Title
A Unified Approach for Integrated Computer-Aided Design and Manufacturing

Author
Huang, Bin

Publication Date
2013

Peer reviewed|Thesis/dissertation

University of California
Los Angeles

A Unified Approach for Integrated Computer Aided Design and Manufacturing

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Mechanical Engineering

By
Bin Huang

2013

ABSTRACT OF THE DISSERTATION

A Unified Approach for Integrated Computer Aided Design

and Manufacturing

by

Bin Huang

Doctor of Philosophy in Mechanical Engineering

University of California, Los Angeles, 2013

Professor Daniel C.H. Yang, Chair

Current research is targeted on automation of the computer aided process planning based on a CAM oriented model. Instead of bonding the CAM data directly on top of the CAD model, a CAM oriented model separates the manufacturing related information from the design process and translate the design model to a set of machining features that is semantically compatible with the CAM operations. With parameterization of the CAM oriented model, another objective of this research is to improve the efficiency and accuracy of current integration system.

Motivated by this goal, there are four relevant topics of this concern are addressed in this research: 1). A data exchange scheme of CAD/CAM integration based on CAM

oriented model. 2). Data representation and architecture design for CAM oriented model. 3). Process planning of roughing machining based on morphing theorems and parameterization of CAM oriented model. 4). A parameterization technology for tool path generation and verification.

To show the feasibility to fully automate process planning and tool path generation with the CAM oriented model, the morphing based theorem is applied to modeling the manufacturing process particularly for machining process. The mathematical model of machining process enables the automation of process planning for the rough operation. Meanwhile the parameterization technology of tool path generation is exploited particularly to for the machining feature with tessellated geometry.

The dissertation of Bin Huang is approved

Jenn-Ming Yang

Rajit Gadh

Tsu-Chin Tsao

Daniel C. H. Yang, Committee Chair

University of California, Los Angeles

2013

Dedication

To My Sweethearts

Benjamin H. Huang and

Rachel X. Huang

Table of Contents

Contents

Acknowledgments

I would like to express my appreciation for the generous helps of many people, without whom it would be impossible for me to accomplish this thesis. First, I would like to thank Professor Daniel C.H. Yang for his great supervision, support and encouragement during the last few years. I would also like to thank all the faculty members of my PHD committee, Professor Jenn-Ming Yang, Professor Petros Faloutsos, Professor Rajit Gadh and Professor Tsu-Chin Tsao for their supports, patience and time to review and share the informative discussion with me. My gratitude is extended to staffs and fellow students of the Department of Aerospace and Mechanical Engineering in UCLA for their help and support. The innumerable discussions have created a stimulating and enjoyable atmosphere to work in. I would like to express my special gratitude to Dr. Chen-han Lee, the former senior research director in Siemens PLM. Thanks for the many valuable discussions during the completion of this thesis and for the companionship in the journey towards the Ph.D. degree.

Last but not least I would like to express the warmest thanks to my lovely parents, Youjing Huang and Yaxi Xiong for their constant support and encouragement. A special gratitude should be forwarded to my wife, Yan Xiong, for her endless spiritual support and accompany along these harshest years in my life.

Los Angeles in March 2013

Bin Huang

VITA

1993- 1997	B.S Mechanical Engineering Shanghai Jiaotong University, P.R.China
	B.S. Science of Business Administration Shanghai Jiaotong University, P.R.China
1997- 2000	Assistant Engineer Shanghai Coal Science Institution, P.R.China
2001- 2004	PH.D. Candidate Mechanical Engineering M. S. Mechanical Engineering University of California, Los Angeles, USA
2005- 2006	Intern Touchdown Technologies Inc. USA
2007 - 2008	Software Engineer, Digital Technology Laboratory, Mori seiki Inc., USA
2009 - 2011	Advanced Software Engineer Siemens PLM Inc., USA
2012 - 2013	Senior Software Engineer / System Architecture, CG Tech Inc., USA

PUBLICATION

Bin Huang, Daniel C.H. Yang, A.M. Madni, J.B. Vuong, "Multi-Electrode Differential Capacitive Sensors - Model Simulation with Design Application". IEEE sensor Journal, Vol 13, April 2013, pp: 1371 – 1379

A.M. Madni, J.B. Vuong, Daniel C.H. Yang, Bin Huang "A Differential Capacitive Torque Sensor with Optimal Kinematic Linearity". IEEE sensor Journal, Jan, 2007, pp: 800 - 807

Bin Huang, Daniel C.H. Yang "A parametric implicit solid modeler based morphing technology for free-form surface machining", Vol. 1, No.2, International Journal of manufacturing research, 2006, pp 122-135.

Bin Huang, Daniel C.H. Yang "Laplace-based implicit solid modeling for surface morphing in 3D CNC machining", the proceeding of ASME IDETC/CIE 2005.

Yanyao Jiang, Bin Huang, "A study of early stage self-loosening of bolted joints", Proceedings of Symp. Reliability, Stress Analysis Failure Prevention, ASME International Mechanical Engineering Congress and Exposition, New York, November 2001.

Bin Huang, Chongjian Wang "An approach based on CBR to decide the initial point for optimization problems," Jan / 2000, Journal of Mechanical Manufacturing and Design, P.R.China

PATENT

Bin Huang, Xiaodong Tian, Tetsuo Ogawa, Bingyan Zhao, "Machining simulation method and machining simulation apparatus", IPC8 Class: AG05B1918FI, USPC Class: 700182, Publication date: 2009-10-22, Patent application number: 20090265030

AIM

With a standard data structure and unified scheme for data exchange, the goal of this research is to build up a portable Computer Aided Manufacturing (CAM) framework that can seamlessly integrate with the current Computer Aided Design (CAD) systems. To achieve this goal, the following five theoretical topics are addressed

1. **A data exchange scheme of CAD/CAM integration based on CAM oriented model** One goal of this research is to exploit a unified data interface between CAD/CAM. By splitting the design-related information and manufacturing-related information into two different levels, this new mechanism enables seamless transfer between the design model and the manufacturing model with the standard protocol of data exchange. It simplifies the product life cycle, such that both planning of manufacturing processes can be performed in a unified frame of data format.

2. Data representation and architecture design for CAM oriented model. A commercial CAD model is normally composed of different types of geometric features. There is a lack of a portable and efficient data format specifically for CAM modeling. CAD and CAM data are identified separately, and are thus not exchangeable. Based on the STL models, a new B-rep structure is developed to represent the machining features in current research.

3. **Modeling manufacturing process based on Morphing theorems** The existing methods of manufacturing planning usually is targeted on a specific type of feature. Therefore, the roughing

and finishing process have to be processed with different strategies. Based on the technology of solid and surface parameterization, this research aims to introduce a model method synthesis for the shape transformation of stock during the manufacturing process. This methodology parametrically controls the intermediate shapes during the roughing process. The intermediate models are subsequently used for tool path generation. Such an approach is very useful for modern machining planning, because the roughing and finish steps can be finished in one machine and share the same fixtures.

4. **Parametric Tool Path Generation** This research aims to exploit a generic method of tool path generation based on the parameterization technology. Once model parameterization is complete, the tool path in 3D space can be generated by conformably mapping from parametric space. By adjusting the embedding distribution of tool path in 2D parametric space, the scallop height of the surface in 3D space can be easily controlled.

5. **Simulation and error control** with the parameterization of both 2D and 3D freeform model, this research also provides a method for tool path optimization. With the error control of scallop height, the tool path will be adjusted to control the error of cut surface from the desired surface. Meanwhile, this research proposes a new method that can numerically simulate the scallop surface of cut stock. Based on the parameterization of stock surface, the simulation is easier and more accurate.

1

INTRODUCTION

1.1. Computer Integration Manufacturing

This research exploits a new methodology for the integration of Computer Aided Design and Computer Aided Manufacturing, and thus involves theoretical research into the topics of system infrastructure design, data integrity and modeling design.

Traditionally, the two major steps from design to manufacturing are bonded tightly. The increased complexity of manufacturing processes on the one hand accelerates the advancement of technologies and on the other further facilities the design technologies. Frequent changes in the manufacturing environment prevent the close bonding between CAD and CAM processes and separates manufacturing from design phase. The globalization of the manufacturing industry has further magnified this separation, which has led to the design works and manufacturing

works being delegated to isolated ranks of people who, therefore, have to keep on update with each other. Although advances in internet technologies have provided efficient ways for people to exchange their ideas and information, there is not yet a standard scheme for the integration of these two processes that pays due respect to the expertise of design engineers as well as manufacturing engineers. Today's costly iterations between design and manufacturing phases of production cycles lack effective means of communication, as the information of the design phases does not necessarily reflect the requirements for the manufacturing process. This chapter presents a brief discussion of major terms and concepts related to the current computer systems used to integrate the design and manufacturing, with a primary focus on the manufacturing technologies.

The traditional technology of Computer Integration Manufacturing (CIM) is defined as the computer systems that integrate the entire process of the product development cycles. It includes all the methods and tools associated with each step of the product life cycle, from the initial concept design to final manufacturing. The research into CIM spans almost all engineering domains and extensively affects the production quality, efficiency and product cost.

In terms of functionality, the lifecycle of product development includes three key phases: design, planning and manufacturing. Respectively, Computer Aided Design (CAD), Computer Aided Process Planning (CAPP) and Computer Aided Manufacturing (CAM) represent the automation systems for each of these three phases. As shown in Figure 1-1, CAD is the system that converts the design concept to a geometric draft form. CAPP transfers the designed

geometries into the operational steps with generic level of machine instructions whereby to interpolate the design information as the steps of manufacturing operations. Each step of operations contains the set of information related to this particular operation process, such as the machining facility, the geometry of desired stock, the setup instruction, specifications of fixtures, machining tools, the execution machine, operation method and BOM (Bill of Material), etc. As the last step, the CAM system targets specific operation processes and controls the machine tools as well as materials flow with programmable automation. It delivers each of these operations through detailed machining commands, the so-called Computerized Numerically Controlled (CNC) program. The CNC program is normally an interpolated language that provides the precise control for the execution of each step in the manufacturing process. Because every

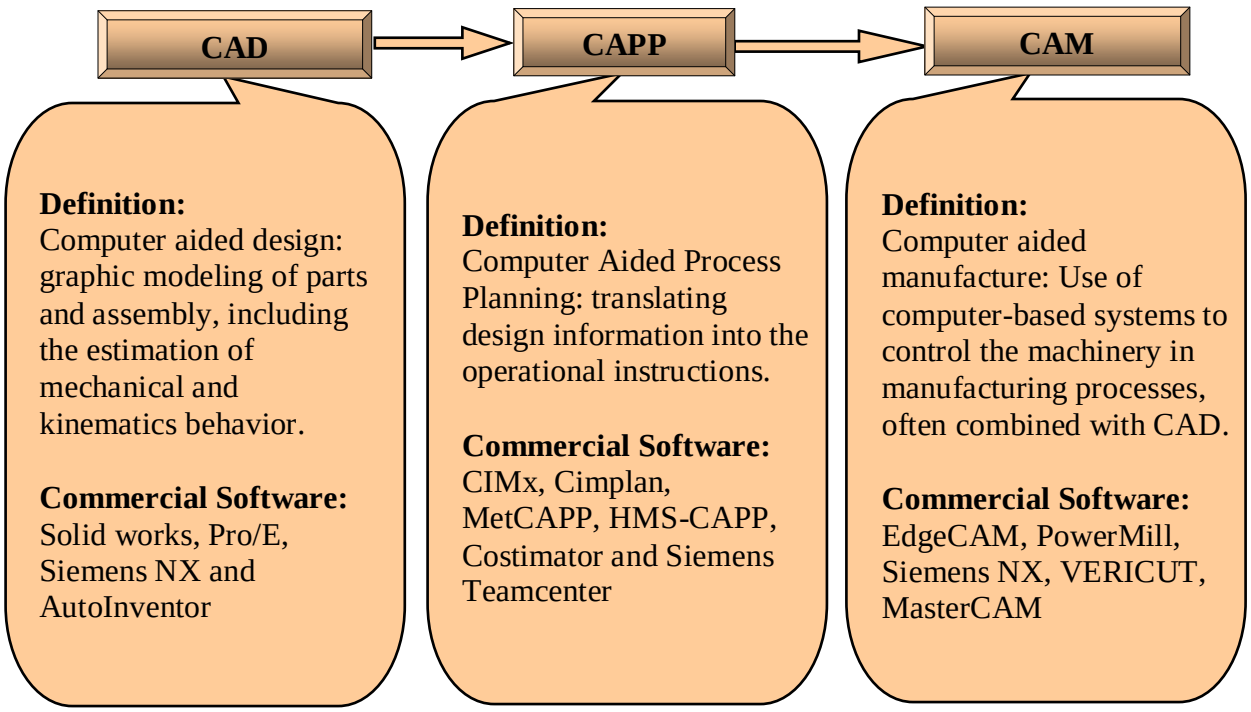


Figure 1-1 Three phases of Computer Integrated Manufacturing

controller company has their own technologies of program control, the CNC programs generated by CAM systems are normally targeted on the particular manufacturing facility. With generic information, CAPP bridges CAD and CAM by translating CAD data into a sequence of machine operations. The detailed execution of each operation is laid out by CAM systems.

With the continuous evolution of manufacturing technologies, and especially the emergence of multi-tasking machine tools and machining centers, the functions of these three developing phases are coming increasingly to overlap. One major goal of CIM technologies is to realize the seamless data flow and integrity from CAD and CAM. However, due to the complexity of CAD geometries and modern manufacturing processes, the link between CAD and CAM is still weak. In recent years, some commercial software has emerged to integrate the CAD/CAM information in a standard and exchangeable way, and therefore certain functions of CAPP can be automated. One example of standard data format is STEP-NC [1], which builds up the manufacturing instruction design bonded with the design model. It provides NC commands as well as the CAD geometry to the CNC machine so that work pieces, fixtures and cutter shapes can be analyzed in the context of the tool path. However, the STEP-NC lacks flexibility because the machining environment is very complicated and has a variety of uncertainties that might influence the manufacturing process. For any change in the manufacturing environment, it is necessary to communicate back and forth, and even repeat the entire design process for the corresponding changes. Therefore, the direct bonding of final NC commands and design model is not the good practice for the complexity of the machine environment. Another example is Product and Manufacturing Information (PMI) [2, 3] first exploited by Siemens Product Life Management (PLM) software. As a new industry-standard format, PMI data can be generated and utilized to

drive manufacturing processes. It defines any annotations and attributes that can be associated to a 3D model, or directly attached on the model. Since the production information is contained in the design model, the requirements or changes made by design engineers can be captured for subsequent CAM operations downstream. However, this also poses the same problem as the STEP-NC standard, i.e. the PMI has to be reposted for any changes of manufacturing environment. Meanwhile, since the PMI feature is attached directly to the CAD model, there is much additional work to be done for the CAM engineers to define these operations. Also, PMI can only be defined and recognized in the NX environment, and there is no way for any other CAM system to either access or translate it into a recognizable format. Because there is no currently no compatibility for the PMI to be imported into other software to generate CAM features, the CAM engineers need to experience some limitations or difficulties to create their own PMI feature if they are not using the same CAD/CAM package.

The geometry information of CAD systems cannot be exchanged with CAM operations directly. Therefore the CAD model normally has to be interpolated with a CAPP system, and divided into fundamental manufacturing operations. For each of these operations, a detailed manufacturing plan is built up in the subsequent CAM systems. As of today, this interpolation process is not fully automated. Intensive human expertise has to be involved in this phase to interpret the design information into the machining operations and to prepare geometry for tool path generation and after the NC programs. The methods for planning are still far from efficient. Meanwhile, as shown in Figure 1-2, the modeling of CAM is processed through iterative attempts. To meet the demands of cycle time reduction, many manufacturing engineers recognize the need to improve their development processes. Process modeling is one way to support this as

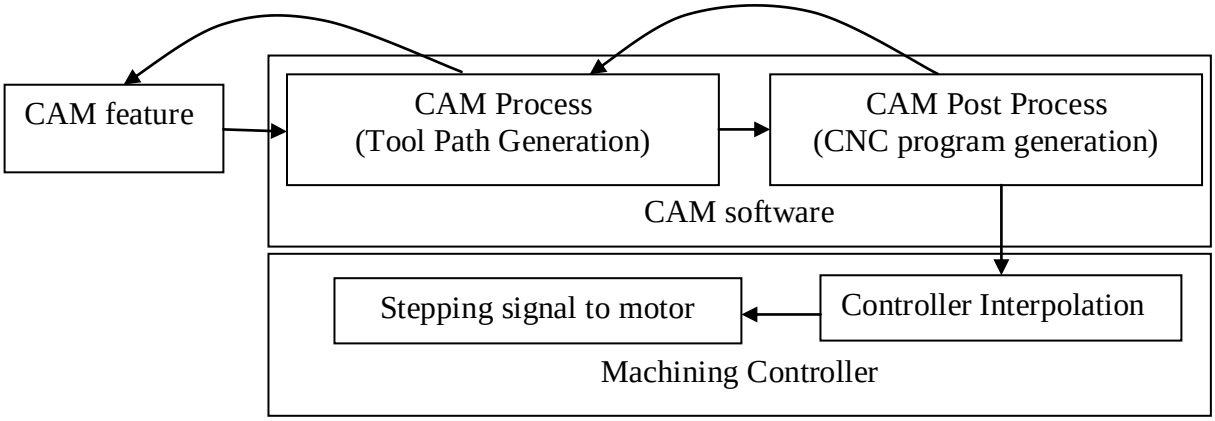


Figure 1-2 Workflow of CNC process

a method to evaluate risks, identify potential improvements and predict outcomes. Also, CAM simulation is a way to estimate the geometrical accuracy of the designed operations in CAM model. However due to the uncertainty surrounding design and production processes, development processes of new product are difficult to model compared to other business workflow. Although research has been conducted into CAD/CAM integration for over 20 years, the time spent on process planning, NC programming and machine setup is still extensive, remaining considerably longer than the actual manufacturing time.

1.2. CNC Manufacturing

CNC (computer numerical controlling machining) is the automation technology most commonly used in manufacturing engineering. By controlling the movements of cutting tools to follow a sequence of electronic commands, CNC makes it possible to automate the manufacturing process, and most Computer Integrated Manufacturing Systems are hence based on it. There are many different types of CNC manufacturing processes, which can generically be

categorized into additive and subtractive processes. Additive process is a forming process to build up the successive layers of material; it is also referred to as rapid prototyping, composite manufacturing or 3D-printing. Based on the base material used in prototyping, there are many different types of additive manufacturing technologies, such as fused deposition modeling (FDM) [4, 5, 6], granular materials binding [6, 7, 8], photopolymerization [6, 9] and composite manufacturing [10]. The opposite of additive processes, subtractive machining tools are normally driven by a set of programmable motors controlled by a numerical controller.

As the process used to generate the driving commands for machine, CAM modeling not only specifies the manufacturing operations but also a set of detailed commands related to each of the operation steps, the so-called CNC program. Figure 1-3 illustrates an example model of a CAM system for the machining process; the bottom surface represents the desired CAD surface, and the curve on the top illustrates a path base for the machining tool to cut through the material and form the stock to be the shape of design. It is a group of curve traces that drives the machining

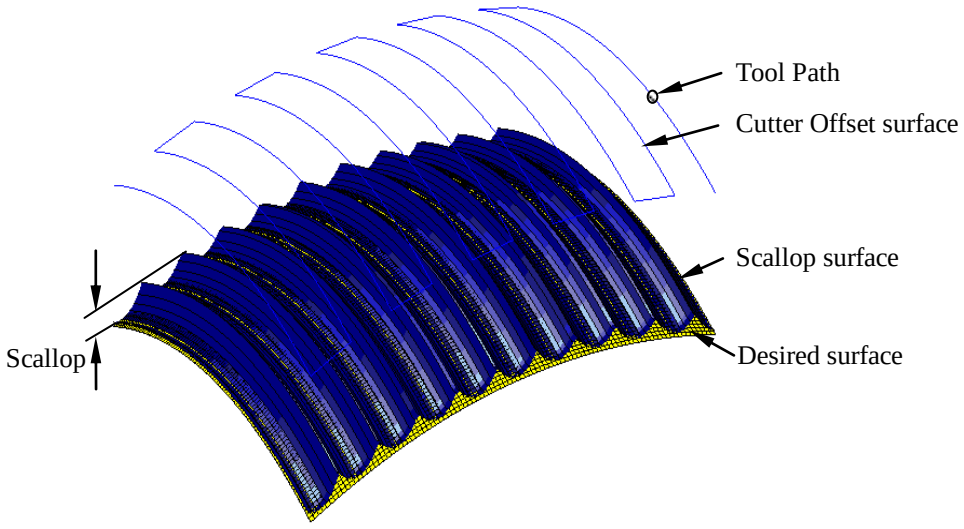


Figure 1-3 CNC machining process

tool during the manufacturing process. The cluster of the curves forms a coverage surface geometrically, and is referred to as a cutter offset surface. Because the machining tool is usually not flat, the movement of the tool cannot cover the entire CAD surface, and the resulting shape of the cutting surface is not exactly like the CAD surface. Based on the shape of the tool, the surface has wave-like remains, called scallops, and the surface that remains after the machining process is called a scallop surface.

For CNC controlling, the controller interpolates the CNC commands and sends out simultaneous signals to the step motors. It controls the movements of the tool and removes the undesired material from the stock until the stock shape approaches the desired shape. In order to generate the CNC commands, each CAD feature has to be redefined as a group of manufacturing operations and interpolated as a cluster of curves in the CAM system, the so-called tool paths. As shown in Figure 1-3, the tool path is designed in such a way that the movements of the tool can cover the desired stock and transform the shape of stock to approach the desired shape. A best tool path should maximally reduce the machining time and avoid any collision between machine components.

Figure 1-4 illustrate a multistep machining process, in which the desired geometry is normally achieved in multiple steps. From the rough stock to the final finished stock, a machining process is normally divided into three steps: roughing process, semi-finishing process and finishing process. Heat treatment will sometimes be conducted following the steps of roughing or semi-finish process.

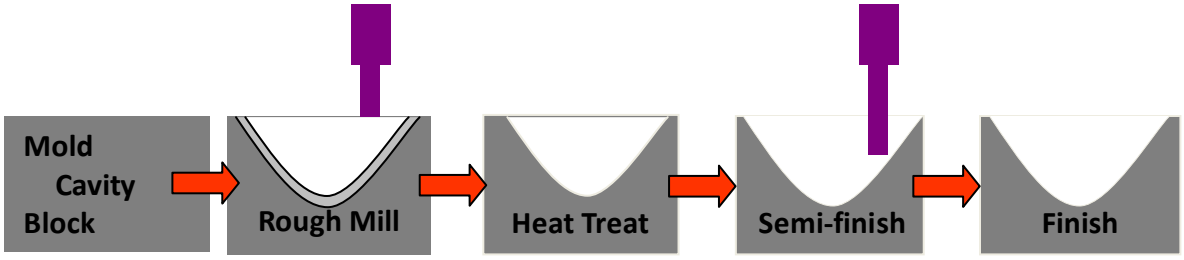


Fig. 1-4 Multistage Machining Process

1.3. Challenge of Modern CAM technologies

The continuous enhancement of CAD technologies enables engineers to design the geometry with higher levels of complexity. The abundance of complex design concepts requires manufacturing companies to develop tooling systems with richer flexibility to handle the geometric features with different levels of complexity. For the wide spectrum of CAD geometries and manufacturing types, there is a lack of a generic modeling method to integrate the CAD and CAM process in a unified manner. Although many studies have been conducted in this field, the connection between CAD modeling and CAM process is still not smooth. All the CAD features have to be interpolated manually as a set of CAM operations with a CAPP package. For any changes in the manufacturing environments, the entire process has to be recycled.

The variable-axis machining process raises another issue. Because there is a lack of format exchange standards or a unified scheme to link the CAD model and CAM feature, different features in the CAD model have to be processed with different strategies in CAM, which

requires intensive involvement of human expertise. This would be more generic and simpler if there was a unified modeling technology to interpolate the CAD features with the semantics of manufacturing operations.

For the machining process, currently different strategies have to be adopted for each step of the machining process. Because of the high efficiency, the roughing processes are normally performed on a three-axis machine, in so-called fixed axis machining. However, for more complex products, fixed axis machining is not the best manufacturing method to ensure the maximal removal of the material. In some models for which the CAD geometry is extremely distorted, such as impellers, turbine blade and the like, the geometry is not accessible with fixed axis machining. With additional degrees of freedom, variable-axis machining can create tool paths across the complex shape that is not accessible to fixed-axis machines. Good examples of these include impellers or blades cutting, five-axis trimming, automotive port finishing, and undercut die mold production.

.

2

CAD/CAM INTEGRATION BASED ON UNIFIED DATA EXCHANGE

2.1 Review of Current CIM systems

A CAD/CAM integration system integrates the design and manufacturing planning by translating the design specifications into the final machine instructions. CAD/CAM integration is quicker and less error prone than human work. Meanwhile CAD/CAM systems allow engineers to see how the various parts of a design interact with each other without having to build a prototype. One of the more recent examples [11, 12] is that of "Boeing having designed and built its 777 wide-body airframe without any prototype work at all: the first physical version was the actual plane that test pilots flew in 1994".

In a Computer integration manufacturing system(CIM), the processing of data is transferred into the production will also be streamlined within hardware and software. This will allow operators to alter and enhance programs in order to improve production. Meanwhile, the CIM system will also provide the necessary algorithms to connect all the data together, which will then be able to intermingle with the sensing and modification components of the system.

Figure 2-1 illustrates the key elements and the framework of an integrated manufacturing process. The CIM framework provides a collection of software and hardware tools to enable the manufacturers to react quickly to any changes of the requirement and minimize cost and errors.

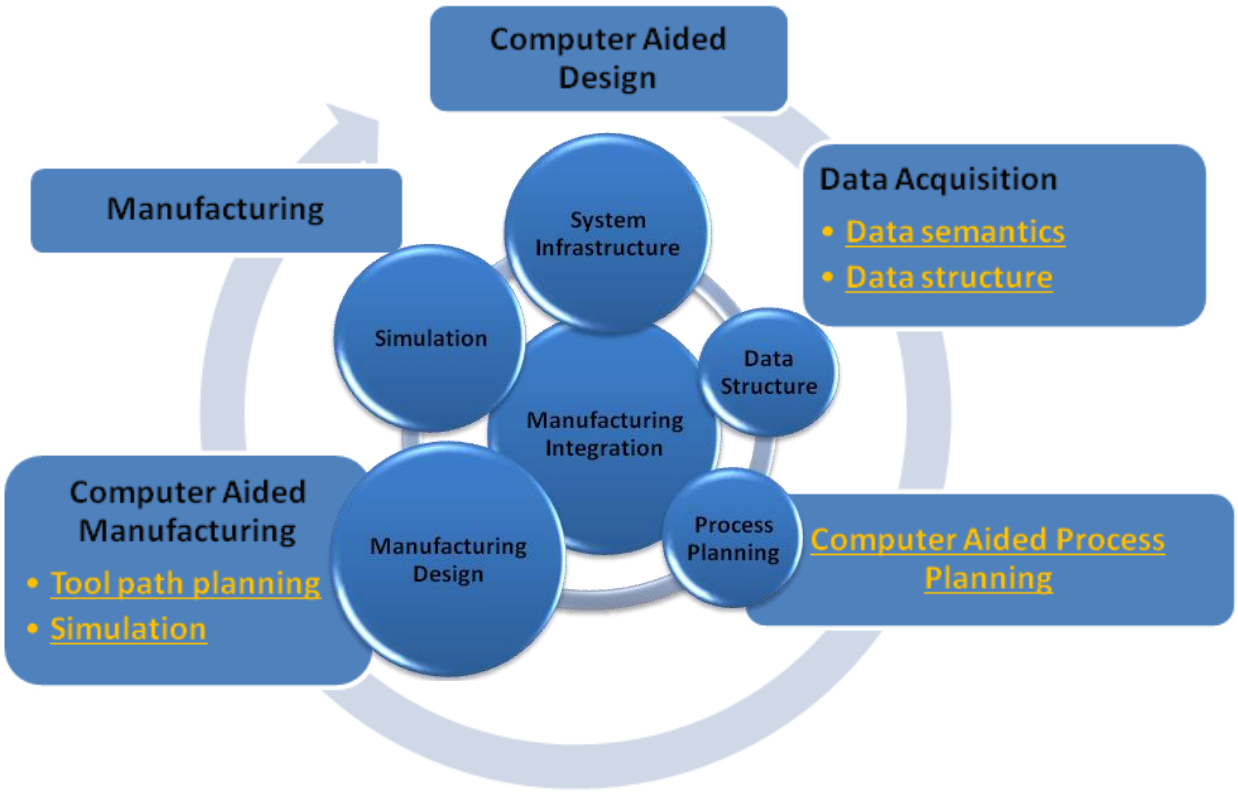


Figure 2-1 Key elements of CIM framework

To build a CIM system, there are five key elements to be addressed for any CIM system: scheme of design, data acquisition, Process Planning, manufacturing design and manufacturing. The major technologies of CIM include Flexible Manufacturing Systems (FMS) [13, 14, 15], Distributed Numerical Control (DNC) [16, 17], Manufacturing and Execution Systems (MES) [18, 19, 20] and distributed manufacturing system [21, 22]. These researches include the areas of building the hardware/software infrastructure, data integrity, process planning and manufacturing design.

2.2 Traditional technologies of integrated manufacturing system

Tu et al. [23] introduced a method called incremental process planning (IPP) for one-of-a-kind production (OKP), where process plan is extended or modified incrementally with a primitive plan by adding new features identified from a product design, until all the features is exhausted.

There are normally two different kinds of systems for process planning. As the more traditional approaches, most of the earlier CIM systems can be categorized under the centralized approaches, which utilize artificial intelligence techniques to automatically generate an optimal process plan according to the part and manufacturing requirements. Such approaches include object-oriented approaches [24, 25], GA-based approaches [26, 27], neural-network–based approaches [28, 29], feature recognition or feature-driven approaches [30, 31] and knowledge-based approaches [32, 33]. All these approaches and their combinations have been applied to the domains of some specific problem, such as tool selection, tool path planning, process sequencing and setup planning. The major weakness of the centralized approach is that the central system

lacks knowledge of each machine and tool availability. The estimate of central expertise for the entire manufacturing process is therefore not accurate. Many of the studies that have been conducted have been based on certain ideal scenarios or empirical data, which cannot reflect the dynamism of the world of industry. Based on these reasons, a truly generative process-planning system that can provide an appropriate generic framework and meet industrial needs has not yet been developed [34].

In addition to the centralized approaches, agent-based technology is emerging in CAPP as a solution for distributed manufacturing process and has attracted wide attention. Cooperative intelligent agents are being used in developing distributed CAPP and CAM systems instead of being one large expert system,. This type of system is targeted at building up a framework that fully integrates design and manufacturing processes, automates the process planning and provides rapid manufacturing service by utilizing the distributed intelligent resources. The agent-based approach is also recognized as an effective way to realize flexibility and dynamism of process planning. Shih and Srihari [35] proposed a distributed AI-based framework for process planning. Their approach decomposes the entire production control task into several sub-tasks, each of which is implemented by an intelligent agent. By working collaboratively, the agents can reach a solution for the problem. Zhao et al. [34, 36] proposed distributed complex process-planning activities to multiple specialized problem-solvers and the coordinate of them to solve complex problems. In their research, cooperation and coordination mechanisms are established among distributed agents by using knowledge-based techniques. Each agent in the system deals with a relatively independent issues. Zhang et al. [25] proposed an agent-based adaptive process-planning system (AAPP) with object-oriented manufacturing resources modeling framework. It

describes the manufacturing resources' capacity in the logic of object-oriented manner, while this approach is implemented as an integrated process-planning platform. Instead of automating process-planning tasks completely, the AAPP system provides an interactive mode to enable experienced manufacturing engineers to make decisions at crucial points. A contract net-based scheme is utilized as the coordination protocol between agents. Other studies include the virtual work system of Sluga et al. [37] and the networked manufacturing service developed by CyberCut [22] for rapid part design and fabrication on the internet. While all these researched or prototyped solutions are successful for a particular area of manufacturing problems, they lack a unified framework to support the data exchange. As introduced in the previous chapter, the major challenge partially derives from the discrepancy of data integrity. All these agents are either simulated modules or are not dedicated to particular manufacturing facilities.

2.3 Challenges of CIM systems

The ultimate goal of the CIM system is to reduce the waste of time and material incurred during the manufacturing process. This is done by taking account of all the elements of production lifecycle, including design, analysis, planning, purchasing, cost accounting, inventory control and distribution departments, and interlinking them with the factory floor, material handling, and management departments. Currently, advances in solid modeling systems offer the better tools to visualize the design concept for the manufacturing process. This development has improved the feasibility of CAD/CAM integration; however, on the other hand, the complexity of CAD models and the emergence of new modeling technologies have introduced new problems and challenges for manufacturing and system integration.

With the globalization of manufacturing processes, many manufacturing processes are now accomplished through multiple facilities located in different areas. Recent research on Computer Integration Manufacturing (CIM) and design has shifted toward the consideration of problems in distributed manufacturing environments or remote manufacturing systems. Traditional CAD/CAM integration is based on a tight bond between design and manufacturing processes. The manufacturing process needs to provide a clear picture to the design engineers, so that they can easily manipulate the CAD model to produce a more efficient manufacturing process. One characteristic of distributed manufacturing is that such systems usually do not have a meaningful manufacturing context laid out to provide a semantic foundation for the purpose of CAD/CAM integration. In such systems, there is no way for CAD designers to control the economics of manufacturing process, such as operation type, tolerance of rough stock etc. Therefore, the traditional integration standard of CAD/CAM integration, such as STEP-NC, cannot be applied to these scenarios.

One challenge is how to coordinate the different machines within the factory. In the workshop enviroment, there are a variety of machines that perform different tasks that are made by a variety of suppliers. It has to be addressed to get every one of these machines to interact with the manufacturing code with a unified interface, and finish the tasks within one mainframe computer.

Another challenge of the CIM system is posed by the data itself and the data entry personnel. There is a need for operators to maintain the integrity of the data that is transmitted to the machines. The challenge is the factory will need to ensure that the individuals working with the

system throughout the factory are competent and knowledgeable. These individuals will need to be well trained, and to update their training periodically.

Last problem remains that Product Life Management (PLM) gives the design engineer the flexibility to design any geometric features. At the CAM facility, an experienced engineer is needed to design the manufacturing process and coordinate the machine resources, but it is still necessary that the design engineers have insightful expertise of the manufacturing process. Recent design-for-manufacturing (DFM) oriented software, such as VLSI-MOSIS [21] and CyberCut [22], requires the designer to place more concern on the manufacturability. A separate software module is usually embedded in the CAD software to assist the design engineer for the purpose of DFM.

2.4 Agent-based CIM design based on unified scheme of data exchange

Figure 2-2 shows a typical infrastructure of an agent-based CIM network, which includes the elements that most of the current CIM system has. The design concepts are independent of facility location, resources, and technical expertise etc. As shown in the figure, the framework of the system comprises three major elements: client site agent, workshop managing module, and virtual CNC machine.

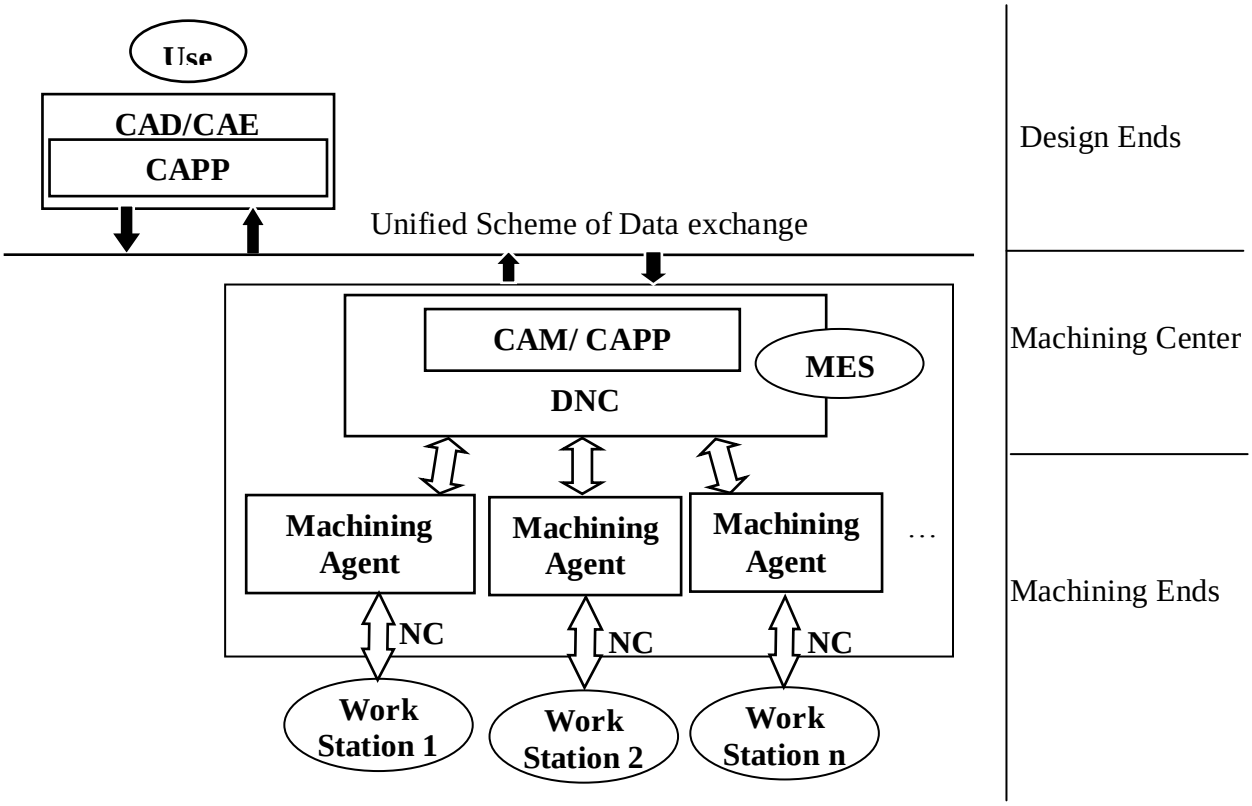


Fig. 2-2 Architecture of agent-based manufacturing system

In an agent system, the functions of CAPP tend to be split in two: one embedded in the design level is used to identify the CAD features to be manufactured and to set up the order for task according to the current status fed back from each local agent. Apart from the traditional CAPP system, the detailed process plan for each manufacturing feature is not completed in this step. Instead, it assigns the tasks to each cell through a unified scheme of data exchange. Combined with the tasks of Management and Execution System (MES) and DNC, a CAPP function is accomplished right at the cell level, where the data from the CAD modeler is accepted, analyzed by the Direct Numerical Controller (DNC), and translated into executable commands for a specific station. Other elements of this framework are as follows:

Machine Agent (MA): In machining agent, the task assigned from CAPP is communicated to the machine in the same way. As the interface between the DNC and the station, MA is basically a small CAM that is dedicated to the particular machine. Because each MA has all the information associated with the machine, this mechanism provides a better way to protect the machine and generate an optimized manufacturing command.

