6 G-buffer Application for NC Machining of 3D Models

The G-buffer method is applied to NC machining. A total NC system is created that consists of all essential functions, such as tool path generation, path verification, and feed rate control. Moreover, any combination of object surface and tool shape is acceptable. By utilizing G-buffers created from a parallel projection, the required NC functions are realized as image processing operations. This ensures that the NC software is independent from surface description. Conventional rendering software can be used to make the G-buffers. Any surface can be milled if it can be rendered by parallel projection. Tool shape changes can be easily handled by changing the image processing filters. 3D examples of geometric surfaces, mesh data, and volume data are milled with this method, and the results show that the method is very effective.

6.1 Introduction

While the thrust of modern computer graphics is the visualization of the real world, people must use actual 3D objects to survive. One interface between conceptual and actual objects is that created by CAD/CAM systems coupled to numerically controlled (NC) milling machines. Unfortunately, this interface is not as efficient or productive as it should be. The problem lies in the large number of factors that must be considered when machining a complex object.

CAD/CAM systems were initially used to produce relatively simple objects whose profiles could be described as combination of geometric primitives. The production of more complex objects demands the creation of highly sophisticated tool paths. This is due to the following factors.

1. Shape accuracy: Over cutting must be prevented and the degree of under-cutting accurately evaluated. This requires the interference* between tool and required surface to be constantly monitored.
2. Tool load: Cutting rate must not exceed the limits of tool, milling machine, or workpiece.
3. Tool movements: Unplanned collisions* between workpiece and tool or tool holder must be prevented.
4. Machining time: Tool selection, tool path, and feed rate should be optimized.
5. Variations in tool types and surface shapes: The tool path must accommodate the available tools and all possible workpiece shapes.

A number of sophisticated methods have been developed to address one or more of these factors, and some have turned into commercial systems. Each method can be characterized as one of two types: tool path generation, or machining simulation/verification. In path generation, the main purpose is to obtain an adequate tool path that produces an accurate shape. The tool path can be obtained from the offset surface, i.e. the trace of the limit position of the tool center, and a lot of research has been performed to calculate an accurate offset surface[Zhang86], [Sakuta87]. Kishinami *et al.* have proposed a flexible algorithm called the Inverse Offset Method [Kishinami87] which uses the rasterization technique. In simulation and verification, the tool path is verified for each factor listed above. Many systems have been developed for this purpose, and one of the major differences among them is the shape representation for interference calculation. Wang *et al.* [Wang86a], [Wang86b], Hook [Hook86], and Atherton *et al.* [Atherton87] all used a projection from a view point and applied a variation of the z-buffer algorithm. Thus, their methods are termed 'view based methods'. Kawashima *et al.*. [Kawashima89] implemented such a method by using an oct-tree data structure. Chappel [Chappel83] and Jerard *et al.*. [Drysdale87], [Jerard89a],

---

* In this chapter, the words interference and collision are used as the following meanings.

interference: Relation between tool and required surface. It is a measure of over-cutting.

collision: Contact of workpiece and upper part (without milling edge) of a tool. It damages the tool and/or the workpiece.

[Jerard89b] represented the shape with 'direction vectors' from discrete points distributed on the object's surfaces.

Unfortunately, in conventional CAD/CAM systems, these two activities, tool path generation and simulation/verification, are usually independent of each other and based on different methodologies. This has two disadvantages. First, the entire software package is huge and excessively complicated. This is because different programs are required for each activity in order to accommodate various shapes and tool types. Second, it is difficult to generate tool paths by using simulation results. This function is necessary because, for example, adequacy of a tool path for fine cutting depends on the result of rough cutting.

To build a total NC program on one methodology, we extend the G-buffer2 method [Saito90] to 3 axis NC machining. A G-buffer (Geometric Buffer) is a 2 dimensional array, like an image. Each G-buffer contains one geometric property for all pixels such as depth, surface normal, etc. By manipulating G-buffers, various kinds of rendering techniques can be realized with image processing operations. Moreover, G-buffers generated from a parallel projection can also easily realize the many functions required for a total NC system by employing image processing operations. For example, it is easy to generate tool paths based on collision as well as interference. Many kinds of scanning operations are also available by referring to G-buffers. Tool paths can be simulated and evaluated with a series of image processing sequences. The optimum tool path and feed rate can be chosen from the evaluation results.

Some functions of the proposed method are identical or similar to previously proposed functions. The offset surface generation algorithm is equivalent to Kishinami's Inverse Offset method [Kishinami87]. The simulation and verification methods can be regarded as simplifications of the conventional view based methods. However, these functions are very synergistic when implemented in a total G-buffer machining system.

One of the notable advantages of the proposed method is that all parts of the NC software are independent of surface description. In order to make the G-buffers, conventional rendering software can be used. This means that any surface can be milled if it can be rendered by parallel projection. Various tool shapes can be employed by simply changing the image processing filters. Dedicated graphics or image processing hardware, such as graphics workstations, can effectively perform all required

computations.

6.2 G-buffers for NC Machining

6.2.1 Concept of G-buffer Method

The G-buffer method [Saito90] was originally developed for comprehensible rendering, and has the two following features:

- The geometric properties for each pixel of the visible object are preserved in G-buffers;
- Various rendering techniques are realized with image processing operations.

By using G-buffers as the intermediate result, geometric procedures (such as scan conversion and hidden surface removal) and other procedures (such as shading, texture mapping, and enhancement) are completely separated. Therefore, they can be performed independently and efficiently.

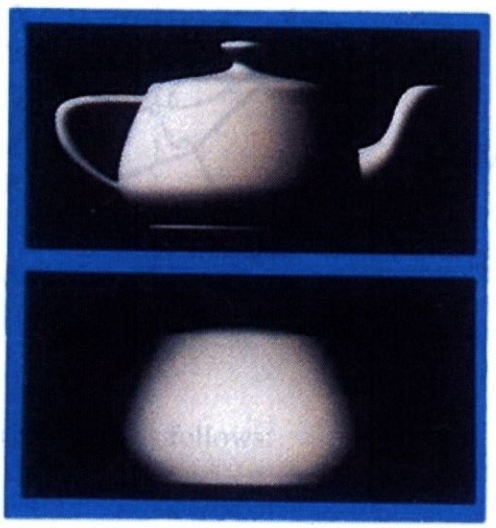
6.2.2 G-buffer Set Required for NC

The concept of the G-buffer method can be effectively extended to 3 axis NC machining. By preparing a G-buffer set from a parallel projection, it is possible to realize many kinds of NC software procedures as image processing operations. For NC machining, the following geometric properties are the typical contents of a set of G-buffers.

