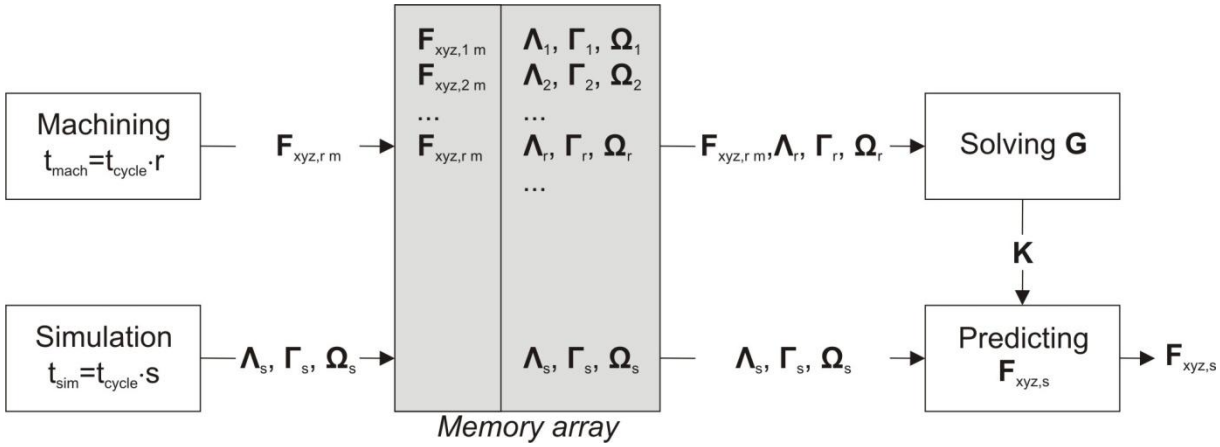

Figure 63. Instant force prediction

With this technique we can overcome the two main obstacles that limit the usability of the mechanistic cutting force model in practice. First of all, we can predict instantaneous cutting force vectors without the need of pre-created coefficient databases, which have to include values for every workpiece material – cutter couple. Secondly, if we update the coefficients several times in the course of the machining, we can avoid the obsolete of them, which is caused by the tool wearing. The best result can be reached with continuous cutting force measuring and synchronized simulation in the course of the machining, with small look-forward period between the simulation and the real machining process. This can be performed in the following way.

Three pointers point to the memory block that gives place to the array of geometrical data and the measured force values (Figure 64). The *manufacturing_time_pointer* gets on with the progressing of the physical manufacturing; the *simulation_time_pointer* is synchronized with the simulation. In each simulation step the Λ, Γ and Ω values are calculated and pushed into the memory block at the position of the *simulation_time_pointer*. After the real cutting forces have been measured, their values can be

put into the memory beside the related geometry data at the *manufacturing_time_pointer*, and all these values can be added to the **G** matrix. Solving of the matrix gives the actual coefficients. On the basis of the prospective cutting geometry information provided by the simulation, and the coefficients just calculated, the forthcoming cutting forces are predicted. After a certain elapsed time the Λ, Γ and Ω values and the measured force values are subtracted from the matrix at the *time_window_pointer*, so they do not take part in the coefficient calculation any more. In this way the coefficients adjust themselves to the altering machining conditions during long manufacturing processes. This method makes it also possible to conclude to the degree of wearing with the observation of the altering coefficients during the machining, relying on the results of experiments [124] in which the relationship between the flank wear and the tangential cutting force coefficients are investigated.

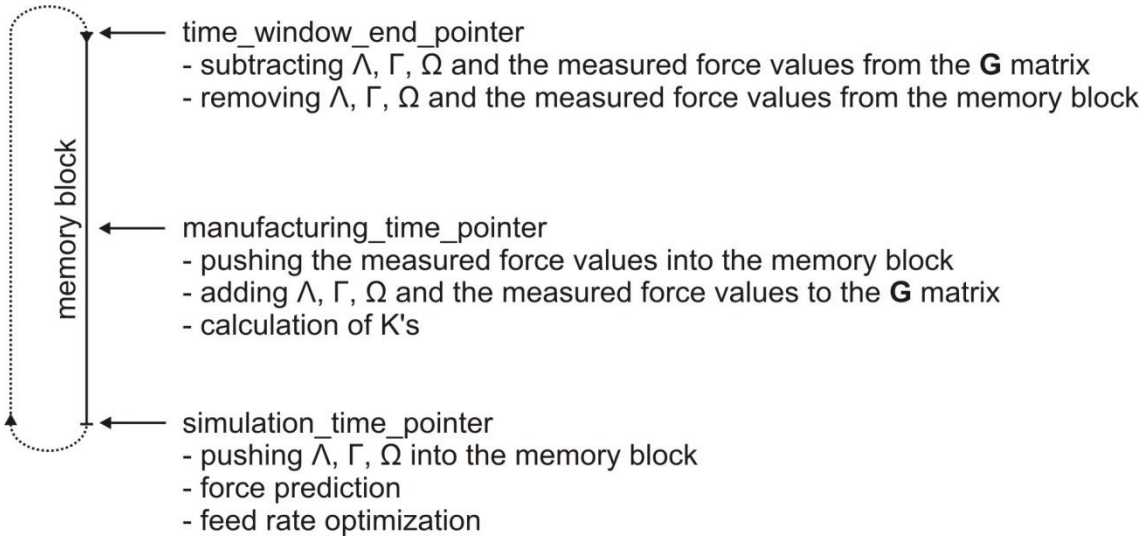

Figure 64. Pointers of the look-forward algorithm

Besides the advantages of the introduced instant cutting force prediction I must mention the difficulties of its realization too. For the accurate force prediction exact synchronization is needed between the simulation and the real machining. This means that not only the force component values but the cutter position values (including spindle turning position) have to be measured and sent to the computing unit to be processed in real-time or at least very fast. The accurate dynamometers are expensive so their application raises economical questions. Lamikiz et al. in [111] detail the possibilities and difficulties of such a kind of force recording in the course of milling of complex parts. They enlighten the main issues to be solved in relation to the high sampling speed, the processing of a big amount of data and the cost of the configuration. All of these are out of the scope of this dissertation – but they will be mentioned in the next section if they caused difficulties during the verifying experiments.

3.3.4 Model validation

To validate the model two tests series with different characteristic have been fulfilled. In both cases the same well-machinable AlMgSi1 aluminum alloy blank was cut with cooling by a two-fluted HSS ball-end cutter with 12 mm diameter, 30° nominal helix angle and 0° rake angle. The milling processes were performed on a Kondia B640 machining center with NCT control in the workshop of the Budapest University of Technology and Economics. The cutting forces were measured by a Kistler 9257B dynamometer. The sampling rate was set to 3000 Hz; +-1000 N measuring range and no filtering were chosen on the connected Kistler 5019 Multichannel Charge Amplifier (Figure 65).

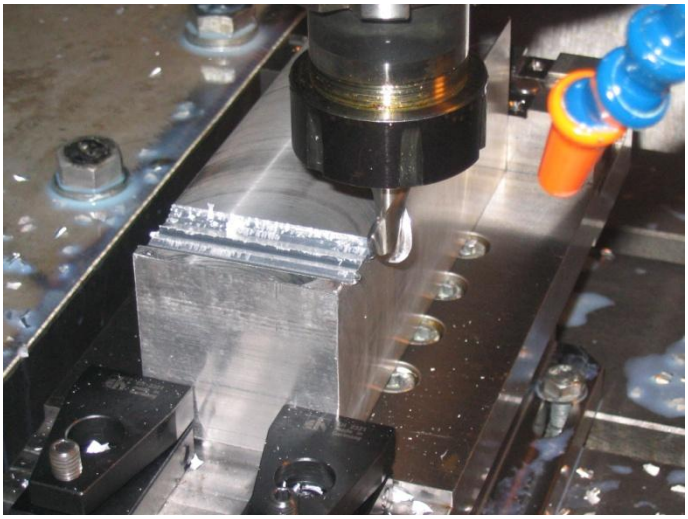

Figure 65. Experiment set-up

The GPGPU based machining simulation ran on the same workstation configuration that was used for the speed comparison tests detailed in Section 3.2.6.2; the hardware components are enumerated in Table 1. Owing to the highly parallelized execution, the performance of the simulation was very high: the cycle time of the visualization and Boolean calculations on about seven hundred thousand dexels together with the cutting force prediction and cutting force coefficient determination was about 0.01 second. For achieving the best accuracy, one turn of the tool was simulated in every cycle; in this case $\frac{1}{0.01} = 100$ spindle revolution per second (6000 RPM) could be simulated in real-time. Otherwise, if we do not calculate the forces in every simulation cycles, only at the third or fourth movement interpolation steps, the 10000-20000 RPM of high-speed machining can also be modeled. In spite of this the simulation was off-line, since the Dynoware software provided by Kistler did not allow instantaneous data processing: the measured force values were first stored and then they were read from a text file for the computations. Anyway, the simulation software perceived and computed as if direct connection had been established with the measuring system.