MA also plays a role as a virtual machine that mimics the behavior of the real machine. Its capability is determined by the specifications of the CNC machine with which it is associated. With the technologies of forward and inverse kinematical analysis, the virtual machine will simulate the machining process, validate the material removal, detect collision and report malfunctions. Through the communication of MA, the CAPP on the server side can be dynamically updated with the status of each machine. This improves the accuracy when CAPP estimates the work load and minimizes the waste to assign the tasks.

Traditional process-planning is to be finished automatically. However, the technologies of these areas are not yet well developed, and there is a need to introduce a fair amount of manual works that will increase the data occupation in a sophisticated manner. A standard means of doing this so far is via dialogs, i.e. the designer is required to plan the manufacturing process manually through a series of interactions. The outcome of this module is not just a machining feature, but the operation flow that also gives important connectivity information bridging one feature to another. If any exemption occurs or the design model changes, the whole process might need to be reset, which causes much waste. Through MA, all the updated information will feed directly back to the server. Through certain algorithms, the CAPP could readjust its plan adaptively.

In addition, MA contains a set of rules to prevent any infeasible features. One example of an infeasible feature is that pocket depth is beyond the access limits of machining tools, and another is that the rib feature is too thin to evade deformation when the feature is being machined.

Management and Execution System (MES) once the manufacturing task is sent to the workshop and the manufacturing quote is in the cell, MES is triggered to check the availability of local machine resources. If there is a local manufacturing facility available, MES will assign the task request to an available machine that is capable of it. CAM module of MA is then launched to set up a specific manufacturing program for this machine. If all the manufacturing stations are busy, MES will append the request and the manufacturing information to a waiting list. Sequencing the manufacturing order is the main function of MES. In cases in which some models with complicated parts cannot be completed in one location, MES needs to arrange the transportation of semi-finished stock and monitor and synchronize the inventory base. For flexible manufacturing of mass production, WMM is also in charge of deploying different workstations and design

2.5 Scheme for data integrity

CAPP plays the role of an interface from the designer to manufacturers for the exchange of design and manufacturing information. Based on the framework of the highly effective industry standards that facility the data exchange and the modeling functionality, processing data is generated to drive manufacturing processes. However, because of the complexity of the products, there is a lack of a standard scheme for the data exchange through the entire network. The communication from design to the downstream applications is far away from exchangeable and

compatible. The design data mostly has to be translated manually into the product definition and accountability enforced for the purpose of manufacturing. The conversion creates redundant data that might introduce manufacturing errors and increase the cost.

The PMI (Product and Manufacturing Information) [3] provided by Siemens is a comprehensive annotation in 3D modeling environment that not only captures and associates manufacturing requirements to the 3D model, but also allows the digital data to be re-used by downstream applications. Although this is step further to the goal of integrity between CAD and CAM, the PMI is not a complete description of the manufacturing features. The 2D drawing or the 3D model generated from the PMI needs to be further analyzed and converted to a set of manufacturing features that is accountable to the machine stations. In Siemens NX, the software switches between the design mode and manufacturing, so the user can define the manufacturing features based on the same CAD model.

Other industry-standard file formats, such as the Initial Graphics Exchange Specification (IGES), are better supported, allowing CAD geometry to reliably transfer between CAD or CAM systems. Industry is also developing new standards, such as the International Standards Organization's (ISO) Standard for the Exchange of Product Model Data (STEP), to meet expanding interoperability requirements. This may allow for the CAM system functions to be boosted by integrating the product specification with the solid model from the CAD system. As it currently stands, the CAM system does not have all the information it needs. Hence, CAM users still have to import CAD geometry into their CAM systems and create the manufacturing system. There is lacks of standard for data exchange between different CAM software tools. STEP-NC [38, 39] is a new data standard CNC machining that combines the product, process information

and manufacturing instructions. The new standard for the CAD geometry and tolerance data is to include CAM tool-path data, process data, and cutting tool data. But with STEP-NC, similar disconnect occurs between a CAM system and a CNC machine. Currently the state-of-the-art in CNC program format tells machine tools exactly where and how to move. For different machine tools, the NC program needs to be re-interpolated.

Figure 2-3 illustrates a structure of the STEP-NC information model. The root of a STEP-NC part program is a project containing a main work plan. Each work plan contains a series of working operations. Each of these operations applies a machining operation to a CAD feature, for example a roughing operation. The same machining operation may be reused in multiple working steps and multiple working steps may be necessary to complete a feature. For example, a roughing operation is required followed by a semi-finish operation. The details of the work plan are encapsulated in each operation, including the tool specification, tool path, heat treatment

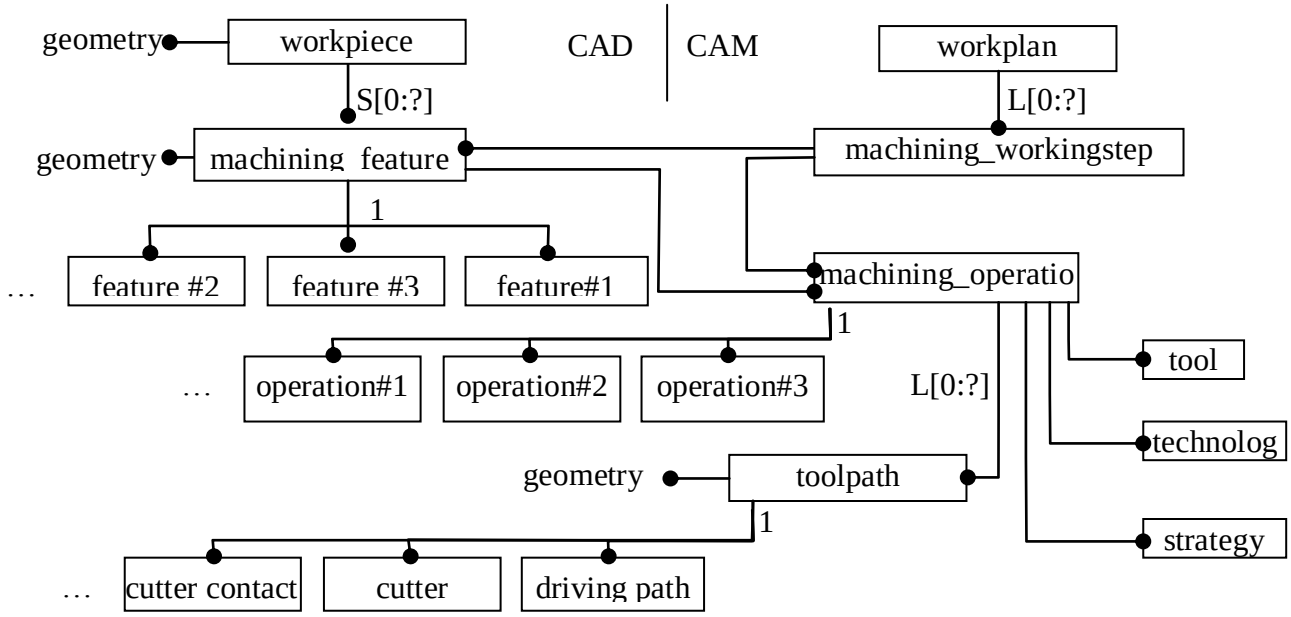


Figure 2-3 Illustrations of STEP-NC architecture

and other related parameters.

As an open architecture module, STEP-NC provides a standard to build up a seamless connection through CAD, CAPP to CAM. However, this standard is not widely accepted by the software providers and machine builders. One major reason for this is the fact that the data integrity of STEP-NC limits the flexibility of manufacturing process planning. In a distributed working environment, the design engineer has the role of setting up the geometrical model of the product. The CAD model is then transferred to the manufacturing engineer, who might be located in a different place. Their jobs include providing a detailed process plan by adding the manufacturing information to the model transferred from the upper-stream department. However, the synchronization of manufacturing data from peer departments is extremely hard, if not impossible. Meanwhile, STEP-NC attempts to connect CAD, CAPP and CAM modules by binding all the information into one file on top of the CAD model. To achieve this goal, the more recent edition of STEP-NC (ISO 10303-238:2007) unifies the methodologies of the post process for the generation of NC commands; that is, the tool path nested in the STEP-NC file can be directly adopted to control the NC machines. This eliminates the post process that is currently used to generate and optimize the NC commands for one specific machine. However, this might cause problems in practice for productivity, because of the diversity of post and controlling technologies adopted by different machine builders to ensure the best manufacturing quality for the particular machine. There are various factors in the real life of a workshop that could influence the productivity and the quality. Especially in a distributed manufacturing environment, it is normally difficult to predict the machine tool used for a certain operation. If the operator decides to replace one machine with another, the corresponding tool path and NC commands

have to be redesigned. Therefore, in many recent studies, the manufacturing information carried by STEP-NC file is omitted [40], as manufacturing engineers prefer to ignore the information stored in the STEP-NC file and to generate their own NC commands. But this leads to STEP-NC losing its original meaning, and also introduces redundant data and discrepancies between the management statistics and real workforce.

2.6 A new scheme of data exchange for agent-based distributed manufacturing

To resolve the problem and make STEP-NC more adaptable to the unpredictable working environment, a new scheme of data exchange is introduced as shown in Figure 2-4. The major difference between the new scheme and the traditional one is that its geometry model is not a

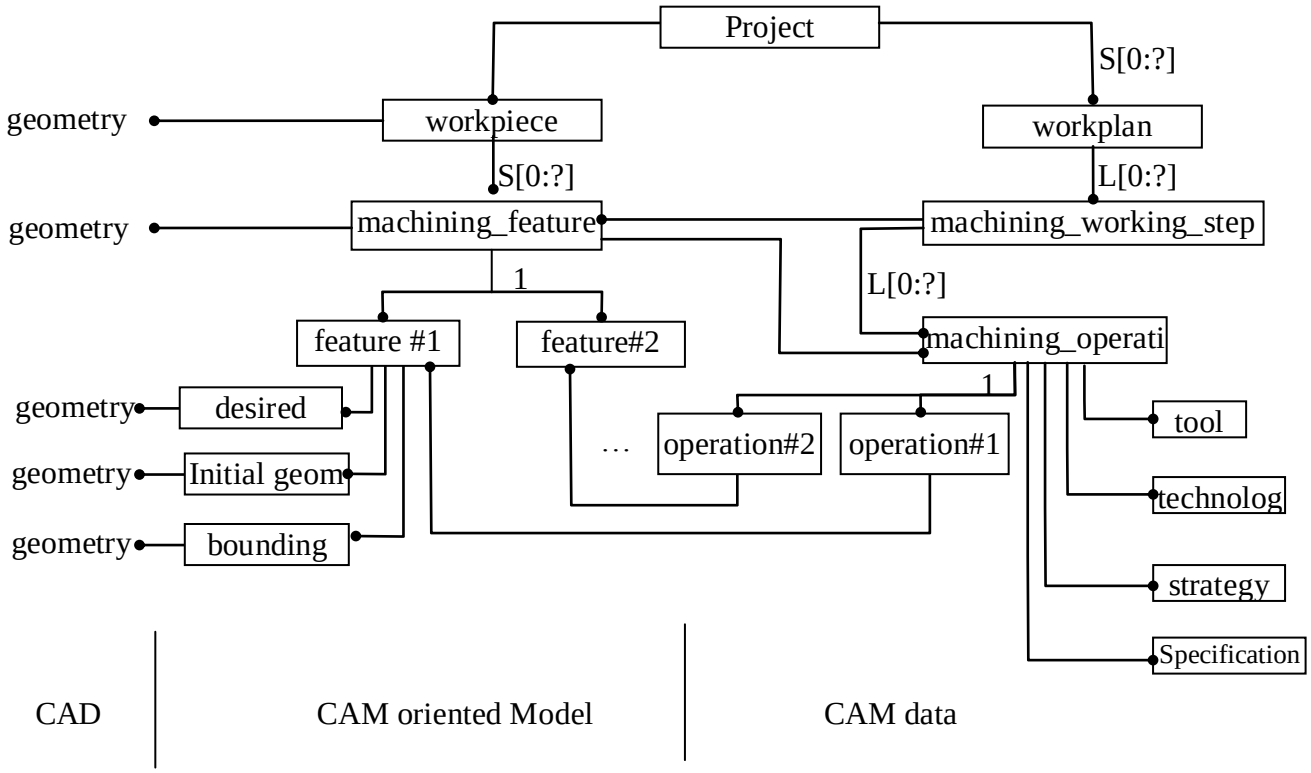


Figure 2-4 New STEP-NC architecture based on distributed

direct copy of the CAD model. Instead, it is built on top of a CAM-oriented geometry model that is abstracted by subtracting the CAD model from the model of rough stock. By peeling off the machine-related information, the CAM-oriented model is left only containing the geometry information of removed material. All the machine information is shifted to another layer of CAM data. The in-process-stock contains a set of to-be-machined features, each of which encapsulates the subgroups of geometries, including the initial geometry, desired geometry and bounding geometry. Instead of one work plan for the project, on the CAM layer there could be multiple work plans for each project. This will help version control and data synchronization. Based on the working plan, the same geometry entity could be interpolated as different machining features, and also handled with different operations.

Figure 2-5 illustrates the data flow of the new scheme. The CAD model is first converted into a CAM-oriented model. This module can be accomplished by an applications built in the CAD systems. After CAM-oriented model is created, the data can be transferred to the CAM system for the NC post. The applications built in the CAM systems retreat the CAM feature from CAM-oriented model and generate operations according to the machine information and specification in real-time. Any new defined CAM operations are kept semantically separate from the CAD information.

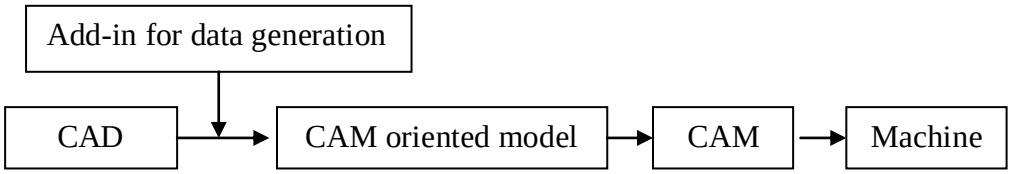


Figure 2-5 data flow of the CAD/CAM integrity

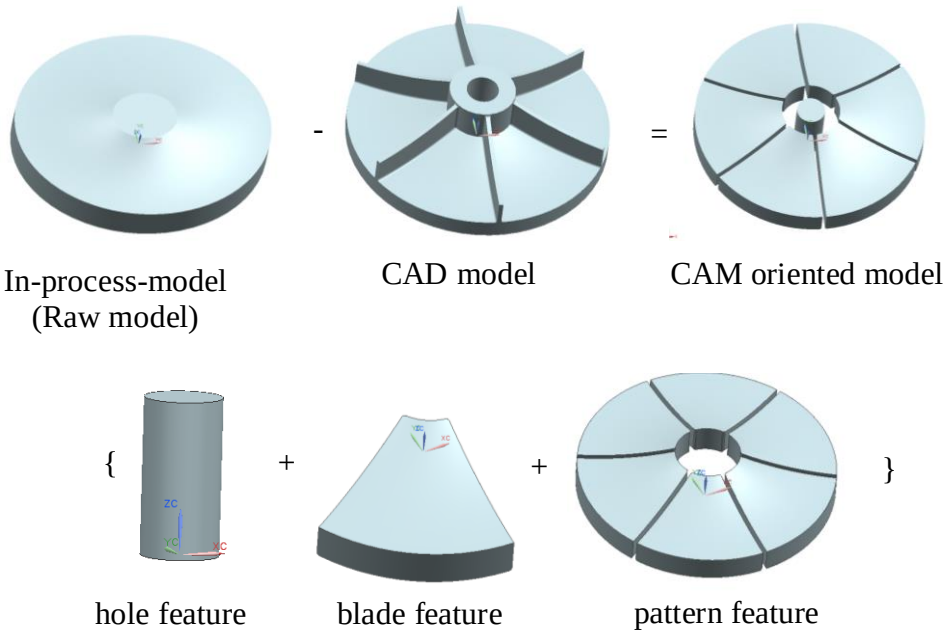


Figure 2-6 Decomposition of machining feature

2.7 Case study

Figure 2-6 illustra tes the decomposition of the machining feature. CAM-oriented model is obtained by subtracting the CAD model (desired model) from the raw model. It represents the volume to be removed during the machining process, and is further decomposed to a set of machining features. Every machining feature is composed of initial stock, desired stock and

machining stock. In such a CAM-oriented model, raw stock is also referred to as in-process stock, which is the initial stock of every machining step. It may also be the desired stock of the preceding machining step. The machining stock could be decomposed as a group of machining features. For the part shown in the figure, it contains one hole feature and six blade features. Because blade features are axis-symmetric, it can also be simplified as a pattern feature whereby radial copies of the blade feature along the axis. As a result, the final machining feature can be translated as one hole feature, one blade feature and one pattern feature. Figure 2-6 illustrates the geometries for the blade feature: it contains the desired surface, bounding surfaces and initial surface. All the decomposition is accomplished at the phase of design. It provides a technical analysis of the product to identify machining requirements and constraints. This phase can be accomplished by the manufacturing experts manually through a series of interactions in the CAD environment. There are also many technologies of feature recognition to automate this process.

The final result of all the working steps is transferred to the local manufacturing center. Based on the current status of the work floor, the second planning is conducted in the machining center to match the required job operations with the operation capabilities of the available production resource. Once the machining station receives the task, it will detail the manufacturing plan and generate the NC code for the assigned machine station. Without the restriction to follow the designed tool path requirement, each station and agent has the flexibility

to optimize the NC code based on the characteristics of the machine station that it is associated with. Meanwhile the data transition between server, machining center and machining station can be more efficient.

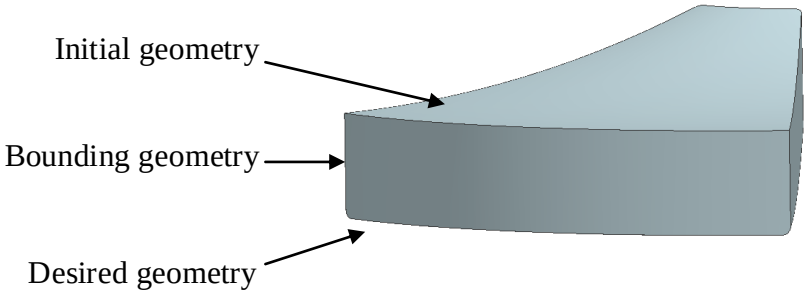


Figure 2-7 A generic machining feature

3

REPRESENTATION STRUCTURE AND ARCHITECTURE DESIGN

3.1 **Overview of Boundary Representation (B-rep) for Solid Modeling**

3.1.1 Representation of boundary surface

As shown in Figure 3-1, the machining feature is defined as the cut volume to be removed during the machining process. It is usually abstracted from the machining feature that contains the boundary entities, such as the bounding geometry entities (surfaces, edges and vertices). Each machining is usually represented as a boundary representation model (B-rep). Solid modeling systems for boundary representation models are mostly 2-manifold. A 2-manifold surface is one for which every point is topologically equivalent to an open disk. The machining features are normally generated with an interactive step-by-step construction procedure by picking up the boundary entities. Many studies have been conducted that attempt to identify the machining features automatically[41, 42].

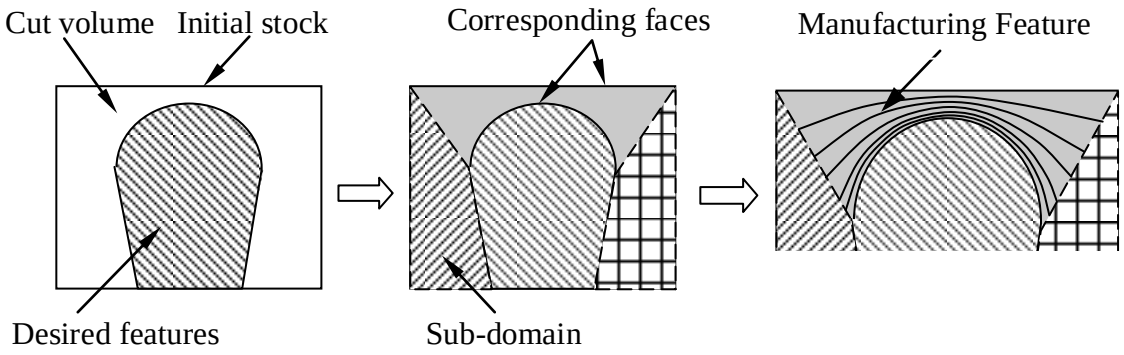


Figure 3-1 Sub-domains of cut volume and their boundaries

Geometric Entities	Topological Entities
Point	Vertex

Figure 3-2 Geometrical entity and their topological set

B-rep is formed on the basis of the assumption that any physical object is enclosed by a set of bounding faces that divide the space into domains of interior and exterior space. The orientation applied to the surfaces identifies the interior and exterior region. Meanwhile, each face has boundary curves and vertices. The database of B-rep comprises the sets of geometrical entity and topological entity. As shown in Figure 3-2, geometrical entities do not carry any boundary information, and therefore the geometrical entities, such as surface and curve, are regioned infinitely. By defining boundaries of the geometrical entities, topological entities trim the geometry entities, such that volume is bounded by faces, face is bounded by edges, and edges are bounded by ending points. Topological entities serve as the "glue" to connect the geometrical entities together. Information needed for a boundary representation is commonly obtained through user input or a boundary evaluation of CSG data. Due to the complexity of 3D geometry, many studies have attempted to simplify the modeling process by using a unified internal form

for geometric representation, including parametric representation and facet representation ([43-47])

3.1.2 Unified Parametric representation

The boundary surfaces of conventional B-rep are composed of patches that have different kinds of primitive geometry, such as polygons, cone, cylinder, sphere, and quadric surface. In recent years, solid modelers have been developed to handle the wealth of not only the primitive models, but also the freeform surfaces. One of the strategies whereby this is done is to develop a unified approach to support the various curve and surface forms. Non-Uniformed Rational B-spline (NURBS) [43, 44] is an example that has been commonly adopted for this purpose. Since NURBS is convertible with most primitive features, its internal operations, such as the calculation of surface-to-surface intersections, can be achieved with a unified algorithm. The surfaces are first specified by the designer and then translated into the unified internal form such as NURBS. Primitives can be created in the form of NURBS and used for model development in CSG or B-rep representations, allowing a large variety of geometric forms to be handled with a manageable amount of computer code and easily accessed by the CAM system. The fundamental theories of NURBS are stated in a number of studies [45, 46].

<Vertex Table>

<Index Number>	<X coordinate>	<Y coordinate>	<Z coordinate>
1	0	0	0
2	1.0	0.0	0
3	1.0	1.0	0
4	0	1.0	0

Table 3-1 B-rep of cube with tessellation

The formulation of parametric representation has a higher order, and it can therefore represent complex patches or curves with a simplified formulation. The CAD model can consequently be built up with small number of features or patches. This could reduce the data size of the model. However, the parametric representation is normally an approximated or interpolated form, and it does not exactly match with the original patch and its boundary edges. It is usually very hard to obtain a waterproof model with parametric representation, that is, there are almost always gaps or mismatches between two neighboring patches.

3.1.3 Tessellated representation

In cases of complicated objects, the exact B-rep models are obtained by the equations that represent their edges and faces. However, these surfaces and curves can be also approximated with planar facets that are referred to as facet representation or tessellated representation [46, 47, 48]. The surface of the object is covered with triangles and quadrilaterals. It is very flexible and

easy to add new surface types, and all needs can be satisfied with very simple geometric data. The problem of surface-to-surface intersection is simplified to calculate the intersection of planar facets. The tessellated representation enables numerical analysis. Many simulations, such as finite element analysis, are based on the facet model. However, since the geometry needs first to be linearized as facet planners, a large quantity of facets has to be generated to maintain an acceptable level of accuracy in the model.

One commonly used data structure for tessellated B-rep is an Index Table. Table 3-1 provides an example of such a facet representation for a unit cube. The table lists the four vertices of the top face and their index number. The face is defined in another table with the indices of the vertices that form each face, also followed with the additional attributes derived from the surface to which the mesh belongs, such as its normal direction and its curvature. These attributes provide geometric and topologic details that are the necessary parameters for the implementation of tool-path generation.

In our system, all the CAD models will be tessellated into triangle facets. However, any sorting or searching operation of the Index Table is inefficient. There is hence a need for a more efficient data structure. A winged-edge data structure is designed to solve this problem ([49-52]). Other traditional data structures such as lists, trees and graphs are by themselves insufficient to represent geometric objects because they are either one dimensional or do not capture the rich structural properties.

3.2 Data structure of B-rep

3.2.1 Winged-edge structure of C style

The data structure used for B-rep is normally called winged-edge structure[46-49]. Unlike regular one-dimensional structures, it is a nonlinear structure that describes the topology of the network formed by geometrical entities. The winged-edge data structure was first proposed by Bruce G. Baumgart in 1975 [50]. It features vertices, edges and faces with a dense pointer structure between incident primitives. Through those links, a fast access of neighboring entities is made possible, although there is some overhead in space and time necessary (memory space and processing time, and sometimes programming time). The winged-edge data structure is capable with certain modifications of overcoming some of these restrictions, and the more complex radial-edge data structure allows for non-manifold meshes altogether. There are various

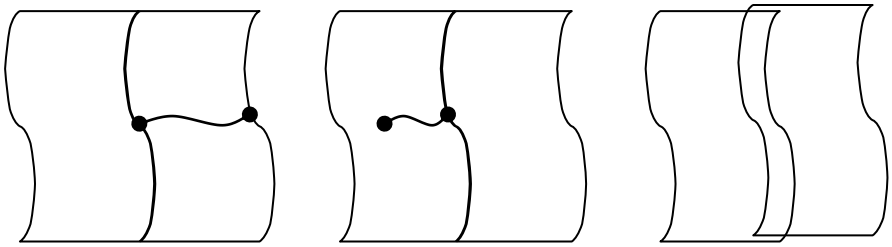


Fig 3-3 Failure cases of B-rep structure

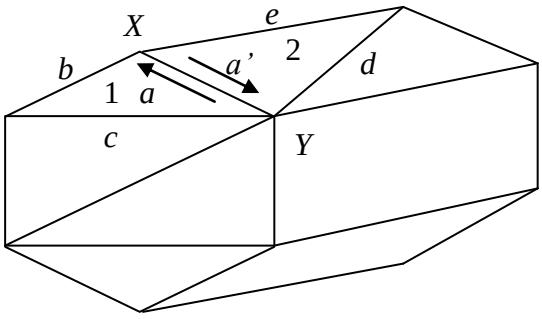


Fig 3-4 B-rep of tessellated model

winged-edge structures, such as quad-edge structure [51], half-edge structure [52, 53], radial-edge structure [54] etc. As mentioned in the introduction, half-edge's restriction to represent manifold surfaces makes it unusable for certain purposes. With the winged-edged structure of B-rep, two surfaces intersect at a common edge curve that has the same ending vertices. The hanging entity, as shown in Figure 3-3, is not allowed. Each entity of B-rep structure intersects at the same entity of lower level in the hierarchy of the data structure. The lower-level entities follow an order such that the higher-level entities are ordered counter-clockwise(CCW) with respect to the innate orientation of the edge. Figure 3-4 shows an example of a model with polyhedron facets. Each face is represented with a list of edges that consist of two ending vertices. If two faces share the same edge, the overlap edges are the wing edge of each face. The figure shows a polyhedron with vertices, edges and faces indicated with upper case letters, lower case letters and digits respectively. Vertex *X* is comprised of the incident vertices of edge *a*, *b* and *e*. When the edges are traversed, the following information is important and must be linked to the vertex: i) vertex position and other geometric attributes, such as normal, u/v tangent, principle curvature etc.; ii) All the edges incident to it, so-called emanating edges; iii) The faces incident to it.

For edge $a = XY$ as an example, it has two incident vertices *X* and *Y*, and two incident faces *1* and *2*. A face is a polygon surrounded by loops of edge. Face *1* has its edges *a*, *b* and *c*, and face *2* has its edges *a'*, *e* and *d*. Edge a and a' represent the same geometric edge, but they are in opposite directions when traversed in the two neighboring faces. Therefore they are called radial edges. In the example case, it is traversed once when traversing face *1* and traversed a second time when traversing face *2*, and is used twice in different directions. These two edges are

defined as radial edges. The radial edges belong to the different loops of neighboring faces. For each edge, the following information is important: i) its two ending vertices; ii) its radial edges; iii) the predecessor and successor of this edge when traversing its left face, and the predecessor and successor of this edge when traversing its right face.

For face *1*, for example, it has one loop that is composed of edges *a*, *b* and *c*. But many faces contain the exterior loop and interior loops. With the implementation of winged-edge structure with C style, the B-rep is built on the hash table of vertices, edges and faces. The entity is recorded with the index number of the corresponding entity stored in the hash table. The traverse is based on each face, then edge, and then vertex.

Vertex Table The vertex table contains a list of vertex structures. Each of them contains the coordinates of the point. In some B-rep models, each vertex also contains the list of the index numbers for the corresponding edges that are incident to it.

Edge Table Each entry in the edge table contains the index information of above entities: edge name, start vertex and end vertex, left face and right face, the predecessor and successor edges when traversing its left face, and the predecessor and successor edges when traversing its right face. Note that clockwise ordering (viewed from the outside of the polyhedron) is used for traverse. Note also that the direction of edge *a* is from *X* to *Y*. If the direction is changed to *Y* to *X*, all entries but the first one in the following table must be changed accordingly.

The winged-edge data structure allows for quick traverse on the faces, edges, and vertices due to the explicitly linked structure of the network. This rich form of specifying an unstructured grid is in contrast to simpler specifications of polygon meshes such as a node and element list, or

the implied connectivity of a regular grid. However, the traditional winged-edge structure is a procedure-based structure. There is a lack of expandability for the data to be handled. The traverse among the geometrical entities is quicker than with the Index Table, but the operation of geometry, such as scaling, boolean, adding or reduction of entities is not easy. To resolve the problem, a new radial-edged architecture has been designed [54]. This new architecture is based on the philosophy of object-oriented programming, which is a more data-centered architecture. It makes all the B-rep operations easier to handle, and also provides great flexibility for expanding functions.

3.2.2 Winged-edge structure of Object-Oriented Programming (C++) style

In the last decade, the development of object-oriented programming (OOP) technologies has increased the demands to redesign the B-rep structure with the concepts of class. As opposed to OOP, the traditional data structure is based on procedure and has limitations in many aspects. First of all, the traditional implementation is not expandable and is designed for only one specific type of model. For example, the data structure of a facet model is dedicated solely to the facet model, which is not compatible with the NURBs model or any other non-facet model. Meanwhile, the traditional structure style of B-rep lacks of flexibility, which can hardly add a new geometric entity in real-time into an existing B-rep. For detailed information regarding OOP and class design, refer to Appendix A.

```
Class Edge
{
Private:
    // The following fields correspond to radial edges
    Edge * mPrecessorRadialE;
    Edge * mSuccessRadialE;
    // The face that the edge belongs to
    Face * mFace;
    // The preceeding and successor edges in the loop
    Edge * mPrecessingEdge;
    Edge * mSuccessorEdge;
   // The ending vertics
    Vertex * mStartVertex;
   Vertex * mEndVertex;
}
```

Figure 3-5 Class definition of edge

Based on the definition of object-oriented programming, the topological information of the B-rep entities can easily be encapsulated as the class attributes. Figure 3-5 shows the example classes of an edge, in which its radial edges are connected with a double-linked list. mPrecessorRadialE and mSuccessRadialE represent the predecessor radial edge and the successive radial edge. All the radial edges will subsequently be bundled into a list. mFace is the

face to which the edge belongs. Also, each instance of edge class is connected to an edge loop with a link-list, and its precessor and successor in the loop is represented as mPrecessingEdge and mSuccessorEdge.

Figure 3-6 shows an example class of a vertex. This class is normally derived from the point class that contains the coordinate information of the vertex. Besides this, the class definition of the vertex also includes other geometric attributes, such as normal vector, tangent vector etc. For the convenience of traversing the emanating edges, mEminatingEdge represents the list of edges that emanate from the vertex. To easily access the edges that are incident from it, each vertex also contains the addresses of the emanating edges.

```
Class Point
{
Private:
        Double x;
        Double y;
        Double z;
}
Class Vertex : public Point
{
Private:
      // The following fields correspond to emanating edges
      Edge* mEmanatingEdge;

     // The face that the edge belongs to
     Face* mFace;
}
```

Fig. 3-6 Class definition of vertex

```
Class Loop
{
Public:
      //Accessors
      …
Private:
     // The following fields correspond to the array of
     Edge * mEdgelist;
     Int mNumEdges;
}

Class Face
{
Private:
 // The following fields correspond to a interior loops and exterior loop that belongs to the face
      Loop* mExteriorLoop;
      Loop* mInteriorLoops
}
```

Fig. 3-7 Class definition of face

Figure 3-7 shows the class definition of a polyhedron face that is composed of a loop of edges. A face can have multiple loops, one of which is the exterior loop. Interior loops are used if the face has holes. Interior loops and exterior loops have opposite directions (CCW and CW). Other entities on the face include composite face, shell and lump. Shell is composed of a set of connected faces, and represents a region of a solid. Some solid modelers support multiple solids, and can therefore have multiple sets of B-rep. Each of them are called lumps, but the top level entity for a polygon B-rep is face.

3.2.3 A new Design pattern of Topology network

Besides the data structure, the operation for a B-rep also follows certain behaviour patterns, and these can be abstracted with some design patterns in software technology. Design pattern is a

general structure or solution that can be re-used for a common type of repeatable problem. (For detailed information regarding OOP and class design, refer to Appendix A.) The major strength of design pattern over regular OOP is that, in addition to the data structure of OOP, design pattern also generalizes the interactions between multiple objects. It describes a semantic solution to abstract the repeatable behavior of real-world objects with interface objects, while OOP only contains data. In such a way, we can use the core pattern over and over again by hooking it with different real world behavior in the application level. Many studies and commercial packages have attempted to design B-rep structure with the traditional OOP structures, but design pattern has not yet been incorporated into these researches. All the OOP programming hitherto provided is simply intended to resolve the issues of how the data is built and connected in the level of concrete classes, but the traverse between different entities is still dependent on the types of B-rep and its geometrical entities. For facet and NURBS models, for example, although their behaviors and data structure follow certain repeatable patterns, the traverse function, visiting function and the entities have to be defined separately for each individual entity. Most codes are insufficiently flexible to handle any types of B-rep, and a new framework is hence developed with the design pattern.

iTopology and iOwningTopology iTopology and iOwningTopology are sets of abstract bases to define the topology and owner of topology. Figure 3-8 shows the definition of abstractive vertex and topology. Each iTopology is a member of a linked list that is owned by an iOwningTopology. It represents a node in the topology owned by iOwningTopology. iOwningTopology is the owner of all the entities in its list, but can also be a topological member of another topology owner. Therefore, iOwningTopology is derived from iTopology. In this framework, each entity class is

derived from the interface of either iTopology or iOwningTopology. With the unified interface class, namely iTopology, visiting the target entity of B-rep can be accepted through the unified interface, which simplifies the traverse. On the other hand, because iOwningTopology could be owned by another iOwningTopology, it can easily build up a composite based on this inheritance.

iVertex Figure 3-8 shows the class definition of iVertex in design. It is the interface class of any vertices in the B-rep. The vertex is a subclass of iTopology. The difference between a point and vertex is that a point contains only geometric information, while in B-rep a vertex has further attributes. It also contains the topology information, including its incident edges that merge at or emanate from it. All the vertices are defined and visited based on the same interface, so that each instance can be traversed. For instance, the work flow that calculates the normal direction is shown in the following figure. The iterator goes through each vertex and calculates the normal direction. Because the type of visited object is all iVertex, they can therefore be visited through a unified interface, i.e. iVertex. The same traverse can be used for other operations on the vertices

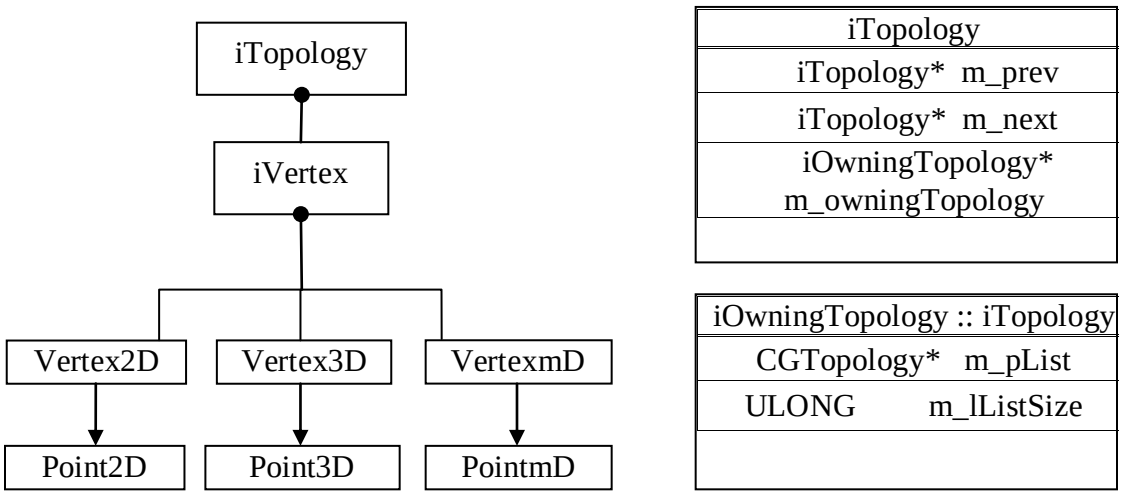


Fig. 3-8 Class definition of vertex with interface

of B-rep. This follows the so-called visitor pattern: when the operation changes, clients simply need to create a different visitor.

iEdge iEdge is the interface base that represents the intersection of any two adjacent faces. It normally contains two ending vertices. The edge of the polygon face is a line segment, and therefore its two ending vertices are sufficient to represent it shape. In other types of model, such as IGES model, the edge needs a parametric function to represents its shape besides its two ending points. In case of a feature model, the edge might contain the feature type and other information. Figure 3-9 shows a definition of a linear edge. It has two vertex members that are connected to the adjacent edges. Meanwhile, the iEdge class is derived from iOwningTopology, as an edge can be a composite of multiple sub-edge segments. Beside its inherited properties, each edge is shared by the neighboring face. In order to easily access these faces, it contains a list of all faces that are connected to the face. In the B-rep of this research, a double-hanged structure is used, as shown in Figure 3-9. Each edge is owned by only one face, and every face has its own edges. The edges at the intersection of the neighboring faces are called radial edges, and they are bundled as a list of radial edges.