- **Z**: world coordinate z (depth)
- **nx**: normal vector x
- **ny**: normal vector y
- **nz**: normal vector z
- **id**: object/patch identifier
- **ou**: patch coordinate u
- **ov**: patch coordinate v

Only **Z** in this list is indispensable; the use of the other G-buffers depends on the procedures required for machining or verification. Figure 6.1 shows a shaded image and a **Z**-image of the famous Utah teapot.

In this chapter, all G-buffers and derived 2 dimensional arrays are called images. Images whose names begin with an upper case character contain absolute or relative height data. The unit height is equal to the pixel interval.



(a) shaded image

(b) depth image (**Z**)

Fig.6.1 Shaded and depth images of the Utah Teapot.

6.3 Image Processing for NC Functions

6.3.1 Basic Tool Path Generation

Once G-buffers of a required object are generated, an offset surface between the object and a tool can be obtained from the **Z**-image with the following operation. Let $\mathbf{L}_I$ be the height field of offset surface, and $h(d)$ be the height of the tool-end at the distance d from the tool axis, where $h(0) = 0$. When the tool-end touches the required surface at (x_t,y_t), the height of the tool center must satisfy the following relation (Fig.6.2):

$$\mathbf{L}_I(x, y) = \mathbf{Z}(x_t, y_t) - h(\sqrt{(x - x_t)^2 + (y - y_t)^2}). \qquad (6.1)$$

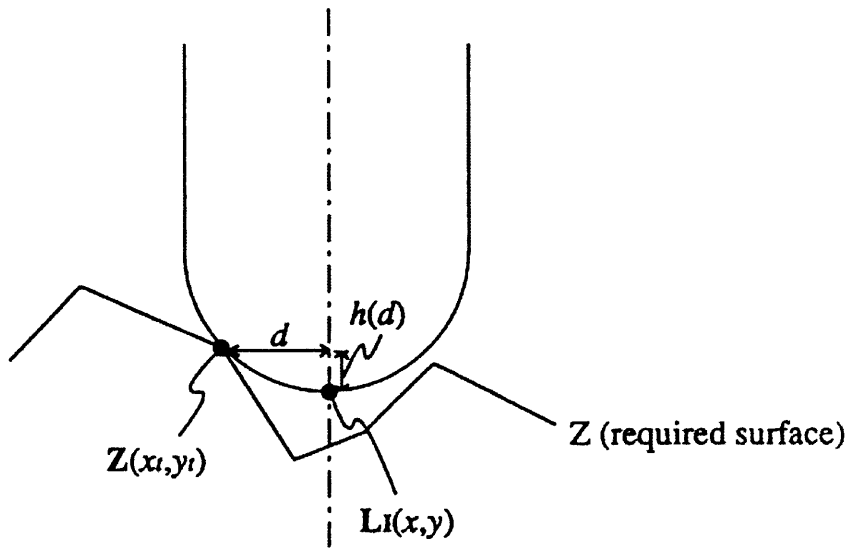


Fig.6.2 Obtaining an offset surface.

Therefore, $\mathbf{L}_I$ -image is obtained as follows:

$$\mathbf{L}_I(x, y) = \max_{i,j} (\mathbf{Z}(x + i, y + j) - H(i, j)) \qquad (6.2)$$

$$(i^2 + j^2 < r^2),$$

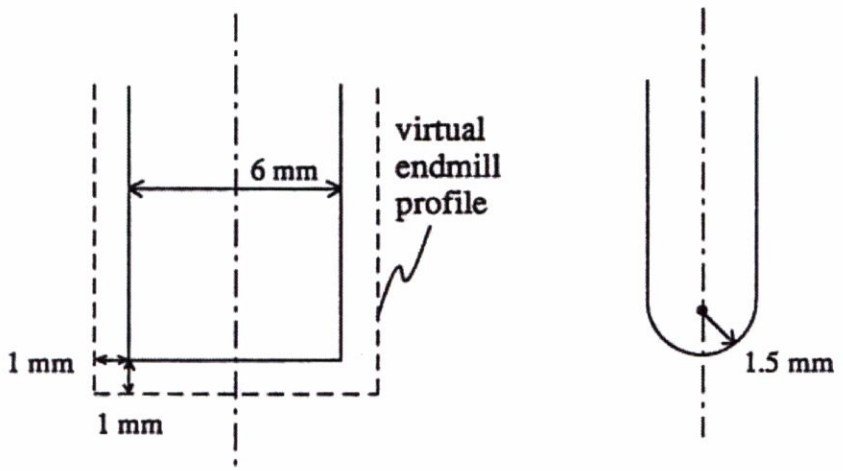
where H is the height field of the tool-end shape, *i.e.*

$$H(i, j) = h(\sqrt{i^2 + j^2}), \qquad (6.3)$$

and r is the radius of the tool. *Equation* (6.2) is equivalent to the Inverse Offset Method [Kishinami87].

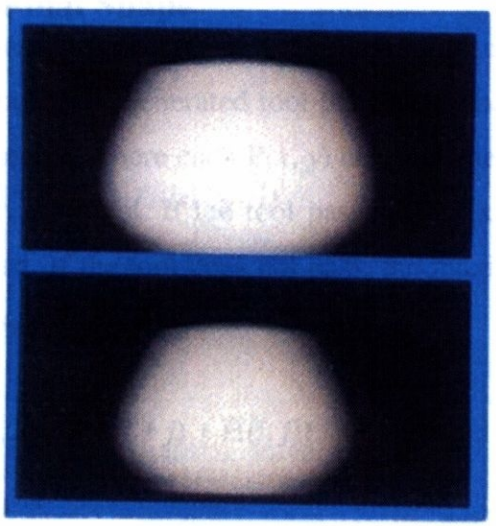
Figure 6.3 shows two examples of tool shapes; (a) is a flat endmill and (b) is a ball endmill. In some cases, especially for rough cutting, some amount of under-cutting is required everywhere. For this requirement, the virtual endmill profile (a) is used. Figure 6.4 shows the offset surfaces when the tools shown in Fig.6.3 are used to produce the teapot shown in Fig.6.1. Here, higher intensity (white region) means larger z values. In this example, the image size is 400×200 pixels, and the pixel interval is $0.2mm$ (i.e. the

simulation size is $80mm \times 40mm$).