The configuration of the first test is shown on Figure 66. The cutting parameters of the machining section, which has been chosen as a typical operation element of an end-milling process along zigzag toolpath with a well-definable asymmetric contact area, are listed in Table 3.

Table 3. Cutting parameters

Spindle speed (S)	Cutting speed (v_c)	Feed rate (f)	Feed per tooth (z_f)	Radial depth of cut (a_e)	Axial depth of cut (a_p)
3200 rpm	120 m/min	512 mm/min	0.08 mm	6 mm	4 mm

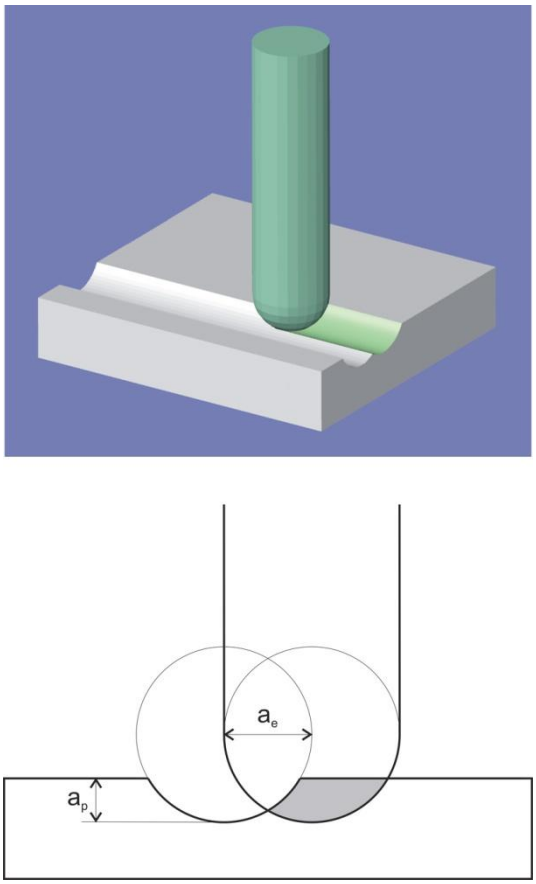

Figure 66. Configuration of the first test

The look-forward period was set to zero second first. This means that the same forces were predicted which had been just measured in the current simulation step. This method hasn't got practical importance but in this way we can validate the whole circle of the process of the force prediction: after we add the measured force values and the geometrical data into the **G** matrix and solve it, considering the measured and simulated data of every past point of time since the beginning of the experiment, we must get such predictions which are in good correspondence with the measured values at the recent point of time. Figure 67 shows the measured and predicted forces during the machining process. It can

be observed that the predicted force values flutter at the beginning due to the lack of the a priori knowledge of the coefficients. But after a very short time, which means a couple of tool revolutions, as enough information has been gained and consumed for the calculation of the coefficients, the curves of the estimated forces approach the curves of the measured forces.

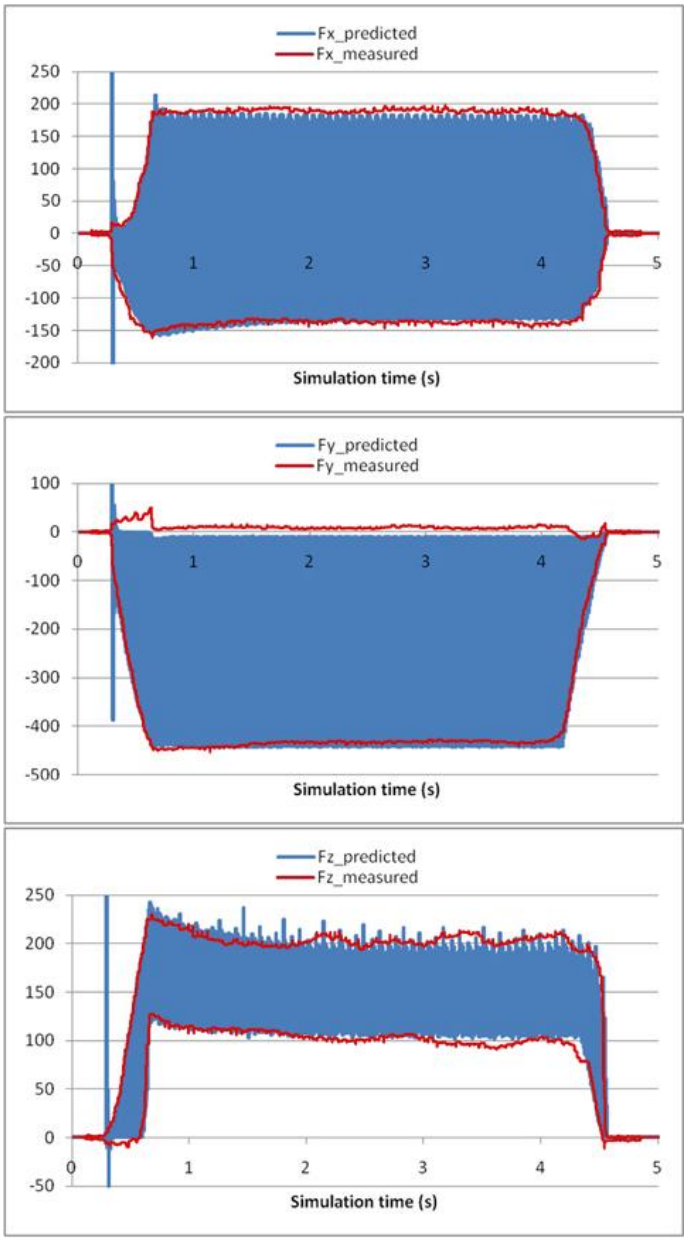

Figure 67. Results of the first experiment, with 0 s look-forward period

I chose the above machining configuration because it equals with the configurations that were commonly used for determining the accuracy of the mechanistic force model in the literature. Due to numerous references [90], [106], [108], [118], [130], the acceptable accuracy of the mechanistic model is about 15% in most cases relating to the peak values of the predicted and measured forces in the course of a simple plain milling with ball-end cutter, without considering cutter runout and deflection

that were not included in our experiment, and without applying special analytical calculations for cycloid tooth trajectory determination. Anyway, the uncertainty of the measuring – which appears on the figure too, in the shape of a slight slope at the beginning of the permanent part, especially at the z-component –, do not let the significant improvement of the accuracy neither with the use of more sophisticated methods [97]. On Figure 68, which has been created by the magnification of a short period on Figure 67, the measured and predicted instantaneous cutting forces can be seen during some revolution of the cutter. The figure shows that the cutting forces obtained by the fulfilled experiment are in good agreement with the earlier published results obtained using the mechanistic cutting force model, considering both the wave form fidelity and the accuracy of the peak values; and this is true for the whole machining process.