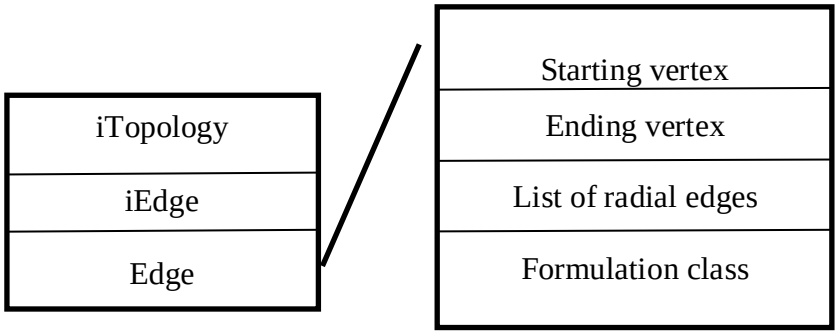


Fig. 3-9 Class definition of edge

iFace Face class is defined as the subclass of interface iFace that is derived from iOwningTopology, as shown Figure 3-10. Each face is composed of lists of edge loops and the mathematic formulation of the surface. If it is a simple polygon face, the mathematic formulation is not needed. Every loop is a closed list of edges. The CCW direction of the loop points to the innate direction of the surface being normal. Face class also has other geometric attributes, such as normal vector etc. In Figure 3-11, Face *A* and *B* intersect at the same edge, but they both have their own edges, namely Edge *CD* and *DC*. Because these two edges actually represent the same edge but in opposite directions, they are bundled into a list, so-called radial edge. Meanwhile, due to their compatible condition, Edge *CD* and *DC* have the same ending vertices but opposite directions. For the non-manifold model, the radial edge list normally has only two edges, which are opposite to each other. For the manifold model, the number of radial edges can be greater

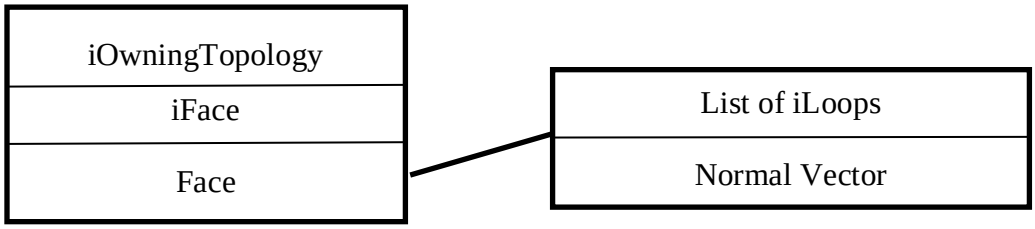


Figure 3-10 Class definition of Face

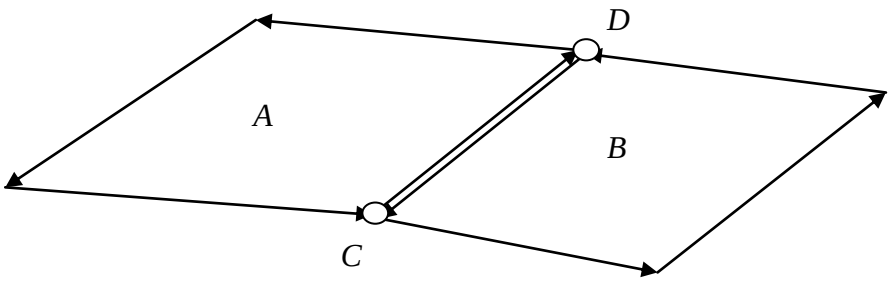


Figure 3-11 Double winged-edge structure

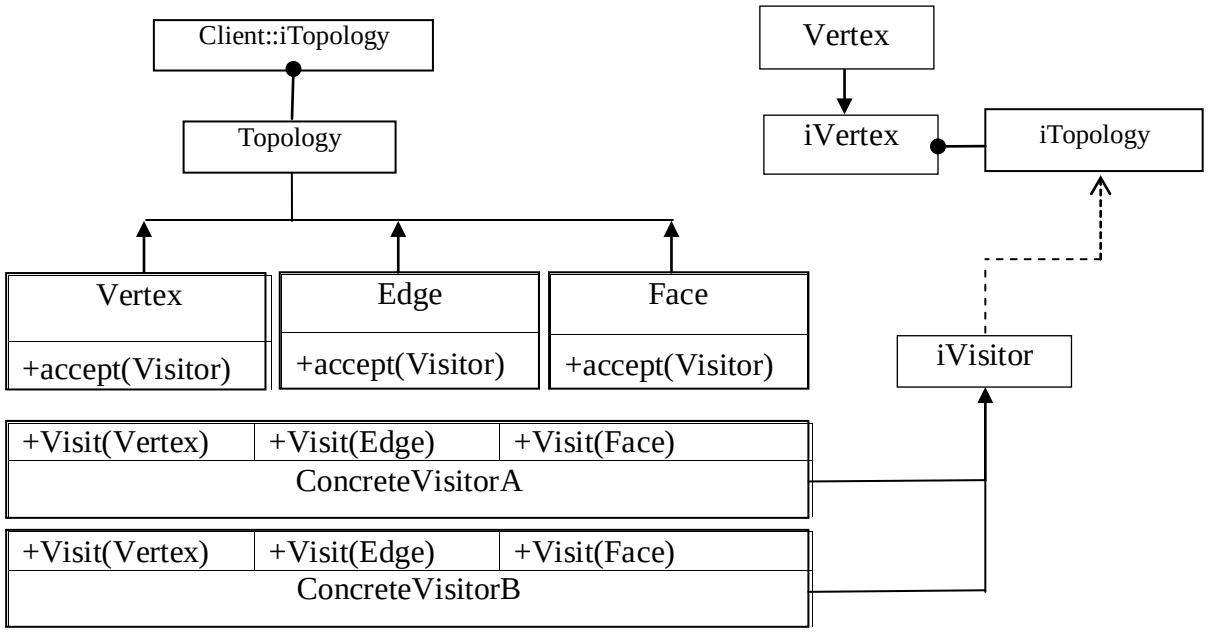


Figure 3-12 Class definition of vertex traverse

than two, but the number of radial edges along one direction has to be equal to the other direction.

iVisitor and Traverse Because of unified interface definition, any operation of B-rep can be implemented through an aggregate hierarchy of interface objects. In design pattern, this approach is accomplished by the visitor pattern. The approach encourages the design of lightweight element classes, because processing functionality is removed from their list of responsibilities. New functionality can easily be added to the original inheritance hierarchy by creating a new visitor subclass. Figure 3-12 shows that the design pattern can be implemented on both client side and visitor side. On the client side, every concrete class of iTPTopology has a common method

iTPTopology :: accept (iVisitor). This method offers a mechanism with which to implement the operation to be applied to a type of element during the traverse. On the visitor side, the concrete visitor inherits the visit methods from the interface. The actual implementation of visit methods is specified on the basis of the input type of topology element.

```
Class DLL_EXPORT
ValidationVistor : public iVisitor
{
public:
//Constructor
        ValidationVistor(B-rep* B-rep);
//Destructor
        ~ValidationVistor();
//Visiting Functions
        virtual void Visit(Topology* myTopol);
        virtual void Visit(Vertex* myVertex);
        virtual void Visit(Edge* myEdges);
        virtual void Visit(Face* myFace);

        virtual void Visit(TreeNode* myNode);
//Status accessor
        bool GetStatus() const;
private:
//No copy constructor
        ValidationVistor(const ValidationVistor&);
        B-rep* m_brep;
};
```

Figure 3-13 Class declaration of validation visitor

Figure 3-13 shows a validation visitor that checks the entities of a B-rep. The validation operation needs to be different for vertex, edge and face. Although all the entities of B-rep are traversed and visited by the same visitor, each element structure will have an associated visitor class. This abstract visitor class declares a VisitConcreteElement operation for each class of concrete element defining the object structure. Each visit operation on the visitor declares its argument to be a particular concrete element, allowing the visitor to access the interface of the concrete element directly. As a concrete visitor, validation visitor overrides each visit operation to implement visitor-specific behavior for the corresponding element class.

Figure 3-14 shows a traverse method implemented in B-rep. As each topology element is iterated, the visitor will implement the operation based on the type of the topology element. The visitor is created by a visitor factory that provides a static method to create all types of visitors and return it as iVisitor.

```
//Class declaration of visitor factory
Class DLL_EXPORT VisitorFactory
{
static iVisitor BuildValidationVisitor()
    {
        Return ValidationVisitor();
    }
}


//Traverse function in B-rep
void  Brep::Traverse(iVisitor* visitor, TOPOL_LIST& topoList)
{
  TOPOL_LIST ::const_iterator cgiTerator;
  for(cgiTerator = topoList.begin();  cgiTerator!=topoList.end(); cgiTerator++)
  {
        if(*cgiTerator)
          (*cgiTerator)->Accept(visitor);
        if(visitor->Terminate())
          break;
  }
};
```

Figure 3-14 Creation and traverse of validation visitor

3.2.4 Reconstruction of B-rep from STL model

The STL model has no topological information. Each triangle has separate vertex coordinates, and a method is required to rebuild the information with the STL model. Based on the above design pattern, the topology can easily be reconstructed by merging the overlapping vertices. The regular method is to import each triangle and search through the B-rep to find its neighboring triangles. This algorithm is time consuming. Figure 3-15 shows an algorithm that is based on the above design pattern. To accelerate the search process, a subdivision tree structure is introduced to divide the space into sub-regions, so that the closet triangles to the input triangle can be sorted out for search and merge. As shown in Figure 3-16, when each triangle is read, the existing

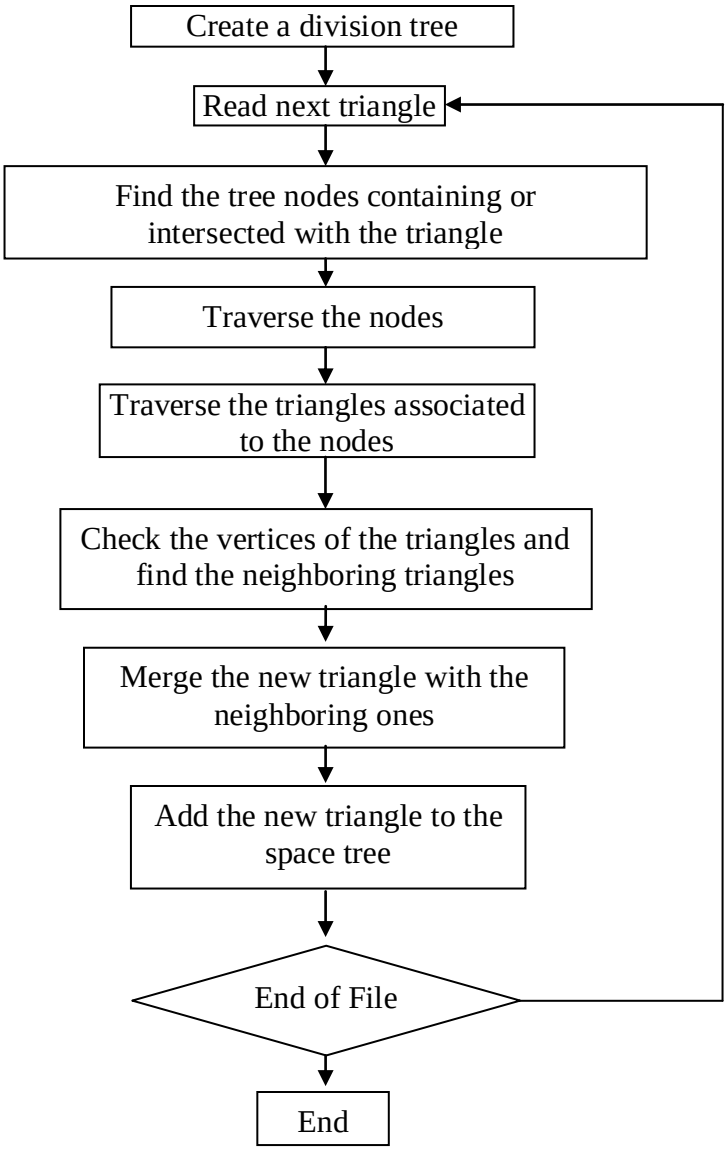


Figure 3-15 Reconstruction of B-rep

vertices will be compared to the new triangle. If none of the existing vertices is close enough to the new vertices, a new vertex will be created, but if there is a vertex that has the same position to the new vertex, the vertex will be re-used to form the new triangle. After the three vertices are obtained, the new face will be created. The neighboring triangles of a vertex can be obtained from its emanating edges. If the input triangle shares the two vertices with any existing triangle,

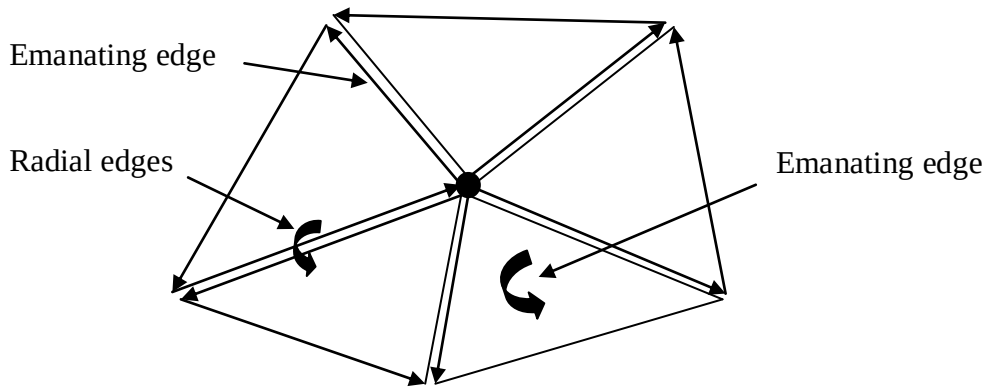


Figure 3-16 Topologies around a vertex

the topological information of its faces and edges are formed to connect the triangle with the other triangles. The topological information to be formed includes the radial edges, the emanating edges of the vertices, and the edge loops of the face, as shown in Figure 3-16.

One advantage of this structure is that the neighboring vertices of any vertex can easily be found, because they are all connected through emanating edges. Also, with this structure, each facet can be polygonal instead of triangular, and can consist of internal loops in a CCW direction. In this part, the STL model is limited to triangles only.

3.2.5 Memory Management

Object and Container classes All the classes are based on either the UObject class or UContainer class. UObject class is a global base for all the objects that need to be "used" (allocated) in the memory pool. As the opposite of this, the UContainer is the class that contains (manages) the memory pool. Figures 3-17 and 3-18 show the class declaration of object and

container. Object is an abstract class for which certain methods are enforced to implement in its sub-classes. It defines a new operator that will take the designated memory pool and reallocate the memory.

Container contains a member of UMemPool. It is a member that maintains and manages all the memory usage in the heap. Because of the large size of B-rep, especially for polygon B-rep, there needs a way to quickly allocate, de-allocate and access the heap. Normally, a memory pool mechanism will be implemented to manage the heap usage for large sized objects [55, 56]. UMemPool is based on a structure very similar to a regular heap manager[57], but provides a mechanism to dynamically allocate and stack heap blocks, and a way to rapidly access the memory block without the limitation of fragmentation.

The UContainer can have multiple memory pools based on the types of object that it can contain. One example of a container is B-rep. The container of B-rep has four memory pools, one for vertices, one for edges, one for loops and one for faces. The major benefits of a mem_pool is improving the speed of allocation, traverse and deletion.

```
namespace UCAM
{
// Context object, used to support the memory management
// It is the base class for the classes to be allocated in the memory pool
class DLL_EXPORT UObject
{
public:
//Constructors
        UObject() : m_index(0), m_visitingmarker(0) {};
        void *operator new(size_t num_bytes, MemPoolHndl heap)
        {
            if(!heap)
                return malloc(num_bytes);
            else
                return (SMEM_create_element(heap));
        }
        virtual ~ UObject ();
        UObject(const UObject &) {};

        virtual void Dump() const;
        virtual void AssertValid() const;
//Accessors of index
        void SetIndex(int id);
        ULONG GetIndex();
//Operators of bounding box
        virtual void JoinDomain(UDomain3D* rDomain) = 0;
        virtual bool IsContainedBy(UDomain3D * rDomain) = 0;
        virtual UDomain3D GetDomain3D() = 0;
//Operators of visting methods
        virtual void Accept(iVisitor* cgVisitor) = 0;
        virtual void IncrementVisitingMarker() { m_visitingmarker++;}
        virtual ULONG GetVisitingMarker() { return m_visitingmarker;}

private:
        ULONG m_index;
        ULONG m_visitingmarker;
};
typedef std::list< UObject *> OBJECT_LIST;
typedef std::vector< UObject *> OBJECT_VECTOR;
}
```

Figure 3-17 Declaration of object

```
namespace UCAM
{
// This is the base level object for memory pool. It provides the mechanism
//memory management and allocation
class DLL_EXPORT UContainer
{
public:
//Constructor
        UContainer () : mem_pool(NULL), m_ownerOfMemory(false){};
        UContainer (int poolCount, bool ownerofMemory);
        virtual ~ UContainer ();

//Modifier of ownership
        inline void SetOwnerOfMemory(const bool isOwnerOfMemory);
        inline bool IsOwnerOfMemory();

//Creator of memory pool handler
        UMemPoolHndl CreateNewPool(int ind, int elementSize, int bufferSize);
        UMemPoolHndl GetMemPoolHndl(int index) const ;
        inline int GetPoolCount() const;

        inline void Free();
        virtual void DumpPool();
protected:
// Copy constructor is not allowed for memory pool management
        UContainer(const UContainer & source) : mem_pool(source.mem_pool),
        m_ownerOfMemory(false) {};
private:
        bool m_ownerOfMemory;
        UMemPool mem_pool;
};
}
```

Figure 3-18 Declaration of container

Figure 3-19 illustrates the declaration of UMemPool, which is based on a stacked heap mechanism. The normal mechanism of the memory management for B-rep is very similar to the memory pool management. But because B-rep normally has triangles of a large size, the memory must have good expandability. The regular heap structure has a fixed size memory pool. If the expanding size is bigger than the total size of the allocated memory, it is sometimes impossible to expand the memory size of the pool in the original address because of memory fragmentation, which blocks the HEAP from expanding from the original address. In this case, the original allocated memory pool is copied to a new address for the new size, as shown in Figure 3-20(a).

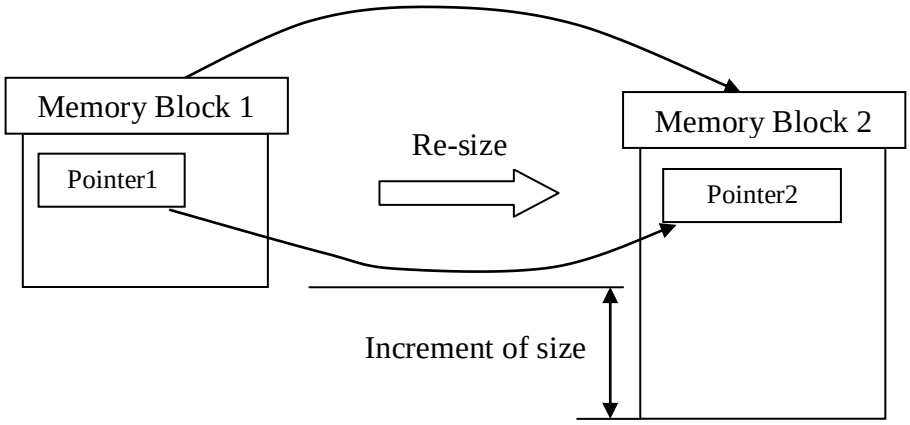
```
// This object contains a system dependent pointer to a memory pool.
// The deletion of this object will clean up the pool.
#define UMemPoolHndl char*
namespace UCAM
{
class DLL_EXPORT UMemPool
{
friend class UContainer;
public:
   UMemPool();
   UMemPool(int m_poolCountl) ;

   virtual ~ UMemPool();

   inline void Free();
   inline void DeletePool(int ind);
   UMemPoolHndl CreateNewPool(int ind, int elementSize, int bufferSize);
   UMemPoolHndl AddNewPool();

protected:
   UMemPoolHndl* m_memPools;
   int m_poolCount;
};
}
```

Figure 3-19 Declaration of UMemPool

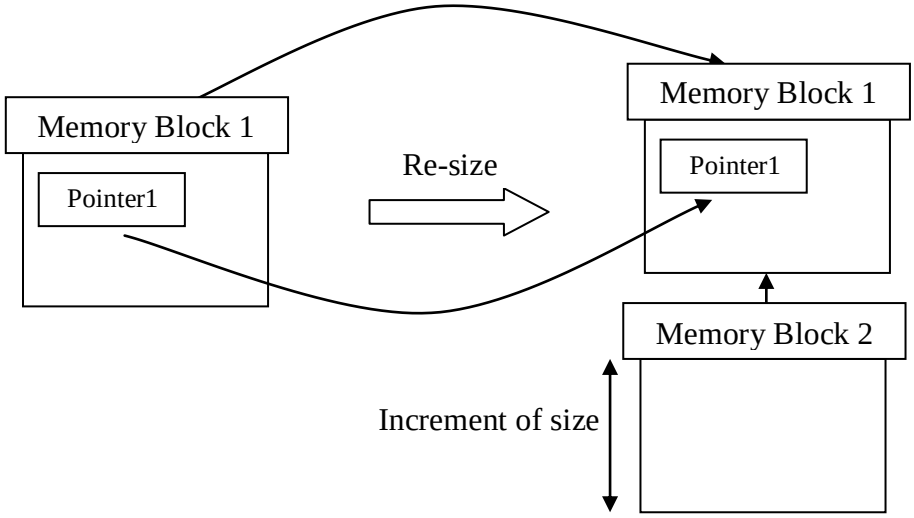
But the memory in the original address will be freed after resizing. If there are external references pointing to these old addresses that are allocated in the original heap, they need to be updated with the new address. This method is not feasible in B-rep, because all the topology information of B-rep entities is carried by referencing the address of the entities. It is difficult to achieve rapid updates for all these topological references with the new addresses. Figure 3-20(b) shows another data structure that preserves all the allocated memory blocks when the pool needs to be resized. Instead of deleting and copying the original memory block to a new address, UMemPool provides a link-list for all the allocated memory blocks. If the expanding size is bigger than the total size of the allocated memory pool, an additional memory block will be allocated in a new address, and this new address will be stacked to the existing list of allocated memory blocks. In this way, the resizing operation does not need to update any external references. Because the memory blocks that have been allocated are preserved in a link-list, the deletion operation of the memory pool simply involves traversing all the elements of this link-list, and freeing each of them. For the detailed function information of stacked heap allocation, please refer to Appendix C.

Meanwhile, the UMemPool can also have multiple pools of different individual element sizes. For example, in polygon B-rep, the objects of vertex, edge, loop and face are all of different size, and when we create the instance for each of them, they should allocate in different pools. Therefore, UMemPool for polygon B-rep should have four memory pools. With this mechanism, all the objects of the same type will be allocated in the array type of memory blocks. Therefore, the traversing of objects is much faster than the mechanism of dynamic allocation. Because we allocate all the memory in a memory block, it is easy to delete. It can contain multiple memory

pools for different types of object. The client can dynamically create the new pool based on the type of the objects to be created in the class. Also, this solution resolves the problem of expandability that we encounter for the regular heap.



(a) Resize operation of Heap



(b) Resize operation of UMemPool

Figure 3-20 Dynamic expansion of stacked memory pool

4

PARAMETERIZATION OF FREEFORM SOLID AND MORPHING BASED MACHINING PLANNING

4.1 **Overview of the multistage rough machining based on solid parameterization and boundary constraint morpheme**

To validate the feasibility of the data scheme of manufacturing oriented model, we use a machining feature as a study case. The machining feature is defined as the feature specifically for a machining operation. It is a volume of material to be removed from the in-process stock by using machining operation, and it normally represented as a desired surface, initial surface and the surrounding boundary geometries.

To protect the tool from damage, the material must not be removed by only one pass if the depth of the cut volume exceeds the permissible depth of cutter. Therefore, the machining procedure begins with stock material normally going into three phases: rough cutting, semi-

finish, and finish, as show in Figure 4-1. Rough cutting brings the material near the required shape. The semi-finish performed on the near shape brings it within the dimension tolerance, and the finishing machining is used to polish or to keep the surface within the finishing tolerance. The tool path for each of these phases has to be generated with different concerns: for rough cutting, the tool path should be designed productive within the limitation of the cutter's life-expectation; for the last two steps, there is more compromise between the speed and surface quality. To meet the seamless integration of CAM, a good machining algorithm should meet the following requirements:

i. It should be independent of the form of geometrical representation. Various algorithms have been created aiming at different geometrical models. But since the CIM is a highly

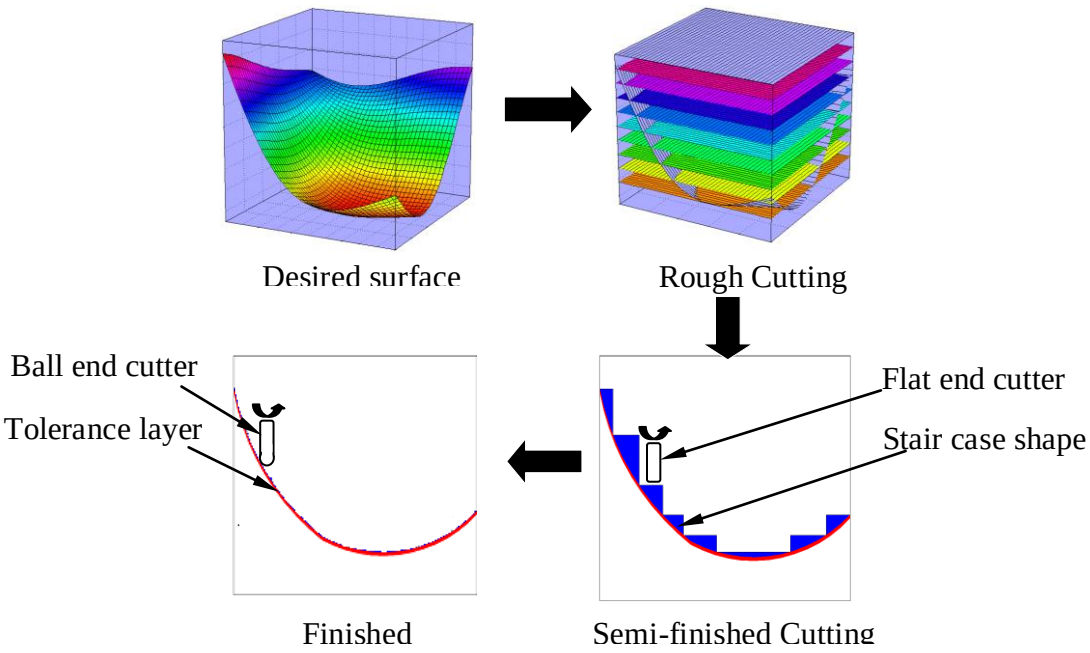


Figure 4-1 Conventional multi-stage machining

intelligent system, it should be applicable to any CAD model provided by any designing features. Regardless of whether it is represented by the discrete data points or parametric description, the algorithm will be able to generate the fabrication plan with a unified scheme.

ii. It should be gauge-free, implicitly avoiding the problems of gouge, over-cutting, and under-cutting. Martin [58] and Tao [59] addressed these issues in their research, but limitations of knowledge about the local analytic properties on the machined surface makes it difficult to avoid gouging and over-cutting.

iii. It should be boundary conformed. Compared with other methods, the boundary-conformed method, in which the cutting tip is pulled up and down to process the disconnected segments in case of non-continuous tool path, is efficient considering the time consumed. Also, the boundary-conformed method can improve the quality of surface at the area around the edge. Although the pocketing method [58] is to some extent boundary conformed by offsetting or scaling the boundary profile, the main application of this machining method includes mold and die cavities only. Yang et al. [60, 61] provide a good alternative to generate the conformed tool path based on the mapping theories.

4.2 Process planning for 5-axis rough milling

In order to meet the requirements of machining alogrithm, handle the complexity of CAD geometry and obtain the maximum cutting speed, most of these researches use 3-axis machining for rough machining [62, 63]. However, 5-axis is superior to 3-axis machining in many respects:

i. Shorter cutting tools can be used for 5-axis machining. Since the tool can be tilted to adjust the angle between the cutter and the part, the increased rigidity of shorter tools can take

advantage of the high-speed options with no loss in accuracy. The result has better surface quality and reduced finishing time.

ii. Generating 5-axis tool paths instead of traditional 3-axis tool paths can result in fewer cut passes and improved surface quality. In many applications, such as blade machining, the accessibility of 3-axis machining is very limited.

iii. By optimizing the angle between the tool and the surface, it is possible to achieve a constant chip load and a high feed rate at the contact point. The result is improved surface finish and extended tool life.

iv. Parts that previously required multiple setups can be machined in a single setup with simultaneous control of the rotary axis. In addition to saving time, this also cuts down on mistakes that might be made during multiple setups.

Nevertheless, although 5-axis is superior to 3-axis machining in these respects, there is lack of methodology to implement the 5-axis machining in the roughing process.

4.3 Morphing-based Process Planning

In the current research, the concept of parameterization technology is extended to 3-D space, where the whole machining volume will be split into surfaces that are conformed to the volume boundary surfaces. The interim surfaces are interpolated from the initial surface and the desired surface. The major improvement of the new cutting strategy is that, while the procedure from rough cutting to finish cutting produces a series of surfaces morphing between the initial surface to the desired surface, as shown in Figure 4-2, the conventional approach of rough cutting [62-66] is to divide the machining volume into parallel slices. Each slice is cut into a staircase-shape

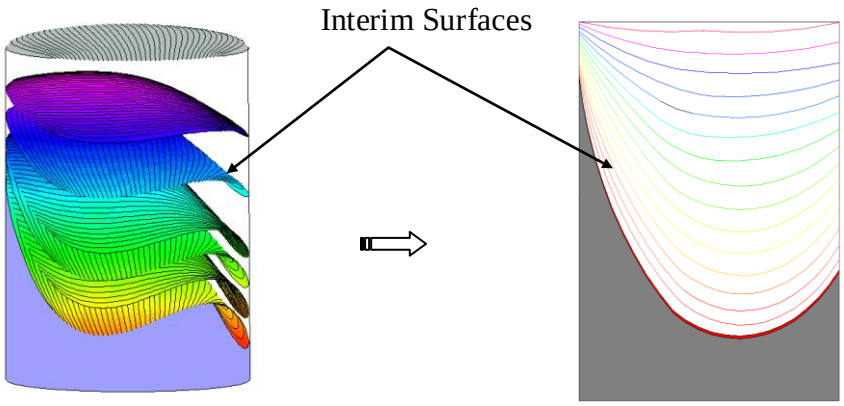


Figure 4-2 the metamorphosis of the intermediate surfaces

leaving a small amount of uncut allowance for semi-finishing or finishing processes. This method is only valid for 3-axis roughing process. Also because complicated numerical methods have to be introduced to calculate the enclosing contour and islands for each slice, it is not an efficient method and cannot be implemented as the unified method for tool path generation. In addition, the tool path does not conform to the boundaries. In practice, additional cutting is needed to clean the remaining material along the boundaries. For complicated geometric models with extreme irregular boundary surfaces, this method is weak, offering few errorless results.

With a morphing-type transformation, the machined profiles are determined in such a way that the intermediate surfaces conform to the initial surfaces, the desired surface and the bounding geometries. The local analytic properties of each intermediate surface are explicitly predicable on the initial surface, final surface and boundary facets. There is no additional driving geometry needed for the tool path generation and collision avoidance.

4.4 **Overview of geometry morphing**

According to Douglas DeCarlo and Jean Gallier [67], the task of transforming one model into another epitomizes the general problem of metamorphosis between two objects, commonly called "morphing". Morphing starts from a correspondence between the two objects, which specifies where features on one object end up on the other objects as a result of the transformation. When the transformation involves topological change, the correspondence must also indicate how the change takes place. The morphing engine effects a transition that realizes the desired correspondence using a method of interpolation.

In recent years, image-morphing techniques have gained considerably in popularity for computer geometry, especially in the entertainment industry. Unfortunately, any intermediate forms produced by image-morphing methods exist only in image form. Current research represents for the first time the concept of surface metamorphosis to be applied to manufacturing process planning.

Meanwhile, most research into 3D surface morphing has focused on either morphing between a restricted, topologically similar class of shapes, or automatically constructing the correspondence between the two shapes for a morph. Very often, the user has little or no say in how the morph takes place. For topologically similar objects, a smooth morphing process has the following properties:

i. Over the course of the transformation, no discontinuous jumps in shape are present

ii. No undesirable topological changes occur, such as the splitting open of a surface or distortion.

The surface topology of the shape is specified by the local metric of the surface. For example, a sphere and torus have different surface topologies. As discussed in the first section, the mesh topology is specified by the graphic connectivity of the mesh. The deformation of a shape (a geometrical change) does not change either the mesh or surface topology.

Introduction of the metamorphosis of 3D surfaces inaugurates a new area for process planning, making possible the unified tool path-planning scheme, namely the manufacturing process can actually be defined as the geometry metamorphosis from initial stock to the final desired object. These two end objects can be linked with a set of interim forms, as shown in Figure 4-2. Instead of applying different strategies for rough cutting and finish cutting, we can plan the tool path on each of these interim surfaces in a unified manner, as what we do in the traditional finish machining. The main advantage of this development is that it dramatically simplifies the process of tool-path planning. The algorithm is independent of feature type. With correct adjustment it can theoretically be applied on any form of surface. Meanwhile, the scheme supplies boundary-conformed tool paths, which allows the machining of the parts to irregular geometrical properties, such as spars or cast raw materials. The description of each to-be-manufactured feature is straightforward and consists of initial surface, desired surface and boundaries. Since the resulting tool paths are boundary conformed, it does not bring into staircase-shape on the surface, ensuring the surface qualities. The process of 3D surface morphing usually consists of two issues:

i. The specification of the correspondence. Particularly for the faces of the control mesh, the correspondences induce correspondences between the points on the initial and final objects.

Moreover, the discrepancies between the structures of the corresponding faces describe the topological evolution that must occur during the transformation.

ii. The interpolation algorithm, which is used to describe the issues arising in the presence of geometric evolution.

In the example illustrated in Figure 4-2, the object model is divided into three machining features. After correspondence, each of them corresponds to one face of the initial stock. The whole cut volume is consequently divided into a set of sub-domains that are bounded with intermediate surfaces. The machining features from CAPP are actually the description of each member in this family. Based on the particular characteristics, the machining agent generates the corresponding intermediate surfaces. In the following section, parametric implicit model will be introduced to present the evolution from initial surface to the desired surface for a generic machining feature.

4.5 **Trivariate Representation**

In order to illustrate the geometry morphing of machining process, we need a modeling methodology that can represent the cut volume as a 3D solid, instead of describing it by specifying the boundary surfaces as CAD modelers do. It in some cases is trivial and inefficient, while the more natural method for CAM is to deal with the object in 3D space as a solid described by parametric function in terms of three variations. This set of description schemes is the so-called trivariate representation and is a direct extension of 2D patch. The trivariate equation can be used to represent either the explicit or implicit solids.