(a) Flat Endmill with Under-cutting (b) Ball Endmill

Fig.6.3 Example tool shapes.



(a) flat endlnill (R = $6mm$) with under-cutting ($1mm$)

(b) ball endmill (ϕ = $1.5mm$)

Fig.6.4 Offset surfaces ($\mathbf{L}_I$).

Basic tool paths are obtained by scanning $\mathbf{L}_I$. In this stage, various kinds of tool paths are possible by referring to the G-buffers. The following paths are typical examples:

- *x- or y-scanning:*
  simply scanning on the x or y axis using an appropriate interval;
- *contour tracking:*
  tracking a constant z value in the $\mathbf{L}_I$-image;
- *u- or v-scanning:*
  tracking a constant u or v value in the **ou**- or **ov**-image.

The tool scanning strategy can be chosen independently for each surface of the object if the **id**-image is referred to.

6.3.2 Tool Path Verification and Evaluation

6.3.2.1 Accuracy of a Final Shape

This section introduces a verification and evaluation method for over- and under-cutting. For this purpose, the generated tool path has to be converted to a height field and preserved in a **P**-image, where each $\mathbf{P}(x, y)$ contains the height of the tool path when the tool goes through the pixel. If the tool passes the pixel more than once, the minimum height is preserved. If the tool does not pass, the maximum z-value is stored. The shape of the milling result **F** is calculated as follows:

$$\mathbf{F}(x, y) = \min_{i,j} (\mathbf{R}(x, y), \mathbf{P}(x + i, y + j) + \mathbf{H}(i, j)) \qquad (6.4)$$

$$(i^2 + j^2 < r^2),$$

where **R** is the shape of the workpiece before milling, which is either the initial shape of the workpiece or the previous milling result **F**. The relations among **Z**-, $\mathbf{L}_I$-, **R**-, **P**-, and **F**-images are shown in Fig.6.5.

By comparing **F** with **Z**, the over/under-cutting amount **D** is obtained for each pixel:

$$\mathbf{D}(x, y) = \mathbf{F}(x, y) - \mathbf{Z}(x, y), \qquad (6.5)$$

where a negative $\mathbf{D}(x, y)$ value indicates over-cutting. Note that this verification is possible for tool paths generated by any method. If they are generated by the method given in Section **6.3.1**, it is obvious that there is no over-cutting.

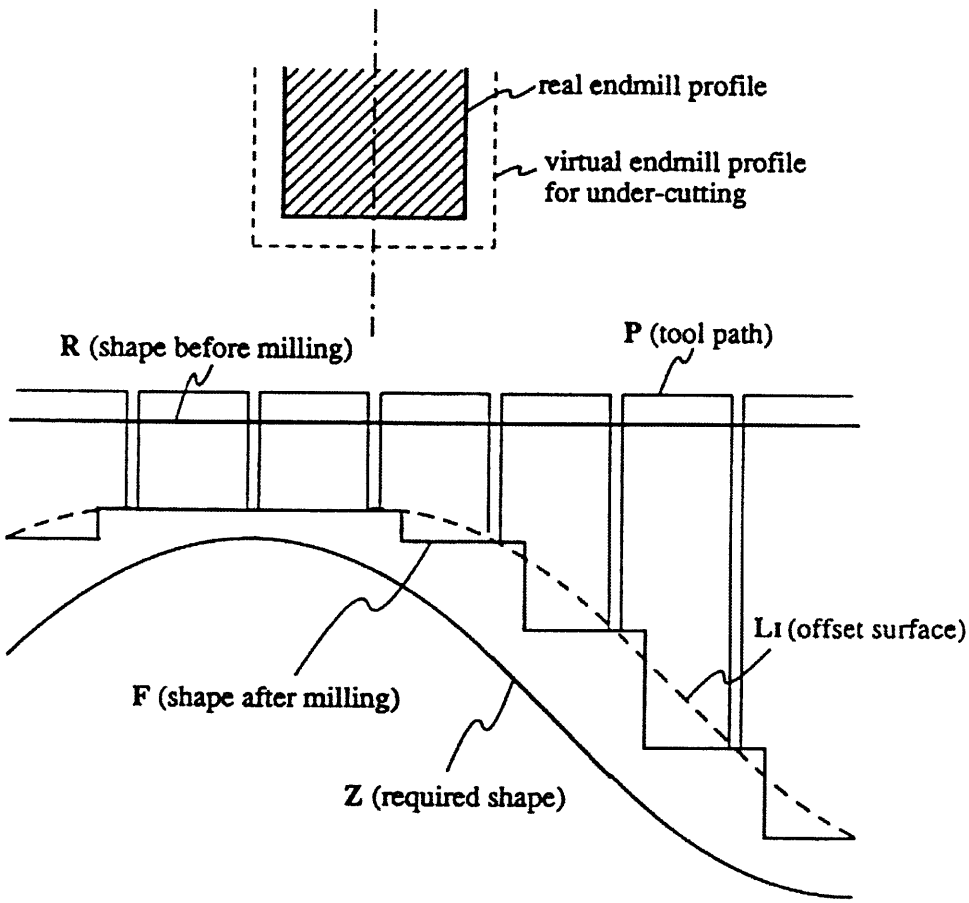
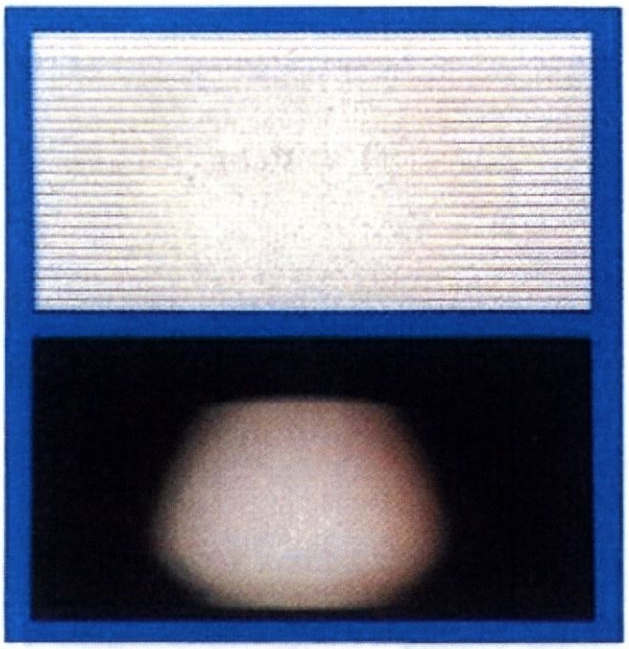


Fig.6.5 Tool path generation and verification.
An example of rough cutting process is described.
The scanning direction of the tool is perpendicular to this chapter.