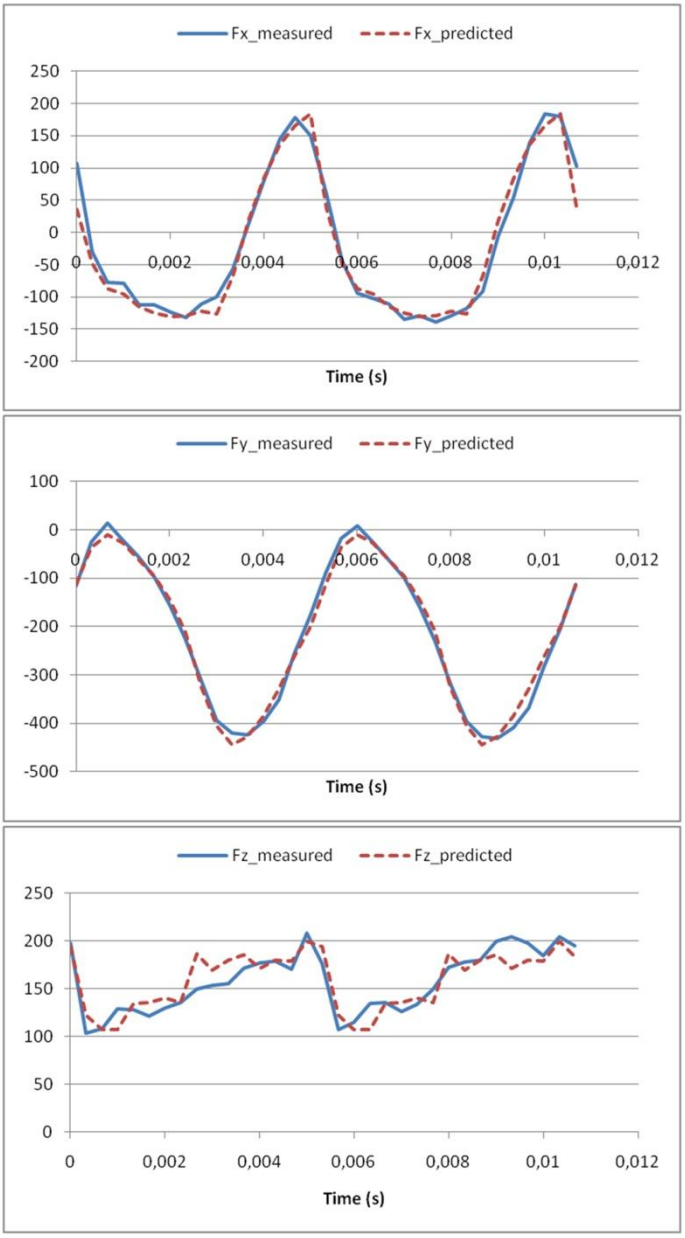

Figure 68. Instantaneous cutting force components

On Figure 69 the results of the simulations with 0.5 second and 1 second look-forward periods can be seen. In these cases the simulation preceded the real machining: the forces were predicted in advance to their measuring. For the sake of the good comparability the measured values are shifted left with the look-forward period on the figures, thus the predicted and measured values which belong together appear at the same point on the time axis. It can be observed that in the first stage of the experiments – according to the length of the look-forward period – the force prediction did not provide any results due to the lack of the measured values. After this period the predicted values shortly approach the measured ones. We can see that the shape of the predicted forces imitates the initial sloping characteristic of the measured ones for a while in the later points of time. That's because earlier measured values are used for the prediction in the present, and the peak forces were slightly higher earlier than later. Relying upon the shape of the curves we can observe that the forward-looking force prediction did not reduce the accuracy of the method.

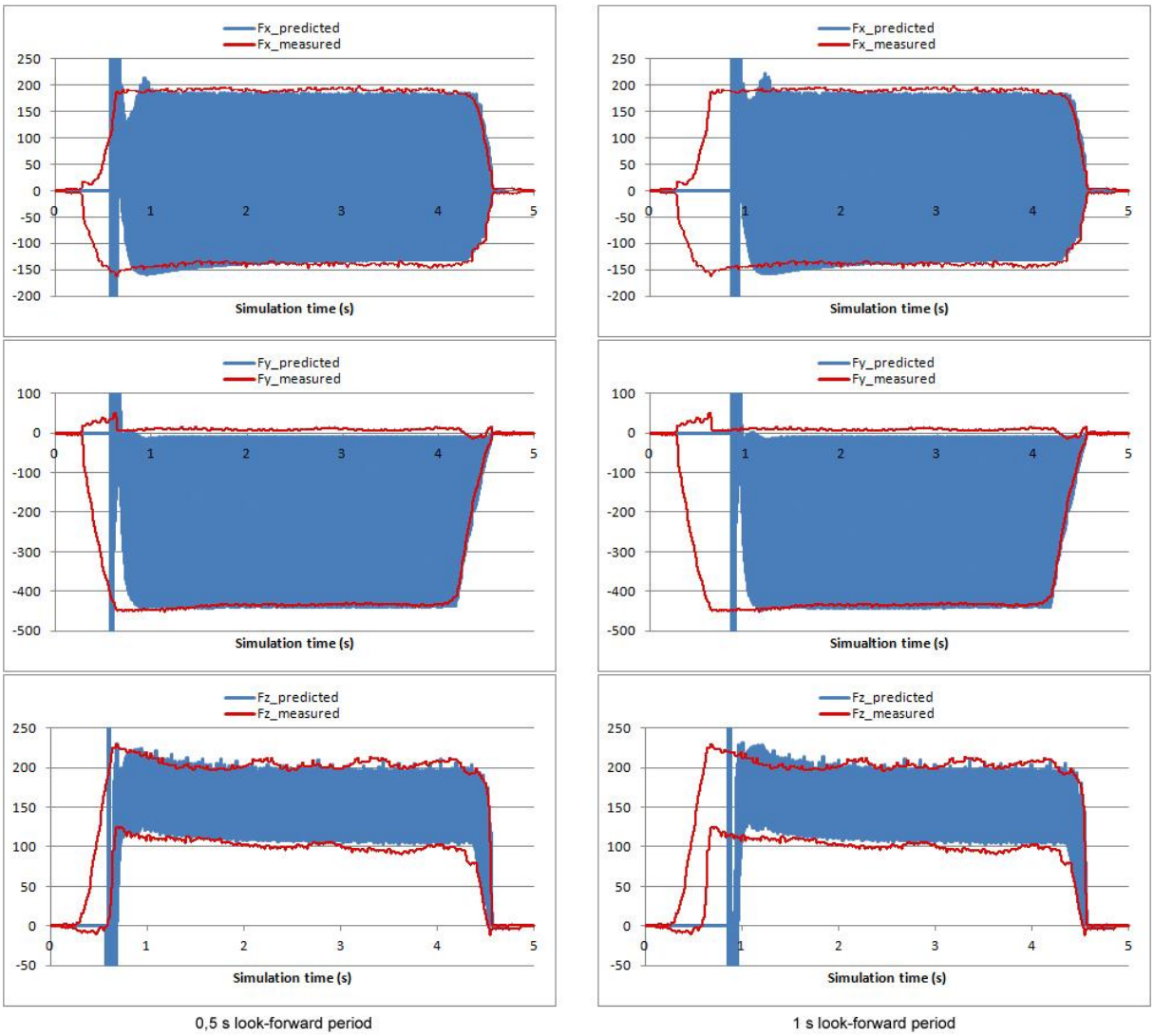

Figure 69. Results of the first experiment, with 0.5s and 1s look-forward period

For the second test the circumstances of machining of sculptured surfaces were created with a simple 3-axis configuration, where the cutting geometry continuously alters along the tool path. A slot with 4 mm maximum depth was milled into a cylindrical blank, at 45 degrees to its symmetry axis (Figure 70). The machining parameters remained the same as in the first test, listed in Table 3.

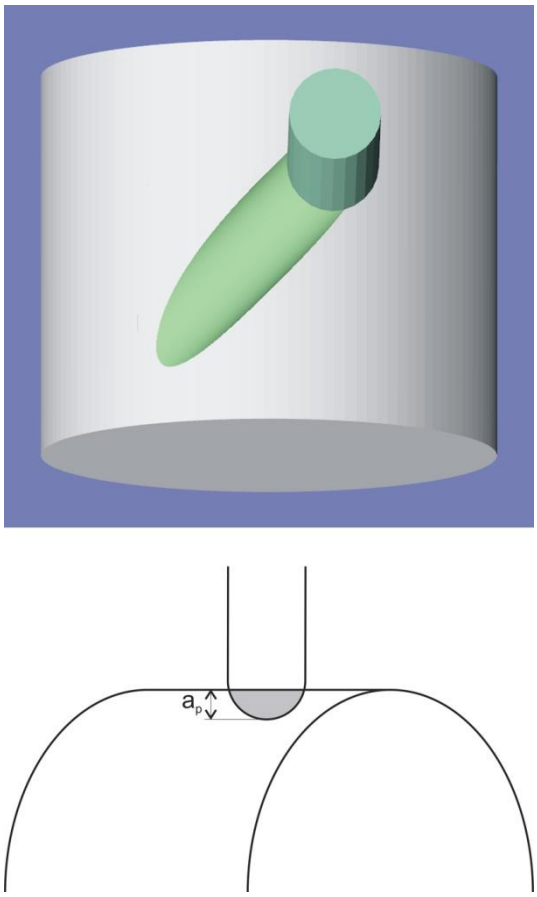

Figure 70. Configuration of the second test

On Figure 71 the results of the force prediction can be seen with zero-second look-forward period. The literature does not mention a common method to validate the cutting force predicting methods for such a complex cutting geometry that appears at the milling of sculptured surfaces; in many cases the authors are satisfied with "good agreement between the predicted and measured values". The reason is the varying accuracy during the machining process as compared with the simple plain machining tests, where the machining parameters are well-controllable and stable. That's why the critical machining sections, where inaccuracy can be expected, seem to be simply avoided in the validating tests. Most problems occur at the exact geometrical modeling of the continuously altering geometry. The force predicting methods require the knowledge of the exact tool position during the whole machining process. Our measurements showed that the difference between the programmed and real spindle

speeds was about 0.46%, while the real feed rate followed the programmed one with 2% accuracy. The cutting geometry greatly depends on the tool position, especially when the growth of the contact area rapidly changes. The referred authors – similarly to us – did not record real tool positions instead of using programmed, thus theoretical values. This brings an inaccuracy into the computations. However, the figure shows that the developed method can be used in the circumstances of machining of sculptured surfaces as well. Minor deviations occur at the *x* and *y* force components, but the envelope of the measured and predicted *z*-directional peak values indicate bigger differences. During the two test series, including experiments which did not bring extra information thus they are not enumerated in this text, the z-directional force components showed the biggest deviation from the expected values. This is not a unique occurrence; Lamikiz in [108] explains this phenomenon with a charge effect of the dynamometer, and mentions that most of the authors do not consider this component – which holds the least importance from the three – in the cutting force model. This allows me accepting the obtained results to validate the introduced method; adding that the trend of the measured and predicted *z*-directional force components does not show such a big difference which could query the validity of the method.