The following is a general form of trivariate equations for explicit solid:

$$\mathbf{F}(u,v,w) = (x(u,v,w), y(u,v,w), z(u,v,w)) \quad \ldots(4\text{-}1)$$

For $u \in [0,1], v \in [0,1], w \in [0,1]$;

To render this model in the conventional surface-based rendering system, the trivariate-represented model usually needs to be converted to B-rep [68, 69]. But as a method of creating, modifying and visualizing freeform geometric models, trivariate representation is efficient and has been shown to be useful for manipulating the dynamic behavior of the object, such as deformation, animation, etc. Examples of explicit models are B-spline hyperpatch [68], COONs, Swept Volume [69] and Loft Volume, etc.

4.6 Parametric Implicit Solid Modeler(PISM)

4.6.1 Boundary surfaces of cut volume

When the trivariate equation is in the form of implicit function, the solid represented by this type of equation is implicit solid modeling (ISM). ISM represents a solid as the set of points at which an implicit global defining function takes on a value less than a given threshold value. One common feature essential to all solid modeling methods is creating a surface that partitions the space into two regions. The natural mathematical description of such a spatially partitioned region consists of an implicit function $\mathbf{F}(x, y, z)$ and a cutoff value associated with the surface, where the interior and exterior of the solid can be set as $\mathbf{F}(x, y, z) < f_0$ and $\mathbf{F}(x, y, z) > f_0$ respectively. This implicit form supplies a very convenient way for modelers to perform various operations on the geometrical model.

The parametric form of ISM in 3D dimension space is $\mathbf{F}(x, y, z, u, v, w) = 0$, for $u \in [0,1], v \in [0,1], w \in [0,1]$, which is implemented to map the coordinate set of interior points to the normalized parametric space. The parametric form of ISM has been shown to be very useful for structured grids, typically employed for FEM problems and FDL problems. One good example is the ellipse equation used to generate the interior grids based on the boundary surfaces where the mapped parametric set (*u, v, w*) equals 0 or 1.

$$\begin{cases} r_{xx} + r_{yy} + r_{zz} = 0 \\ s_{xx} + s_{yy} + s_{zz} = 0 \\ t_{xx} + t_{yy} + t_{zz} = 0 \end{cases} \quad \ldots(4\text{-}2)$$

Given the implicit properties of smoothness and continuity, it has shown many advantages over other grid methods. Most of the existing solid modelers accommodate the creation and manipulation of the complete, unambiguous mathematical representations for 3-D objects. The usual purpose is to provide information needed to perform the calculations associated with the process of geometric design, visualization, and calculation of the shape-related properties such as volume, mass, moments of inertia, surface area, and convex hulls. Solid modelers usually do not support just one category of scheme, but combine all these methods to reach an optimized trade off, allowing data exchanges between different models.

The machining feature can be described in a variety of ways. Various representation schemes can be used in the commercial CAD programs. Inspired by the work of Yang et al.[70], in the current research boundary-fitted structured grids are selected to represent the surface in the form

of a network of curvilinear coordinate lines, as shown is Figure 4-5. One-to-one mapping can be established between 3D and parametric domain, and the curvilinear grid points also conform to the solid boundaries. Various parametric representations of freeform surfaces have been studied thoroughly in many works: B-spline surface [71], NURBS surface [45], COONs patch [46], sixteen-point form, four-curve form, ruled surface, Bezier surface, etc. Most of the parameterization methods design surfaces are based on the interpolation of driving nets or driving curves, of which the initiative curves are generated, based on design parameters or constraints, and then faired along the boundaries. While a few algorithms, e.g. ruled surface and COONs patch, can derive the iso-parametric patches directly from boundary curves or surfaces, they are all linear interpolation technology and are too simple to be used for more complicated cases, such as surfaces with trimmed edges.

Figure 4-3 illustrates the new parameterization method for trimmed surface[60, 61], where $\boldsymbol{S}_o$ in Fig. 4-3(a) represents the original surface; $\boldsymbol{C}_1$, $\boldsymbol{C}_0$, $\boldsymbol{D}_1$, $\boldsymbol{D}_0$ the bounding curves of $\boldsymbol{S}_a$, which is a portion of $\boldsymbol{S}_o$ trimmed by $\boldsymbol{C}_1$, $\boldsymbol{C}_0$, $\boldsymbol{D}_1$, and $\boldsymbol{D}_0$. If we set the function of $\boldsymbol{S}_o$ as

$$S_o(u,v) = S_o(x(u,v), y(u,v), z(u,v)) \quad (u,v \in [0,1]) \qquad \ldots(4\text{-}3)$$

The parametric curves of $\boldsymbol{C}_1$, $\boldsymbol{C}_0$, $\boldsymbol{D}_1$, $\boldsymbol{D}_0$ in (u, v) space can be represented as $\boldsymbol{C}_1'$, $\boldsymbol{C}_0'$, $\boldsymbol{D}_1'$, $\boldsymbol{D}_0'$

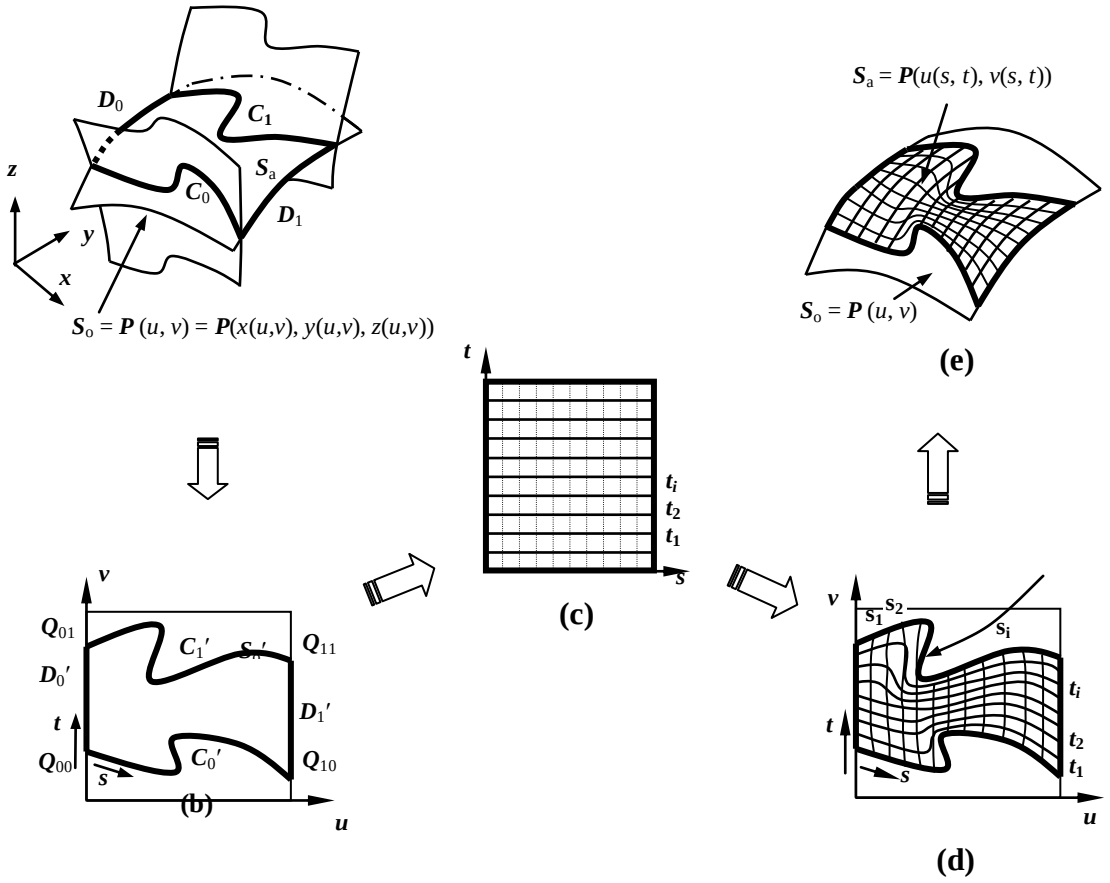


Figure 4-3 Two-phase mapping of boundary-conformed grids

as shown in Figure 4-3(b). $\boldsymbol{S}_a'$ represents the 2D parametric domain of $\boldsymbol{S}_a$. These four parametric curves in terms of (s,t) $s,t \in [0,1]$ can form a new parametric space (s,t). The domain $\boldsymbol{S}_a'$ can subsequently be extracted to a unit square $\boldsymbol{S}_a''$ in (s,t) space as shown in Figure 4-3(c), and iso-t_i's network in Figure 4-3(d) is the new boundary-conformed iso-parametric curve. The resultant boundary-conformed grids after the second phase mapping back to the *xyz* object space are illustrated in Figure 4-3(e).

In the first phase of mapping, $\boldsymbol{C}_1$', $\boldsymbol{C}_0$', $\boldsymbol{D}_1$', $\boldsymbol{D}_0$' in (u, v) can be obtained easily with RE techniques. Assuming these four parametric curves can be represented as,

$$\begin{cases} C_1' = C_1'(s,1) & s \in [0,1] \\ C_0' = C_0'(s,0) & s \in [0,1] \\ D_1' = D_1'(1,t) & t \in [0,1] \\ D_0' = D_0'(0,t) & t \in [0,1] \end{cases} \quad \ldots(4\text{-}4)$$

The major problem is how to grid the domain $\boldsymbol{S}_a'$ with the boundary interpolation. Thus far, there are three methods available to solve this problem: algebraic interpolation, partial differential equations, and hyperbolic systems.

If a similar concept is extended to the domain of 3D volume description, we can obtain a new evolution algorithm based on structured parametric grids. Figure 4-4 illustrates the mapping between physical space and the parametric space. The volume $\boldsymbol{V}$ in physical space represents the cut volume. Assuming $\boldsymbol{f_6}$ as the face on the rough stock, $\boldsymbol{f_5}$ the corresponding to-be-manufactured feature, and each of them enclosed by four edges, since both of these two surfaces have

equivalent topology, we can link them with four edges, l_1, l_2, l_3, l_4. Eventually, the internal grids in physical space can be obtained by mapping it to an orthogonal cubic in parametric space. From the perspective of surface morphing, the transformation between these two surfaces falls into the problem of structured grid generation, which uses a network of curvilinear coordinates to represent the 3D volume such that a one-to-one mapping can be determined between the physical and parametric space.

l_1, l_2, l_3, l_4 is the swept trace of the four vertices on the surface. If they are described as a function in terms of the time parameter $l_1(t), l_2(t), l_3(t), l_4(t)$, the interim surface at any instant time t can be represented as:

$$F(r,s,t)\ ,\ r,s,t \in [0,1] \qquad \ldots (4\text{-}5)$$

Where the boundary condition applies on the surface, i.e.

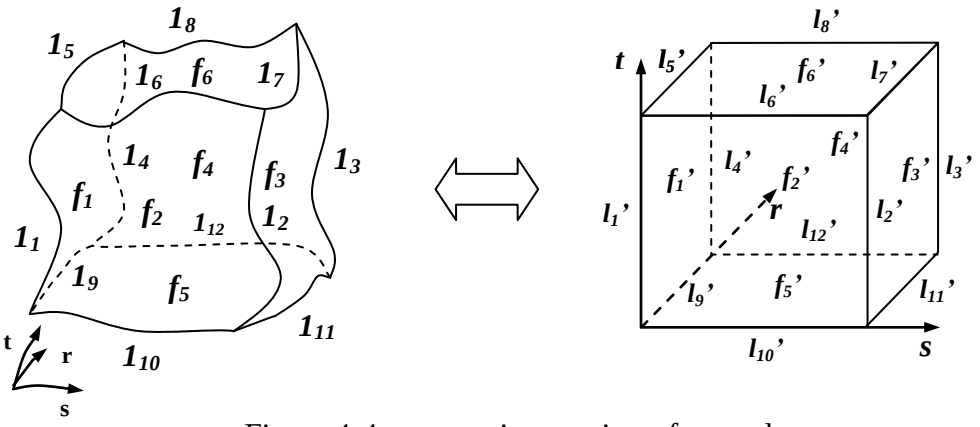


Figure 4-4 parametric mapping of cut volume

$F(r,s,0) = f_5;\ F(r,s,1) = f_6;$

$F(r,0,t) = f_1;\ F(r,1,t) = f_3;$

$F(0,s,t) = f_2;\ F(1,s,t) = f_4;$

The boundary conditions on the edges are:

$F(0,0,t) = l_1;\ F(0,1,t) = l_2; F(1,0,t) = l_4; F(1,1,t) = l_3$

$F(0,s,0) = l_{10};\ F(0,s,1) = l_6; F(1,s,0) = l_{12}; F(1,s,1) = l_8$

$F(r,0,0) = l_9;\ F(r,0,1) = l_5;\ F(r,1,0) = l_{11};\ F(r,1,1) = l_7$

On the vertices:

$F(0,0,0) = l_1(0);\ F(0,1,0) = l_2(0); F(1,0,0) = l_4(0); F(1,1,0) = l_3(0)$

$F(0,0,1) = l_1(1);\ F(0,1,1) = l_2(1); F(1,0,1) = l_4(1); F(1,1,1) = l_3(1)$

4.6.2 Laplace solution of PISM

Imposing these boundary conditions, the algorithm in our scheme involves two phases: initialization and fairing.

Initialization: The purpose of this phase is to obtain initial grids from known boundaries. This is done directly by linear interpolation from the boundaries in the algebraic grid generation

technique that builds the grid by using transfinite interpolation in parametric form and taking the Boolean sum of the interpolation projectors as:

$$P_r \oplus P_s \oplus P_t = P_r + P_s + P_t - P_r P_s - P_r P_t - P_s P_t + P_r P_s P_t \quad \ldots (4\text{-}6)$$

where the shearing transformation between two opposite surfaces is expressed as:

$$\begin{aligned} P_r &= (1 - f(r))F(0,s,t) + f(r)F(1,s,t) \\ P_s &= (1 - g(s))F(r,0,t) + g(s)F(r,1,t) \\ P_t &= (1 - h(t))F(r,s,0) + h(r)F(r,s,1) \end{aligned} \quad \ldots (4\text{-}7)$$

The sum $P_r + P_s + P_t$ represents the interpolant, namely

$$P_r + P_s + P_t = \begin{bmatrix} 1 - f(r) & f(r) \end{bmatrix} \begin{bmatrix} F(0,s,t) \\ F(1,s,t) \end{bmatrix} + \begin{bmatrix} 1 - g(s) & g(s) \end{bmatrix} \begin{bmatrix} F(r,0,t) \\ F(r,1,t) \end{bmatrix} + \begin{bmatrix} 1 - h(t) & h(t) \end{bmatrix} \begin{bmatrix} F(r,s,0) \\ F(r,s,1) \end{bmatrix}$$

$\ldots . (4\text{-}8)$

where $f(r)$, $g(s)$ and $h(t)$ are the interpolation functions, and the simplest form is *r, s, t*. Higher order interpolation function can be used to control the distribution of the grid points.

In order to coincide with the boundary values on the edge, it is necessary to construct a second order interpolant $P_r P_s$, $P_r P_t$, $P_s P_t$, that contains all the redundant boundary interpolants, namely:

$$P_r P_s = [1-f(r),\ f(r)]\begin{bmatrix} F(0,0,t) & F(1,0,t) \\ F(0,1,t) & F(1,1,t) \end{bmatrix}\begin{bmatrix} 1-g(s) \\ g(s) \end{bmatrix} \quad \ldots(4\text{-}9)$$

$$P_s P_t = [1-g(s),\ g(s)]\begin{bmatrix} F(r,0,0) & F(r,1,0) \\ F(r,0,1) & F(r,1,1) \end{bmatrix}\begin{bmatrix} 1-h(t) \\ h(t) \end{bmatrix} \quad \ldots(4\text{-}10)$$

$$P_r P_t = [1-f(r),\ f(r)]\begin{bmatrix} F(0,s,0) & F(1,s,0) \\ F(1,s,0) & F(1,s,1) \end{bmatrix}\begin{bmatrix} 1-h(t) \\ h(t) \end{bmatrix} \quad \ldots (4\text{-}11)$$

Similarly, the third order is used to fit the values on the vertices:

$$\begin{aligned} P_r P_s P_t = {} & f(r)\begin{bmatrix} g(s) & 1-g(s) \end{bmatrix}\begin{bmatrix} F(1,1,1) & F(1,1,0) \\ F(1,0,1) & F(1,0,0) \end{bmatrix}\begin{bmatrix} h(t) \\ 1-h(t) \end{bmatrix} + \\ & (1-f(r))\begin{bmatrix} g(s) & 1-g(s) \end{bmatrix}\begin{bmatrix} F(0,1,1) & F(0,1,0) \\ F(0,0,1) & F(0,0,0) \end{bmatrix}\begin{bmatrix} h(t) \\ 1-h(t) \end{bmatrix} \end{aligned} \quad \ldots (4\text{-}12)$$

Assuming the boundary surfaces and edges result from COONs interpolation, one can simplify Equation (4-11) to:

$$\begin{aligned} \vec{f}(s,t,r) = {} & \frac{2}{3}\{([1-r,r]\begin{bmatrix} F(0,s,t) \\ F(1,s,t) \end{bmatrix} + [1-s,s]\begin{bmatrix} F(r,0,t) \\ F(r,1,t) \end{bmatrix} + \\ & [1-t,t]\begin{bmatrix} F(r,s,0) \\ F(r,s,1) \end{bmatrix}\} - \frac{1}{3}\{\ [t,1-t]\begin{bmatrix} F(r,0,1)\ F(r,1,1) \\ F(r,0,1)\ F(r,1,0) \end{bmatrix}\begin{bmatrix} 1-s \\ s \end{bmatrix} + \\ & [t,1-t]\begin{bmatrix} F(0,s,1)\ F(1,s,1) \\ F(0,s,0)\ F(1,s,0) \end{bmatrix}\begin{bmatrix} 1-r \\ r \end{bmatrix} + [r,1-r]\begin{bmatrix} F(1,0,t)\ F(1,1,t) \\ F(0,0,t)\ F(0,1,t) \end{bmatrix}\begin{bmatrix} 1-s \\ s \end{bmatrix}\ \} \end{aligned} \quad \ldots (4\text{-}13)$$

Figure 4-5 shows the initiative result from the above interpolation. Equation (4-13) is actually not a generic form of derived interpolation function. The traditional process planning is

normally based on this parameterization model. COONs method is a linear method, and it therefore preserves the discontinuity of boundary geometries. It is inaccurate in the cases in which the boundary surfaces are not well interpolated. In addition, the grid generated by means of COONs is simple and might contain self-intersection for complicated shapes. Because it is a first-order interpolation, the discontinuity on the boundary surface will be propagated to the intermediate surfaces. The grid will probably exceed the constraints of the boundary surface, and the second phase is therefore implemented to smooth and fair the grid through nonlinear interpolation.

Fairing: Due to the limitations of COONs interpolation, a further adaptation and smoothing of the interior grid is implemented to smooth the overlapping or nonlinear portions. By introducing the partial differential equation, elliptic equation offers a natural smoothing effect that inhibits jumps or discontinuities, and this makes them ideally suitable for re-adaptation. Due to its propensity to minimize tension energy, elliptic equations are able to smooth boundary data, and this affords a most desirable property. Many elliptic equations are based on Laplace's equation

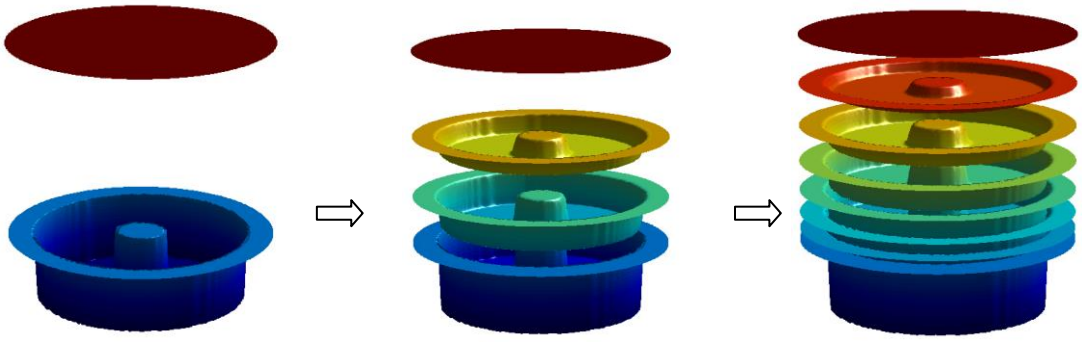


Figure 4-5 Case of concave surface, with COONs

that is well known as the smoothing operator. Laplace's equation is most appropriate in a computational fluid dynamic, which is used to describe the non-viscous steady and incompressible flow, i.e.

$$\begin{cases} r_{xx} + r_{yy} + r_{zz} = 0 \\ s_{xx} + s_{yy} + s_{zz} = 0 \\ t_{xx} + t_{yy} + t_{zz} = 0 \end{cases} \quad \ldots (4\text{-}14)$$

With boundary condition:

$$\begin{aligned} r = 0,1: &\quad (x, y, z) = f_5; f_6 \\ s = 0,1: &\quad (x, y, z) = f_1; f_3 \\ t = 0,1: &\quad (x, y, z) = f_2; f_4 \end{aligned}$$

where *(r, s, t)* are the parametric set of (*x, y, z)*. When the points are located on the boundary surfaces, one of the parameters should be a constant of 0 or 1. The coordinates of interior grids are derived from the boundaries. To utilize these ideas for mesh generation, it is more convenient to transform these equations so that (*x, y, z*) become the dependent variables. In such a case, it is then possible to apply boundary conditions to (*x, y, z*), which in general will be the known boundary. Transforming the variable in (4-14) leads to

$$\alpha_{11}\mathrm{r}_{rr} + \alpha_{22}\mathrm{r}_{ss} + \alpha_{33}\mathrm{r}_{tt} + 2(\alpha_{12}\mathrm{r}_{rs} + \alpha_{13}\mathrm{r}_{st} + \alpha_{23}\mathrm{r}_{rt}) = 0 \quad \ldots (4\text{-}15)$$

where

$\mathbf{r} = (\mathrm{x}, \mathrm{y}, \mathrm{z})^{\mathrm{T}}$, $\qquad \alpha_{ij} = \sum_{m=1}^{3} \gamma_{mi}\gamma_{mj}$

γ_{ij} is the ij-th cofactor of the Jacobian matrix:

$\begin{bmatrix} x_r & x_s & x_t \\ y_r & y_s & y_t \\ z_r & z_s & z_t \end{bmatrix}$ and J presents the determinant of the Jacobian matrix.

$$J = \begin{vmatrix} x_r & x_s & x_t \\ y_r & y_s & y_t \\ z_r & z_s & z_t \end{vmatrix}$$

It has been proved [72] that these coupled nonlinear partial differential equations maximize mesh smoothness. The solution of this equation system can be obtained by linearization and an overridden representation of the derivatives, i.e. the residual on a square mesh with $r = ih_r, s = jh_s, t = kh_t$ can be represented with the override value of previous nodes:

$$\begin{aligned} R^n_{i,j,k} &= \alpha^n_{11}(\mathbf{r}^{\mathbf{n}}_{i+1,j,k} - 2\mathbf{r}^{\mathbf{n}}_{i,j,k} + \mathbf{r}^{\mathbf{n+1}}_{i-1,j,k}) + \alpha^n_{22}(\mathbf{r}^{\mathbf{n}}_{i,j+1,k} - 2\mathbf{r}^{\mathbf{n}}_{i,j,k} + \mathbf{r}^{\mathbf{n+1}}_{i,j-1,k}) + \alpha^n_{33}(\mathbf{r}^{\mathbf{n}}_{i,j,k+1} - 2\mathbf{r}^{n}_{i,j,k} + \mathbf{r}^{\mathbf{n+1}}_{i,j,k-1}) \\ &+ [\alpha^n_{12}(\mathbf{r}^{\mathbf{n}}_{i+1,j+1,k} - \mathbf{r}^{\mathbf{n+1}}_{i-1,j+1,k} - \mathbf{r}^{\mathbf{n}}_{i+1,j-1,k} + \mathbf{r}^{\mathbf{n+1}}_{i-1,j-1,k}) \\ &+ \alpha^n_{13}(\mathbf{r}^{\mathbf{n}}_{i+1,j,k+1} - \mathbf{r}^{\mathbf{n+1}}_{i-1,j,k+1} - \mathbf{r}^{\mathbf{n}}_{i+1,j,k-1} + \mathbf{r}^{\mathbf{n+1}}_{i-1,j,k-1}) \\ &+ \alpha^n_{23}(\mathbf{r}^{\mathbf{n}}_{i,j+1,k+1} - \mathbf{r}^{\mathbf{n+1}}_{i,j-1,k+1} - \mathbf{r}^{\mathbf{n}}_{i,j+1,k-1} + \mathbf{r}^{\mathbf{n+1}}_{i,j-1,k-1})]/2 \end{aligned} \quad \ldots (4\text{-}16)$$

The overridden difference form of Jacobian matrix is represented as:

$$J^n_{i,j,k} = \begin{bmatrix} x^n_{i+1,j,k} - x^{n+1}_{i-1,j,k} & x^n_{i,j+1,k} - x^{n+1}_{i,j-1,k} & x^n_{i,j,k+1} - x^{n+1}_{i,j,k-1} \\ y^n_{i+1,j,k} - y^{n+1}_{i-1,j,k} & y^n_{i,j+1,k} - y^{n+1}_{i,j-1,k} & y^n_{i,j,k+1} - y^{n+1}_{i,j,k-1} \\ z^n_{i+1,j,k} - z^{n+1}_{i-1,j,k} & z^n_{i,j+1,k} - z^{n+1}_{i,j-1,k} & z^n_{i,j,k+1} - z^{n+1}_{i,j,k-1} \end{bmatrix} \Big/ 2 \quad \ldots (4\text{-}17)$$

Where (h_r, h_s, h_t) are the increments of parameters arrayed along the direction of *r*, *s* and *t*. Since the range of the parameter space is unit [0, 1], maximal *i*, *j*, *k* are, respectively, determined by $(1/h_r, 1/h_s, 1/h_t)$.

If $\mathbf{r}^{\mathbf{n}}_{i,j,k}$ represents the unknown coordinates at the point (*i*, *j*, *k*) at iteration level *n*, it follows that the solution to Equation (4-15) can be obtained using the over-relaxation scheme:

$$\mathbf{r}^{\mathbf{n+1}}_{i,j,k} = \mathbf{r}^{\mathbf{n}}_{i,j,k} + \mathbf{R}^{n}_{\mathbf{i,j,k}} \Big/ [3(\alpha^{n}_{11} + \alpha^{n}_{22} + \alpha^{n}_{33})] \quad \ldots (4\text{-}18)$$

The solution of Equation (4-16) renders discrete parameterizations of 3D grids. This method is the general one; it can be applied to either simple or complex solid models bounded by eight facets. The inherent smoothing property of Laplace's equation ensures that the mesh points are smoothly distributed. Usually, the grid points will become more closely spaced near convex boundaries, while the mesh spacing will be sparser near concave boundaries.

Figures 4-6 to 4-9 compare the converging speed of Laplace iteration for the case shown in Figure 4-4. The X-axis represents the iteration number, and the Y-axis the corresponding deviation of each loop. Figure 4-6 depicts the convergent curve of the grids initialized with the COONs interpolation, Figure 4-7 shows the convergence by means of simplified COONs generic formula, and Figure 4-8 compares these two initialization methods. Based on these figures and other cases that have been observed, the following conclusions are obtained:

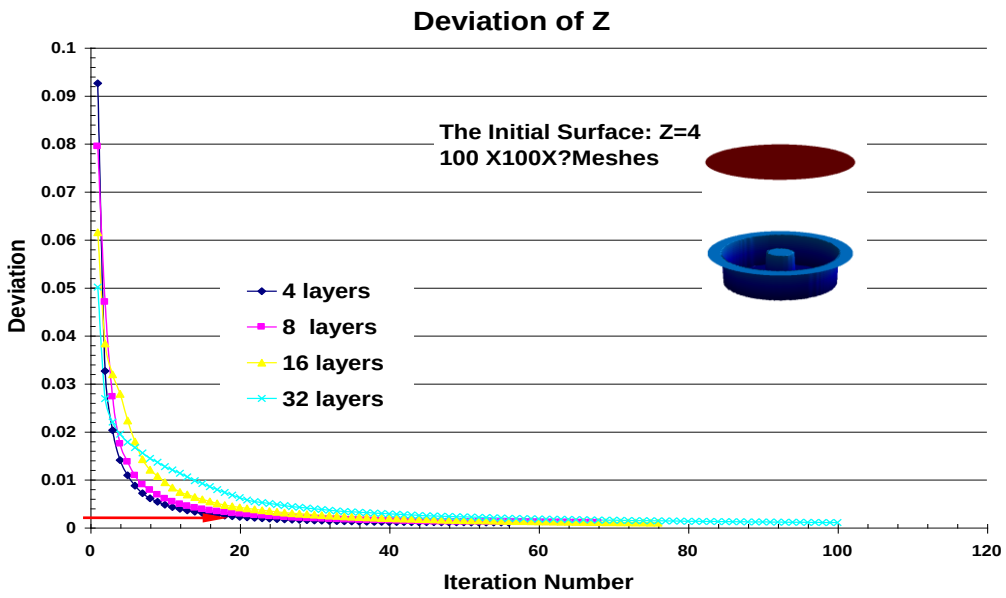


Figure 4-6 Convergent line of Laplace with COONs

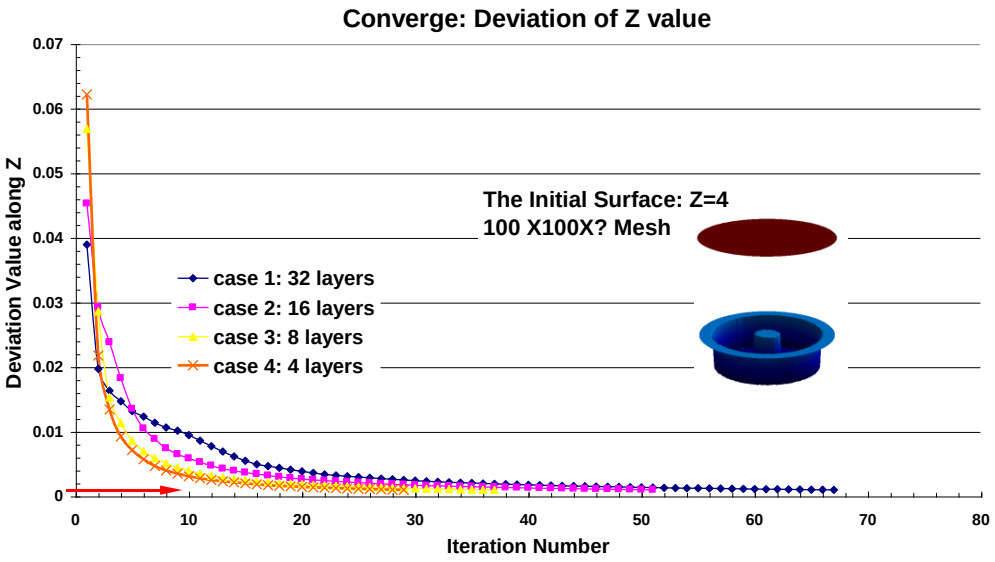


Figure 4-7 Convergent line of Laplace with simplified initialization

i. The convergence of Laplace is independent from the density of the mesh. As a matter of fact,

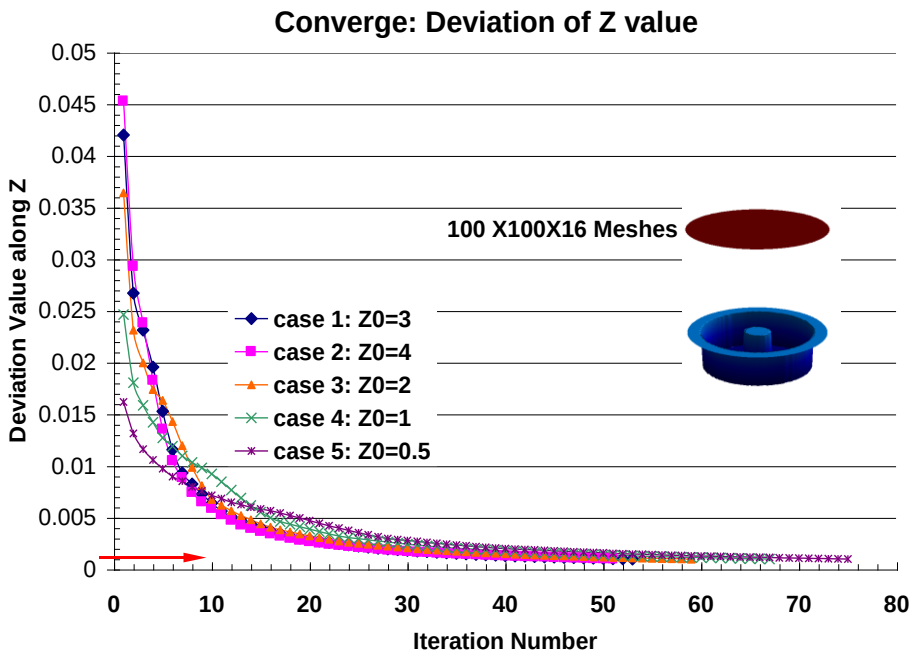


Figure 4-8 Effect of the mesh density on the convergence

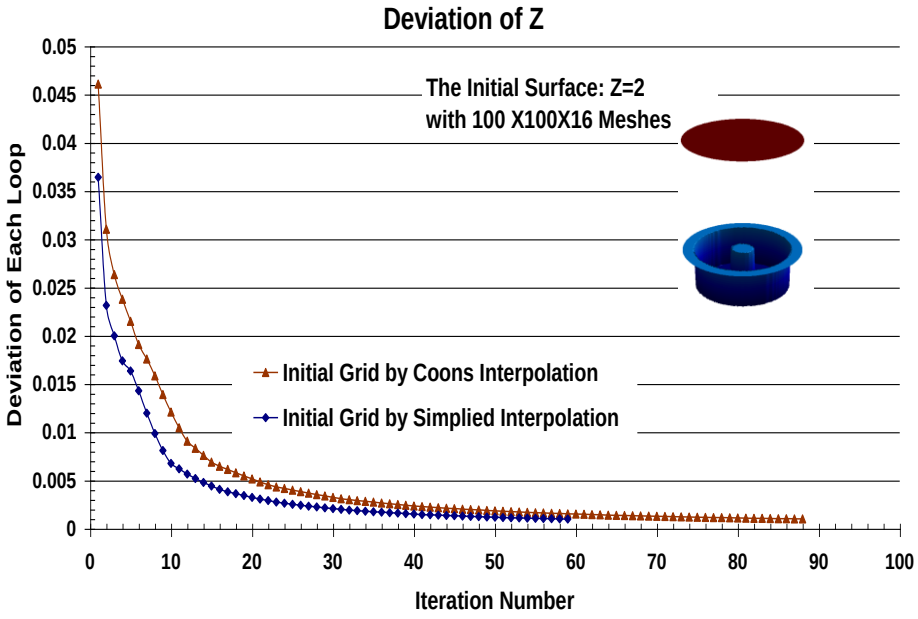


Figure 4-9 Comparison of convergence speed

both Figure 4-7 and Figure 4-8 have verified that the convergent iteration increases with the

number of intermediate surfaces. Also, if the distance between initial surface and desired surface increases, Laplace needs more iteration to reach convergence.

ii. However, the initial grids affect the convergent speed of Laplace. A more reasonable initialization scheme can dramatically speed up the convergent marching.

4.7 Case study

4.7.1 Results of surface morphing

Figure 4-10, 4-11 and 4-12 illustrate three cases of morphing with different geometry characteristics. The surface in Figure 4-10 has two features, a dome and a cylinder. They are connected with first-order continuity: the left picture represents the target surface and initial surface; the middle picture is the expanded view of the intermediate surfaces calculated with PISM; the right picture represents the generated surfaces selected for machining. In Figure 4-11, these two features are disconnected. It can be seen that Laplace PISM can converge and generate smooth intermediate surfaces in both scenarios after a few iterations. Figure 4-12 is another case with a deep cavity. In comparison to the linear interpolation method as shown in Figure 4-5, Laplace PISM provides intermediate surfaces with no self-intersections and excess. Normally, the weakness of Laplace PISM is its convergence speed. With an increase in mesh, it will take a longer time to reach the solution.