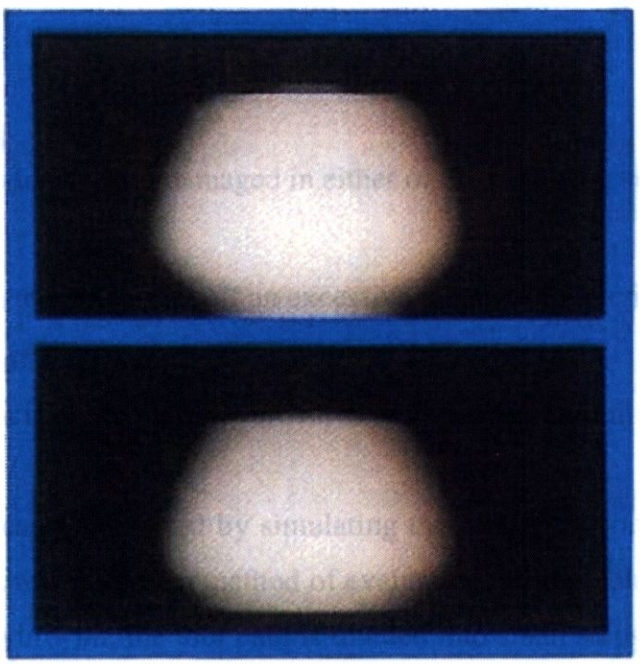
Example images from the teapot data are shown in Figs.6.6-6.8. Figure 6.6 is the tool path images (**P**). The path interval for rough cutting is $3mm$, while that for fine cutting is the same as the pixel interval ($0.2mm$). The cutting simulation results (**F**-images) are presented in Fig.6.7. These results can be evaluated by calculating the under-cutting amount (**D**-images) shown in Fig.6.8.



(a) rough cutting with the flat endmill

(b) fine cutting with the ball endmill

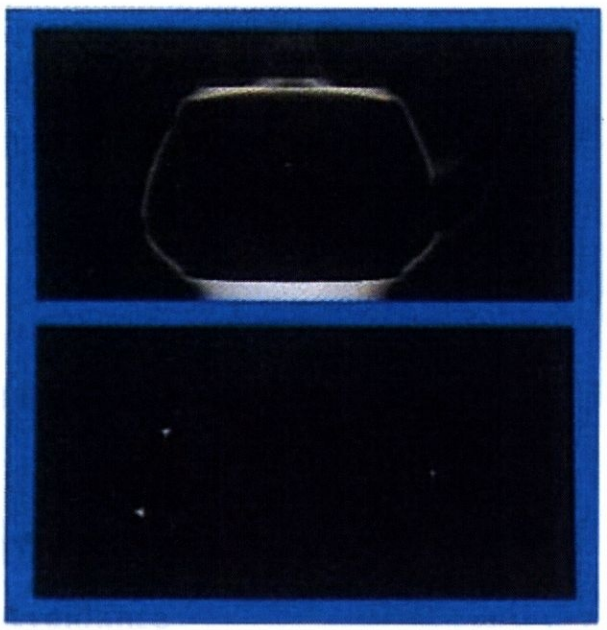
Fig.6.6 Path images (**P**).



(a) rough cutting with the flat endmill

(b) fine cutting with the ball endmill

Fig.6.7 Cutting simulation results (**F**).



(a) rough cutting with the flat endrnill (max: 17.47*mm*)

(b) fine cutting with the ball endmill (max: 2.99*mm*)

Fig.6.8 Under-cutting amount (**D**).

6.3.2.2 Tool Load

A tool can be overloaded and damaged in either of the following situations.

- The machine attempts to mill off an excessive amount of material.
- Milling direction is illegal.

  One example is to try to make a vertical hole with a flat endmill.

These situations can be detected by simulating the milling operations step by step using the **R**-image. In this section, a method of evaluating milling volume is introduced as an example. To begin with, the tool path is divided into small steps so that each step consists of only one pixel movement or vertical tool setting. Let the location of the tool at the s-th step be (x_s, y_s, z_s), and the intermediate resulting shape at that time be $\mathbf{F}_s$. Then, $\mathbf{F}_s$ and the cutting volume c_s are obtained as follows:

$$\mathbf{F}_s(x,y) = \begin{cases} \mathbf{F}_{s-1}(x, y) & (\text{if } (x - x_s)^2 + (y - y_s)^2 \geqq r^2) \\ \min(\mathbf{F}_{s-1}(x, y), zs + H(x - x_s, y - y_s)) & (\text{if } (x - x_s)^2 + (y - y_s)^2 < r^2), \end{cases} \quad (6.6)$$

$$cs = \sum_{x,\,y} (\mathbf{F}_{s-1}(x,y) - \mathbf{F}_s(x,y)). \quad (6.7)$$

Note that the initial shape $\mathbf{F}_0$ is **R**, and the final shape is **F**.

6.3.3 Advanced Tool Path Generation

6.3.3.1 Collision Avoidance

Collision avoidance between the workpiece and tool or tool holder is important in path generation. In Fig.6.9, for example, the length of the milling edge, h_1, restricts the maximum depth of lateral side milling. The depth of the tool must be limited to a safe z value during each scanning path. In this case, deep regions must be scanned many times while steadily increasing the depth. This method is effective for the initial rough cutting process since the shape of the workpiece is flat. However, it is inefficient for later cutting processes, and efficient tool paths for such cases must take the constraints of the tool or tool holder into account.