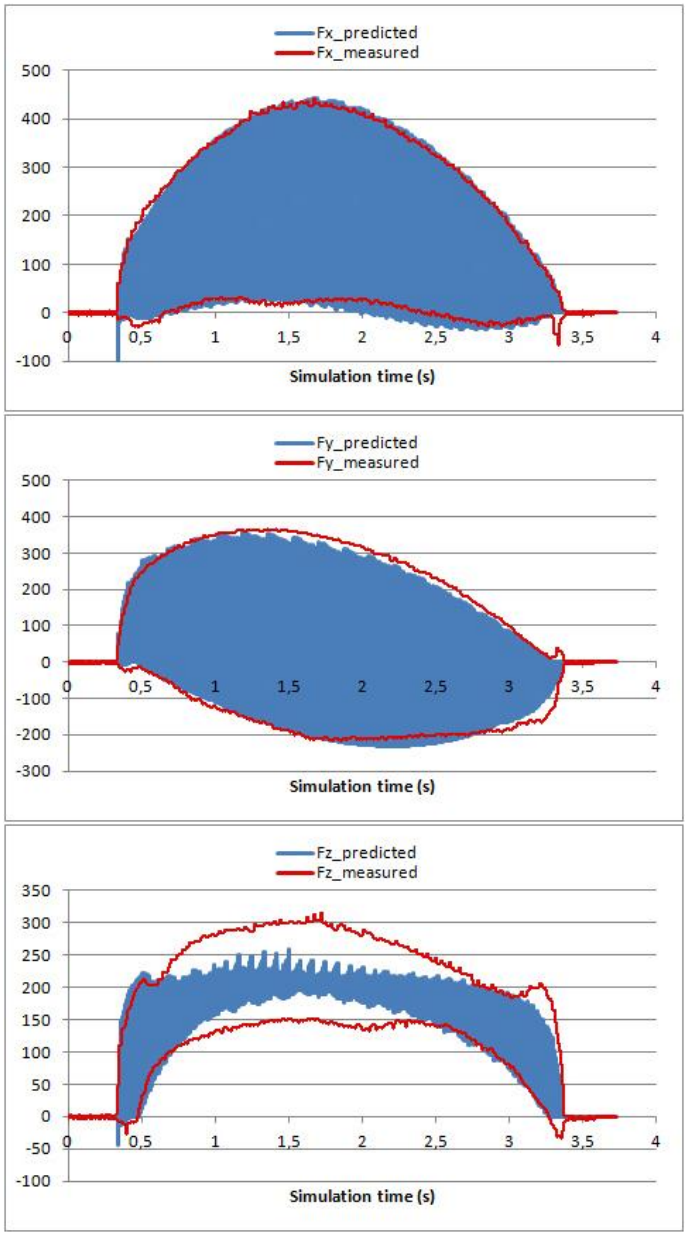

Figure 71. Results of the second experiment, with 0s look-forward period

On Figure 72 we can see the measured and predicted force components in the case of 0.5 second and 1 second look-forward periods. Similarly to the first test, the forward-looking does not decrease the accuracy of the method. In both cases the predicted and measured instantaneous force values are in strong correlation after the early "learning" period. The experimental results also prove – in agreement with the statement of the original mechanistic cutting force model – that the cutting force coefficients, which were determined and applied at different instantaneous cutting geometries, are valid to a wide range of machining conditions.

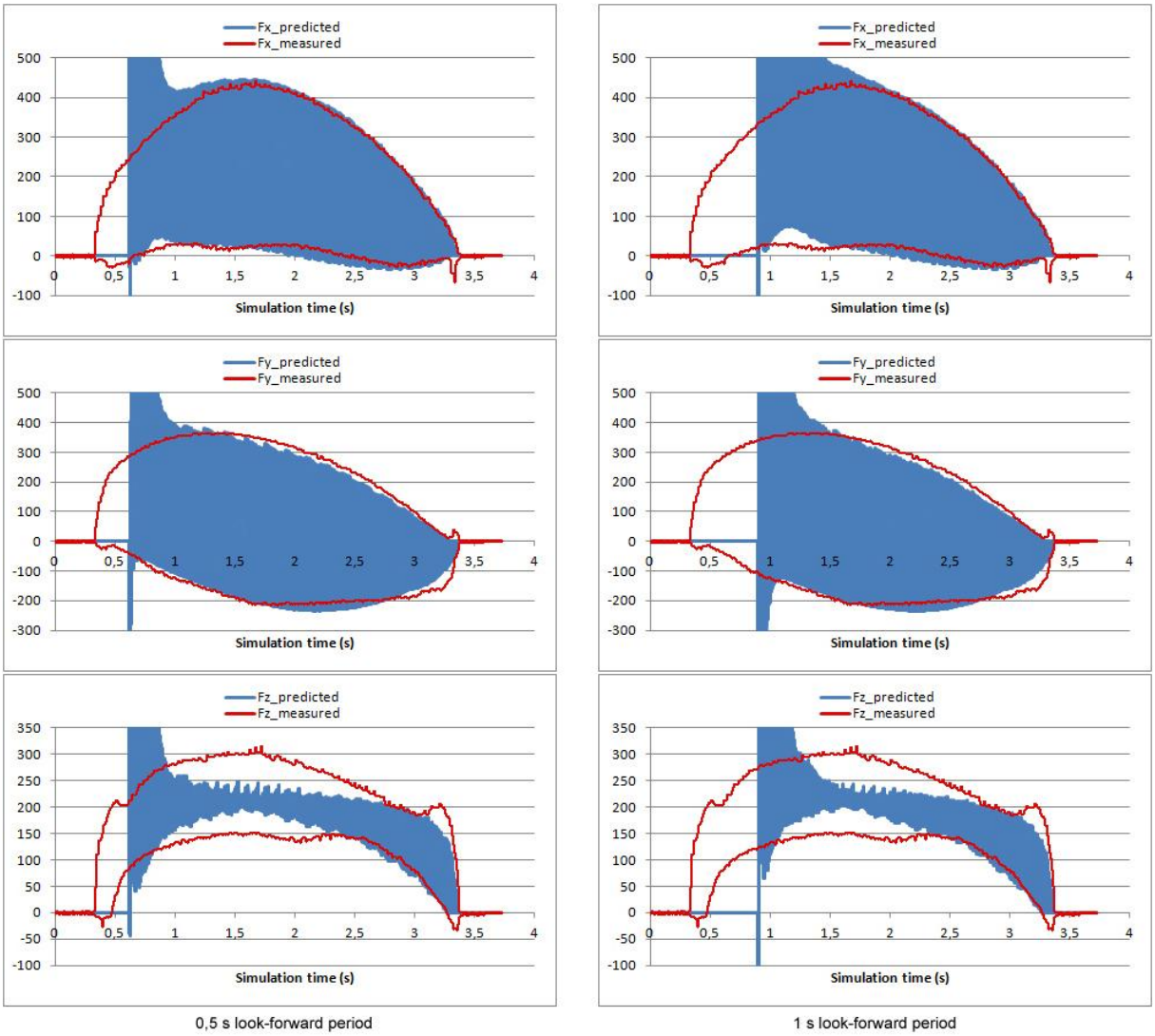

Figure 72. Results of the second experiment, with 0,5s and 1s look-forward period

4 Scientific results

4.1 Theses

According to the foregoing I sum up the scientific results of the fulfilled research work in the following theses:

Thesis 1

I have pointed out the shortcomings of the traditional CPU based multi-axis material removal simulation methods. I found that the two main reasons of the decreased performance as compared to the 3-axis simulation are the followings:

- Well parallelizable computations, where the same operations are performed on several independent data elements, are executed by the CPU, without exploiting the capacity that the graphics hardware offers in the course of a highly parallel program execution;
- Due to the big amount of data that is transferred from the host to the device memory to be displayed, the CPU-GPU data transfer bottleneck prevents the fast process visualization and notably slows down the simulation.