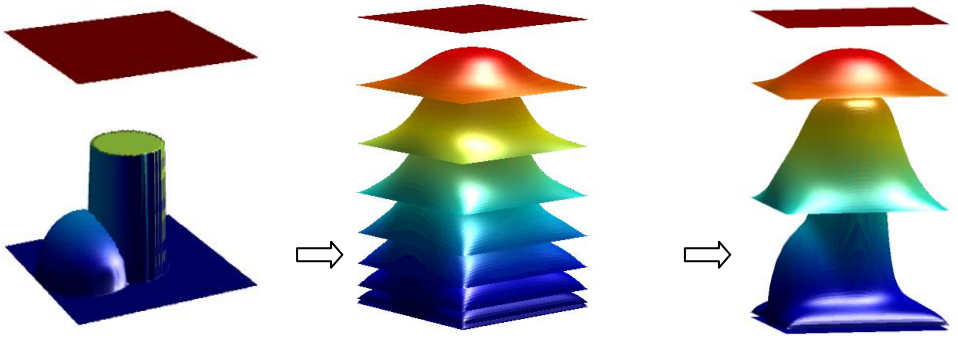


Figure 4-10 Morphing of surfaces with first-order continuity

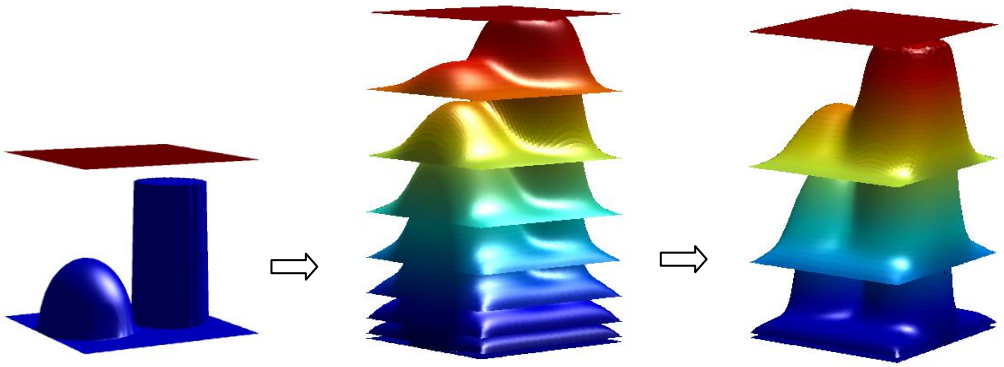


Figure 4-11 morphing of surface with discontinuous feature

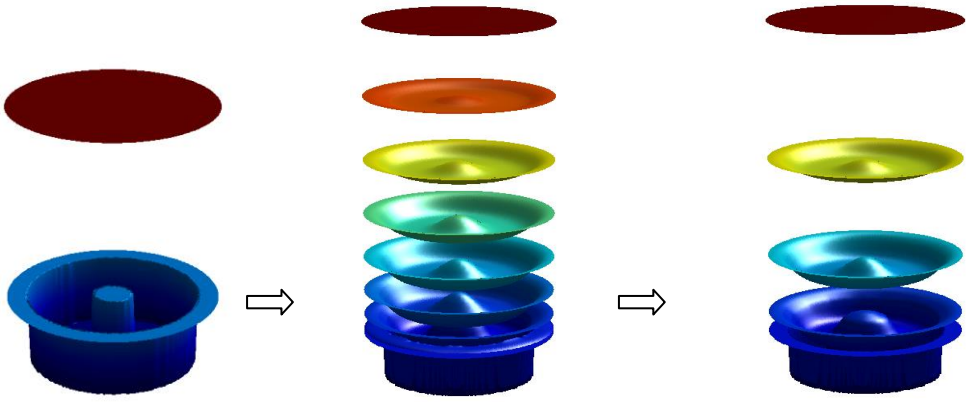
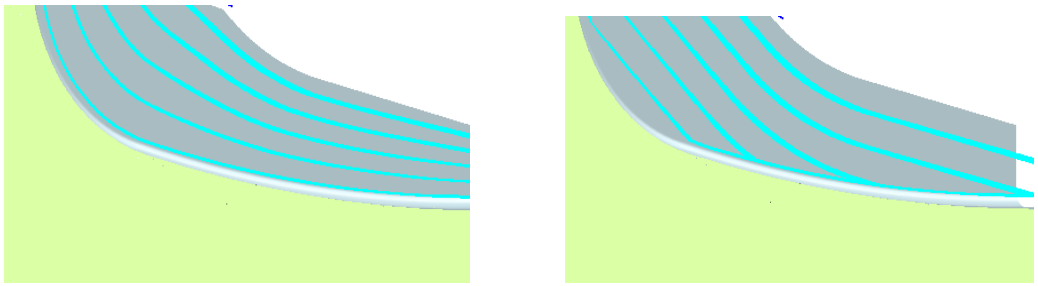


Figure 4-12 Morphing of cavity surface

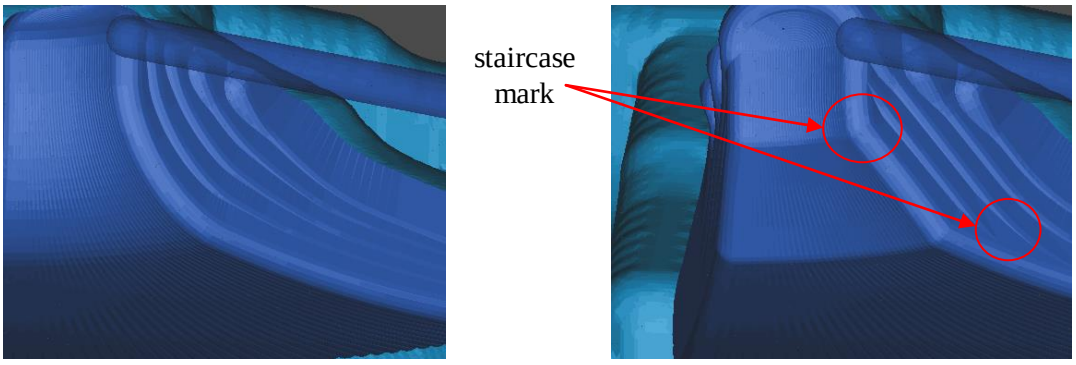
4.7.2 Comparison of tool path

The intermediate surfaces generated by Laplace are actually composed of iso-parametric curves that supply the tool path of intermediate steps for roughing machining. For the case as shown in Figure 2-6 and Figure 2-7, Figure 4-13 illustrates a comparison of the tool path generated from morphing-based multistage machining and that from conventional parallel offset



(a) Tool path of morphing (b) Tool path of parallel offset

Figure 4-13 Comparison of tool path



(a) Morphing based roughing (b) Parallel offset roughing

Figure 4-14 Comparison of cut surface

roughing process. The simulation for both cases is conducted in Siemens NX8 with the same CAD model, same tolerance and in-process stock, and the tool path generated with a ball mill tool of a diameter of 8 mm. There are five levels of cut. Tool path in Figure 4-13(a) is generated from a morphing strategy, and Figure 4-13(b) illustrates the tool path from the parallel cutting. It can be seen that the tool path in the higher level has redundant passes with the lower level ones along the bottom level.

Figure 4-14(a) shows the roughness of the cutter surface with morphing style machining strategy, and Figure 4-15(b) shows the parallel offset machining strategy. Figure 4-14(a) shows a better finish on the final surface than Figure 4-14(b). In Figure 4-14(b), the parallel rough machining leaves a stage case shape of the blade surface, while the morphing strategy does not have the same problem. In order to clean up the extra material around the corner, additional machining steps are needed, called semi-finish and finish machining, which will introduce extra machining time. Based on the simulation result of Siemens NX8, Table 4-2 compares the performance of these two methods. If both feed rates are 250 mmpm, morphing-based roughing is slightly slower than the parallel offset roughing, but since the parallel offset method cannot fully removed the material, additional machining process is required to clean up the stair case marks around the corner of the boundary surfaces, and therefore the total performance of the morphing method should be better than the parallel offset method. Table 4-2 lists some simulation data comparing the efficiency of these two methods. In the morphing strategy, roughing takes a little longer than parallel cutting, but because additional processes are required

to clean up the corners before it is ready for finishing, the total time used by parallel offset roughing might be longer than the morphing roughing.

	Morphing-based strategy	**Parallel offset strategy**
Tool Description	Ball mill tool with 6mm diameter, 20mm/length, 10mm/ flute length	Ball mill tool with 6mm diameter, 20mm/length, 10mm/ flute length
Feed Rate	250 mmpm	250 mmpm
Performance time	1:15:55.6	1:13:28.0

Table 4-2 Comparison of simulation data

5

REVIEW OF CNC MACHINING AND TOOL PATH GENERATION

5.1 Review of CNC machining

Numerically controlled (NC) machines are the machine tools programmed to automate the manufacturing process. NC machines follow a sequence of preprogrammed instructions, and drive the cutting tools to drill, turn, bore, or mill different parts in various shapes.

As the most commonly used form of programmable automation today, CNC machine received their instructions from a microcomputer. NC and CNC machines rank just after CAD in terms of most popular CIM technologies. It has become a popular technology to automate the process from design to manufacturing, increasing productivity, improving quality, meeting customer needs faster, and offering more flexibility. Therefore, most setups of Computer Integrated Manufacturing System are based on the frameworks of CNC machines. Generally, modern CNC machines have two types of applications, as shown in Figure 5-1:

1) One is the duplication of a physical object from measured data [64, 65], normally called prototyping, where the shape of original object is measured and rebuilt virtually via reverse engineering. The geometrical information is partially unavailable due to the discreteness of measured points. A prototyping procedure normally includes three steps: (1) create measure data (2) rebuild the physical object with measuring devices in the form of a geometric model and (3) realizing the geometric model by means of CNC machining or prototyping machining(3D printer). The second phase creates a bottleneck in the automation of duplication procedures. It falls into an open area involving many disciplines [64, 65 and 76].

2) Another category is production of CAD models, where the geometrical information of the

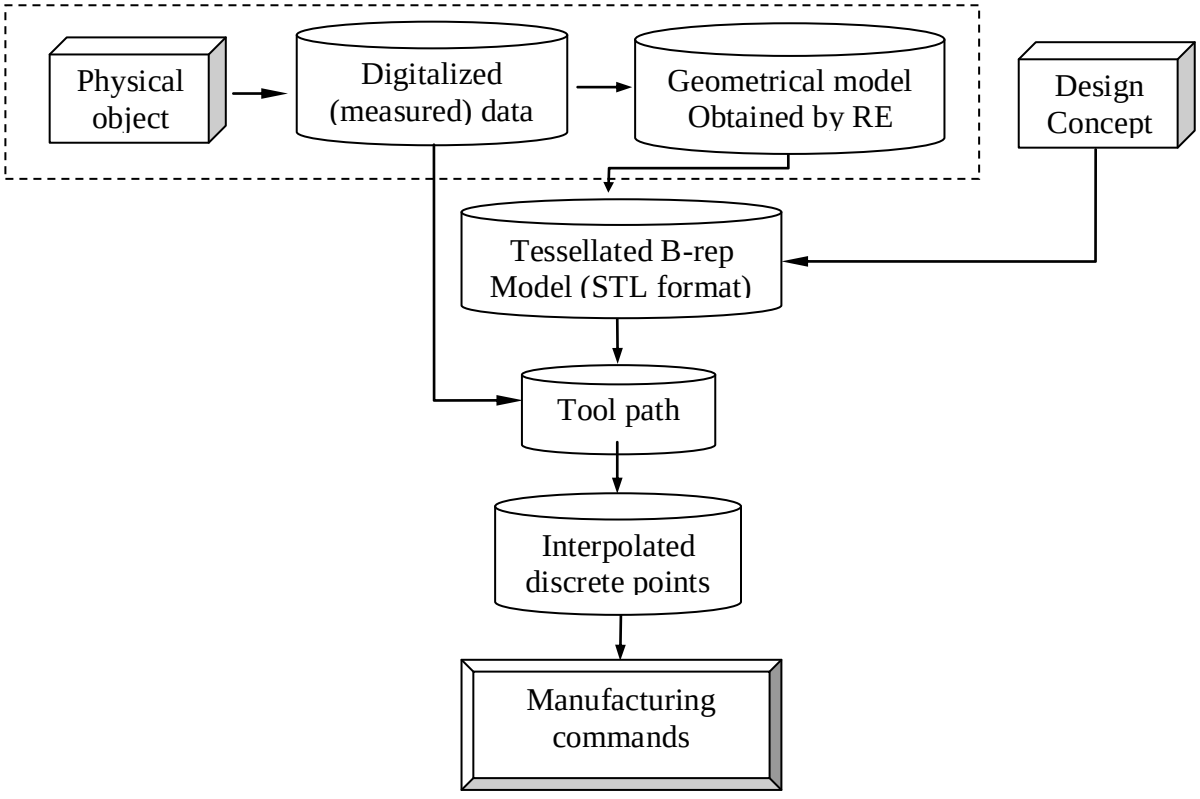


Figure 5-1 Process of integrated manufacturing

design is interpolated into the controlling commands of CNC machine. Differing from the first case, the graphical data has to be identified as a sequence of features [77, 61]. Each of them will be processed individually with different setup instruction, tools and cutting methods.

The CNC cutter driven by a predefined program traces and changes the shape of the rough stock into the designed profile by removing the material. According to Daniel C.H. Yang [61], Lartigue [71] and others, a CAM system for CNC machining planning usually has three modules: tool path generation, interpolation and command generation, as shown in Figure 5-1. After a CAD model is built, CAPP block identifies the features used to compose the geometrical model and divide them into single primitives with lower level configurations. For each of these, a detailed fabrication process plan will be established. This fabrication plan contains not only the setup instructions and other configurations but also a set of commands that control the motion of the machine. In order to generate the controlling commands, each feature that is to be cut is interpolated as a cluster of curves, called tool paths, as shown in Figure 1-3. Interpolation starts with the CAD model by approximating these tool paths by a set of line or circular segments. The maximum chordal deviation is calculated for each segment. In cases the deviation is greater than the prescribed tolerance; the curve is subdivided until the chordal deviation is less than the tolerance. Then, each approximating segment is further processed by an interpolator, which converts the path into a sequence of discrete points incremented by $V\Delta T$, where V is the specified cutting feed rate and ΔT is a fixed-time interval.

For the cases of duplicating physical objects, reverse engineering (RE) is adopted to obtain the geometrical model based on the information provided by the measured data, as shown in Figure 1-3. However, some papers [64] present schemes through which 3-axis NC tool paths (for

roughing and finishing) can be directly generated from measured data. Figure 5-2 illustrates the terminologies of cutting tools, cut center (CL), scallop surface, and their relations to the desired surface.

5.2 Multistage rough machining

In most cases, the material cannot be removed by only one pass since the depth of the cut volume exceeds the permissible depth of cut. Instead, the machining procedure begins with stock material going into three phases: rough cutting, semi-finish and finish, as shown in Figure 4-1. Rough cutting brings the material to near shape, semi-finish performed on the near shape brings it to within the dimension tolerance, and the finishing machining is used to polish or to keep the surface within the finishing tolerance. The tool path for each of these phases has to be generated with different considerations. For example, the tool path for rough cutting should be designed to be productive; for the last two steps, there is more compromise between the speed and surface quality.

5.3 Current methods of tool path planning

5.3.1 Finish Machining

Methods of tool path generation for finish machining of three-axis CNC can be divided into three categories [79]:

i. The Apt-based real-time method: Explicit tool drive surfaces are introduced in this method. The tool paths are defined along the intersection curves between the designed surfaces and driving surfaces. At each step, numerical iteration searches are made to locate the cutter

position within a specified tolerance limit. The disadvantage of this method is that the iterative computation is time consumptive and there is no certainty that the iteration converges, especially for irregular geometry.

ii. The Euclidean machining method, in which the tool path is generated within the frame of work-piece space in Euclidean coordinate system. According to how the individual tool paths are generated, Cartesian methods are classified into guide-planes [79], Z-map [64], pocket approaches [80, 81], iso-scallop [82, 83] or iso-phote [84]. The Cartesian machining generates the tool paths from surface represented either by parametric functions or from measured data points.

iii. The parametric machining method. The tool paths are generated on the parametric space. Afterwards, the results are mapped onto the Cartesian space. This method is widely used in the commercial CAD/CAM programs. Typical parametric machining methods include offset, iso-parametric method [70], and iso-distance method [59].

5.3.2 Rough machining

Most of the recent studies on tool path generation concentrate on the phase of finished cutting. However, it is usually the least amount of material that is removed during the cutting procedure. When parts such as molds and dies are machined, about 70% of the raw material is removed by rough cutting [65, 85]. Therefore, a minimization of the rough machining time can effectively improve the productivity. As shown in Figure 5-2, the rough machining of a 3D-sculptured surface is usually converted to layer-by-layer 2D pocket cutting. When the desired sculptured surface is intersected with a set of parallel guiding planes, the area on these planes is

enclosed by intersection curves [45]. The cutter passes over the enclosed area and removes the volume, leaving small amount of material with staircase shapes, as shown in Figure 5-2. The most commonly used cutters for rough cutting are flat end tool or chamfer end mill.

As shown in Figure 5-2, according to the main cutting direction (MCD) of tool path segments, the tool path patterns for rough cutting can be categorized as the zigzag tool path and the contour-offset tool path. Figure 5-2(a) shows the zigzag tool path that is normally generated from a set of parallel lines lying within the area to be machined. These line segments intersect the boundary elements or contours of inside islands, and are divided into several line segments. The final tool path is generated by merging all these segments by following certain specific rules. Figure 5-2(b) shows another pattern of tool path, called contour-offset pattern, where the boundary curve seeds the resulting tool path by iteratively offsetting itself. The main disadvantage of these two methods is that they are restricted to pockets the boundary elements of

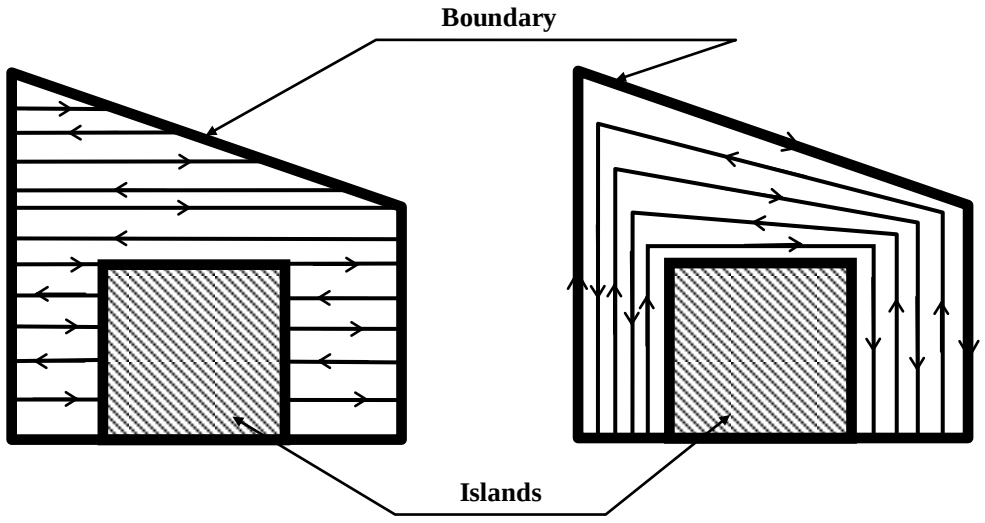


Figure 5-2 (a) zigzag tool path and (b) contour-offset tool path

which are in specific shapes, such as line or quadric curves. Extremely time-consuming calculation is needed if the pocket is composed of complex topological pattern.

In recent years, some studies have attempted to obtain more efficient and robust strategies of tool path generation for pocket cutting. Shaobin Tao et al. [63] and Huang et al. [66] propose a method based on pixel maps, by way of which the search of intersection points between the sweep line and the boundary is more efficient than before, and the boundary is therefore not restricted to specifics shapes. Meanwhile this method resolves the problems of boundary line elements being parallel to the sweep line in conventional approaches. However, this research only considers the 2D pocket, and the author did not mention how to convert the surface information into pixel maps. Y.N. Hu and Y.H. Chen [85, 86] present a robot-based approach that generates the tool path for rough machining using a grid height. This method takes the rough stock into account, but it is only limited to some special boundary conditions and fails to consider how to avoid the collision between bounding features and the cutting tool. In addition, the implementation of robot machining has some deficiencies, including lacks of stiffness and also the non-orthogonal movements during prototype machining. Y.S Huang **[**88**]** et al. developed an NC algorithm based on the grid height model that is obtained by Boolean operation subtracting the solid model from the model of rough material. With ray-casting method, this grid height model is represented with a spatial array of z heights. It will be changed automatically during the machining process and utilized as an image for further roughing and verification. Since the height changes on each grid of the spatial array need to be recorded as an image for every stage of the simulation process, the efficiency of this method depends extensively on the selection of grid size and the performance of the processor.

Other studies take into consideration the features and the topology of multi-features. Gan Ping [87] presented a method to assemble complicated, multi-featured surfaces into one using blending formulism, which has been verified to be efficient and robust. Balasubramaniam Mahadevan et al. [88] described an algorithm to generate NC paths directly from the shape of the part using geometric volume filling. One distinguishing feature of their research is that the slice planes are variably placed with special consideration of the important features, such as horizontal and vertical faces. Instead of using an infinitely long cylinder as in other systems, the cutting tool in their works is modeled as the tool assembly of shank, holder and cutting portion.

5.4 Major machining deficiencies

The cutting problems incurred in a machining process are normally the resulted of one or more of three primary reasons: gouge, collision and inaccessibility. As shown in Figure 5-3(a), gouging usually occurs on the concave surface when the local radius of curvature is smaller than the cutter radius. As shown in Figure 5-3(b), the collision problem happens when the non-cutting portion of the tool, such as tool neck, shank or holder, collides with stock surface or other bounding features. The examples shown in Figure 5-2(c) are also examples of collision problems, but they are practically called an "inaccessible area." Inaccessibility is also caused by the tool enveloping a surface that cannot be exactly matched with the target surface. When the tool is removing the material from the stock by following the tool path, technically its envelope surface should be exactly matched with the target surface. In practice, however, it is impossible to move the tool continuously in such a way that its envelope can cover the whole target surface without any deviation.

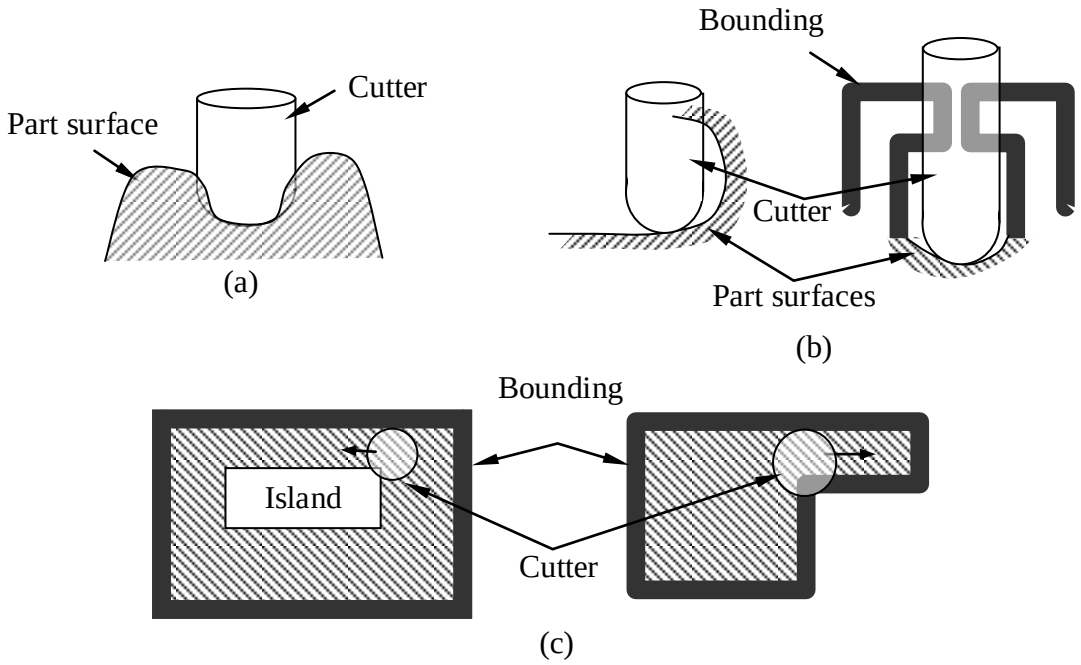


Figure 5-3 Major machining problems

Most of the commercial CAD/CAM systems will check the accessibility of the input model before launching the subroutine of tool path generation. This is called tool axis tiling, and is used to adjust the machining axis to the orientation of the part model and select the manufacturing direction that allows access all the required part features. After checking manufacturability, an acceptable cutting orientation will be defined to avoid the collision area as shown in Figure 5-2(c). An alternative solution to avoid collision or inaccessible areas is to implement a collision detection process after the tool path is generated, so that the regions of the tool path will be trimmed away if it will introduce any collision.

In 3-axis machining, the gouge and collision problems can also be resolved by selecting the correct tool size. In rough cutting, the issue of top-priority is to maximize the speed of material

removal, because the roughing process normally consumes the majority of the machining time. In most cases, the surface of the stock will receive heat treatment after the roughing process is completed, and it is therefore unnecessary to maintain a high precision on the surface. While a cutter with a larger radius removes more material in the unit time, the compromise is that a big radius introduces a higher possibility of overcut or undercut, especially in the complex areas as shown in Figure 5-3(a), (b). Therefore, the optimal tool selection is that which maximizes the tool size within the constraints to prevent collision and gouging. Most of the existing methods for tool selection are simulation-based and sequence the tools in a trial-error-correction process [66]. Bala and Chang [89] reported a method by which every tool in the tool library is projected along the spindle direction to form a set of circles. The largest tool the projected circles of which do not intersect with island or work-piece contours is then selected. On the same basis Lin [45] address a scheme that uses multiple tools for each cutting slice. The main consideration of this research is to increase the accessibility of each cutting slice, but the problem is that the NC machine has to reload tools frequently, while is time consuming.

5.5 Major challenges of tool path generation

No matter what method is used, the implementation of pocket cutting for rough machining has implicit deficiencies. None of these pocket-cutting–based attempts are directed to a generic purpose, and most of them are only valid for some specific applications. The scheme of adjusting the material stock slice by slice is inconsistent with the transformation of objects in the real world and will inevitably introduce many problems:

i. For different types of boundary surface and geometry features, there need to be corresponding algorithms to search the intersection curves.

ii. There is a lack of concerns for the geometric generality of rough stock, and time consuming computational work is required for extracting features in the to-be-machined area of each layer.

iii. In the industrial world there are two types of raw materials. The first type arises when the raw material already has a shape close to the final product, fabricated in advance by a preceding operation. The second type is that in which this differs from the finish-machined shape, usually encountered when parts are machined from rectangular blocks or cylinders. The existing systems produce the manufacturing plans for these two cases with different strategies: in the first case, cutter paths for rough cutting are generated from the offset of the final product; in the second case, the raw material is removed in layers by consecutive passes of the cutter.

iv. Current tool path generation schemes for 2D pockets in rough machining are boundary un-conformed and restricted to some specific boundary elements, such as line, arc etc. The generated tool path must be trimmed on the boundary entities, and reconnected with a step-over movement or non-cutting movement.

v. Different cutting strategies have to be implemented for finish cutting and rough cutting individually. Since the roughing cutting is a sequence of planner cutting, while finish cutting is cutting directly on the final geometry. Therefore the strategy of machining for these two cutting process are normally different.

6

PARAMETERIZATION OF TESSELLATED SURFACE AND TOOL PATH GENERATION FOR FINISH MACHINING

6.1 A new iso-parametric methodology for finish machining

Advances in solid modeling systems have, on the one hand, improved the feasibility of the CAD/CAM integration, the complexity of existing CAD models has, on the other hand, introduced new challenges for the automatic generation of NC tool path. The diversity of mathematic representation for CAD model makes it very hard to process with a unified methodology.

Meshed model is one of the most popular formats in contemporary solid modeling technology, CAD data transformation, prototyping and CNC machining. Due to its flexibility

and generality, it is widely used for rapid prototyping and Computer Aided Manufacturing. In the CAM-oriented model, each machining feature is identified and represented with the format of tessellated meshes. However, since the surface is defined with a set of polygonal vertices, the complexities of the meshed geometry and their topology make the generation of a tool path quite difficult. In practice, various methods have been attempted to obtain the tool path by interpolating the discrete faces with certain forms of parametric functions [90, 91]. As the indirect approaches, these methods either lack accuracy or are time consuming. Also because the local metric is easily lost due to the intrinsic smoothness of many existing interpolation functions, they are not robust and accurate especially when the local property is not smooth.

Most operations of the current CAM systems are targeted on certain types of primitive geometry features, with which the geometry is represented in the forms of interpolation functions or parametric functions. A few examples include drilling operations, plan pocketing, and some COONs-based milling operations. Because these features are represented in mathematical formulations, different algorithms are applied on the basis of selected features and their boundary information. For the composite features that include complex geometries, additional routines are required to merge the tool paths generated from each single subset of surfaces. Meanwhile, intensive user interactions have to be involved in order to select the features to be machined. In the scenario of integration manufacturing, there is not yet a unified method whereby the tool path can be generated efficiently for any type of freeform surface. Because of this, the flexibilities of current CAD systems have been restrained by the manufacturability of CAM systems.

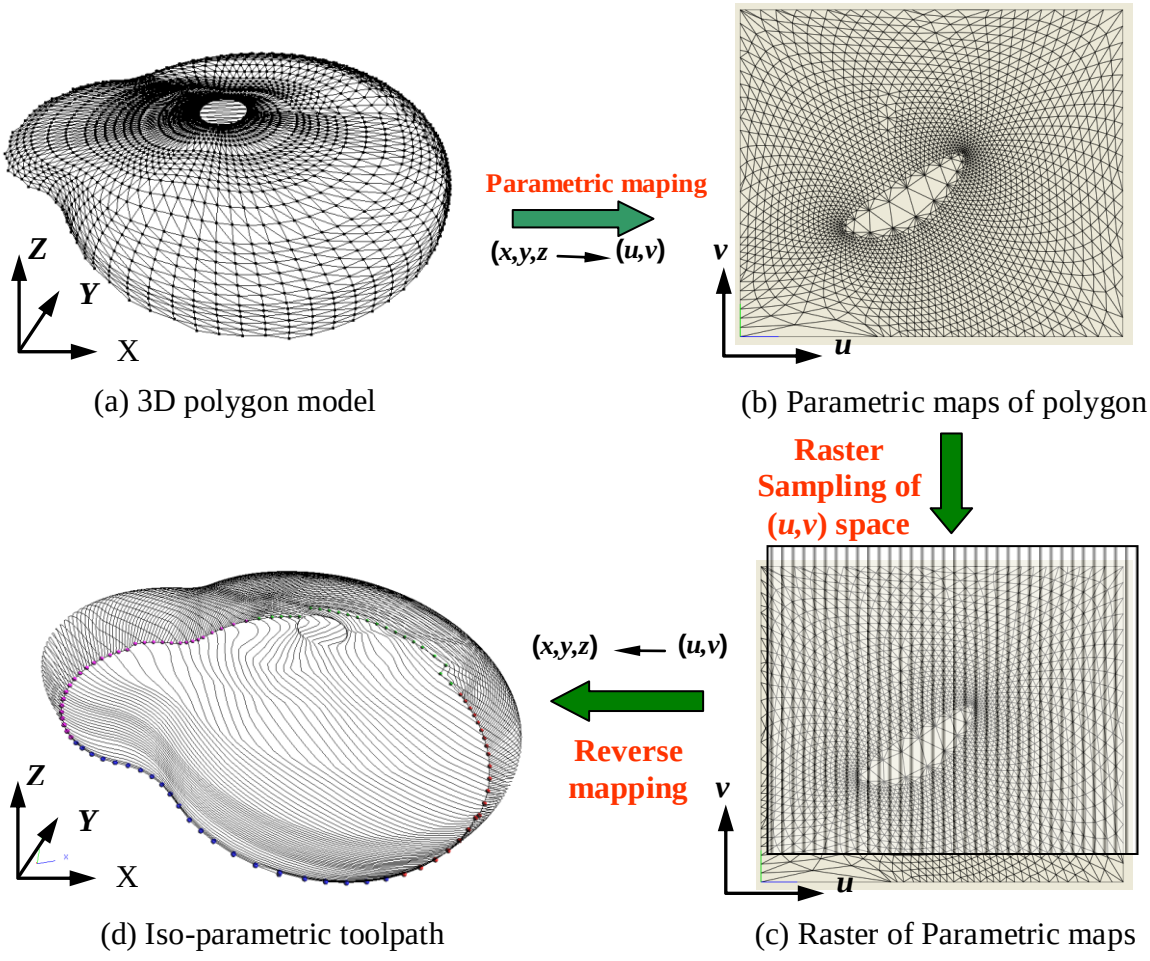
The second issue comes from the boundary curves. Although the commercial CAD/CAM software provides various options of patterns to fit the tool path with boundary conditions, the

boundary geometries for 3D freeform surface with trimmed boundaries normally come from the intersect of the guiding plan and the target surface. The existing patterning algorithms lack intrinsic conformance to irregular boundaries.

If data is measured directly from real objects, the measured surface is represented with polygonal facets or point clouds. In such cases there is no information of advanced features to be selected for tool path generation. A commonly used method is to reconstruct the features by identifying the local geometrical information, such as the continuity of normal direction. This additional step of reverse engineering will introduce errors, and is not a good strategy for finish machining. Some CAM systems now machine on a tessellated solid or surface model. Compared to machining directly on a solid or surface, machining on a tessellated model is a good alternative solution to minimize machining errors and improve machining performance.

The primary advantage of the tessellation modeling and machining technique is that the mathematical description of the geometry is exchangeable and is easy to import from different software platform. Meanwhile it more straightforward than working with parametric surfaces, it is also accepted for the prototyping process. For a complex surface represented as a group of single entities, it can avoid missing data, or ambiguity. However for a tessellated mesh model, most of the existing CAM systems generate the tool path by using iso-plane curves. All these method are expensive in terms of the computational time.

In current research, a new iso-parametric method is introduced to generate the tool path directly based on the tessellated surface. In comparison to the traditional iso-parametric methods, the new method vias the polygons directly and abandon the explicit parametric functions that are used to describe the machining features in the traditional methods. Figure 6-1 illustrates the major steps of this strategy, where the tool path is generated through the steps of mapping and reverse mapping. Assuming a polygon model that results from the tessellated CAD model or



(a) 3D polygon model

(b) Parametric maps of polygon

(d) Iso-parametric toolpath

(c) Raster of Parametric maps

Figure 6-1 Parameterization of polygon surface and iso-parametric

measured surface is represented with $Q(x, y, z)$ in Euclidean space, as shown in Figure 6-1(a), it has been proven that a meshed surface with single boundaries is hormeomorphic to a disk in parametric space R^2. It can be flattened to a disk in R^2 with the certain constraints on the boundaries. If $P(u, v)$ represents the embedding of S in R^2, $P(u, v)$ owns the same topology as $Q(x, y, z)$, as shown in Fig. 6-1(b), namely all the vertices of P have the same connectivity as the vertices of Q. By reverse mapping the parametric square with iso-u or iso-v curves, a 3D surface can be represented as a group of parametric grids, as shown in Fig 6-1(d).

6.2 The parameterization of freeform surfaces

Definition 1. A meshed surface is defined as point set ***P*** with manifold connectivity that is locally homeomorphic to a disk.

Definition 2. Iso-curves of surface S is defined as a set of curves i.e. C that is conformal on the boundaries of the surface, and for any point Pc on the curves, one can always find the point Ps such that $\|P_c - P_s\| \le \phi$ and ϕ is any small positive.

Additional Scenarios The primary target of this research is to map the set of topological points directly into a group of iso-curves that covers the entire domain and conforms to the boundaries.

According to Gerson Elber [92], the iso-curves ***C*** of freeform surface ***S*** can be defined as a valid coverage of the surface with the required deviation of less than δ. specifically for a meshed surface, the commonly used iso-curves are iso-plane curve and iso-parametric curve. Iso-plane curve is formed by the intersection curve between the surface and a set of parallel planes. Iso-parametric curve is generated by representing a 2D-manifold surface in Euclidean space with

two parameters $P(\mathrm{u}, \mathrm{v}) -> (x(u,v), y(u,v), z(u,v)) \in R$. If one keeps one parameter member of (u, v) as a constant, the dimensionality of Q(u, v) is reduced to 1, namely $Q(u, v_j)$ or $Q(u_i, v)$. Changing this constant value will produce a family of curves with single parameters, which are called iso-parametric curves.

The local tangent space T_pQ can be equally described with a 2D Riemannian metric by using the form of inner product, i.e. $\langle \cdot, \cdot \rangle$. One example of a Riemannian metric is the first fundamental form that is commonly used to describe the local tangent space in Euclidean space. It can be written in the notation of a metric tensor.

$$I = \langle u_p, v_p \rangle = \begin{pmatrix} \left\langle \frac{\partial}{\partial u}, \frac{\partial}{\partial u} \right\rangle & \left\langle \frac{\partial}{\partial u}, \frac{\partial}{\partial v} \right\rangle \\ \left\langle \frac{\partial}{\partial u}, \frac{\partial}{\partial v} \right\rangle & \left\langle \frac{\partial}{\partial v}, \frac{\partial}{\partial v} \right\rangle \end{pmatrix} \quad \ldots(6\text{-}1)$$

Where u_p and v_p are two points in the tangent space at point p. Similarly, the second fundamental form can be written in a form of metric tensor

$$II = \begin{pmatrix} \left\langle N, \frac{\partial^2}{\partial u^2} \right\rangle & \left\langle N, \frac{\partial^2}{\partial v \partial u} \right\rangle \\ \left\langle N, \frac{\partial^2}{\partial u \partial v} \right\rangle & \left\langle N, \frac{\partial^2}{\partial v^2} \right\rangle \end{pmatrix} \quad \ldots(6\text{-}2)$$

The third fundamental form is given by:

$$III = \left\langle Sh(u_p), Sh(v_p) \right\rangle = \begin{pmatrix} \langle N_u, N_u \rangle & \langle N_u, N_v \rangle \\ \langle N_u, N_v \rangle & \langle N_v, N_v \rangle \end{pmatrix} \quad \ldots(6\text{-}3)$$

These three fundamental forms are commonly used to describe the local geometrical properties of freeform surface. For the purpose of surface parameterization, the distortion metric is the major metric to be preserved. In terms of first fundamental form, a surface mapping is undistorted (isometric) if and only if the first fundamental form ***I*** is identity, namely

$$I = \mathbf{I}^* \quad \ldots(6\text{-}4)$$

Unfortunately, isometric mapping is only possible in cases of a developable surface, such as a flat surface or cylinder etc. In most cases, a best maps $f : P \text{-} > Q$ is the one that minimizes the distortion. In terms of first fundamental form, there are two distortion metrics that are normally used for the mapping measurement: one is angle preserving and the other is area preserving [93]. The first, angle-preserving method is conformal maps that preserve the tangent angles. For this map, the first fundamental form is an orthogonal matrix.

$$I = \eta(u, v)(\mathbf{I}^*) \quad \ldots (6\text{-}5)$$

The second, area preserving method is called equiareal, and preserves the area ratio of each triangle. Its determinant of ***I*** is equal to a constant.

$$\det(I) = 1 \quad \ldots (6\text{-}6)$$

In most cases the area and angle cannot be preserved at the same time. Many maps in practice are the tradeoff between conformal maps and equiareal maps. Depending on a specific application, many studies [94, 95] define different sorts of energy functions. In terms of the first fundamental form, an optimal map is that which minimizes this energy, and it is called a harmonic map when Dirichlet Energy is being minimized.

$$E_D(f) = \frac{1}{2}\int_S \|\nabla f\|^2 ds \qquad \ldots(6\text{-}7)$$

Where ∇f is the gradient vector field. It has been proved that to find the minimal Dirichlet Energy is actually a weak form of Laplace equation. Many existing parameterization schemas are based on Dirichlet Energy [95-98]. The singular values of the first fundamental form are

$$\lambda_{\min} = \frac{(a+c) - \sqrt{(c-a)^2 + 4b^2}}{2} \text{ and } \lambda_{\max} = \frac{(a+c) + \sqrt{(c-a)^2 + 4b^2}}{2} \qquad \ldots(6\text{-}8)$$

Where $a = \left\langle \frac{\partial}{\partial u}, \frac{\partial}{\partial u} \right\rangle$, $b = \left\langle \frac{\partial}{\partial u}, \frac{\partial}{\partial v} \right\rangle$ and $c = \left\langle \frac{\partial}{\partial v}, \frac{\partial}{\partial v} \right\rangle$

It can be proved that

$\boldsymbol{f}$ is isometric $\Leftrightarrow \lambda_{\min} = \lambda_{\max} = 1$

$\boldsymbol{f}$ is conformal $\Leftrightarrow \lambda_{\min} = \lambda_{\max}$

$\boldsymbol{f}$ is equiareal $\Leftrightarrow \lambda_{\min} \cdot \lambda_{\max} = 1$

Therefore, a map is conformal iff $\sqrt{(c-a)^2 + 4b^2} = 0$.