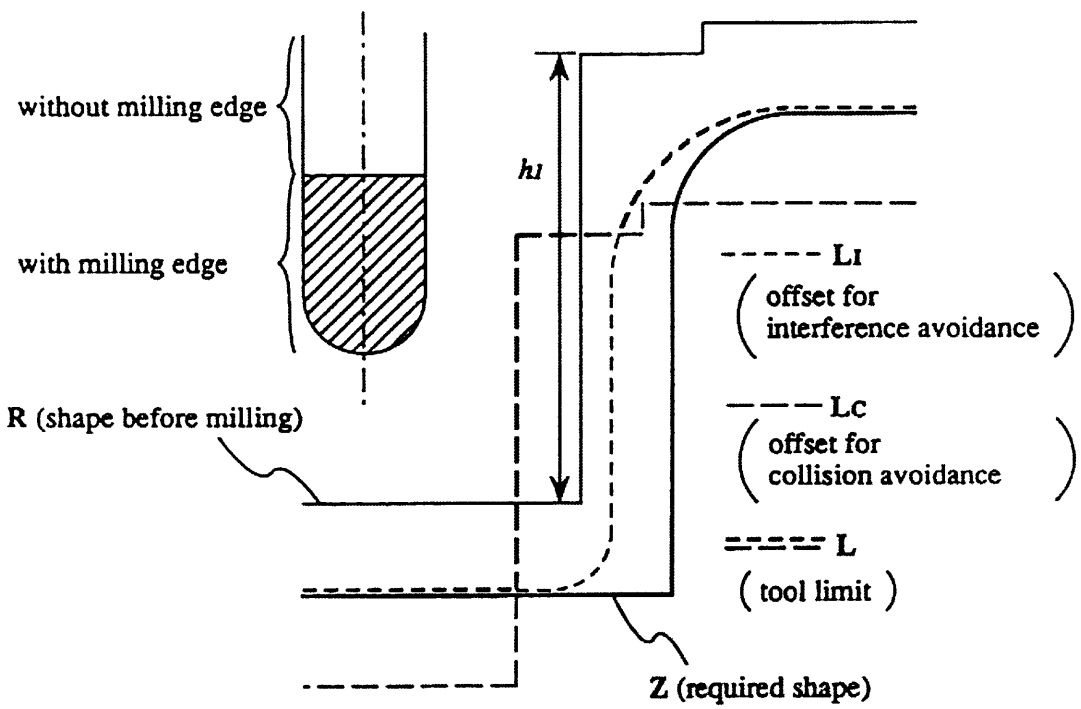


Fig.6.9 Tool path with collision avoidance.

This problem is simply solved with image processing operations. Let H_c be a height field of the upper part of the tool (without milling edge), and let $\mathbf{L}_c$ be the offset surface between **R** and H_c, *i. e.*

$$\mathbf{L}_c(x, y) = \max_{i,j} (\mathbf{R}(x + i, y + j) - H_c(i, j)) \quad (6.8)$$

$(i^2 + j^2 < r^2)$.

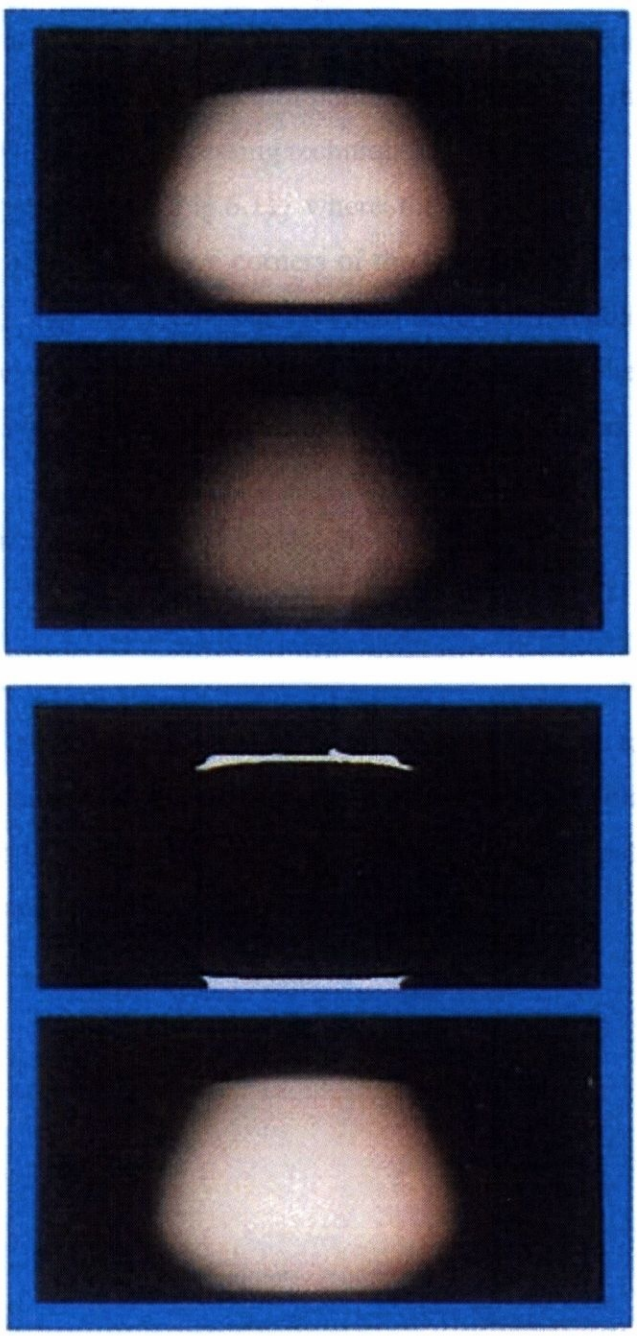
$\mathbf{L}_c$ is the limit surface to avoid tool collision. The tool can move above both offset surfaces $\mathbf{L}_I$ and $\mathbf{L}_c$ (Fig.6.9). Therefore, the tool limit surface L is defined as follows:

$$\mathbf{L}(x, y) = \max(\mathbf{L}_I(x, y), \mathbf{L}_c(x, y)). \quad (6.9)$$

After milling along surface **L**, **R** is changed and deeper milling may be possible in the next stage. This can also be simulated easily.

Figure 6.10 shows an example using the teapot. For fine cutting after rough cutting,

$\mathbf{L}_I$- and $\mathbf{L}_c$-images are calculated. If the tool path is generated from just the $\mathbf{L}_I$-image, collision occurs in the white region displayed in (c).



(a) offset surface for interference avoidance ($\mathbf{L}_I$)

(b) offset surface for collision avoidance ($\mathbf{L}_c$)

(c) difference between (a) and (b) ($\mathbf{L}_I < \mathbf{L}_c$)

(d) tool limit surface ($\mathbf{L}$)

Fig.6.10 Tool path generation with collision avoidance.