Relying upon these I proposed the redistribution of the computing tasks between the CPU and the GPU for the purpose of a more effective processing. I assigned the simulation steps that require SIMD like execution – namely the swept volume generation, the Boolean subtraction and the conversion of the multi-dexel representation to polyhedral surface model – to the GPU to be executed in a highly parallel manner. This solution also assures that no considerable data transfer happens outside the device memory, thus eliminating the disadvantageous effect of the data transfer bottleneck. In this way the same level of graphics hardware support has been reached in the case of a multi-axis material removal simulation technique that belonged only to the 3-axis simulation methods before. [A1][A2][A3][A6]

Thesis 2

I have adapted the multi-dexel based object description to fit the requirements of parallel processing by general-purpose graphics processing units, by creating a three-directional independent dexels set to be stored in vertex buffers and processed by GPU shaders. I let the graphics hardware into the creation of the initial dexel data by means of applying layered depth-normal images (LDNIs) for transitional representation of the object volume, thus making the fast rebuilding of a portion or the whole object available to be magnified with increased resolution. For the visualization I defined the way of reconstructing the object volume descriptor data set, composed by independent three-directional dexels, into displayable polyhedral surface. The method does not require an additional data reordering step that was essential in the earlier published techniques, thus makes the unbroken execution of the

simulation steps by the GPU possible, without CPU intervention and unnecessary CPU-GPU data transfer. [A3][A6]

Thesis 3

I have worked out a method of executing the volume subtraction operations of the material removal simulation in a parallel manner by the graphics hardware. Besides the traditional analytical calculations I created an image space based method in which LDNIs are applied for representing the cutter volume to be subtracted from the workpiece volume represented by multi-dexel structure, thus getting complex cutting tools and polygonal swept volumes involved into the simulation process. For serving the image space based volume subtraction process I developed a special GPU based swept surface generating algorithm that produces the LDNI representation of the swept volume of the moving cutter during its linearized movement between two consecutive cutter location points. With the help of the method the quality of the displayed machined surface can be effectively improved without the use of a traditional analytical swept volume generation technique. [A3][A4][A5][A6]

Thesis 4

I have reconstructed the equations of the mechanistic cutting force model in such a way that the geometrical information, which is gained from the GPGPU based material simulation in a highly parallelized manner for the sake of predicting the instantaneous cutting force components, is directly consumed from the multi-dexel structure on the whole cutter envelope, without the need of rearranging its data structure. This provides a general solution of the mechanical cutting force equations to milling tools with arbitrary shape, in the course of three or multi-axis machining processes, including the forming of sculptured surfaces. [A5][A7][A10]

Thesis 5

On the basis of the GPGPU based material simulation and cutting force prediction I have created a method for determining the cutting force coefficients acting in the force equations, which is free from the geometrical restrictions of the earlier published solutions, and it can be fulfilled in the course of a single machining experiment in real-time, in parallel to the modeled machining process. Relying upon these I have introduced a look-forward algorithm, with which the instantaneous cutting forces can be predicted in the course of milling processes, including multi-axis machining of freeform surfaces, without the need of the a priori knowledge of cutting force coefficients or other material properties, by means of measuring the orthogonal cutting forces and simulating the material removal process at the same time. [A7][A8][A9]

4.2 Usefulness of the results, directions of the future work

In respect to the usefulness the two main parts of the research work ought to be treated separately. The GPGPU based material removal simulation exploits the intensive development of the massively parallel hardware components, and the comparative diagrams clearly show that the difference in speed between the parallelized and the traditional CPU based execution further grows in the course of time. There is a couple of software companies in the world (e.g. MachineWorks in Sheffield, UK, or the German ModuleWorks), which has become specialized in writing graphical, modeling and simulation modules for the CAM and PLM system providers [131][132]. The introduced method can be easily inserted into these modules as an optional alternative for visualization and verification of multi-axis machining. Some of these companies have inquired for the presented solution – via the internet and in person –, so it has a good chance to find it in one of the CAM programs in the future. For safeguarding the intellectual product created during the research work, my employee, the University of Pécs, has begun a national and later on an international (PCT) patenting process, which are in an advanced state: the search report of the International Searching Authority has established the *novelty*, the *inventive step* and the *industrial applicability* of the intellectual product – so we can hopefully expect the early and successful closing of the lengthy patenting process.

In relation to the applicability of the GPGPU based cutting force prediction and especially the introduced look-forward algorithm more questions are raised. The mechanistic cutting force model is not used in CAM programs due to the detailed reasons. The only software package available on the market is CutPro [133], being developed with the leading of Yusuf Altintas, which offers cutting coefficient determination and instantaneous cutting force prediction for institutes and companies that deal with research and development in the field of manufacturing. The presented methods can facilitate the work of the users who need this kind of functionality. Nevertheless, if our aim is solving the validity problem of the coefficients with applying the look-forward algorithm, we must go beyond the off-line CAM systems. The simultaneous cutting force prediction and coefficient determination requires continuous – or at least frequent – cutting force measuring on the machine and real-time simulation in a connected computing unit. This can be a computer, but if we want to utilize the predicted values for the purpose of feed rate optimization, the simulation and force predicting functions must be integrated into the controller of the machine tool (Figure 73). The advantage of executing the force calculation on the controller is that some important information is still not available at the off-line state, but can be gained during the machining process. This means that the controller should take over such functions, which traditionally belong to the CAM software. Some of the newest CNC controllers include machining simulation module but only for visualization (see the solution of MachineWorks for Hurco machining centers [134]). The integration of the advanced functions may hide difficult software technology problems, as they must be merged into the

interpolator module, which has nowadays such sophisticated tasks like NURBS interpolation. In any case, the obtained results give a good basis for further researches in this field.

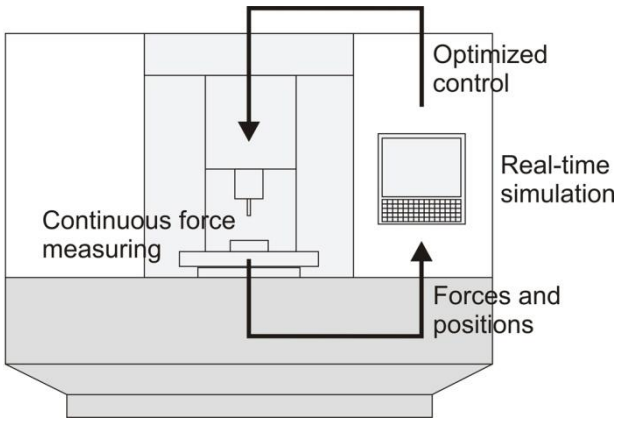

Figure 73. Advanced machine configuration

5 References

[1] Mastercam - Dynamic milling demonstration video. http://www.mastercam.com/Products/Enhancements/X5/simulation.wmv. Accessed on 7.8.2011.

[2] Chappel Y.T.: The use of vectors to simulate material removed by numerically controlled milling. Computer-Aided Design 15 (3), pp. 156–158, 1983.

[3] Jerard R.B., Hussaini S.Z., Drysdale R.L. Schaudt B.: Approximate methods for simulation and verification of numerically controlled machining programs. The Visual Computer 5 (6), pp. 329-348., 1989.

[4] Oliver J.H. Goodman E.D.: Direct dimensional NC verification. Computer-Aided Design 22 (1), pp. 3-10. , 1990.

[5] Oliver J.H.: Efficient intersection of surface normals with milling tool swept volumes for discrete three-axis NC verification. ASME Journal of Mechanical Design 114 (2), pp. 283-281. , 1992.

[6] Huang Y., Oliver J.H.: NC Milling Error Assessment and Tool Path Correction. Proceedings of the 21st annual Conference on Computer Graphics, pp. 287-294., 1997.

[7] Drysdale R.L., Jerard R.B., Schaudt B., Hauck K.: Discrete Simulation of NC Machining. Algorithmica Special Issue on Computational Geometry 4 (1), pp. 33-60., 1989.

[8] Jerard R.B., Angleton J.M., Drysdale R.L. and Su P.: The Use of Surface Points Sets for Generation, Simulation, Verification and Automatic Correction of NC Machining Programs. Proceedings of NSF Design and Manufacturing Systems Conference, pp 143 -148., 1990

[9] Anderson R.O.: Detecting and eliminating collisions in NC machining. Computer-Aided Design 10 (4), pp. 231-238., 1978.

[10] Wang W.P., Wang K.K.: Real time verification of multiaxis NC programs with raster graphics. Proceedings of IEEE, International Conference on Robotics and Automation, pp. 166–171., 1986.