6.3 Parameterization of facet surface with arbitrary topology

6.3.1 Mapping of faceted surface

In case of facet surface, its mapping in parametric space is so called embedding. To parameterize the facet surface, the first issue is to preserve the connectivity of the vertices on S. Tutte [96] was the first to provide the most basic way to compute embedding by using barycentric coordinates with a weight matrix. The boundary vertices are mapped to a convex position and each interior vertex is simply a centroid of its adjacent vertexes. Computing the embedding involves solving a set of linear equations derived from the connectivity graph matrix. Floater [97] and Levy et al. [98] further proved that not only barycentric form but any convex linear combination of adjacent nodes can be used to generate the embedding of 3D meshed surface in a parametric plane.

As shown in Figure 6-2(a), if the facet surface is represented as three sets, as {$\boldsymbol{Q}$, $\boldsymbol{E}$, $\boldsymbol{F}$}, where {$\boldsymbol{Q}$} is the set point { $\boldsymbol{q_i} \in \boldsymbol{R^3}$ }, {$\boldsymbol{E}$} is the edge set defined by any two neighboring vertices of Q, it can be defined as a set of its two ending vertices {q_i, q_j}. {$\boldsymbol{F}$} is the face set that is constructed by the elements of {E}. For any point in Q, namely $\boldsymbol{q_i} \in \boldsymbol{R^3}$, it is a map from $\boldsymbol{p_i} \in \boldsymbol{R^2}$.

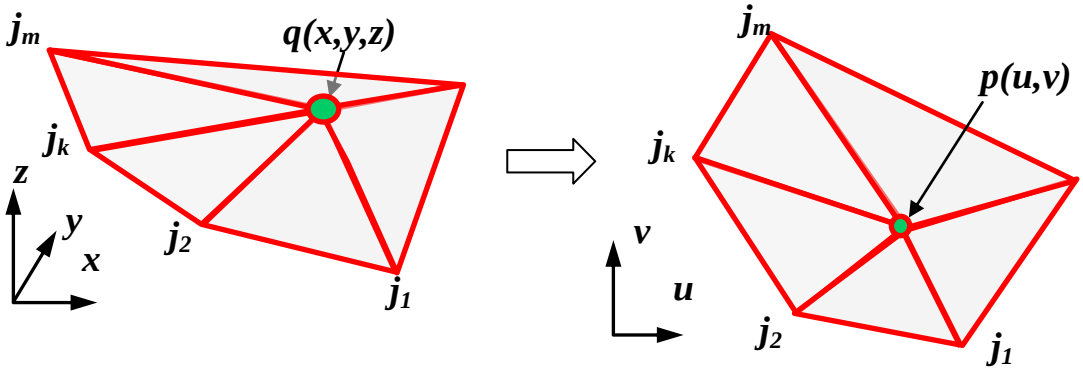
Embedding points of Q in $\boldsymbol{R^2}$ have a linear relationship such that

$$[\lambda_{i,j}][P]=[P], \qquad \ldots(6\text{-}9)$$

where P is the set of p_i. and $[\lambda_{i,j}]$ is a coefficient matrix that satisfies

$\lambda_{i,j}=0$ if $\{q_i,q_j\}\notin E$ and $\lambda_{i,j}>0$ if $\{q_i,q_j\}\in E$

To form a one-to-one map between P and Q, the coefficients of its neighboring vertices must satisfy $\sum_{j=1}^{n}\lambda_{i,j}=1$. As long as the coefficient matrix satisfies all the above conditions, embedding of $\{Q\}$ on $\boldsymbol{R}^2$ can be found by solving the linear Equation (6-9).



(a) Centroid in 3D space (b) Centroid in parametric space

Figure 6-2 Centroid preserve in parametric space

With $[\lambda_{i,j}]$ changes, embedding is not unique. The simplest method [96] is to divide Q into two subsets, i.e. the set of the internal vertices and the set of vertices on the boundary, called Q_I and Q_E respectively. The embedding of boundary points is predefined, and therefore its corresponding coefficients on all other vertices are 0, except the one corresponding to itself, namely

$$\lambda_{i,j} = \begin{cases} 0 & for \quad i \neq j \\ 1 & for \quad i = j \end{cases} \quad \ldots (6\text{-}10)$$

where $p_i \in Q_E$

When $p_i \in Q_I$, the coefficients on its neighboring vertices can be set equal, namely:

$$\lambda_{i,j} = \begin{cases} 1/m & for \quad \{pi, pj\} \in E \\ 0 & for \quad i = j \end{cases} \quad \ldots (6\text{-}11)$$

Where m is equal to the total number of neighboring vertices to p_i

In the theory of Dirichlet Energy, the best embedding is usually defined as the one that can preserve the geometric properties and minimize the distortion of the original surface, as Equation (6-7) shows. Hence, the selection of $[\lambda_{i,j}]$ is intended primarily to minimize distortion. But with different applications, the selection of $[\lambda_{i,j}]$ has to meet various criteria. Specifically for tool path generation of finish machining, what we need control the scallop height of the cut surface to be consistent throughout in order that the surface can have better smoothness. That is, the

interval of the tool path that must be fairly distributed and cover the entire cut area. This is normally measured by the distribution of normal curvature. Particularly in facet model, one good way to preserve the normal vector is to determine the $[\lambda_{i,j}]$ by the area information of its surrounding facets. Particularly for a triangle model, the area of each facet is the cross product of its two edges. Therefore $[\lambda_{i,j}]$ of vertex $q_{i,}$ can be selected as weighted by the reverse factor of the cross product of the edges that are connected to $q_{i,}$, namely,

$$\lambda_{i,j} = \begin{cases} \dfrac{1}{\left\|(q_j - q_i)\times(q_{j-1} - q_i)\right\|} \Big/ \displaystyle\sum_{k=1}^{k=m} \frac{1}{\left\|(q_k - q_i)\times(q_{k-1} - q_i)\right\|} & for \quad \{pi, pj\} \in E \\ 0 & for \quad i = j \end{cases} \qquad \ldots (6\text{-}12)$$

Where q_k is the neighboring vertices of $q_{i,}$ namely $\{q_k, q_j\} \in E$, and m is the total number of vertices of q_i. $q_j \times q_i$ represents the area of the triangle formed by the central vertex and the two neighboring vertices, namely $\{q_i, q_j, q_{j-1}\}$. In the facet surface, it will be proved later that the normal direction is not continuous along the edge, and therefore the normal direction on the edge $\{q_i, q_j\}$ is adjusted by using the average value of the normal direction of its two adjacent triangles, namely

$$\lambda_{i,j}=\begin{cases}\dfrac{\left\|\dfrac{1}{(q_j-q_i)\times(q_{j-1}-q_i)}+\dfrac{1}{(q_j-q_i)\times(q_{j+1}-q_i)}\right\|}{\sum_{k=1}^{k=m}\left\|\dfrac{1}{(q_k-q_i)\times(q_{k-1}-q_i)}+\dfrac{1}{(q_k-q_i)\times(q_{k+1}-q_i)}\right\|} & for \quad \{pi,pj\}\in E \\ 0 & for \quad i=j\end{cases} \qquad \ldots (6\text{-}13)$$

6.3.2 Reverse mapping

After the triangle surface is mapped to an isomorphism embedded in 2D parametric space, embedding of this surface can be reversely mapped for any points in the parametric space. Fig. 3 illustrates how to derive the embedding from the parametric space, namely ***p***, to the Euclidean space, namely ***q***.

By using area coordinates, the map of ***p*** in Euclidean space, namely ***q*** is:

$$q=\frac{\Delta pp_2p_3\cdot q_1+\Delta pp_1p_3\cdot q_2+\Delta pp_1p_2\cdot q_3}{\Delta p_1p_2p_3} \qquad \ldots(6\text{-}14)$$

where Δ represents the area of the triangle.

In addition, since the surface is discretized with small triangles, the first partial derivatives with respect to the parameters ***u*** and ***v*** in each triangle are:

$$\begin{cases}\dfrac{\partial q}{\partial u}=\dfrac{v_3-v_2}{\Delta p_1p_2p_3}q_1+\dfrac{v_1-v_3}{\Delta p_1p_2p_3}q_2+\dfrac{v_2-v_1}{\Delta p_1p_2p_3}q_3 \\ \dfrac{\partial q}{\partial v}=\dfrac{u_2-u_3}{\Delta p_1p_2p_3}q_1+\dfrac{u_3-u_1}{\Delta p_1p_2p_3}q_2+\dfrac{u_1-u_2}{\Delta p_1p_2p_3}q_3\end{cases} \qquad \ldots(6\text{-}15)$$

It can be observed that with Equation (6-13), the first derivative is a constant in any triangle element. But on the boundary of any two triangles, the first derivative is not continuous. It can be adjusted by using the average of first derivative at its two adjacent triangles.

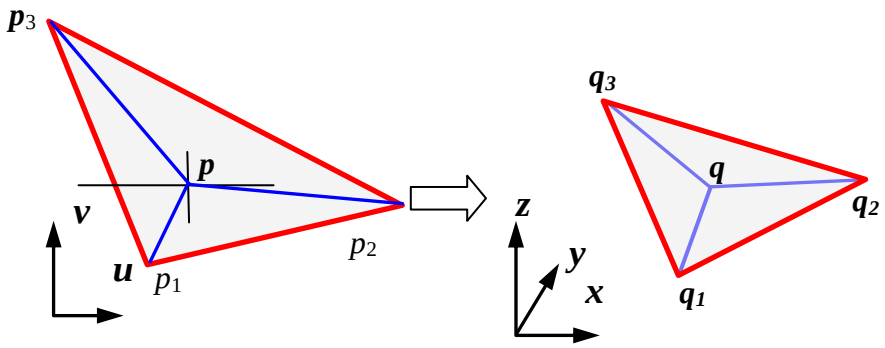


Figure 6-3 Remapping from parametric space to Euclidean space

6.3.3 Tool path generation for parameterized surface

With the embedding formed from the parametric mapping, the original surface can easily be converted into a structured grid. The iso-parametric embedded curves are generated in the Euclidean space from parametric space by reverse mapping. Once these curves are obtained, the remaining task is to determine the step over between two consecutive tool paths. Theoretically the tool path can be obtained directly from each iso-parametric curves, as long as the interval of the iso-parametric curve is sufficiently small to keep the scallop height smaller than the required tolerance. But this solution compromises the overall machining efficiency, because too small a step over will cause many unnecessary passes. The optimal tool path pattern is to maximize the step over size of the tool path while keeping the scallop height within the required tolerance.

The theory to determine the step over of the adjacent tool path was originally developed for primitive surfaces by Chuang and Yang [99]. As shown in Figure 6-4, the path interval depends on the cutter radius, i.e. R; the signed radius of curvature along the intrinsic normal direction for iso-curve Ψ is notated as ρ_Ψ and scallop height allowance as h. The allowable path interval, d_{ij}, at point $\boldsymbol{q}_{ij}$ can be determined according to the surface geometry. Depending on geodesic curvature at $\boldsymbol{q}_{ij}$ being convex, concave, or flat, the path step over along the binormal direction will be:

$$d_{ij} = \frac{\|\rho_\Psi\|\sqrt{(2\rho_\Psi + h)(2\rho_\Psi + h + R)(2R - h)h}}{(\rho_\Psi + h)(\rho_\Psi + R)} \quad \ldots(6\text{-}16)$$

where R is the tool radius and ρ_Ψ represents a signed curvature radius that indicates the local concavity at q_{ij}. The surface is concave if ρ_Ψ is less than 0 and convex if ρ_Ψ is greater than 0. A special case of flat surface for curvature is great enough, then

$$d_{ij} = 2\sqrt{h(2R - h)} \text{ (for flat)} \quad \ldots (6\text{-}17)$$

For most cases, the step over can be calculated by either Equation (6-16) or Equation (6-17). For 2D manifold surface represented in forms of tessellated meshes, the local area of the surface can be approximated to a flat surface, namely, Equation (6-17) can be used to simplify the calculation of the corresponding path interval d_{ij}. As shown in Figure 6-4, assuming the generated boundary conformal tool paths are in iso-*v* direction, denoted as $\boldsymbol{\Psi}_j$, and letting $\boldsymbol{\Phi}_i$ be an iso-$\boldsymbol{u}$ curve, the path step over will be determined along the direction of $\boldsymbol{\Phi}_i$. By taking the first derivative of q with respect to *v and u*, we obtain the surface normal $\boldsymbol{n}_q(u,v)$ at $\boldsymbol{q}$:

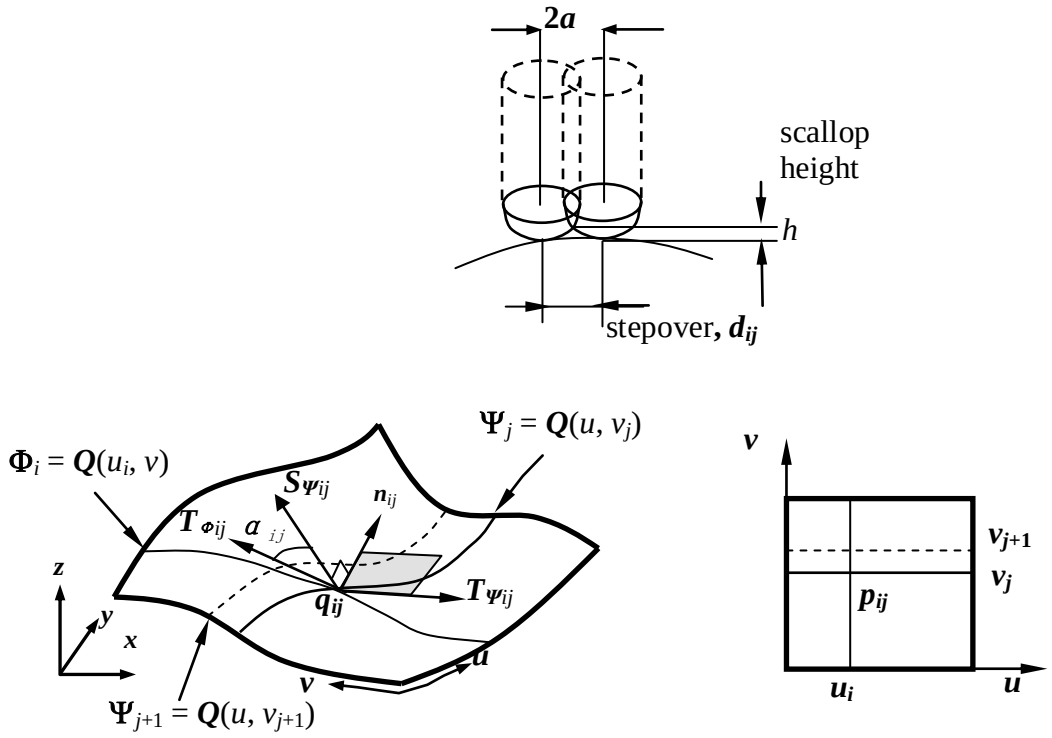


Figure 6-4 Tool path interval and scallop height

$$\mathbf{n_q}(u,v) = T_\Psi(u,v) \times T_\phi(u,v) = \frac{\partial q}{\partial u} \times \frac{\partial q}{\partial v} \quad \text{... (6-18)}$$

Where T_Ψ and T_ϕ are the tangent vector at point q. If set $\boldsymbol{S}_\Psi$ the binormal vector of iso-curve $\boldsymbol{\Psi}_j$ at point $\boldsymbol{q_{ij}}$, that is

$$\mathbf{S}_\Psi(u,v) = \mathbf{n}(u,v) \times T_\Psi(u,v) \quad \text{... (6-19)}$$

Binormal i.e. $\boldsymbol{S}_\Psi(u,v)$ is in most cases not co-linear with the tangent vector $T_\Phi(u,v)$. The angle between $\boldsymbol{S}_\Psi(u,v)$ and $T_\Phi(u,v)$ can be denoted as α, which can be calculated with $\boldsymbol{S}_\Psi(u,v)$ and $T_\Phi(u,v)$, namely

$$\cos(\alpha) = \mathbf{S}_\Psi \cdot T_\Phi(u,v) \quad \text{...(6-20)}$$

Thereafter, the path step over in the parametric space, Δv_{ij}, is the path interval d_{ij} divided by the derivative along the step over direction, namely

$$\Delta v_{ij} = \frac{\left\|\Delta P_{ij}\right\|}{\left\|\frac{\partial P}{\partial v}\right\|} = \frac{d_{ij}}{\cos(\alpha)\left\|\frac{\partial P}{\partial v}\right\|} \quad \text{... (6-21)}$$

where allowable path step over d_{ij} is taken as the component of $\left\|\Delta P_{ij}\right\|$ along the binormal direction at point $\boldsymbol{q}_{\mathrm{ij}}$. The final path interval $\Delta \boldsymbol{v_{ij}}$ for the next iso-parametric curve to $\boldsymbol{\Psi}_j$ is

determined by selecting the minimum path interval along the entire $\boldsymbol{\Psi}_j$. The procedures to generate the tool paths based on the calculated sidesteps in parametric space are illustrated in Figure 6-4. First, let the boundary curve, $\boldsymbol{\Psi}(u,0)$, be the first tool path with parameter $v = 0$. With Equation (6-18), the interval of every point on $\boldsymbol{\Psi}(u,0)$ can be predicted. This process of tool path generation can be repeatedly applied to cover the entire surface.

6.4 Case Study

Figure 6-5 shows a parametric representation of a pocket feature. With the parametric implicit modeler, the cut surface can be converted to a parametric surface. With good tolerance, the tool path can be fitted to the real CAD surface very well. Figure 6-7 shows the tool path that is generated from iso-parametric surface, the step over of the tool path is not a constant. Based on Equation 6-20, the step over of the tool path is selected based on the input scallop height and the cutter radius. CL data is obtained by pulling the tool along the tool axis to avoid any gouging against the stock. For the CL shown in Figure 6-7(e), the tool axis is constantly pointing upward along the z axis.

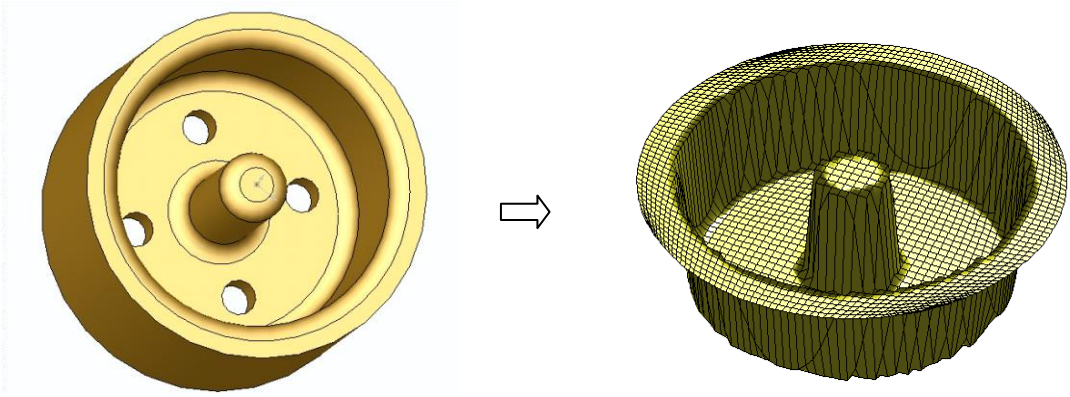


Figure 6-5 Parametric representation of CAD model

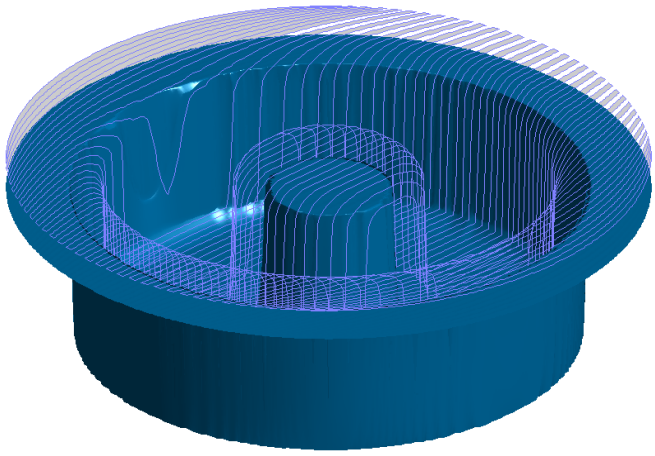
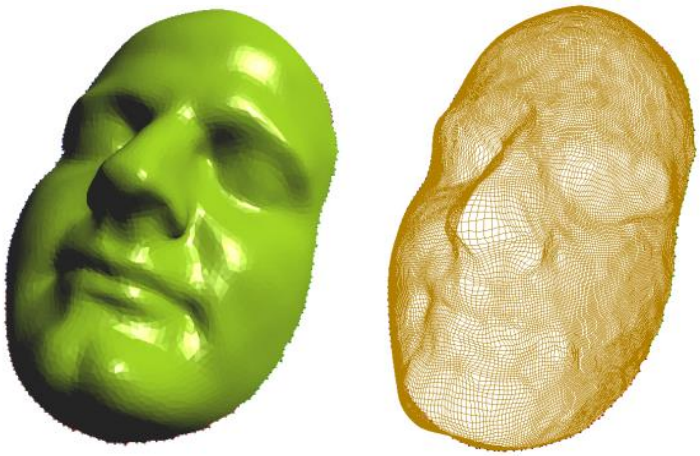


Figure 6-6 Iso-parametric tool path



(a) Original meshed surface (b) Parametric grid

(c) Zag-pattern tool path (d) Zig-pattern tool path (e) Cutter location

Figure 6-7 Iso-parametric tool path with adaptive step over

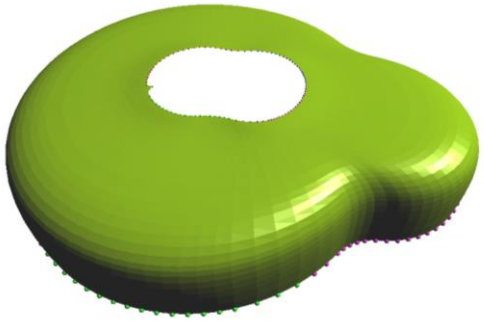
(Tool radius: 20, scallop height: 4)

6.5 Parameterization of surface with open hole

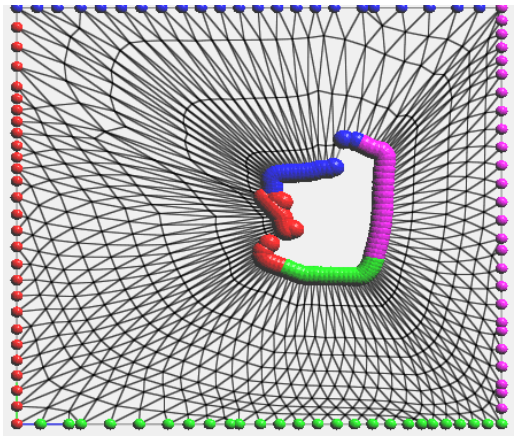
In the real applications, the cut surface is not always topologically equivalent to a disk. The boundary geometries might trim the cutting surface and form the internal holes. Figure 6-8(a) shows another case where the surface has a hole. When the points are mapped to parametric space, any open area in the surface will leads to a similar hole in the parametric surface, as shown in Figure 6-8(b). The hole of the parametric surface might cause failure for the re-sampling process as the point grids in the parametric space might end up falling into the open space as shown in Figure 6-8(b). Equation 6-4 has shown that the re-sampling of the parametric point must find a corresponding facet on the 3D surface in order to find the corresponding Euclidean coordinates; otherwise the re-sampling process will fail. In order to fill the gap while preserving the topology of the original surface, the open hole is degenerated to a singularity point in the parametric space. The singularity point can be chosen as any point, but to preserve the topology of the neighboring facet without flipping the normal direction, the singularity point must be located in the hole. The easiest method is to pick the center point of the points along the hole boundary as the singularity point. Figure 6-8(c) shows the singularity point after the merge, with the topology of the original surface preserved. Figure 6-8(d) shows the iso-parametric tool path in 3D space when the merged parametric point is re-sampled back onto the original 3D surface. Because of the intrinsic property of conformal mapping, it can be seen that the iso-parametric tool path around the hole conforms to the boundary.

The same methodologies can even be applied to surfaces with multiple holes. Figure 6-9 shows the iso-parametric tool path for a surface with two holes. Figure 6-9(a) shows the two

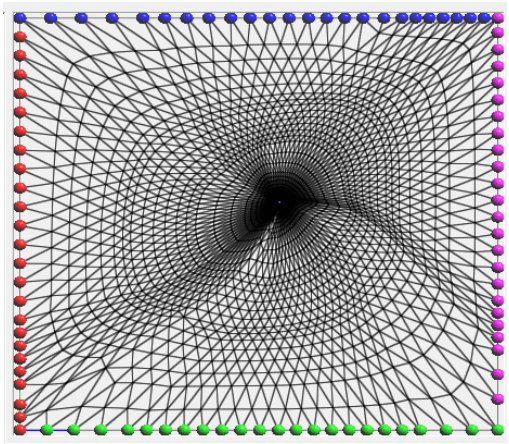
singularity points in the parametric space, and Figure 6-9(c) shows the resultant iso-parametric tool path, which has quite good conformal property along the two holes.



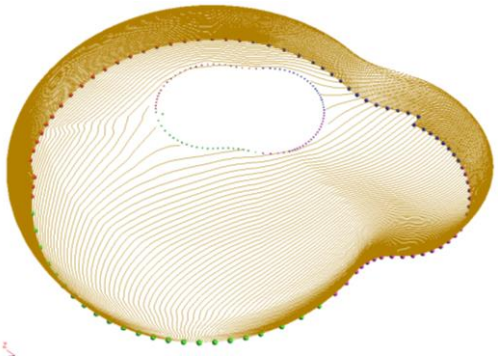
(a) Original surface with hole



(b) Parametric surface with hole

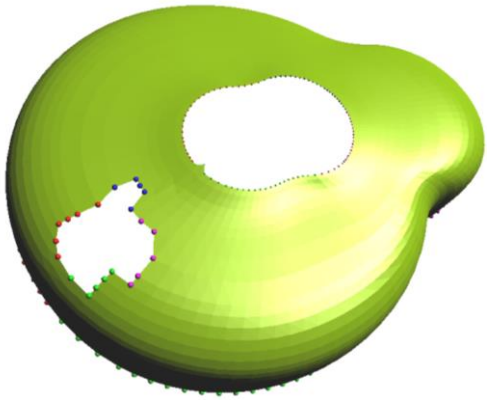


(b) Parametric surface with merged point

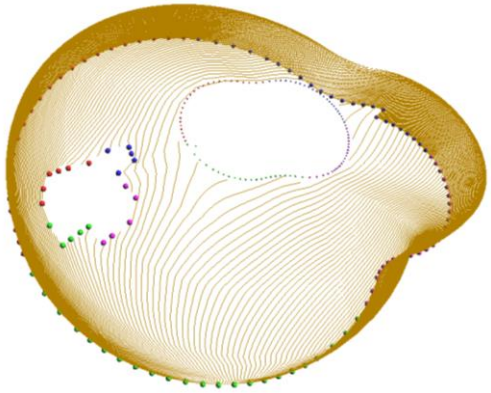


(d) iso-parametric toolpath

Figure 6-8 Iso-parametric tool path for open surface



(b) Original surface with hole of random shape



(b) Parametric surface with hole

Figure 6-9 Iso-parametric tool path with two holes

7

REAL TIME SIMULATION OF PARAMETRIC TOOL PATH

7.1 Gouge-free CL data generation for parameterized surface

7.1.1 Representation of cutting tools

Simulation is a very important phase in CAM. As the last step after NC post, it verifies the NC code or tool path and ensures that the NC code does not crash the machine. There are two major types of simulation process: material removal and machine simulation. Each of them has different purposes. As shown in Figure 7-1, material removal is intended to visualize altering the shape of stock while the cutter is moving along the tool path. It verifies the cutting surface quality and accuracy of the tool path or NC code and provides information about the continuously changing geometry in the course of the cutting process. There are two major types of material removal simulation, each of which is needed individually for different phases of CAM process. The first is tool path verification, which removes material on the basis of the tool

path. This simulation is performed before the NC codes are generated, and it verifies the accuracy of the tool path. The second is NC code verification conducted after NC post. This interpolates the NC program into the machine commands, and drives a virtual tool to remove the material from virtual stock. By comparing the cut stock with the desired stock, this simulation verifies the accuracy of NC programs.

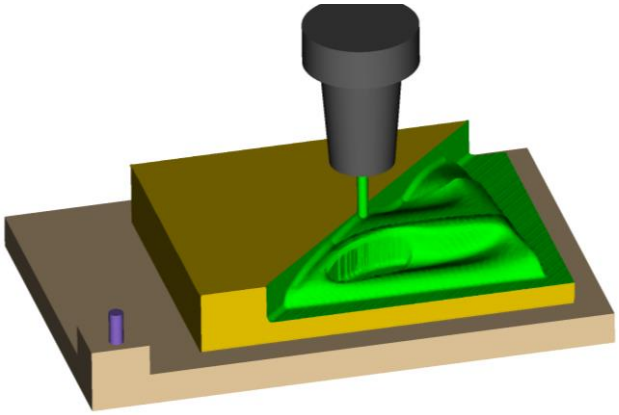


Figure 7-1 Material removal simulation [7-1]

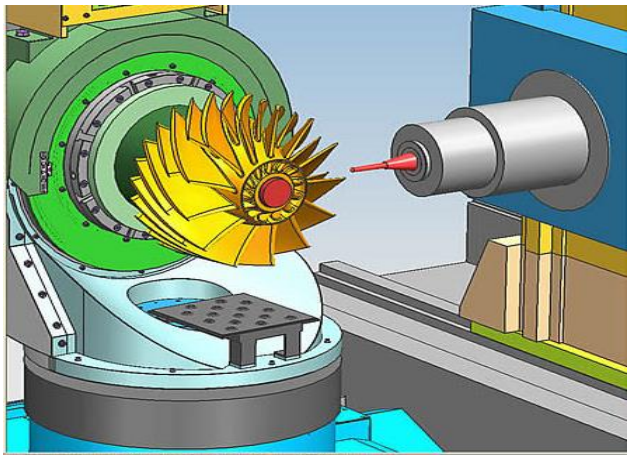


Figure 7–2 Machine simulations [7-2]

As shown in Figure 7-2, the machine simulation basically mimics the machine movement driven by the NC program through a virtual machine, and visualizes the motion of the machine components. The major purpose of this simulation is to display the kinematics of all the moving parts and performs collision detection among the moving and non-moving objects in the workspace.

When the cutter penetrates into the stock and removes material from it, it is geometrically represented by the Boolean subtraction of the cutter swept volume from the stock volume. The volume swept off by the moving tool subtracted from the volume of the in-process-stock results in the final shape of the work piece at the end of the material removal process. While the tool is moving along the tool path, the graphic engine visualizes the shape changes of in-process-stock. The core engine of simulation relies on the solid object modeling technology that is used in the CAM system. The subtraction of two solids can be converted as the Boolean operation of two B-reps. However with the early stage of PC technology, there is no hardware with sufficient power to do the calculation required for a continuous Boolean operation while the tool is moving. Therefore, many technologies have emerged in the industry to simplify the modeling of solids. Many of them are quite successful. There are varieties of simulation technologies depending on the core modeling technologies. The voxel-based method was first proposed by Chappel [102] and developed by Oliver and Goodman [103]. The result of the preceding Computer Aided Design forms the source of the Computer Aided Manufacturing process. As shown in Figure 7-3, the surface is converted into a point cloud and the local normal vector of the surface is assigned to each point. When the tool is moving over the surface, it clips these vectors. The vectors consequently grow shorter and shorter as the tool approaches the desired surface in the course of

the cutting process. The final vectors show the remaining surface errors after the machining, i.e. vectors with positive length indicate excess material, while vectors with negative length indicate an undercut in the given area. The errors are displayed by the coloring or shading of the surface model. Hook [104, 105] introduced the concept of dexels, with the name derived 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. Dexels are determined by the intersections of the ray array and the surface of the object: a dexel begins where the ray penetrates into the stock model, and ends where the line leaves it (at the back face). Dexels are described by the coordinates of the entering point and leaving point, but other attributes can be attached to them too, such as the normal vectors of the

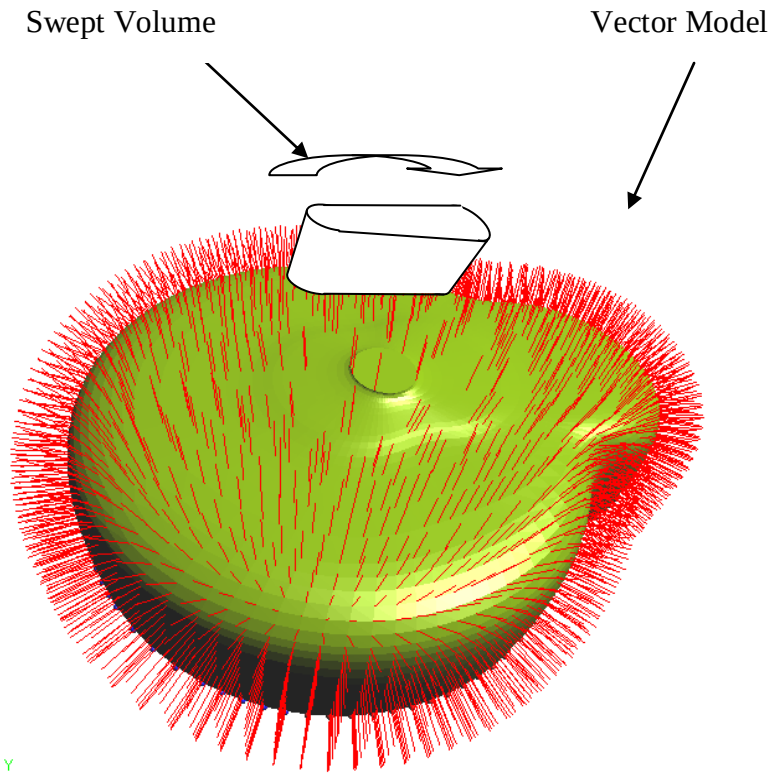


Figure 7-3 Vector-based simulation

surface at the entering and leaving points. If the ray intersects the stock model more than once, a number of dexels are created, which are linked in the memory from the front up to the rearmost face. A multi-dexel or tri-dexel method was proposed by Benouamer et al. [106] to improve the sampling accuracy of the dexel model by creating three sets of dexels in three orthogonal directions, along the three respect coordinate axes. The volume is entirely described by each of the three dexel sets, so three complete dexel descriptions are created. With this procedure significant increases of sampling resolution can be reached with little growth of memory consumption [107]. In recent years, with the development of CPU/GPU, many technologies directly based on B-rep [108] have emerged. A simplified version of B-rep is represented by a collection of connected, non-overlapping triangles, without topological information. The computational growth rate of B-rep is 0 (1.5), which can cause problems in long tool paths. Fleisig and Spence [109] attempted to improve the efficiency of the method with topology and algorithm optimization. However, the computational efficiency of B-rep–based simulation slows dramatically with an increase in the number of triangles. Dexel-based technology is more consistent. Its performance is not influenced by the length of the cutting process. However, since the elements of dexel array are not topologically connected, it is very hard to connect the dexel with its neighbors to form a facet with which current GPU technologies are compatible. There are some temporary methods to display dexel model, but the visual quality of these is usually not as good as the regular B-rep model. Other technologies include pixel-based simulation [110, 111] and voxel-based simulation [112]. But all these modeling methods are based on either 2D image buffer or an approximation voxel model. The model is not rotatable while the tool is moving, and the simulation is not accurate, especially for very complicated geometry and varieties of tool axis. With the parameterization technology proposed in this research, the B-rep can be represented as

a triangle-based facet model, and thereafter parameterized into a grid type structure. This parametric representation is actually quite similar to the dexel model. However, it preserves the topology of the model as the regular B-rep does. Because the parametric surface is represented as an array of grid points, instead of randomly distributed dexels, it resolves the graphic problem of the dexel model. Parametric grid representation of CAD model simplifies the simulation process and also provides an exchangeable data structure that is compatible with any existing modeling standards. Therefore, it has both benefits of computational efficiency and high graphic quality. In addition, because the stock surface in this method is being tessellated as a parametric grid, it is not necessary to generate swept volume for the tool movement. Instead, the material removal of stock is calculated by "plunging" the tool with every grid.