6.3.3.2 Final Cutting for Concave Regions

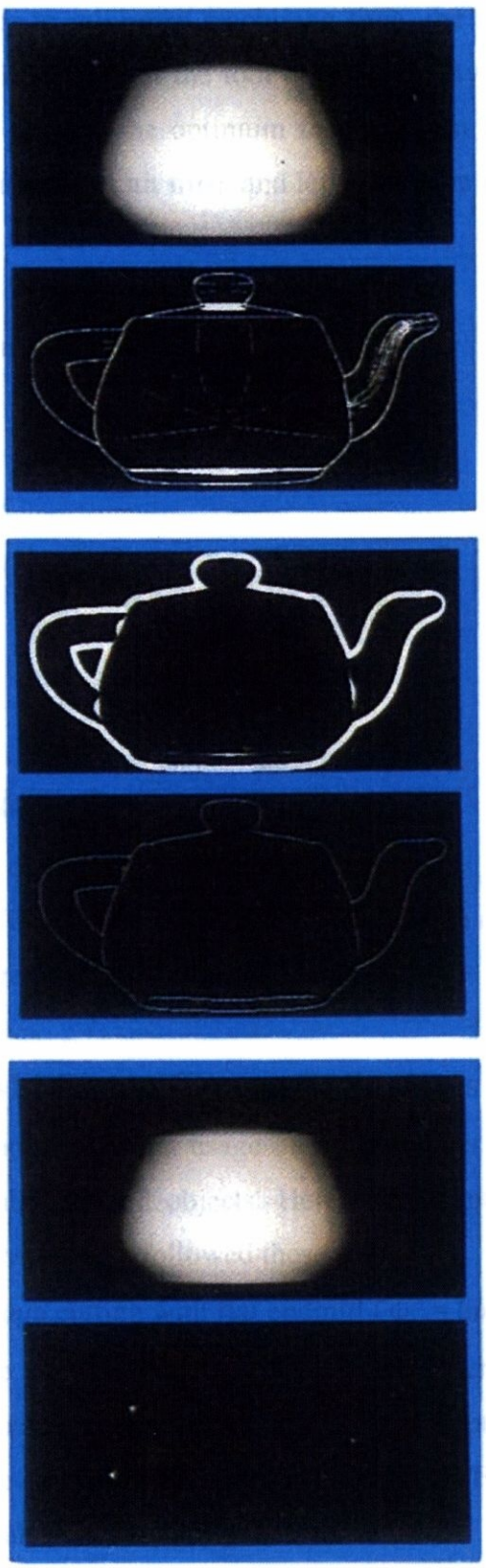
To minimize under-cutting in concave regions, the tool appropriate for the concave shape must be selected. In this case, the tool path should trace only the essential points. This is also possible with image processing techniques.

An example is presented in Fig.6.11, where the tool path for a flat endmill is generated in order to mill the concave corners of the teapot after fine cutting. First, the offset surface $\mathbf{L}_I$ (a) and its Laplacian (second order differential) image (b) are calculated. A positive value in the Laplacian image means that the offset surface is concave at that point. Next, static cutting volume **c** (c) is calculated as follows:

$$\mathbf{c}(x, y) = \sum_{i,j} \max(\mathbf{R}(x + i, y + j) - (\mathbf{L}_I(x, y) + H(i,j)), 0) \qquad (10)$$

$(i^2 + j^2 < r^2)$.

If this volume is large at a pixel, then it is effective to pass the tool over the pixel. By tracing the peak values in the **c**-image (c) only when the Laplacian value (b) is positive, the tool path (d) can be obtained. The effect of the tool path is shown in the simulation results (e) and (f).



(a) offset surface of flat endmill ($\mathbf{L}_l$) (R = 3*mm*)

(b) Laplacian of (a) (concave region)

(c) static milling volume (**c**) (white is > 1.0)

(d) tool path for corner cutting

(e) simulation result (**F**)

(f) under-cutting amount (**D**) (max: 2.94*mm*)

Fig.6.11 Tool path generation for corner cutting.

6.3.4 Feed Rate Control

In order to minimize machining time, the maximum feed rate should be selected within the load limit of the tool or material. By using the load evaluation discussed in Subsection **6.3.2.2**, the optimum feed rate can be selected. Let c_{max} be the maximum cutting volume per unit time, and let v_{max} be the maximum feed rate. Then, the feed rate is obtained as follows:

$$v = \min (c_{max}\, l/c_s,\ v_{max}) \tag{6.11}$$

where

$$l = \sqrt{(x_s - x_{s-1})^2 + (y_s - y_{s-1})^2 + (z_s - z_{s-1})^2}. \tag{6.12}$$

6.4 Examples

Some examples of NC machining results are shown in this section. A low-cost personal NC machine (Roland DG, CAMM-3) connected to a workstation (Sun-4) was used. Figure 6.12 shows the NC machine.

6.4.1 Machining Geometric Surfaces

The most important usage of NC machines is to produce accurate 3D shapes from geometrically defined objects. Here, the teapot was machined by using the G-buffer method. The process followed these steps.

1. Rough cutting with flat endmill (ϕ = 6*mm*, *x-scanning*).

   The path interval was 3*mm*. 1*mm* under-cutting was specified. A maximum of four scans was needed, and the tool depth was increased by 5*mm* for each scan. The simulation results are shown in Figs.6.4, 6.6-6.8(a), and the machining result is shown in Fig.6.13(a).

2. Fine cutting with ball endmill (*R* = 1.5*mm*, *x-scanning*).

   The path interval was 0.2*m*m, the same as the pixel interval. A maximum of two

scans was needed to ensure collision avoidance. The simulation results are shown in Figs.6.4, 6.6-6.8(b) and Fig.10, while the machining result is given in Fig.13 (b).

3. Fine cutting with ball endmill (R = 1.5*mm*, *y-scanning*).

   Since step 2 resulted in rippled surfaces where the gradient along the y direction was large, y-scans were generated from the same $\mathbf{L}_I$-image. The machining result is shown in Fig.6.13 (c).

4. Corner cutting with flat endmill (ϕ = 3*mm*).

   The simulation results are shown in Fig.6.11, and the machining result is given in Fig.6.13 (d).