[11] Nehéz Károly Róbert.: A marás számítógépes szimulációja és optimálási kérdései. Ph.D. dissertation, Miskolci Egyetem Gépészmérnöki Kar, 2002.

[12] Zhanga Y., Xub X., Liua Y.: Numerical control machining simulation: a comprehensive survey. International Journal of Computer Integrated Manufacturing 24 (7), pp. 593-609., 2011.

[13] Atherton P.R., Earl C. and Fred C.: A graphical simulation system for dynamic five-axis NC verification. Proceedings of Autofact, 2.1-2.12., 1987.

[14] Hook van T.: Real time shaded NC milling display. Computer Graphics 20 (4), pp. 15-20., 1986.

[15] Benouamer M. O., Michelucci D.: Bridging the gap between CSG and Brep via a triple ray representation. ACM Symposium on Solid Modeling and Applications, pp. 68-79., 1997.

[16] Muller H., Surmann T., Stautner M., Albersmann F., Weinert W.: Online Sculpting and Visualization of Multi-Dexel Volumes. ACM Symposium on Solid Modeling and Applications, pp. 258 - 261., 2003.

[17] Gossard D.C., Tsuchiya F.S.: Application of set theory to the verification of NC tapes. Proc. North American Metalworking Conference, April, 1978.

[18] Voelcker H.B., Hunt W.A.: The role of solid modeling in machining. Process Modeling and NC Verification, SAE Technical Paper 810195, 1981.

[19] Altintas Y., Spence A.: End milling force algorithms for CAD systems. CIRP Annals 40, pp. 31–34., 1991.

[20] Demo of constructive solid geometry tree. http://commons.wikimedia.org/wiki/File:Csg_tree.png. Accessed on 5.25.2011.

[21] Walstra W.H., Bronsvoort W.F., Vergeest J.S.M.: Interactive simulation of robot milling for rapid shape prototyping. Computers & Graphics 18 (6), pp. 861–871., 1994.

[22] Manner K.J.: Lecture notes for the Geometric Modeling for Engineering Applications course. http://homepages.cae.wisc.edu/~me232/lecture_notes/solid2.pdf. Accessed on 7.26.2011.

[23] Karunakaran K.P., Shringi R., Ramamurthi D., Hariharan C.: Octree-based NC simulation system for optimization of feedrate in milling using instantaneous force model. International Journal of Advanced Manufacturing Technology 46, pp. 465-490., 2010.

[24] Ensz M.T., Storti D.W. and Ganter M.A.: Implicit Methods for Geometry Creation. International Journal of Computational Geometry and Applications 8 (5-6), pp. 509-536, 1998.

[25] Kawashima, Y., Itoh K., Ishida T.: A flexible quantitive method for NC machining verification using a space-division based solid model. The Visual Computer 7, pp. 149-157. , 1991.

[26] Ayala D., Brunet P., Juan R., Navazo I.: Object representation by means of nonminimal division quadtrees and octrees. ACM Transactions on Graphics 4 (1), pp. 41–59., 1985.

[27] Brunet P., Navazo I.: Solid representation and operation using extended octrees. ACM Transactions on Graphics 9 (2), pp. 170–197., 1990.

[28] Roy U., Xu Y.: Computation of geometric model of a machined part from its NC machining program. Computer-Aided Design 31, pp. 401–411., 1999.

[29] Mounayri H.E.I., Spence A.D., Elbestawi M.A.: Milling process simulation—a generic solid modeller based paradigm. Journal of manufacturing science and engineering, Transactions of the ASME 120, pp. 213–221., 1998.

[30] Fleisig R.V., Spence A.D.: Techniques for accelerating B-Rep based parallel machining simulation. Computer-Aided Design 37, pp. 1229–1240., 2005.

[31] Laure A, Altintas Y.: Simulation of flank milling processes. International Journal of Machine Tools & Manufacture 40, pp. 549-559., 2005.

[32] Salami R, Sadegi M.H., Motakef B.: Feedrate optimization for 3-axis ball-end milling of sculptured surfaces. International Journal of Machine Tools & Manufacture 47, pp. 760-767., 2007.

[33] Fleischmann P.: Mesh generation for technology CAD in three dimensions. Ph.D. dissertation, Technischen Universität Wien, 1999.

[34] Jamieson R., Hacker H.: Direct slicing of CAD models for rapid prototyping. Rapid Prototyping Journal 3, pp. 12-19., 1995.

[35] Wang C. C. L., Leung Y.-shan, Chen Y.: Computer-Aided Design Solid modeling of polyhedral objects by Layered Depth-Normal Images on the GPU. Computer-Aided Design 42 (6), pp. 535-544, 2010.

[36] Chen Y., Wang C.C.L.: Layered depth-normal images for complex geometries - part one: Accurate sampling and adaptive modeling. Proceedings of ASME IDETC/CIE, pp. 717-728., 2008.

[37] Wang C.C.L., Chen Y.: Layered depth-normal images for complex geometries - part two: manifold-preserved adaptive contouring. Proceedings of ASME IDETC/CIE, pp. 729-739., 2008.

[38] Ren Y., Lai-Yuena S.K., Lee Y-S.: Virtual prototyping and manufacturing planning by using tri-dexel models and haptic force feedback. Virtual and Physical Prototyping 1 (1), pp. 3-18., 2006.

[39] Lorensen W.E., Cline H.E.: Marching Cubes: A high resolution 3D surface construction algorithm. Computer Graphics 21 (4), pp. 163-169., 1987.

[40] Ju T., Losasso F., Schaefer S., Warren J.: Dual contouring of hermite data. ACM Transactions on Graphics 21 (3), pp. 339–346, 2002.

[41] Kobbelt L., Botsch M., Schwanecke U., Seidel H.-P.: Feature Sensitive Surface Extraction from Volume Data. Proceedings of SIGGRAPH 2001, pp. 57-66., 2001.

[42] Ren Y., Zhu W., Lee Y-S.: Feature conservation and conversion of tri-dexel volumetric models to polyhedral surface models for product prototyping. Computer-Aided Design and Applications, 5 (6), pp. 932-941., 2008.

[43] Muller H., Surmann T., Stautner M., Albersmann F., Weinert K.: Online sculpting and visualization of multi-dexel volumes. Proceedings of ACM SM'03, pp. 258-261., 2003.

[44] Geiss R.: Generating Complex Procedural Terrains Using the GPU. GPU Gems 3, Part I: Geometry. [online] http://http.developer.nvidia.com/GPUGems3/gpugems3_ch01.html. Accessed on 8.1.2011.

[45] Du S., Surmann T., Webber O., Weinert K.: Formulating swept profiles for five-axis tool motions. International Journal of Machine Tools & Manufacture 45, pp. 849-861., 2005.

[46] Frey D.D., Otto K.N., Pflager W.: Swept envelopes of cutting tools in integrated machine and workpiece error budgeting. Annals of CIRP 46 (1), pp. 475-480., 1997.

[47] Aras E.: Generating cutter swept envelopes in five-axis milling by two-parameter families of spheres. Computer-Aided Design 41, pp. 95-105., 2009.

[48] Wang W.P., Wang K.K.: Geometric modeling for swept volume of moving solids. IEEE Computer Graphics and Applications 6(12), pp. 8-17., 1986

[49] Abdel-Malek K., Yeh H-J.: Geometric representation of the swept volume using Jacobian rank-deficiency conditions. Computer Aided Design 29 (6), pp. 457-468., 1997.

[50] Blackmore D., Leu M.C., Wang L.P.: The sweep-envelope differential equation algorithm and its application to NC-machining verification. Computer Aided Design 29 (9), pp. 629-637., 1997.

[51] Chung Y.C., Park J.W., Shin H., Choi B.K.: Modeling the surface swept by a generalized cutter for NC verification. Computer Aided Design 30 (8), pp. 587-594., 1998.

[52] Sheltami K., Bedi S., Ismail F.: Swept volumes of toroidal cutters using generating curves. International Journal of Machine Tools and Manufacture 38, pp. 855-870., 1998.

[53] Roth D., Bedi S., Ismail F., Mann S.: Surface swept by a toroidal cutter during 5-axis machining. Computer Aided Design 33 (1), pp. 57-63., 2001.

[54] Mann S, Bedi S. Generalization of the imprint method to general surfaces of revolution for NC machining. Computer Aided Design 34, pp. 373-408., 2002.

[55] Chiou C-J., Lee Y-S.: Swept surface determination for five-axis numerical control machining. International Journal of Machine Tools and Manufacture 42, pp. 1497-1507., 2002.

[56] Weinert K., Du S-J., Damm P., Stautner M.: Swept volume generation for the simulation of machining processes. International Journal of Machine Tools and Manufacture 44 (6), pp. 617-628., 2004.