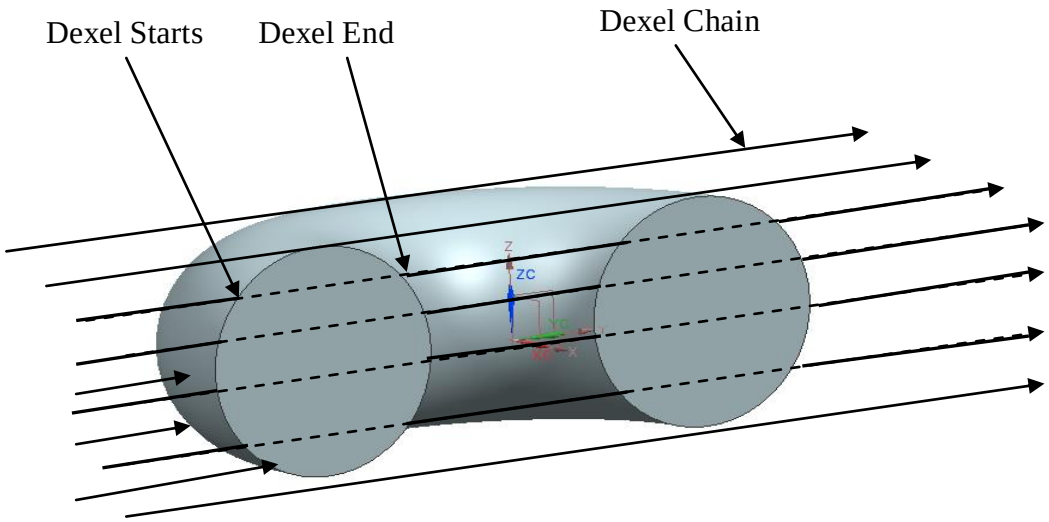


Figure 7-4 Dexel-based simulation

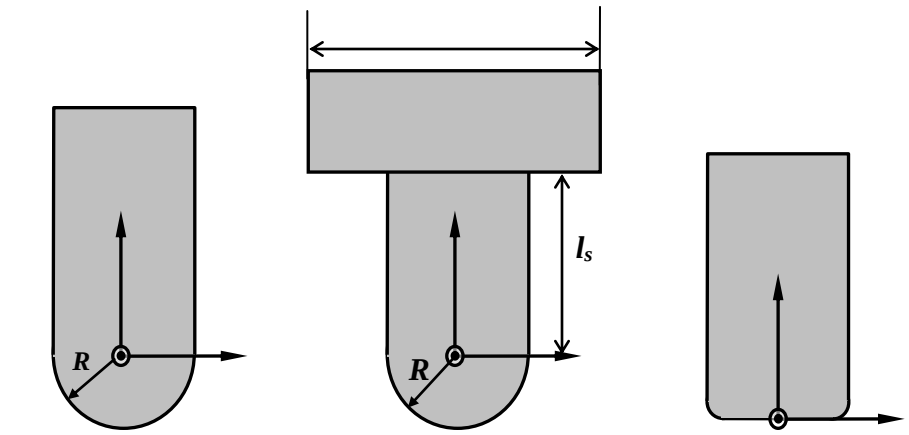


Figure 7-5 Geometrical model of cutting tools

Assuming the machining tools have simple shapes, as shown in Figure 7-5(a), their geometrical models can be easily represented as a parametric function. Presuming the cutter to be a ball-end mill tool with radius ***R*** and an infinite rod, its mathematical description is:

$$\begin{Bmatrix} X_{CT} \\ Y_{CT} \\ Z_C - \sqrt{R^2 - (X_{CT} - Xc)^2 - (Y_{CT} - X_C)^2} \end{Bmatrix} \quad \ldots (7\text{-}1)$$

where (X_C, Y_C, Z_C) is the coordinate vector of the cutter center, and (X_{CT}, Y_{CT}, Z_{CT}) represent the coordinates of the cutting surface. R is the radius. Considering the cut holder and shank as shown in Figure 7-5(b), a more precise model of the tool can be represented as:

$$\begin{Bmatrix} X_{CT} \\ Y_{CT} \\ Z_{CT} \end{Bmatrix} = \begin{cases} when \quad R \le \sqrt{(X_{CT} - X_C)^2 + (Y_{CT} - Y_C)^2} \le R_h : & \begin{Bmatrix} X_{CT} \\ Y_{CT} \\ Z_C + l_s \end{Bmatrix} \\ when \quad \sqrt{(X_{CT} - X_C)^2 + (Y_{CT} - Y_C)^2} \le R : & \begin{Bmatrix} X_{CT} \\ Y_{CT} \\ Z_C + l_s \end{Bmatrix} \end{cases} \quad \ldots (7\text{-}2)$$

There are also other types of cutting tools, such as fillet end mills or flat end mills etc. They can all easily be depicted in similar functions as Equation (7-2). In the current research the ball mill tool is implemented, with the cutting tip presumed to be an infinite rod with a semi-spherical end.

7.1.2 Generation of cutter center location

Once the tool path has been obtained, the precise estimation of the cut error is needed to determine the scallop height, pick feed and the cut center. The most straightforward way to do this is to offset the tool path points along the norm direction by the radius of the cutter. However, this method is faulty where local geometry has a small curvature radius. It is also impossible to derive an accurate norm direction based on the discrete parameterizations in the absence of differential properties, especially for our scheme where the surface is described in the form of the parametric gridded points. An alternative solution for this problem is to transfer it to a Maximal Marching Algorithm (MMA). As shown in Figure 7-6, if CL is located at $\mathbf{G}_{i,j}$, where *i, j* is the index number of the node in the array of grid matrix.

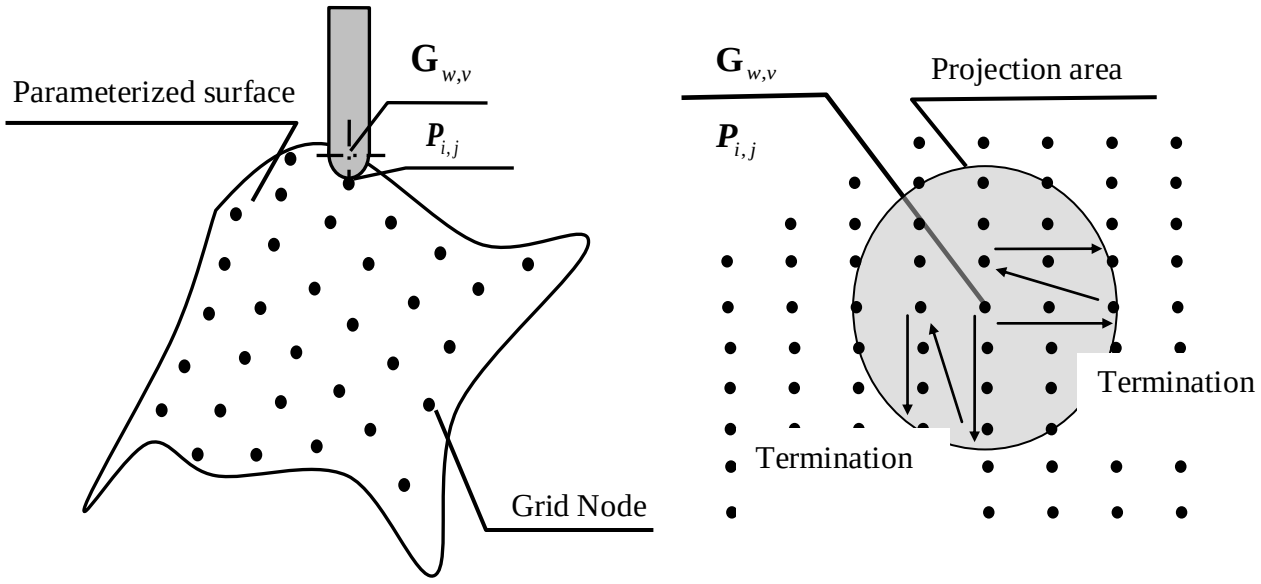


Figure 7-6 Node sorting process: the termination of (*i, j)* increment

The initial set of the cut center location is represented as

$$\mathbf{C}^{0}{}_{w,v} = \mathbf{G}_{w,v} + \mathbf{R} \qquad \ldots (7\text{-}3)$$

where *w, v* is the index number of cutter center location $\mathbf{C}^{0}{}_{w,v}$, and $\boldsymbol{R}$ is an approximation of the cutter radius vector along the tool axis direction. The tool axis can be set as surface normal. Alternatively, in fixed axis machining, the tool axis is constantly along Z the axis. The *X* and *Y* components of $\boldsymbol{R}$ are equal to 0, namely $\boldsymbol{R} = (0,0,|\boldsymbol{R}|)$.

The steps of the process are described in detail as the following iteration. Presuming the cutter touches with the desired surface at the position of (*w, v*), the location of the cutter center in this case should be $\mathbf{C}^{\mathbf{0}}{}_{w,v} = \mathbf{G}_{w,v} + \mathbf{R}$. This is an initial position determined by the central point

of the tool and its corresponding grid on the desired surface. To ensure there is no interference between the cutter and the desired surface, let *i, j* iterate around grid point (*w, v*). Given the ball-end cutter model, for each pair of (*i, j*) one needs to sort out all the grid nodes that are covered by the projection area of cutter tip along Z direction, that is:

$$\left|\left(X_{\mathbf{G}_{i,j}} - X_{\mathbf{C^0}_{w,v}}\right)^2 + \left(Y_{\mathbf{G}_{i,j}} - Y_{\mathbf{C^0}_{w,v}}\right)^2\right| < |\mathbf{R}|^2 \quad \ldots (7\text{-}4)$$

where (*X, Y, Z*) represents the components of the subscript vector. In order to expedite the process, the sorting process starts from the point at (*w, v*) that is located at the center of the projected area along the cutter axis. As shown in Figure 7-6, each loop moves from the center point and terminates at the first grid point that is not covered by the projection area of the cutter.

If the grid node is located in the projection area of the cutter tips, check if it satisfies:

$$Z_{C_{i,j}} - \sqrt{|\mathbf{R}|^2 - \left(X_{\mathbf{G}_{i,j}} - X_{\mathbf{C^0}_{i,j}}\right)^2 - \left(Y_{\mathbf{G}_{i,j}} - Y_{\mathbf{C^0}_{i,j}}\right)^2} < Z_{G_{i,j}} \quad \ldots (7\text{-}5)$$

If No: Jump to the next grid by increasing/decreasing (*i, j*) pair by 1.

If Yes: adjust $\mathbf{C^0}_{w,v}$ by $\mathbf{C^1}_{w,v} = \mathbf{C^0}_{w,v} + \mathbf{\Delta C}$. Figure 7-7 has indicated the geometrical meaning of $\mathbf{\Delta C}$, which is the adjusted distance for the position of the tool center and ensures there is no over-cutting when the tool tip moves along the tool paths. $\mathbf{\Delta C}$ is the offset of the cutter along the tool axis, when the tool axis is along the Z axis,

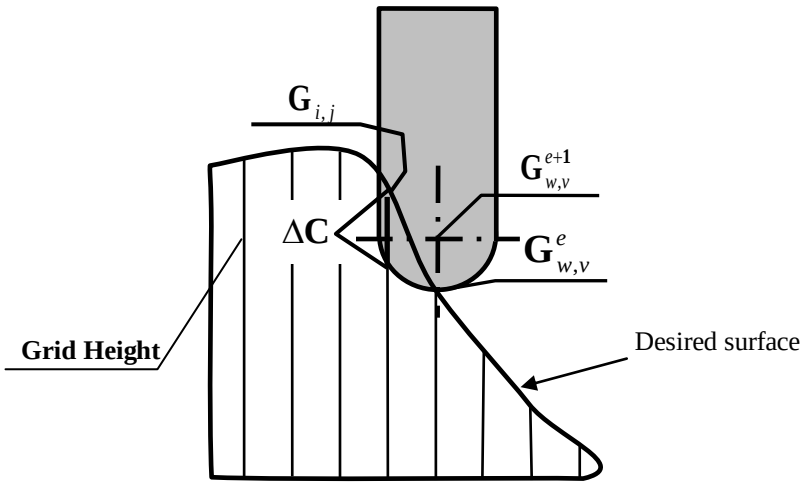


Figure 7-7 Maximal marching algorithm

$$\Delta \mathbf{Z} = \begin{Bmatrix} 0 \\ 0 \\ \sqrt{|\mathbf{R}|^2 - \left(X_{\mathbf{G}_{i,j}} - X_{\mathbf{C}^0_{i,j}}\right)^2 - \left(Y_{\mathbf{G}_{i,j}} - Y_{\mathbf{C}^0_{i,j}}\right)^2} + Z_{(\mathbf{G}_{i,j} - \mathbf{C}^0 i,j)} \end{Bmatrix} \quad \ldots (7\text{-}6)$$

Iterating through all the grids covered by the projection area of the tool along the tool axis, the final cutter gouge-free center coordinate, represented as $\mathbf{C}^e_{w,v}$, can be obtained with Equation (7-6). After all these iterations, the cutter will only contact the desired surface at one grid point, and all other grid nodes are located outside of the tool. If the grid size is sufficiently small, it is acceptable for there t be only one contact point between cutter and the desired surface when the cutter locates $\mathbf{C}^e_{w,v}$, which is used to compose the tool path.

7.2 Simulation of the iso-parametric tool path

Once the cutter center is determined, the resulting scallop surface can be derived by iterating and adjusting the position of the grid points that are cut by the tool. The heights of the grid points on the initial scallop surface are adjusted based on the deviation distance between the tool surface and scallop surface.

First initialize the grid of the cut surface as in-process stock. Let *i, j* increase/decrease around (*w, v*) iterating as shown in Figure 7-6. Each pair corresponds to one cutter location, where the material is removed from the initial surface. When the tool tip moves to a new position, i.e. $\boldsymbol{C}_{w,v}$, verify if the material between the initial surface and the desired surface is removed by the tool. As shown in Figure 7-8, given a grid node on the initial surface, $\boldsymbol{G}_{i,j}$, if it is located inside the tool when the tool is being placed at $\boldsymbol{C}_{w,v}$, the material should be removed. In this case, the coordinate vector of $\boldsymbol{G}_{i,j}$ should be adjusted to the lower position, at which machining surface has no interference against tool. Assuming the tool is a ball-end cutter, its geometrical model can be simplified to an infinite rod with a half-spherical end, as depicted in Equation (7-2), namely

$$\mathbf{G_{i,j}} = \begin{Bmatrix} X_{\mathbf{G}i,j,l} \\ X_{\mathbf{G}i,j,l} \\ Z_{\mathbf{G}i,j,l} \end{Bmatrix} = \begin{Bmatrix} X_{\mathbf{G}i,j} \\ X_{\mathbf{G}i,j} \\ Z_{\mathbf{G}i,j} - \sqrt{|R|^2 - \left(X_{\mathbf{C}_{w,v}} - X_{\mathbf{G}_{w,v}}\right)^2 - \left(Y_{\mathbf{C}_{w,v}} - Y_{\mathbf{G}_{w,v}}\right)^2} \end{Bmatrix} \quad \ldots (7\text{-}6)$$

where only the Z-coordinate of the cut surface needs to be changed, while other two coordinates keep the same value. Figure 7-9 shows the case in which $\mathbf{G}_{i,j}$ has already been adjusted before the cutter moves to $\mathbf{C}_{w,v}$. However, $\mathbf{G}_{i,j}$ is possibly still inside of the tool, the cutter has moved to $\mathbf{C}_{w,v}$, and consequently $\mathbf{G}_{i,j}$ should be adjusted to a lower position until there is no interference between the grid height of the grid node at (i, j) and the tool tip when it moves along the tool path.

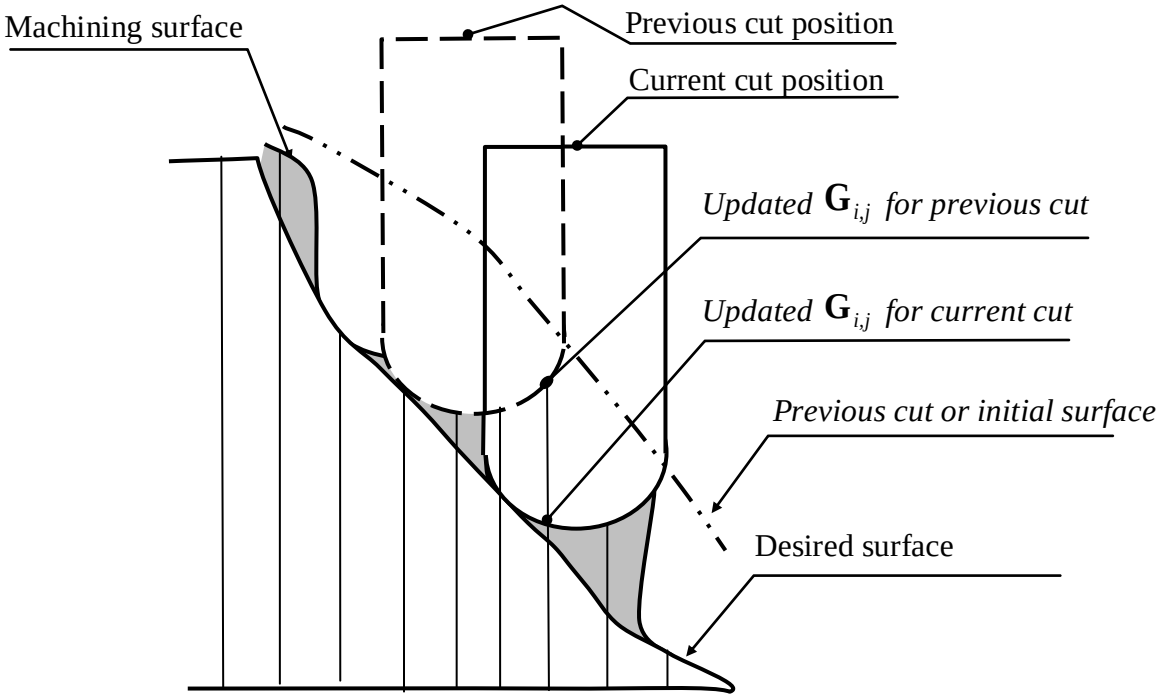


Figure 7-8 Update of scallop surface

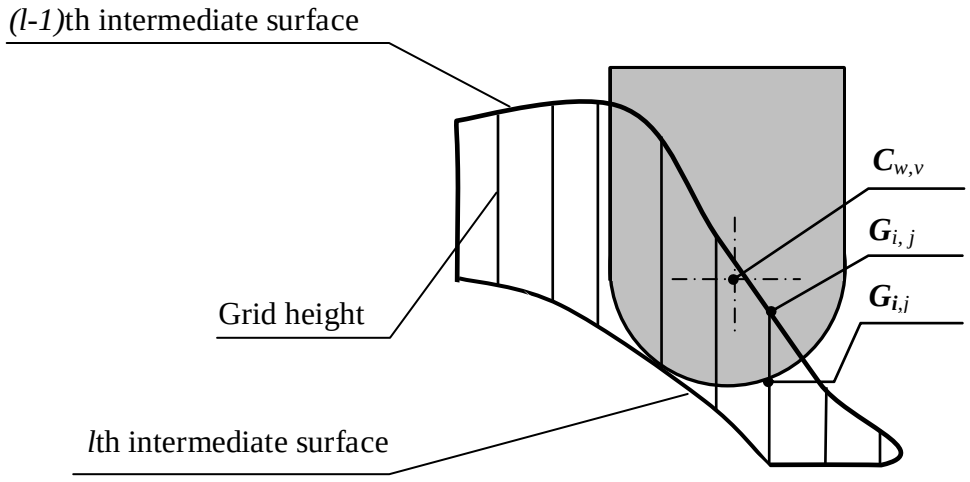


Figure 7-9 Simulation of scallop surface

7.3 Error evaluation

The cutting error is the deviation between the actual cut surface and the desired surface along its normal direction. This definition applies only where the normal direction can be accurately obtained. Most applications, however, such as the facet model or dexel model, do not have an explicit description of normal direction at a specific point. It is time consuming to calculate the distance between the two surfaces with tessellated meshes time consuming

Moreover, the scallop of the surface produces a major deviation from the target surface. Although the scallop height is controlled within the required criterion, the deviation can be longer along one axis than other. This happens especially easily for three axes CNC machine. As shown in Figure 7-10, the more reasonable definition of machining error on a point is the minimal distance of the nodes on scallop surface from the desired surface, that is

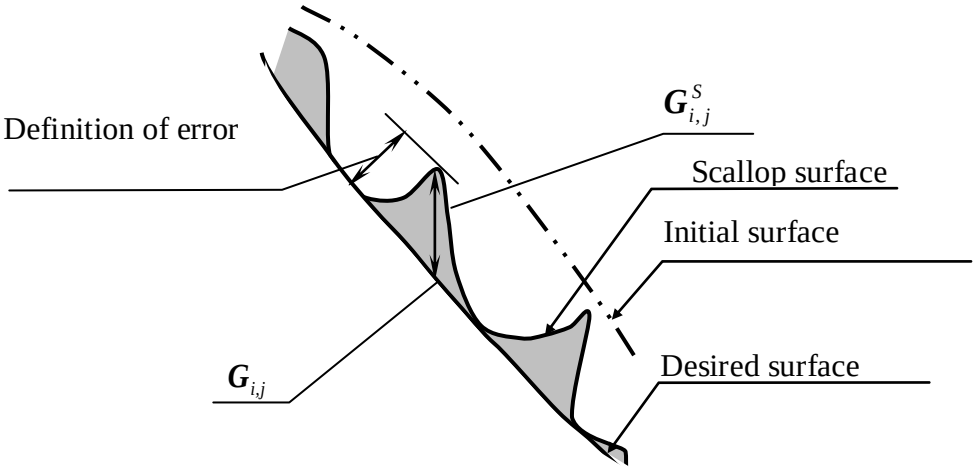


Figure 7-10 New definition of Cutting Error for three-axis CNC

$$E_{i,j} = Max\left\{\left|\boldsymbol{G}_{i,j} - \boldsymbol{G}^{s}_{i,j}\right|\right\} \quad \ldots(7\text{-}7)$$

where $\boldsymbol{G}_{i,j}$ is the grid point on the desired stock and $\boldsymbol{G}^{S}_{i,j}$ is the corresponding grid point on the scallop surface. Their difference does not always point to the normal direction of the desired surface. Measurement of distance between the two points represents exactly the shift of the grid point of the surface while the shape of stock is transforming. $E_{i,j}$ is the max value of the shift value for all the grid points across the entire cut surface.

7.4 Case Study

Figure 7-11, Figure 7-12 and Figure 7-13 have shown some results of simulation with this algorithm for a fixed axis tool path. It can be seen that this algorithm has provided a highly realistic picture of the machined surface as well as an accurate estimate of the deviation between the scallop and the desired surface. Its scallop height is relatively consistent along the entire cut surface, while the distribution of the tool path is not overly condensed; instead the step over is determined by surface curvature and the required scallop, as indicated in Equation 6-21. The curves in the pictures are the cutter center locations resulting from the algorithm discussed in the prior chapter. Figure 7-11 and Figure 7-12 compare the influence of tool path step over with the scallop height of the resulting cut surface. If the step over is bigger, the scallop is normally higher as long as the curvature of the surface does not change dramatically.

On the other hand, Figure 7- 12 and Figure 7-13 show another scallop surface resulting from the tool path that is obtained for different tool sizes. It can be observed from these two pictures that the smaller tool normally has a smooth surface quality. However, a small tool needs a smaller step over, which will greatly change the time cost. As mentioned before, the best tool path is to optimize the step over of the tool path to eliminate the unnecessary passes and minimize the total cutting length, while controlling the scallop height within the required range. As shown in Figure 7-14, Figure 7-15 and Figure 7-16, the tool path passes are not distributed evenly on the surface. The density of passes varies according to the local curvature of the cut surface as well as the required scallop height.

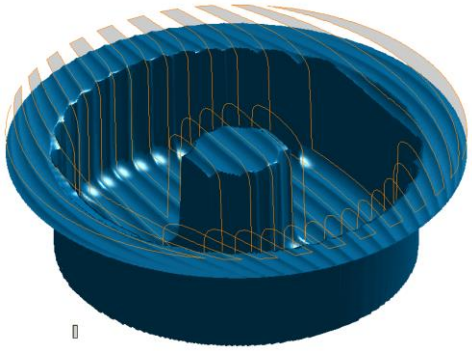


Figure 7-11 The scallop surface by tool-nose of R0.05,
(Mesh size 200x200; tool path step over: 10 parametric increments)

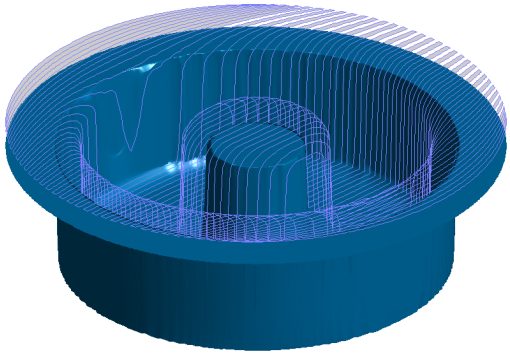


Figure 7-12 The scallop surface by tool-nose of R0.05
(Mesh size 200x200; tool path step over: 3 parametric increments))

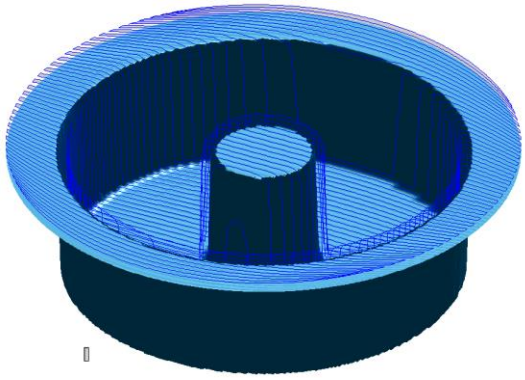


Figure 7-13 The scallop surface by tool-nose of 0.02
(Mesh size 200x200; tool path step over: 3 parametric increments))

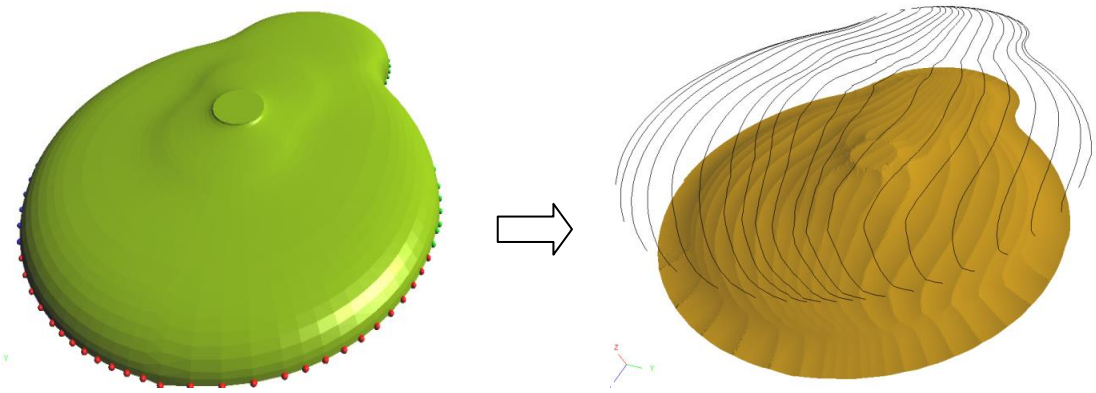


Figure 7-14 Cut surface for the tool path with scallop control

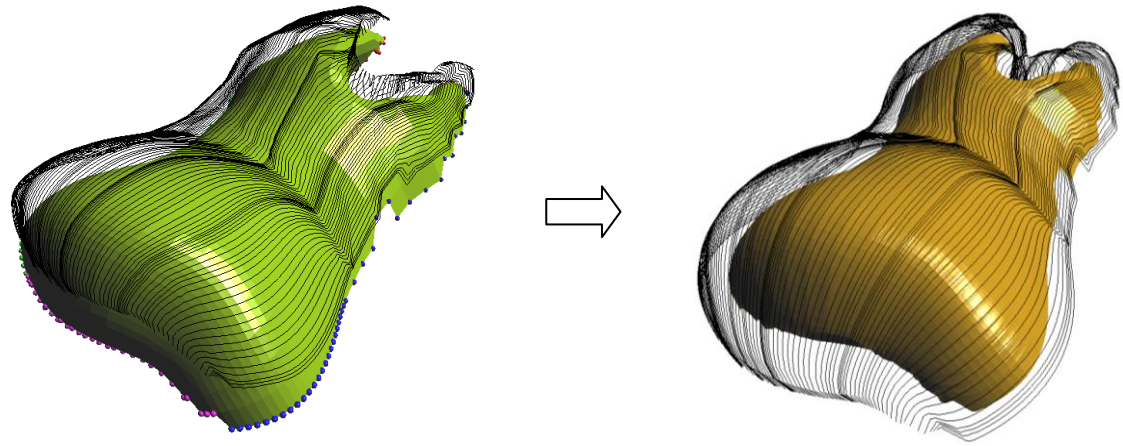


Figure 7-15 The scallop surface by tool-nose of 5
(Mesh size 200x200; expected scallop height: 0.8)

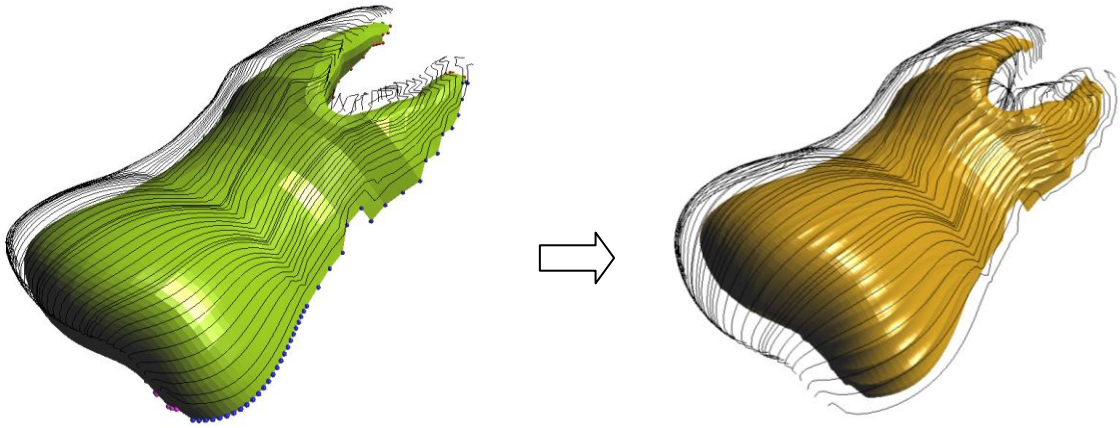


Figure 7-16 The scallop surface by tool-nose of 5
(Mesh size 200x200; expected scallop height: 1.6)

8

CONCLUDING REMARKS

Based on a unified CAM model, this research attempted to develop a framework for the integration of design and manufacturing systems. Because in the current integration system the CAM process is based on the CAD model, there is no CAM-oriented model that is exchangeable with CAM process. Current research proposes a new scheme of data acquisition and data handling based on a CAM-oriented model that removes any manufacturing information from the design process. Instead, the CAM-oriented model only provides the geometrical information of cut volume by subtracting the designed stock from the in-process stock. Thereafter, the machining features are identified within the context of manufacturing process. Unlike STEP-NC, the semantics of the proposed CAD data is not dependent on the manufacturing context. Changes of manufacturing environment do not affect the data exchanges between CAD/CAM. It neither advocates that the design process should mimic or simulate the manufacturing process, nor that the CAM process should be built on the basis of the design; instead a CAM-oriented model is obtained from the CAD model by subtracting the design model from the in-process stock. The

CAM agents will identify the machining features in CAM-oriented model and convert it into a set of machining features based on the available manufacturing resources in real-time. The machine feature to be created depends on the real-world condition of the manufacturing workshop. Since the CAM-oriented model translates the CAD model into a set of features that is specifically compatible with CAM process, it can easily achieve automation for the tool path generation and NC post. Meanwhile, the conversion from CAD features to CAM features is not a one-to-one mapping; that is, one CAD feature can be designed or converted to multiple CAM operations and multiple features. It is up to the manufacturing engineers to specify the operation type in the context of the manufacturing facility. This data scheme increases the flexibility of process planning. This new concept does not promote CAD engineers to be experts in the manufacturing process, but the economics of design can be defined and dealt with by designers. The operational nature of manufacturing is still reflected on the manufacturing side. A manufacturing context is not needed as the foundation of a design language, such as operations, tolerances, etc.

Once the machining feature is established with the CAM model, the geometric information of each feature can easily be processed in a unified manner. In order to show the merits of this new scheme, a unified method, particularly for freeform milling feature, is exploited to automate the process planning and tool path generation. Instead of the three phases that are adopted in the conventional manufacturing methods, with parameterization technology, current research unifies the whole machining process to several morphing surfaces. Since the shape of stock is altered in gradually approaching the desired shape, the surface quality will retain good finish quality and have minimal surface stress. Based on the same parameterization on each morphing surface, the

tool path can be generated specifically for finish milling. Also, the simulation can easily be performed on the basis of the same modeling data and algorithm.

As discussed above, future work in CAD/CAM integration will be targeted on the area of remote manufacturing systems, where the manufacturing context is not, or is only partially, clear. Consequently, full automation of CAD/CAM integration requires a simple, exchangeable semantics and unified scheme for data acquisition in CAM.

The next stage of this research will focus on system synthesis and adding the new machining features for different operation types:

i. Although the morphing algorithm of Laplace solution is feasible for most model. As a partially differential method, the current numerical solution for Laplace is not stable enough, and it generates parameterization anomalies as experienced in algebraic methods. As mentioned by Daniel C.H. Yang and Oulee. T. H. in their research [60], Laplace tends to provide parameterization with quite uneven grid distribution around the areas with high boundary irregularities. An alternative solution of this problem is to have a new parameterization strategy that can control the morphing of facet models directly with the constraints of boundary surfaces.
ii. Extend the idea of morphing-based unified process planning to multi-axis lathe operation. Normally CNC lathe machine is applied to revolving feature that has 2D profile.
iii. The same morphing-based planning can also be used for other manufacturing mechanisms. Machining is a subtractive process, that is, it consists in removing material from the stock. Some new manufacturing methods, such as composite manufacturing and 3D printing, are

additive. They obtain the desired shape by building up the material layer by layer. The morphing strategy can potentially be used to model the additive manufacturing process as well.

iv. Create a prototype of this technology with the commercial software. The work includes building API in the commercial CAD software to convert the CAD model into a CAM-oriented model, and developing a software package in CAM software to recognize the machining features and generates process planes as well as tool path.

Appendix A

Object Oriented Programming and Design Pattern

A-1 Definition

Object Oriented Programming (OOP) is a programming philosophy, in which both the data type and the collection of operations are encapsulated into one data structure. The data structure becomes an object that includes both data and behavior of the object to be described. In addition, this programming mythology allows specifying the relationships between objects, i.e. objects can inherit characteristics from other objects. One of the principal advantages of object-oriented programming techniques over the procedural programming techniques is that they enable programmers to preserve the existing modules when a new type of object is to be added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to expand its function-abilities without the risks to create any regressions on the existing module. The classes are usually divided into two major levels, one in the abstraction level is called abstract class, and one in the implementation level is called concrete class

A-2 Definition

Interface class (abstract class) is class that abstracts the object meaning from the real world. As opposed to the concrete class, it abstracts the behavior of object to be described and only defines generic methods in the class. Instead of actual implementing the actual behavior, it only provides semantic definition of the behaviors that are common among a specific type of objects. Those methods will be implemented later in specialized sub-classes. This is how we get interfaces-

interfaces are the implementation of the abstract class. With the interface, the common behavior of target objects can be abstracted as core architecture through the unified interfaces to interact with each other. The actually implementation is defined with the concrete classes.

A-3 Definition

Concrete Class is a derived class of interface class. It provides implementations for the member functions that are not implemented in the base class. A derived class that actually implements all the missing functionality is called a concrete class.

A-4 Definition

Design Pattern is a general repeatable solution to a commonly occurring problem in OOP software design. Built on top of OOP classes, it describes the relationship between the objects and is a description or template for how to solve a problem that can be used in many different situations. Effective software design requires considering issues that may not become visible until later in the implementation. Design patterns can speed up the development process by providing tested, proven development paradigms. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns. Often, people only understand how to apply certain software design techniques to some specific problems. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem. In addition, patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs. There

are three major catergories of design pattern: creational design patterns, structure design patterns and Behavioral design patterns.

Appendix B

Computational Geometry

B-1. Definition

A meshed surface is defined point set P with manifold connectivity that is locally hormorphic to a disk.

B-2. Definition

Iso-curve coverage of surface S is defined as a set of curves i.e. C that is conformed on the boundaries of the surface and for any point Pc on the curves, always can find the point Ps so that $\|P_c - P_s\| \le \phi$ and ϕ is any small positive.