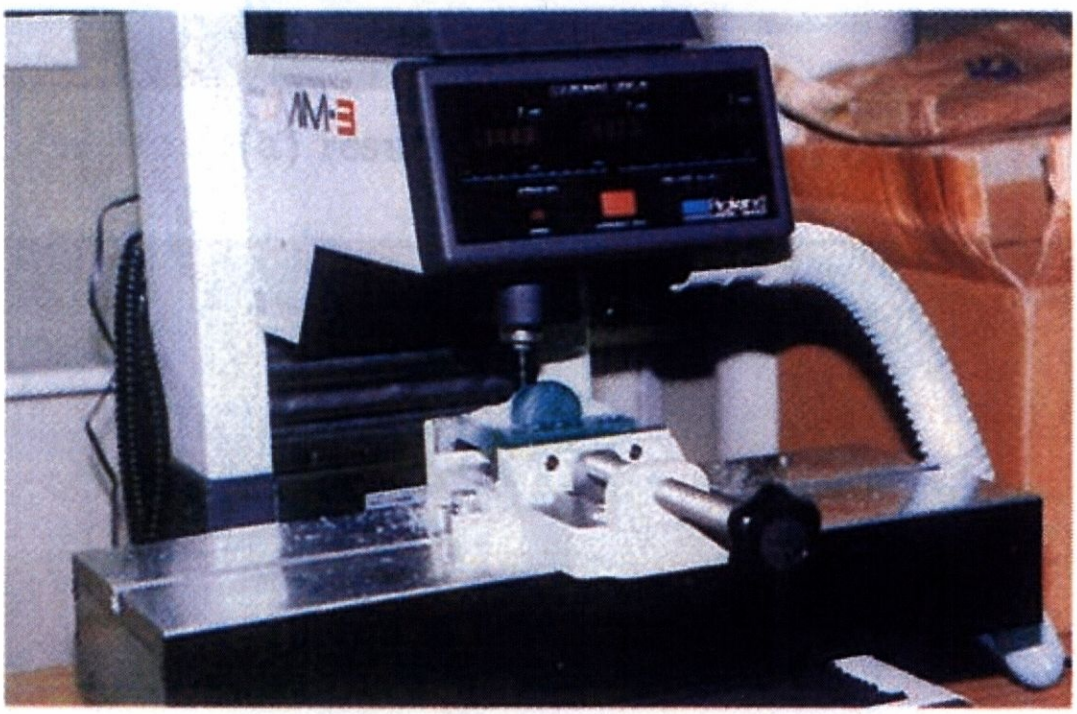


Fig.6.12 The NC Milling Machine.

(a) result of rough cutting

(b) result of fine cutting (only x-scanning)

Fig.6.13(1/2) Process of machining the teapot.

(c) result of fine cutting (x- and y-scanning)

(d) result of corner cutting

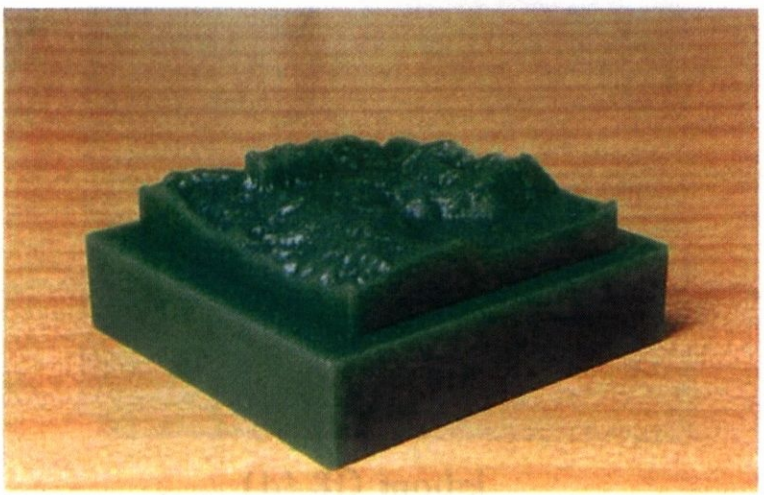
Fig.6.13 Process of machining the teapot.

6.4.2 Machining Meshed or Volume Data

In addition to geometric surfaces, 3D shapes defined by meshed or volume data can be easily created with the G-buffer method. Thus, it is a powerful tool for visualization. Two examples are shown in this section.

The first one is a topographical map. The G-buffer method is useful to draw maps with various enhancements, however, it is also possible to produce 3D maps with G-buffer machining. Fig.6.14 (a) is a regular topographical map of the region around the NTT Yokosuka R&D Center. Fig.6.14 (b) shows a 3D map of the same region. Both were made with the G-buffer method from the same height data.

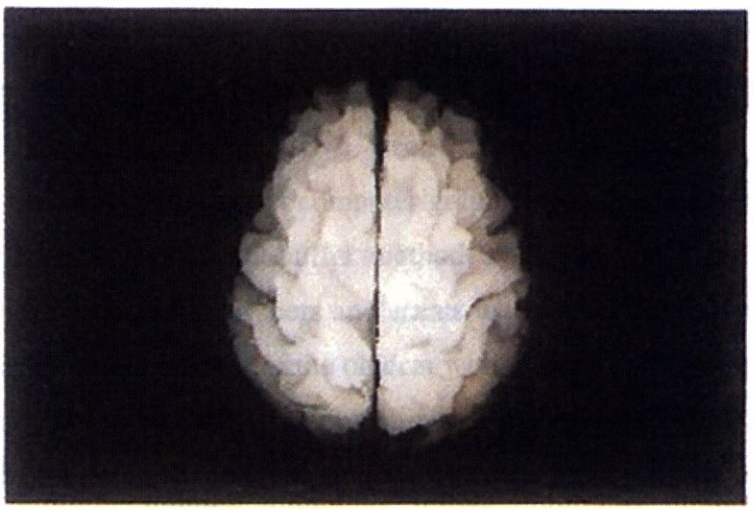
(a) enhanced 2D map by G-buffer rendering