[57] Edwards W.R.V., Lin F.: Machining simulation method and apparatus, US Patent No. 6,862,560., 2005.

[58] Review: NVIDIA GeForce GTX 680 2GB graphics card. http://hexus.net/tech/reviews/graphics/36509-nvidia-geforce-gtx-680-2gb-graphics-card/?page=3. Accessed on 4.12.2012.

[59] NVIDIA CUDA C Programming Guide Version 3.2., 10/22/2010., http://developer.nvidia.com., Accessed on 8.8.2011.

[60] Technical Brief - NVIDIA GeForce 8800 GPU Architecture Overview, November 2006, www.nvidia.com/object/IO_37100.html. Accessed on 8.8.2011.

[61] Liu W., Schmidt B., Voss G., Müller-Wittig W.: Molecular Dynamics Simulations on Commodity GPUs with CUDA. Lecture Notes in Computer Science 4873, pp. 185-196., 2007.

[62] Manavski S.A.: Cuda compatible GPU as an efficient hardware accelerator for AES cryptography. Proceedings of IEEE International Conference on Signal Processing and Communication, pp.65-68., 2007.

[63] Nyland L., Harris M., Prins J.: Fast N-Body simulation with CUDA. GPU Gems 3, Addison Wesley, pp. 677–795., 2007.

[64] Rodrigues C.I., Hardy D.J., Stone J.E., Schulten K., Hwu W-M.W.: GPU acceleration of cutoff pair potentials for molecular modeling applications. Proceedings of the 2008 Conference on Computing Frontiers, pp. 273-282., 2008.

[65] General-Purpose Computation on Graphics Hardware. http://www.gpgpu.org. Accessed on 4.12.1012.

[66] Lysenko M., D'Souza R.M.: Interactive machinability analysis of free-form surfaces using multiple-view image space techniques on the GPU. Robotics and Computer-Integrated Manufacturing 26, pp. 703–710., 2010.

[67] Goetz F., Junklewitz T., Domik G.: Real-Time Marching Cubes on the Vertex Shader. EUROGRAPHICS 2005 Short Presentations, 2005.

[68] Johansson G., Carr H.: Accelerating Marching Cubes with Graphics Hardware. Proceedings of CASCON '06, doi.:10.1145/1188966.1189018., 2006.

[69] Oberg E., Jones F.D., Ryffel H.H., McCauley C.J., Heald R.M., Hussain M.I.: Machinery's Handbook, 28th Edition. Industrial Press, ISBN 0-8311-2700-7, 2008.

[70] Bailey T., Elbestawi M.A., El-Wardany T.I., Fitzpatrick P.: Generic simulation approach for multi-axis machining, part 2: model calibration and feedrate scheduling. Journal of Manufacturing Science and Engineering 124, pp. 634–642., 2002.

[71] Makhanov S.: Optimization and correction of the tool path of the five-axis milling machine Part 1. Spatial optimization. Mathematics and Computers in Simulation, 75, pp. 210–230., 2007.

[72] Makhanov S.: Optimization and correction of the tool path of the five-axis milling machine Part 2: Rotations and setup. Mathematics and Computers in Simulation, 75, pp. 231-250., 2007.

[73] Oysu C., Bingul Z.: Application of heuristic and hybrid-GASA algorithms to tool-path optimization problem for minimizing airtime during machining. Engineering Applications of Artificial Intelligence 22 (3), pp. 389-396., 2009.

[74] Castagnetti C., Duc E., Ray P.: The Domain of Admissible Orientation concept: A new method for five-axis tool path optimisation. Computer-Aided Design 40 (9), pp. 938-950., 2008.

[75] Lauwers B., Dejonghe P., Kruth J.P.: Optimal and collision free tool posture in five-axis machining through the tight integration of tool path generation and machine simulation Computer-Aided Design 35 (5), pp. 421-432., 2003.

[76] López de Lacalle L.N., Lamikiz A., Sánchez J.A., Salgado M.A.: Toolpath selection based on the minimum deflection cutting forces in the programming of complex surfaces milling. International Journal of Machine Tools & Manufacture 47, pp. 388–400., 2007.

[77] Lazoglu I., Manav C., Murtezauglu Y.: Tool path optimization for free form surface machining. CIRP Annals - Manufacturing Technology 58, pp. 101–104., 2009.

[78] Wang W.P.: Solid modeling for optimizing metal removal of three-dimensional NC end milling. Journal of Manufacturing Systems 7(1), pp. 57–65., 1998.

[79] Karunakaran K.P., Shringi R.: A solid model-based off-line adaptive controller for feedrate scheduling for milling process. Journal of Materials Processing Technology 204(1–3), pp. 384–396., 2008.

[80] Kaymakci M., Lazoglu I., Murtezaoglu Y.: Machining of complex sculptured surfaces with feedrate scheduling. International Journal of Manufacturing Research 1(2), pp. 157–175., 2006.

[81] Kurt M., Bagci E.: Feedrate optimisation/scheduling on sculptured surface machining: a comprehensive review, applications and future directions. The International Journal of Advanced Manufacturing Technology 55, pp. 1037–1067., 2011.

[82] Yazar Z., Koch K-F., Merrick T., Altan T.: Feedrate optimization based on cutting force calculations in-axis milling of dies and molds with sculptured surfaces. International Journal of Machine Tools and Manufacture 34 (3), pp. 365–377., 1994.

[83] Ko J.H., Yun W.S., Cho D-W.: Off-line feedrate scheduling using virtual CNC based on an evaluation of cutting performance. Computer-Aided Design 35, pp. 383–393., 2003.

[84] Guzel B.U., Lazoglu I., Increasing productivity in sculptured surface machining via off-line piecewise variable feedrate scheduling based on the force system model. International Journal of Machine Tools and Manufacture 44, pp. 21–28., 2004.

[85] Erdim H., Lazoglu I., Ozturk B.: Feedrate scheduling strategies for free-form surfaces. International Journal of Machine Tools & Manufacture 46, pp. 747–757., 2006.

[86] Martellotti M.E.: An analysis of the milling process, Transactions of the ASME 63, pp. 667-700., 1941.

[87] Koenigsberger F., Sabberwal A.J.P.: An investigation of the cutting force pulsations during the milling process. International Journal of Machine Tool Design and Research 1, pp. 15-33., 1961.

[88] Yang M.Y., Park H.D.: The prediction of cutting force in ball end milling. International Journal of Machine Tools & Manufacture 31 (1), pp. 45–54., 1991.

[89] Tai C.C., Fuh K.H.: A predictive force model in ball-end milling including eccentricity effects. International Journal of Machine Tools & Manufacture 34 (7), pp. 959-979., 1994.

[90] Lee P., Altintas Y.: Prediction of ball-end milling forces from orthogonal cutting data. International Journal of Machine Tools & Manufacture 36, pp. 1059-1072., 1996.

[91] Yucesan G., Altintas Y.: Mechanics of ball end milling process. ASME Winter Annual Meeting PED 64, pp. 543–51., 1993.

[92] Altintas Y., Engin S., Generalized modeling of mechanics and dynamics of milling cutters. CIRP Annals - Manufacturing Technology, 50 (1), pp. 25-30., 2001.

[93] Omar O.E.E.K., El-Wardany T., Ng E., Elbestawi M.A.: An improved cutting force and surface topography prediction model in end mining, International Journal of Machine Tools & Manufacture 47, pp. 1263-1275., 2007.

[94] Li H.Z., Liu K., Li X.P.: A new method for determining the undeformed chip thickness in milling. Journal of Materials Processing Technology 113, pp. 378–384., 2001.

[95] Kumanchik L.M., Schmitz T.L.: Improved analytical chip thickness model for milling. Precision Engineering 31, pp. 317–324., 2007.

[96] Sai L., Bouzid W., Zghal A.: Chip thickness analysis for different tool motions: for adaptive feedrate. Journal of Materials Processing Technology 204, pp. 213–220., 2008.

[97] Sun Y., Ren F., Guo D., Jia Z.: Estimation and experimental validation of cutting force in ball-end milling of sculptured surfaces. International Journal of Machine Tools and Manufacture 49, pp. 1238-1244., 2009.

[98] Schmitz T.L., Couey J., Marsh E., Mauntler N., Hughes D.: Runout effects in milling: surface finish, surface location error and stability. International Journal of Machine Tools and Manufacture 47, pp. 841–851., 2007.

[99] Desai K.A., Agarwal P.K., Rao P.V.M.: Process geometry modeling with cutter runout for milling of curved surfaces. International Journal of Machine Tools and Manufacture 49, pp. 1015-1028., 2009.