B-3 Definition

Suppose for every point x in a manifold M, an inner product $<\cdot,\cdot>_x$ is defined on a tangent space T_xM of M at x. Then the collection of all these inner products is called the Riemannian metric.

B-3 Definition

The metric tensor is defined abstractly as an inner product of every tangent space of a manifold such that the inner product is a symmetric, nondegenerate, bilinear form on a vector space. This means that it takes two vectors v, w as arguments and produces a real number $<v, w>$ such that

$<kv, w>=k<v, w>=<v, kw>$

$<v+w, x>=<v, x>+<w, x>$

$<v, w+x>=<v, w>+<v, x>$

$<v, w>=<w, v>$

$< \mathbf{v}, \mathbf{v} >\geq \mathbf{0}$ with equality iff v=0

B-4 Definition

The inner product of R^3 generates a quadratic form that takes vectors in the tangent space to the real line, that is, $I_p : T_p(S) \to R^3$. Given a vector w in the tangent plane, the first fundamental form is given by

$$\mathbf{I_p}(w) = < w, w > = |w|^2 \geq 0$$

Given a parameterization $f(u,v)$ of the surface, we can express the first fundamental form in the basis $\{\boldsymbol{x}_u, \boldsymbol{x}_v\}$. Recall that a tangent vector $\boldsymbol{w}$ is by definition the tangent of some curve $\boldsymbol{s}(t) = \boldsymbol{x}(u(t), v(t))$ with $\boldsymbol{s}(0) = \boldsymbol{p}$. Expanding the first fundamental form, we get

$$\boldsymbol{I_p}(w) = \boldsymbol{E}u'^2 + 2\,\boldsymbol{F}\,u'\,v' + \mathbf{G}\,v'^2$$

where $\boldsymbol{E}=<\boldsymbol{r_u}, \boldsymbol{r_u}>$, $\boldsymbol{F}=<\boldsymbol{r_u}, \boldsymbol{r_v}>$, $\boldsymbol{G}=<\boldsymbol{r_v}, \boldsymbol{r_v}>$

B-4 Theorem

Coordinate patches of surfaces $\boldsymbol{f}$ and $\boldsymbol{f}'$ are isometric if and only if there exist parametrizations $\boldsymbol{f} : \boldsymbol{V} \to \boldsymbol{R}^3$ and $\boldsymbol{f}' : \boldsymbol{V} \to \boldsymbol{R}^3$, respectively, with the same first fundamental form.

B-5 Definition

The area of an open continuous surface can be represented in terms of first fundmental form:

$$Area = \int_v^f |f_u \times f_v| dvdu = \int_v^f \left|\sqrt{EG - F^2}\right| dvdu$$

B-6 Definition

Let $\boldsymbol{I_p}(w) = \boldsymbol{E}u'^2 + 2\,\boldsymbol{F}\,u'\,v' + \mathbf{G}\,v'^2$ be the first fundamental form of

Then the second fundamental form of $\boldsymbol{s}(u, v)$ is the first order form of its first derivative

$$\boldsymbol{L}du^2 + 2\,\boldsymbol{M}\,dudv + \mathbf{G}\,dv^2$$

Where $L = \boldsymbol{s}_{uu} \bullet \boldsymbol{n}$, $M = \boldsymbol{s}_{uv} \bullet \boldsymbol{n}$, $\mathbf{G} = \boldsymbol{s}_{vv} \bullet \boldsymbol{n}$, and $\boldsymbol{n}$ is the normal vector

B-6 Definition

Definition 3.1. The Gaussian curvature K of a surface in $\boldsymbol{R}^3$ is

$$K = \frac{LN - M^2}{EG - F^2}$$

It is independent of the parameterization

B-5 Theorem

The Gaussian curvature depends only on the first fundamental form, which can be expressed as

$$K = \frac{\begin{vmatrix} -\frac{1}{2}E_{vv} + F_{vv} - \frac{1}{2}G_{uu} & \frac{1}{2}E_u & F_u - \frac{1}{2}E_v \\ F - \frac{1}{2}G_u & E & F \\ \frac{1}{2}G_v & F & G \end{vmatrix} - \begin{vmatrix} 0 & \frac{1}{2}E_v & \frac{1}{2}G_v \\ \frac{1}{2}E_v & E & F \\ \frac{1}{2}G_u & F & G \end{vmatrix}}{\left(EG - F^2\right)^2}$$

This expression indicates that surfaces that have the same first fundamental form, or, equivalently, surfaces that are isometric, have the same Gaussian curvature. This is enough to conclude that the parabolic cylinder has 0 Gaussian curvatures, as it is isometric to another surface, the plane, with 0 Gaussian curvature.

B-6 Definition

The curvature of a curve is the signed inverse of the radius of the osculating circle, which is the circle tangent to the curve that also matches the second derivative of the curve. The sign is positive if the circle is on the same side of the surface as the normal vector, or negative if it is on the opposite side of the normal vector.

B-7 Definition

The principal curvatures of a point are the extremal values of the curvatures of all curves through that point obtained by intersecting the surface by all planes containing the normal vector.

B-8 Theorem

The Gaussian curvature at a point is the product of the two principal curvatures at that point.

Though we state this as a theorem, we can also define the Gaussian curvature to be the product of the two principal curvatures; the definitions are equivalent. Using this definition of Gaussian curvature, it is easy to see how the parabolic cylinder has 0 Gaussian curvature. At any point, there is a plane containing the normal whose planar curve is a straight line parallel to the y-axis. This curve has curvature 0, as the osculating circle has infinite radius, and this is a minimal value, so the Gaussian curvature is 0.

Append C

Memory Management of stacked heap

Following functions define a new mechanism for the memory pool allocation. Deferent from the regular heap mechanism, the allocated memory of heap is always preserved for the expansion. When the allocated memory is exhausted, a new memory block is allocated, and the memory address of new memory block is appended to stacks of memory addresses. To delete the memory blocks, the stack elements are traversed to be freed. Following code is implemented as C style, with a structure ss_HEAP defined.

```
#define HEAP_ALLOCATION_SIZE 100
#define MAX_STORAGE_COUNT 50
/* ********************************************************************** */
typedef struct ss_HEAP
{
char **heap_storage;                /* address stack of allocated heap blocks */
int storage_count;                  /* count of used heap blocks */
int max_storage_count;              /* maximum size of heap stacks*/
int  heap_size;                     /* current number of elements in ALL heap block */
int  heap_total_size;               /* maximum element number in ALL heap blocks */
int  heap_buffer_size;              /* realloc incremental number of elements, which is the
                                    number of elements for each newly allocated heap block */
int  element_size;                  /* size of the allocated element type */
} ssHEAP;

/* ********************************************************************** */
/**
* function definition
* @file
*
* This module contains a set of functions which allows for managment of dynamic data.
* The data is stored in an object refered to as a heap.
* The allocated memory are stacked chunks of memory blocks
*  </pre>
/* ********************************************************************** */
```

```
/*****************************************************************************/
/**
 * Initializes a heap.
 *
 * @param heap1
 */
void SMEM_initialize (char *heap1)
{
        ssHEAP *heap2;

        heap2 = (ssHEAP *)heap1;

        if (heap2 == NULL) return;
        heap2->heap_storage = (char**) calloc(MAX_STORAGE_COUNT, sizeof(char*));
        heap2->storage_count = -1;
        heap2->max_storage_count = MAX_STORAGE_COUNT;
        heap2->heap_size = 0;
        heap2->heap_total_size = 0;
        heap2->heap_buffer_size = HEAP_ALLOCATION_SIZE;
        heap2->element_size = 0;
        heap2->alloc_notify_func = NULL;
        heap2->alloc_notify_data = NULL;
}

/*****************************************************************************/
/**
 * Creates and initializes a heap by specifying buffer size
 *
 * @param element_size
 * @param buffer_size
 *
 * @return char*
 */
char *SMEM_create (int element_size, int buffer_size)
{
      ssHEAP *heap;
      //return Null if the element_size is negative value
      if (element_size < 1) return(NULL);

      //Allocate the memory
      heap = (ssHEAP *)calloc(1, sizeof(ssHEAP));
      if (heap == NULL)          return(NULL);

      //Initialize the memory
```

```
    SMEM_initialize((char *)heap);

    heap->element_size = element_size;
    heap->heap_total_size = 0;
    SMEM_set_buffer_size((char*)heap, buffer_size);

    return((char *)heap);
}


/*****************************************************************************/
/**
 * Allows user to set incremental buffer size to size other than
 * HEAP_ALLOCATION_SIZE.
 *
 * @param heap1 heap
 * @param size new buffer size
 *
 */
void SMEM_set_buffer_size (char *heap1, int size)
{
    ssHEAP *heap2;

    heap2 = (ssHEAP *)heap1;
    if (heap2 == NULL) return;
    if (size < 1) return;
    heap2->heap_buffer_size = size;
}


/*****************************************************************************/
/**
 * loop through all the storage chunks and free heap only.
 *
 * @param heap1 heap
 *
 */
void SMEM_free (char *heap1)
{
    ssHEAP *heap2;
    int index;
    heap2 = (ssHEAP *)heap1;

    if (heap2 == NULL) return;
    if (heap2->heap_storage)
```

```
    {
        for (index = 0; index<heap2->max_storage_count; index++)
        {
                if (heap2->heap_storage[index])
                        free(heap2->heap_storage[index]);
        }
        free(heap2->heap_storage);
        heap2->max_storage_count = 0;
    }
    heap2->storage_count = -1;
    heap2->heap_size = 0;
    heap2->heap_total_size = 0;
}


/*****************************************************************************/
/**
 * Frees heap storage and heap data.
 *
 * @param heap1 heap
 *
 */
void SMEM_destroy (char *heap1)
{
    ssHEAP *heap2;

    heap2 = (ssHEAP *)heap1;

    if (heap2 == NULL) return;
    SMEM_free((char *)heap1);
    free(heap2);
}


/*****************************************************************************/
/**
 * Copy and append the input data to heap.
 * NOTES: Additional heap block will add if there is no enough space.
 *
 * @param heap1 heap
 * @param element address of data being stored on heap
 *
 * @return index of element on heap
 */
int SMEM_append_data (char *heap1, char *element)
```

```
{
    int   alloc_notify, ind = 0;
    char  *ptr;
    ssHEAP *heap2;

    heap2 = (ssHEAP *)heap1;

    if (heap2 == NULL)
        return(-1);
    if (element == NULL)
        return(-1);
    if (heap2->element_size < 1)
        return(-1);

    //Allocate the memory pool if the storage_count is negative
    if (heap2->storage_count<0)
    {
        heap2->storage_count++;
        heap2->heap_storage[heap2->storage_count]  =  (char*)calloc(heap2->heap_buffer_size,
        heap2->element_size);
        if (heap2->heap_storage[heap2->storage_count] == NULL)
            return(-1);
        heap2->heap_total_size = heap2->heap_buffer_size;
    }

    //If the assigned size is less than total allocated size, copy the data to the last available address
    if (heap2->heap_size < heap2->heap_total_size)
    {
        memcpy((heap2->heap_storage[heap2->storage_count]+heap2->element_size  *  (heap2-
        >heap_size % heap2->heap_buffer_size)), element, heap2->element_size);
        heap2->heap_size++;
    }

    //If all the allocated memory block are assigned, allocated a new block and append is into //the
    storage stack
    else
    {
        if (heap2->storage_count == heap2->max_storage_count -1)
        {
                heap2->max_storage_count += MAX_STORAGE_COUNT;
                heap2->heap_storage=(char**)realloc(heap2->heap_storage,
                                                heap2->max_storage_count * sizeof(char*));
                for(ind = heap2->storage_count+1; ind<heap2->max_storage_count; ind++)
                {
                        heap2->heap_storage[ind] = NULL;
```

```
                }
        }
        ptr =  (char *)calloc(heap2->heap_buffer_size, heap2->element_size);
        if (ptr == NULL) return(-1);

        heap2->heap_storage[++heap2->storage_count] = ptr;
        heap2->heap_total_size += heap2->heap_buffer_size;

        memcpy((heap2->heap_storage[heap2->storage_count] + heap2->element_size * (heap2-
        >heap_size % heap2->heap_buffer_size)), element, heap2->element_size);
        heap2->heap_size++;
    }

    return (heap2->heap_size - 1);
}


/*****************************************************************************/
/**
 * Create an element from the next unused address
 * Notes: if there is no enough space, additional storage block will be added
 * @param heap
 *
 * @return char*
 */
char* SMEM_create_element(char *heap1)
{
        int   alloc_notify, ind = 0;
        char  *ptr;
        ssHEAP *heap2;

        heap2 = (ssHEAP *)heap1;
        if (heap2 == NULL) return(NULL);
        if (heap2->element_size < 1) return(NULL);

        //Allocate the memory pool if the storage_count is negative
        if (heap2->storage_count < 0)
        {
                heap2->heap_storage[++heap2->storage_count]=(char*)calloc(heap2-
                >heap_buffer_size, heap2->element_size);
                heap2->heap_total_size += heap2->heap_buffer_size;
                alloc_notify = 0;
                heap2->heap_size++;
                if (heap2->heap_storage == NULL)
                        return(NULL);
```

```
            else
                return heap2->heap_storage[heap2->storage_count];
    }

    //If the assigned size is less than total allocated size, copy the data to the last available
    address
    if (heap2->heap_size < heap2->heap_total_size)
    {
        heap2->heap_size++;
        if((heap2->heap_size-1)%(heap2->heap_buffer_size)==0 && (heap2->heap_size-
        1)/(heap2->heap_buffer_size)!=0)
            heap2->storage_count++;
    }

    //If all the allocated memory block are assigned, allocated a new block and append is into
    //the storage stack
    else
    {
        if (heap2->storage_count == heap2->max_storage_count -1)
        {
            heap2->max_storage_count += MAX_STORAGE_COUNT;
            heap2->heap_storage = (char**) realloc(heap2->heap_storage, heap2-
            >max_storage_count * sizeof(char*));
            for(ind = heap2->storage_count+1;
                ind<heap2->max_storage_count;
                ind++)
            {
                heap2->heap_storage[ind] = NULL;
            }
        }
        ptr =  (char *)calloc(heap2->heap_buffer_size, heap2->element_size);
        if (ptr == NULL) return(NULL);

        heap2->storage_count++;
        heap2->heap_storage[heap2->storage_count] = ptr;
        heap2->heap_total_size += heap2->heap_buffer_size;

        heap2->heap_size++;

    }

    return(heap2->heap_storage[heap2->storage_count]+heap2->element_size
        *((heap2->heap_size - 1) % heap2->heap_buffer_size));
}
```

```
/*****************************************************************************/
/**
 * Frees heap storage only.
 *
 * @param heap1 heap
 *
*/
void SMEM_flush (char *heap1)
{
        ssHEAP *heap2;

        heap2 = (ssHEAP *)heap1;

        if (heap2 == NULL) return;
        SMEM_free(heap1);
}


/*****************************************************************************/
/**
 * Resets the number of elements stored on the heap to zero without freeing any memory.
 * NOTES --- so the allocated member can be reused
 * @param heap1 heap
 *
 */
void SMEM_reset (char *heap1)
{
        ssHEAP *heap2;

        heap2 = (ssHEAP *)heap1;
        if (heap2 == NULL) return;

        heap2->heap_size = 0;
        heap2->storage_count = 0;
}


/*****************************************************************************/
/**
 * Resets the number of elements stored on the heap to smaller value without freeing any
memory.
 *
 * NOTE: CAN NOT be used if heap element contains allocated entity!
 *
 * @param heap1 heap
```

```
 * @param count count
 *
 */
void SMEM_set_count (char *heap1, int count)
{
        ssHEAP *heap2;

        heap2 = (ssHEAP *)heap1;
        if (heap2 == NULL) return;

        if (heap2->heap_size >= count)
        {
                heap2->heap_size = count;
                heap2->storage_count = (int)(count/heap2->heap_buffer_size);
        }
}


/*****************************************************************************/
/**
 * Returns the number of elements in heap storage.
 *
 * @param heap1 heap
 *
 * @return number of elements on heap
 *
 */
int SMEM_get_count (char *heap1)
{
        ssHEAP *heap2;

        heap2 = (ssHEAP *)heap1;
        if (heap2 == NULL) return(0);

        return(heap2->heap_size);
}


/*****************************************************************************/
/**
 * Gets the last item in the heap, decrements the number of
 * items in heap and returns the number of items in heap or -1.
 *
 * Allows the heap to be used as a stack.
 *
```

```
 * @param heap1 heap
 * @param[out] element last element from heap
 *
 * @return number of elements on heap after operation is performed
 *
 */
int SMEM_get_last (char *heap1, char *element)
{
        ssHEAP *heap2;

        heap2 = (ssHEAP *)heap1;
        if (heap2 == NULL) return(-1);
        if (element == NULL)return(-1);

        if (heap2->heap_size <= 0)     return(-1);

        heap2->heap_size--;
        heap2->storage_count = heap2->heap_size == 0 ? 0 : (int)(heap2->heap_size/heap2-
        >heap_buffer_size);

        memcpy(element, (heap2->heap_storage[heap2->storage_count] + heap2->element_size
        * ((heap2->heap_size-1) % heap2->heap_buffer_size)), heap2->element_size);

        return(heap2->heap_size);
}


/*****************************************************************************/
/**
 * Gets the last item in the heap, decrements the number of
 * items in heap and returns the number of items in heap or -1.
 *
 * Allows the heap to be used as a stack.
 *
 * @param heap1 heap
 * @param[out] element last element from heap
 *
 * @return number of elements on heap after operation is performed
 *
 */
char * SMEM_retrieve_last (char *heap1)
{
        ssHEAP *heap2;
        char* element;
```

```
	heap2 = (ssHEAP *)heap1;
	if (heap2 == NULL) return( NULL );

	if (heap2->heap_size <= 0)	return(NULL);

	element = heap2->heap_storage[heap2->storage_count] + heap2->element_size *
	((heap2->heap_size - 1) % heap2->heap_buffer_size);

	return element;
}


/*****************************************************************************/
/**
 * Sets the memory allocation notify function.
 *
 * @param heap1 heap
 * @param func address of function to be called everytime heap
 *		storage changes.
 * @param appdata application data passed to caller via "func"
 *
 */
void SMEM_set_alloc_notify_func (char *heap1, void (*func)(int, char *), char *appdata)
{
	ssHEAP *heap2;

	heap2 = (ssHEAP *)heap1;
	if (heap2 == NULL) return;

	heap2->alloc_notify_func = func;
	heap2->alloc_notify_data = appdata;
}
```

Reference

[1] Xu, X; Klemm, P; Proctor, F; Suh. S H (September 2006). "STEP Compliant Process Planning and Manufacturing". International Journal of Computer Integrated Manufacturing, Vol.19, Issue 6, 2006, Special Issue: STEP-Compliant Process Planning and Manufacturing

[2] "ISO Products, Standards". ISO TC 213 Dimensional and geometrical product specifications and verification. Retrieved 2009-06-17

[3] "Product and Manufacturing Information (PMI) management". Siemens PLM Software. 2007-11-07.

[4] Anna Bellini, Selçuk Güçeri, "Mechanical characterization of parts fabricated using fused deposition modeling", Rapid Prototyping Journal, Vol. 9 Iss: 4, 2003, pp:252 – 264

[5] Agarwala, M.K.; Bandyopadhyay, A., Weeren, R. van, Safari, A., "FDC, rapid fabrication of structural components", American Ceramic Society Bulletin, Vol. 75, Journal Issue: 11, Nov 1996, pp: 60-65

[6] Chee Kai Chua; Kah Fai Leong; Chu Sing Lim, "Rapid prototyping: principles and applications", Singapore; New Jersey: World Scientific, ©2003.

[7] Lilli Manolis Sherman, "3D Printers lead growth of Rapid prototyping", Plastics Technology, Issue August 2004

[8] S.G. Bardenhagena, J.U. Brackbill, b, D. Sulskyc, "The material-point method for granular materials", Computer Methods in Applied Mechanics and Engineering, Vol. 187, Issues 3–4, 7, July 2000, pp:529–541

[9] Green Arthur W, "Industrial photoinitiators: a technical guide", CRC Press, Taylor & Francis Group, @2010 By Taylor and Francis Group, LLC

[10] Z. Zhanga, M. Sarhadib, "An integrated CAD/CAM system for automated composite manufacture", 12th International Conference on Computer Aided Production Engineering, Vol. 61, Issues 1–2, August 1996, pp: 104–109

[11] O.Felix Offodilea, Layek L Abdel-Malekb, "The virtual manufacturing paradigm: The impact of IT/IS outsourcing on manufacturing strategy", International Journal of Production Economics, Vo. 75, Issues 1–2, 10 January 2002, pp: 147–159

[12] Grant W. Lawless, "Information Technology (IT) For Manufacturing: Where Has It Been, Where Is It Heading?" Journal of Industrial Technology, Vol 16, Number 4, August 2000 to October 2000

[13] Mohsen Attaran, "Flexible manufacturing Systems Implementing an Automated Factory ", Information Systems Management, Vol. 9, Issue 2, 1992, pp: 44-47

[14] Yuri Merkuryev, Galina Merkuryeva, " Simulation-Based Case Studies in Logistics: Education and Applied Research" , Springer, ISBN978-1-84882-187-3

[15] Peter Kostal, Karol Velisek, " Flexible Manufacturing System" , World Academy of Science, Engineering and Technology, 77 2011, pages 825-829

[16] M Seki, T Takegahara, " Direct numerical control (DNC) system including one high-speed data processing unit for each NC machine tool", US Patent 5,388,051, 1995

[17] Lynch, Mike. "Running programs with direct numerical control.(CNC Tech Talk)." Modern Machine Shop. Gardner Publications Inc. 2005. HighBeam Research. 22 Jun. 2012

[18] Michael McClellan, "Introduction to Manufacturing Execution Systems", MES Conference and exposition, Phoenix Arizona, Jun12-14, 2000

[19] Greene, Christopher M; Jackson, Yasmin; Allen, Shakerha S; Pennington, James. "Manufacturing Execution Systems: A Practical View." IIE Annual Conference. Proceedings. Institute of Industrial Engineers, Inc. (IIE). 2009.

[20] Jürgen Kletti, "Manufacturing Execution Systems – MES", Berlin ; London : Springer, 2007.

[21] MOSIS, "University of Southern California's Information Science Ins.—The MOSIS VLSI Fabrication Service" http://www.isi.edu/mosis/, 2000

[22] Wright, P.K. and Sequin, C.H., "CyberCut: A networked manufacturing system," Proceedings of the managing Enterprise Conference, Loughborough University, England, July, pp605-614, 1997

[23] Y. Tu, X. Chu, and W. Yang, "Computer-aided process planning in virtual one-of-a-kind production", Comput, Ind., vol 41, pp99-110, 2000

[24] D.N.Sormaz and B. Khoshnesive, "Process planning knowledge reprsentation using an object-oriented data model", International Journal Computer Integration Manufacturing, vol. 10, no. 1-4, pp. 92-104, 1997

[25] Y. Zhang, S. C. Feng, X. Wang, W. Tian, and R. Wu, "Object oriented manufacturing resource modeling for adaptive process planning,", Int. J. Prod. Res., vol. 37, no. 18, pp. 4179–4195, 1999.

[26] N. Morad and A. Zalzala, "Genetic algorithms in integrated process planning and scheduling,", J. Intell. Manuf., vol. 6, pp. 169–179, 1999

[27] F. Zhang, Y. F. Zhang, and A. Y. C. Nee, "Using genetic algorithms in process planning for job shop machining,", IEEE Trans. Evol. Comput., vol. 1, no. 4, pp. 278–289, Nov. 1997.

[28] C. R. Devireddy and K. Ghosh, "Feature-based modeling and neural network-based CAPP for integrated manufacturing,", Int. J. Comput. Integr. Manuf., vol. 12, no. 1, pp. 61–74, 1999.

[29] L. Monostori, Z. J. Viharos, and S. Markos, "Satisfying various requirements in different levels and stages of machining using one general ANNbased process model" , J. Mater. Process. Technol., vol. 107, pp. 228–235, 2000.

[30] Y. J. Tseng and S. B. Joshi, "Recognizing multiple interpretations of interacting machining features" , Comput.-Aided Des., vol. 26, no. 9, pp. 667–688, 1994.

[31] L. Wang and D. H. Norrie, "Process planning and control in a holonic manufacturing environment", J. Appl. Syst. Stud., vol. 2, no. 1, pp. 106–126, 2001.

[32] G. C. Vosniakos and B. J. Davies, "Knowledge-based selection and sequencing of hole-making operations for prismatic parts", Int. J. Adv. Manuf. Technol., vol. 8, pp. 9–16, 1993.

[33] J. A. Stori and P. K. Wright, "A knowledge-based system for machining operation planning in feature based, open-architecture manufacturing", presented at the ASME Design Technical Conf., Irvine, CA, 1996.

[34] F. L. Zhao and P. S. Y. Wu, "A cooperative framework for process planning", Int. J. Comput. Integr. Manuf., vol. 12, no. 2, 1999, pp: 168–178,

[35] W. Shih and K. Srihari, "Distributed artificial intelligence in manufacturing systems control", Comput. Ind. Eng., vol. 29, no. 1–4, pp. 199–203, 1995.

[36] F. L. Zhao, S. K. Tso, and P. S. Y. Wu, "A cooperative agent modeling approach for process is planning", Comput. Ind., vol. 41, no. 1, pp. 83–97, 2000.

[37] A. Sluga, P. Butala, and G. Bervar, "A multi-agent approach to process planning and fabrication in distributed manufacturing", Computer. Ind. Eng., vol. 35, no. 3–4, pp. 455–460, 1998.

[38] Yusri Yusof , "Exploring the ISO14649 (STEP-NC) for Intelligent Manufacturing System", European Journal of Scientific Research, Vol. 36 No.3(2009), pp 445-457

[39] Suk-Hwan Suh, Jung-Hoon Cho & Hee-Dong Hong, "On the architecture of intelligent STEP-compliant CNC", International Journal of Computer Integrated Manufacturing, Volume 15, Issue 2, 2002

[40] Wonseok Lee and Young-Bong Bang, "Development of ISO14649 Compliant CNC Milling Machine Operated by STEP-NC in XML Format", International Journal of the KSPE Vol. 4, No. 5.

[41] Han, Junghyun Hyun, "Manufacturing feature recognition from solid models: a status report", Robotics and Automation, IEEE Transactions on, Vol. 16, Issue 6, Pages 782 – 796

[42] Martín G. Marchettaa, Raymundo Q. Forradellasa "An artificial intelligence planning approach to manufacturing feature recognition", Computer-Aided Design, Vol. 42, Issue 3, March 2010, Pages 248–256

[43] G.V.V. Ravi Kumar, Prabha Srinivasan, K.G. Shastry, B.G. Prakash, "Geometry based triangulation of multiple trimmed NURBS surfaces", Computer-Aided Design, Volume 33, Issue 6, May 2001, Pages 439–454

[44] Nathan Litkea, Adi Levinb, Peter Schrödera, "Trimming for subdivision surfaces", Computer Aided Geometric Design, Volume 18, Issue 5, June 2001, Pages 463–481

[45] W. Chung, A.C. Lin and W. F. Lu, "Automated generation of machining data for NURBS-based sculptured surfaces", ASME Proceedings of the Japan, USA Symposium on Flexible automation, San Francisco, CA, pp. 401-408, 1992

[46] Xiao-ming, Ma, "Applied Skill of Coons Surface in MasterCAM", Machine Building and Automation, Vol 02, 2007.

[47] Yong-Jin Liu, "Exact geodesic metric in 2-manifold triangle meshes using edge-based data structures", Computer-Aided Design, Volume 45, Issue 3, March 2013, Pages 695–704

[48] Daniel Sieger, Mario Botsch, "Design, Implementation, and Evaluation of the Surface_mesh Data Structure", Proceedings of the 20th International Meshing Roundtable, 2012, pp 533-550

[49] Ramakrishnan Mukundan, "Mesh Processing", Advanced Methods in Computer Graphics, 2012, pp 179-229

[50] Leonidas J. Guibas and Jorge Stolfi, "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams", ACM Transactions on Graphics, 4(2), 1985, 75–123

[51] Bruce G. Baumgart. “Winged Edge Polyhedron Representation”, Technical Report. Stanford University, Stanford, CA, USA. 1972

[52] K. Weiler, "The Radial-Edge Structure: A Topological Representation for Non-Manifold Geometric Boundary Representations", Geometric Modeling for CAD Applications, North Holland, pp. 3-36, 1988.

[53] L. Kettner. “Using generic programming for designing a data structure for polyhedral surfaces”, Computer. Geom. Theory Appl., 13:65–90, 1999.

[54] K. Weiler, “Topological Structures for Geometric Modeling”, PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, August 1986.

[55] Walls, Keith G. "Method for improving the performance of dynamic memory allocation by removing small memory fragments from the memory pool", U.S. Patent No. 5,675,790. 7 Oct. 1997.

[56] Jung-hyun Yoo, Sung-min Lee, and Sang-bum Suh. "Management of memory pool in virtualization environment”, U.S. Patent No. 20,120,221,765. 30 Aug. 2012.

[57] Dahlin, Michael D., and Stephen W. Keckler. "Memory management for high-performance applications", Doctoral Dissertation, 2002, The University of Texas at Austin

[58] Martin Held, “On the computational geometry of pocket machining (Lecture Notes in Computer Science)”, Berlin: Spring, 1991 LNCS500

[59] Tao Chen, Peiqing Ye, “ A tool path generation strategy for sculptured surfaces machining”, Journal of Materials Processing Technology, vol 127(2002) , PP 369-373

[60] Daniel C.H. Yang, Oulee. T. H, “Boundary Conformed Toolpath Generation via Laplace Based Parametric Redistribution Method”, Journal of Manufacturing Science and Engineering, Vlo126, May 2044, pp 345-354

[61] D.C.H. Yang, T. Kong, “Parametric interpolator versus linear interpolator for precision CNC machining”, Computer Aided Design 26 3 (1994), pp. 225–234

[62] M. Held, “A geometry-based investigation of the tool path generation for zigzag pocket machining”, The Visual Computer, 1991, Volume 7, Issue 5-6, pp 296-308

[63] Shaobin Tao, Kwun-lon Ting, “Unified rough cutting tool path generation for sculptured surface machining”, International Journal Production Research, 2001, Vol. 39, No. 13, 2973-2989

[64] Sang C. Park, Yun C. Chung, “Tool-path generation from measured data”, Computer-Aided Design, Volume 35, Issue 5, 15 April 2003, Pages 467-475

[65] A.C. Lin, R. Gian, “A Multiple-Tool Approach to Rough Machining of Sculptured Surfaces”, International Journal of Advanced Manufacturing Technology, 1999, Volume 15, pp 387-398

[66] Y.S. Huang, P.D.Webster and T.A.Dean, “An image detection approach to NC rough-cut milling from solid models”, International Journal of Machining Tools Manufacturing, Vol. 36 1996, No.12 pp1321-1333

[67] Douglas DeCarlo and Jean Gallier, “Topological Evolution of Surfaces”, In Graphics Interface, 1996, pp. 194-203.

[68] Kenneth I. Joy, “Utilizing Parametric Hyperpatch Methods for Modeling and Display of Free form solids”, Proceedings of the Symposium on Solid Modeling Foundations and CAD/CAM Applications, 1991, pp 455-472

[69] Kenneth I. Joy, "Visualization of Swept Hyperpatch Solids", in "Proceedings of the Computer Graphics International 1992", pp 567-582, 1992

[70] Daniel C.H. Yang, J. J. Chuang, and T.H. Oulee, "Boundary conformed toolpath generation for trimmed free-form surfaces", Computer aided design, Vol 35 2003, pp 127-139

[71] C. Lartigue, F. Thiebaut and T. Maekawa, "CNC tool path in terms of B-spline curves, Computer-Aided Design", Volume 33, Issue 4, 2 April 2001, Pages 307-319

[72] J. U. Brackbill and J.S. Saltzman, "Adaptive zoning for singular problems in two dimension", J.Comp. Physics, Vol.46, No.3, June 1982.

[73] Les A. Piegl, "The NURBS Book (Monographs in Visual Communication)", 2nd Edition, Springer

[74] Richard H. Bartels, John C. Beatty and Brian A. Barsky, "An introduction to Splines for use in computer graphics and geometric modeling", Morgen Kaufmann publishers, ISBN 0-934613-27-3

[75] Balasubramaniam Mahadevan, "Tool selection and path planning for 3 axis rough machining,", Master Thesis MIT, 1997

[76] K.L. Chui, W.K. Chiu , K.M. Yu, "Direct 5-axis tool-path generation from point cloud input using 3D bi-arc fitting", Robotics and Computer-Integrated Manufacturing, Vol. 24, Issue 2, April 2008, pp 270–286

[77] Warren, MI, Marin, S.P. "Feature-Based Surface Design and Machining", IEEE Computer Graphics and Applications, September, Vol. 12 1992, Issue: 5, pp 61 – 68

[78] His-Yung Feng, Huiwen Li, "Constant scallop-height tool path generation for three axis sculptured surface machining", Computer aided design Vol. 34, 2002, pp: 647-654

[79] S. Marshall, J.G. Griffiths, "A survey of cutter path construction techniques for milling machines", International Journal of Production Research, Vol. 32(12), pp: 2861-2877

[80] Persson H., "NC machining of arbitrary shaped pockets", Computer-Aided Design, Vol 10(3), pp: 169-174

[81] Eun-Young Heoa, Dong-Won Kimb, Jong-Young Leeb, Cheol-Soo Leec, F. Frank Chend, "High speed pocket milling planning by feature-based machining area partitioning", Robotics and Computer-Integrated Manufacturing, Volume 27, Issue 4, August 2011, pp: 706–713

[82] C. Lartigue, E. Duc and A. Affouard, "Tool path deformation in 5-axis flank milling using envelope surface", Computer-Aided Design, Vol 35, 2003, pp. 375-382

[83] Ahmet Can, Ali Ünüvar, "A novel iso-scallop tool-path generation for efficient five-axis machining of free-form surfaces", The International Journal of Advanced Manufacturing Technology, Vol. 51, Numbers 9-12, 2010, pp: 1083-1098

[84] Zhonglin Han, D. C. H. Yang and Jui-Jen Chuang, "Isophote-based ruled surface approximation of free-form surfaces and its application in NC machining" , International Journal of Production Research, Volume 39, Issue 9, 2001, pp: 1911-1930

[85] Y. N. Hu, Y. H. Chen, "Implementation of a Robot System for sculptured Surface Cutting. Part 1. Rough Machining", The International Journal of Advanced Manufacturing Technology, Vol. 15, Number 9 (1999), pp: 624-629

[86] Y. N. Hu, Y. H. Chen, "Implementation of a Robot System for sculptured Surface Cutting. Part 2. finish Machining", The International Journal of Advanced Manufacturing Technology, Vol. 15, Number 9, 1999, pp: 630-639

[87] Ganping Suna, Carlo H. Sequinb and Paul K. Wrighta, "Operation decomposition for freeform surface features in process", Computer-Aided Design, Vol. 33, Issue 9, August 2001, pps 621–636

[88] M. Balasubramaniama, P. Laxmiprasada and S. Sarmaa, Z. Shaikhb, "Generating 5-axis NC roughing paths directly from a tessellated representation", Computer-Aided Design, Vol. 32, Issue 4, 1 April 2000, pp: 261–277

[89] M. Bala, T.C. Chang, "Automatic cutting selection and optimal cutter path generation for prismatic parts", ASME Winter Annual Meeting, San Francisco, CA, pp. 57-68, 1988

[90] Jianhong Zhao, Masanori Kunieda, Guilin Yang and Xue-Ming Yuan, "Tool Path Planning Assist System for Freeform Surface Machining", Key Engineering Materials, Vol. 447 – 448, 2010, pp 321-325

[91] Liancheng Zhao, Haowei Wang and Changfa Xiao, "Research of Spiral Tool Path Generation Based on Point Cloud", Vol. 538 – 541, 2012, pp: 1308-1311

[92] Gerson Elber, E. Cohen, "Tool Path Generation for Freeform Surface Models" , Proceeding of 2nd ACM Solid Modeling 93-5/93/Monreal Cananda

[93] Michael S. Floater, Kai Hormann, "Surface Parameterization: a Tutorial and Survey", Advances in Multiresolution for Geometric Modelling, Mathematics and Visualization 2005, pp 157-186

[94] N. Pietroni, "Almost Isometric Mesh Parameterization through Abstract Domains", Visualization and Computer Graphics, IEEE Transactions on, Vol. 16, 2010, pp: 621 – 635

[95] Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman and Steven J. Gortler, "A local/Global Approach to Mesh Parameterization", Computer Graphics Forum, Vol. 27, Issue 5, July 2008, pp: 1495–1504,

[96] W. Tutte, "Convex representation of graphs", Proc. London Math. Soc., Vol 10 1960, pp: 305-320

[97] M. S. Floater. “Parameterization and smooth approximation of surface triangulation”, computer Aided Geometric Design, Vol. 14:231-250, 1997

[98] Levy and J.L Mallet, “Non-distorted texture mapping for sheared triangulated meshes”, ACM SIGGRAPH conference Proceedings, pp. 343.-352, 1998

[99] J.-J.Chuang, T.H.Oulee and D.C.H. Yang, “Boundary-conformed toolpath generation for trimmed free-form surface”, Computer aided design 35(2003) 127-139

[100] http://www.cimatron.com/general.aspx?FolderID=2683, last accessed on March 26th, 2013

[101] http://m.plm.automation.siemens.com/en_us/products/velocity/camexpress/5_axis.shtml, last accessed on March 26th , 2013

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

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

[104] J.H. Oliver, “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.

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

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

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

[108] Mounayri H.E.I., Spence A.D. and 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.

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

[110] 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.

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

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