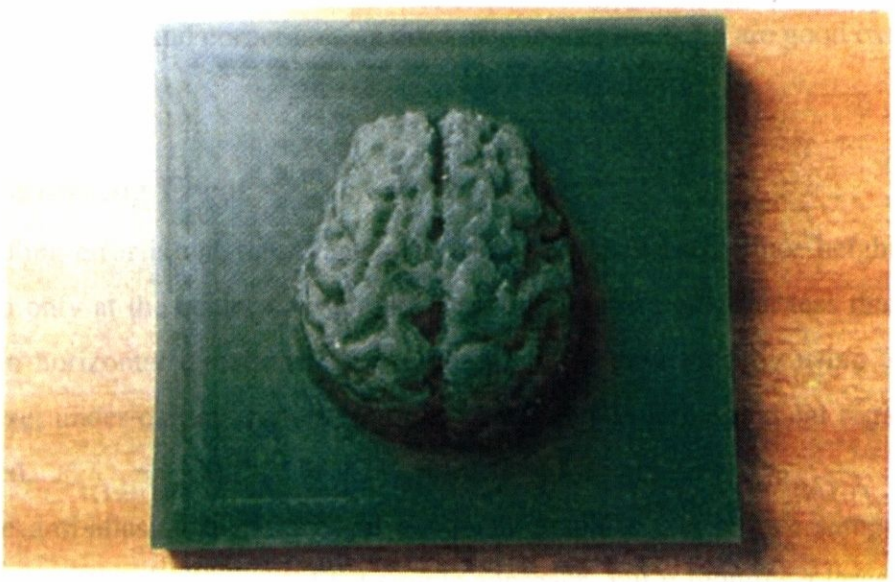
(b) 3D map by G-buffer machining

Fig.6.14 Topographical map.

The next example is a medical application. Figure 6.15 (a) shows a depth image of a human brain. The original data consists of 256×256×30 voxels from MRI. Figure 6.15 (b) is a 3D model of the brain. Although reentrant machining was impossible, the shape of the brain's upper surface can be recognized much more easily than is possible with 2D images.



(a) depth image



(b) 3D model

Fig.6.15 Human brain.

Data courtesy of Dr. Jin Tamai of Nippon Medical School.

6.5 Discussion

6.5.1 Advantages of the G buffer Machining

One of the notable advantages of the proposed method is its simplicity. Actually, our experimental NC machining system consists of about 20 small independent software modules. Most of them perform simple image processing operations, and all the functions described in Section **6.3** are realized by combining them. Another program is required to generate the G-buffers. However, conventional rendering programs, such as z-buffer, ray-tracing, or volume rendering routines, can be used without major changes.

For NC machining, people have been required to possess some skill in manufacturing technology, and most computer graphics people could not produce actual 3D objects by themselves. The G-buffer method, however, can make NC technology accessible to these people. Researchers and artists who render 3D shapes with computer graphics can easily obtain models of the objects with G-buffer machining.

Another advantage of G-buffer machining is its good combination of path generation and simulation. In conventional systems, the main purpose of simulation and verification is the error detection of tool paths. In G-buffer machining, on the other hand, the simulation result is effectively used ill the path generation for the next cutting phase. Collision avoidance and corner cutting described in Section **6.3.3** are good examples.

6.5.2 Sampling Error

Sampling error is one disadvantage of G-buffer machining. Since height fields are calculated only at the center of each pixel, sampling errors occur at less than the pixel interval in horizontal directions. Over-cutting can be prevented by using tool shapes with 1 pixel under-cutting, however, *aliasing artifacts* in large gradient regions cannot be avoided.

Some anti-aliasing techniques in computer graphics can reduce sampling errors. Sub-pixel sampling is a powerful yet simple approach, but it requires a large amount of memory and computing power. According to Kishinami *et al.* [Kishinami87], they succeeded in reducing the required memory capacity by employing the quad-tree data structure for path generation in their *Inverse Offset Method.* However, the quad-tree data

structure makes the image processing operations for the NC functions so complicated that one of the most significant advantages of G-buffer machining, its simplicity, may be lost.

The tolerance of the machining result depends on the user's purpose. If the machined object is only to verify or evaluate the designed or given shape, then G-buffer machining is acceptable. On the other hand, if the object is the final product, much more investigation about sampling error is required.

6.5.3 Computation Cost

The proposed method incurs high computation cost when processing large objects with high precision. Table 6.1 shows the computation time to machine the examples in Section **6.4**. This table presents the total time of path generation, simulation, and evaluation for each cutting phase. Although the values are reasonable for these examples, they become large for large objects. If the operations are simply implemented, the required memory space is $O(S_xS_y)$ and the computation cost is $O(S_xS_yH_r^2)$, where S_xS_y is the G-buffer size in pixels and H_r is the tool radius in pixels.

However, the problem of memory size will become less important in the near future since memory space of commercial computers is still increasing. Computation time can be reduced by using dedicated image processing hardware. Most operations are simple and iterative, so that vector processors or massively parallel processors can effectively accelerate the calculation speed. There is also some possibility to reduce the computation cost with efficient algorithms, especially for flat endmills.

Table 6.1 Required memory space and computation time for G-buffer machining. Sun-4/370 was used for this experiment. Computation time for G-buffer generation is not included.

Tool types are indicated as follows.

F6: ϕ = 6mm flat endmill;

F3: ϕ = 3mm flat endmill;

F1: ϕ = 1mm flat endmill;

B3: R = 1.5mm ball endmill.

Shape	Fig.13 (Tea pot)	Fig.14 (Map)	Fig.15 (Brain)
G-buffer Size (pixels)	400×200	301×301	200×200
Pixel Size	0.20 *mm*	0.20 *mm*	0.30 *mm*
Required Memory Space	1.14 MB	1.26 MB	0.56 MB
Rough Cutting	1007 sec (*F6*)	1140 sec (*F6*)	224 sec (*F6*)
Fine Cutting	378 sec (*B3* : 1*st*) 181 sec (*B3* : 2*nd*) 7 sec (B3 : *y*)	207 sec (B3)	112 sec (*B3* : 1*st*) 54 sec (*B3* : 2*nd*) 188 sec (*F1* : 1*st*) 94 sec (*F1* : 2*nd*)
Corner Cutting	257 sec (*F3*)		

6.6 Conclusion

We applied the G-buffer method to NC machining. By preparing G-buffers from a parallel projection, the various functions required for a NC system were realized with image processing operations. This allows any surface description and any tool shape to be used. Conventional hardware and software for computer graphics and image processing can be employed with this method. This makes NC system development much easier. Experimental results show that our method can be effectively used in an actual machining process. The method should be a great tool, not only for CAD/CAM, but also for scientific visualization.