[100] Sutherland J.W., DeVor R.E.: An improved method for cutting force and surface error prediction in flexible end milling systems. ASME Journal of Engineering for Industry 108, pp. 268–279., 1986.

[101] Feng H.Y., Menq C.H.: A flexible ball-end milling system model for cutting force and machining error prediction. Journal of Manufacturing Science and Engineering 118, pp. 461–469., 1996.

[102] Kim G.M., Kim B.H., Chu C.N.: Estimation of cutter deflection and form error in ball-end milling processes. International Journal of Machine Tools & Manufacture 43, pp. 917-924., 2003.

[103] Omar O.E.E.K., El-Wardany T., Ng E., Elbestawi M.A.: An improved cutting force and surface topography prediction model in end mining. International Journal of Machine Tools & Manufacture 47, pp. 1263-1275., 2007.

[104] Ozturk E., Budak E.: Modeling of 5-axis milling processes. Machining Science and Technology 11 (3), pp. 287-311., 2007.

[105] Kline W.A., DeVor R.E.: The effect of runout on cutting geometry and forces in end milling. International Journal of Machine Tool Design and Research 23, pp. 123-140., 1983.

[106] Kim G.M., Cho P.J., Chu C.N.: Cutting force prediction of sculptured surface ball-end milling using Z-map. International Journal of Machine Tools & Manufacture 40, pp. 277-291., 2000.

[107] Roth D., Gray P., Ismail F., Bedi S.: Mechanistic modeling of 5-axis milling using an adaptive and local depth buffer. Computer-Aided Design 39, pp. 302-312., 2007.

[108] Lamikiz A., Lopez de Lacalle L.N., Sanches J.A., Bravo U.: Calculation of the specific cutting coefficients and geometrical aspects in sculptured surface machining. Machining Science and Technology 9, pp. 411-436., 2005.

[109] Feng H.Y., Menq C.H.: The prediction of the cutting forces in the ball-end milling process - I. Model of formulation and model building procedure. International Journal of Machine Tools and Manufacture 34, pp. 697–710., 1994.

[110] Feng H.Y., Menq C.H.: The prediction of the cutting forces in the ball-end milling process - II. Cut geometry analysis and model verification. International Journal of Machine Tools and Manufacture 34, pp. 711–720., 1994.

[111] Lamikiz A., López de Lacalle L.N., Sánches J.A., Salgado M.A.: Cutting force estimation in sculptured surface milling. International Journal of Machine Tools & Manufacture 44, pp. 1511-1526., 2004.

[112] Kim G.M., Chu C.N.: Mean cutting force prediction in ball-end milling using force map method. Journal of Material Processing Technology 146, pp. 303-310., 2004.

[113] Azeem A., Feng H-Y., Wang L.: Simplified and efficient calibration of a mechanistic cutting force model for ball-end milling. International Journal of Machine Tools & Manufacture 44, pp. 291-298., 2004.

[114] Fussel B.K., Jerard R.B., Hemmett J.G.: Modeling of cutting geometry and forces for 5-axis sculptured surface machining. Computer-Aided Design 35, pp. 333-346., 2003.

[115] Kline W.A., DeVor R.E., Lindberg R.: The prediction of cutting forces in end milling with application to cornering cuts. International journal of machine tool design & research 22, pp. 7–22., 1982.

[116] Boothroyd G., Knight W.A.: Fundamentals of Machining and Machine Tools. Marcel Dekker, 1989.

[117] Ko J.H., Yun W-S., Cho D-W., Ehmann K.F.: Development of a virtual machining system, part 1: approximation of the size effect for cutting force prediction. International Journal of Machine Tools & Manufacture 42, pp. 1595-1605., 2002.

[118] Milfelner M., Cus F.: Simulation of cutting forces in ball-end milling. Robotics and Computer Integrated Manufacturing 19, pp. 99-106., 2003.

[119] Wang J.-J.J., Zheng C.M.: Identification of shearing and ploughing cutting constants from average forces in ball-end milling. International Journal of Machine Tools & Manufacture 42, pp. 695-705., 2002.

[120] Armarego E.J.A., Deshpande N.P.: Computerized end-milling force predictions with cutting models allowing for eccentricity and cutter deflections. Annals of the CIRP 40, pp. 25–29., 1991.

[121] Yucesan G., Altintas Y.: Improved modeling of cutting force coefficients in peripheral milling. International Journal of Machine Tools and Manufacture 34 (4), pp. 473–487., 1994.

[122] Budak E., Altintas Y., Armarego E.J.A.: Prediction of milling force coefficients from orthogonal cutting data. ASME Journal of Manufacturing Science and Engineering 118, pp. 216–224., 1996.

[123] Yun W.S., Cho D.W.: An improved method for the determination of 3D cutting force coefficients and runout parameters in end milling. International Journal of Advance Manufacturing Technology 16, pp. 851–858., 2000.

[124] Choudhury S. K., Rath S.: In-process tool wear estimation in milling using cutting force model, Journal of Materials Processing Technology 99, pp. 113-119., 2000.

[125] Lee J.G., Cho Y.H., Yang S.J., Park J.W.: Near Net-shape Five-axis CL Data Generation by Considering Tool Swept Surface in Face Milling of Sculptured Surface. Computer-Aided Design & Applications 5(1-4), pp. 442-451., 2008.

[126] Kirk D.B., Hwu W-m. W.: Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann – Elsevier, Burlington, USA, ISBN: 978-0-12-381472-2, 2010.

[127] Microsoft Co.: Pipeline stages (DirectX 10) – Direct3D 10 Graphics online documentation. http://msdn.microsoft.com/en-us/library/bb205123%28v=VS.85%29.aspx Accessed on 9.26.2011.

[128] NVIDIA Corporation.: Technical Brief – NVIDIA GeForce 8800 GPU Architecture Overview, 2006.

[129] Nakamoto K., Tsunoda M., Shirase K., Moriwaki T.: Rapid removal volume acquisition for tool posture decision in 3+2-axis control milling. Journal of Advanced Mechanical Design, Systems, and Manufacturing, 2 (4), pp. 464-473., 2008.

[130] Salami Naserian R., Sagedi M.H., Haghighat H.: Static rigid force model for 3-axis ball-end milling of sculptured surfaces. International Journal of Machine Tools and Manufacture 47, pp. 785-792., 2007.

[131] MachineWorks Ltd. http://www.machineworks.com. Accessed on 12.1.2011.

[132] ModuleWorks GmbH. http://www.moduleworks.com. Accessed on 12.1.2011.

[133] Manufacturing Automation Laboratories Inc. http://www.malinc.com. Accessed on 12.1.2011.

[134] WinMax CNC controller of Hurco with machining visualization http://www.hurco.de/service/oavengl/artikel.asp?lnr=60&gkat=3&kat=3. Accessed on 12.1.2011.

6 Publications of the author

[A1] Tukora B., Szalay T.: High performance computing on graphics processing units. Pollack Periodica 3(2), pp. 27-34., 2008.

[A2] Tukora B., Szalay T.: Manufacturing Simulation in the Light of the Recent GPU Architecture. Proceedings of Sixth Conference on Mechanical Engineering, Paper C20., 2008.

[A3] Tukora B., Szalay T.: Fully GPU-based volume representation and material removal simulation of free-form objects. Innovative Developments in Design and Manufacturing: Advanced research in virtual and rapid prototyping, pp. 609-614., 2009.

[A4] Tukora B., Szalay T.: LDNI alapú söpört térfogat generálása anyageltávolítás jellegű gépészeti szimulációhoz. V. Magyar Számítógépes Grafika és Geometria Konferencia, pp. 193-197., 2010.

[A5] Tukora B., Szalay T.: GPGPU-based Material Removal Simulation and Cutting Force Estimation. Proceedings of the Seventh International Conference on Engineering Computational Technology, Paper 17., 2010.

[A6] Tukora B., Szalay T.: Az új generációs grafikus hardverek bevonása a megmunkálás-szimulációs eljárásokba. Gép LXI (4), pp. 35-40., 2010.

[A7] Tukora B., Szalay T.: Real-time Cutting Force Prediction and Cutting Force Coefficient Determination during Machining Processes. Advanced Materials Research: Modelling of Machining Operations, pp. 85-92., 2011.

[A8] Tukora B., Szalay T.: Cutting force prediction with adaptive method in the course of milling processes. Proceedings of Factory Automation 2011, pp. 89-92., 2011.

[A9] Tukora B., Szalay T.: Real-time determination of cutting force coefficients without cutting geometry restriction. International Journal of Machine Tools and Manufacture 51, pp. 871-879., 2011.

[A10] Tukora B., Szalay T.: Multi-dexel based material removal simulation and cutting force prediction with the use of general-purpose graphics processing units, Advances in Engineering Software 43, pp. 65-70., 2012.