CAD/Graphics'2001
August 22-24, Kunming
International Academic Publishers

LOCAL MESH DECIMATION FOR REAL-TIME THREE-AXIS NC MILLING GRAPHICAL SIMULATION

Sang Hun Lee and Kang-Soo Lee
Graduate School of Automotive Engineering
Kookmin University
Seoul, Korea
shlee@kookmin.ac.kr

ABSTRACT

The view dependency of typical z-buffer-based NC simulation methods is overcome by a polygon rendering technique that generates polygons to represent the workpiece, thus enabling dynamic viewing transformations without reconstruction of the entire data structure. However, the polygon rendering technique still has difficulty in realizing real-time simulation due to the limited performance of current graphics devices. Therefore, it is necessary to develop a mesh decimation method that enables rapid rendering without loss of display quality. To meet this requirement, in this paper, we propose a new mesh decimation algorithm for the workpiece which is represented by the z-map structure and whose shape is dynamically changed by the tool. In this algorithm, the z-map data for the workpiece are divided into several regions, and a triangular mesh is constructed for each region. Then, if the tool cuts any region, its mesh is regenerated and decimated again. Since the range of mesh decimation is confined to a few regions, the reduced triangles for rapid rendering can be obtained in a short time. This local mesh decimation method may contribute toward realizing the polygon rendering-based NC simulation in real time on computers equipped with general graphics cards.

KEYWORDS: NC simulation, three-axis milling, real-time simulation, z-map, polygon rendering, mesh decimation.

INTRODUCTION

In addition to traditional research on the development of computer-aided manufacturing (CAM) systems for the generation of numerically-controlled (NC) tool paths from a given computer-aided design (CAD) model, research on the verification and simulation of NC programs has been carried out to prevent the overcut or interference problems in actual NC machining processes [1]. Currently, most of the existing CAM systems usually have graphical simulation capabilities for NC programs. There are two approaches in representing workpieces: one is to adopt the representation schemes for solid models [2, 3, 4], and the other is to use discrete models, such as the z-map [5], dexel [6], and discrete vector models [7, 8]. The solid representation approach can describe a workpiece more precisely including vertical walls, but it requires considerable computation time for the Boolean operations between the workpiece and the tool-swept volumes. In addition, the Boolean operations are not stable due to the numerical errors of the intersections. Conversely, the discrete model approach provides very stable and fast Boolean operations, although the model accuracy is lower than that of the solid representation. In addition, algorithms for efficient graphical simulation and database management can be easily devised and implemented in a simple manner. Thus, most of the existing NC simulation systems use discrete models. The z-map model is widely used, where the z-values for grid points on the x-y plane are stored in the form of a one- or two-dimensional array, particularly in three-axis milling simulations. Thus, we also adopt the z-map as a topological framework for representing the workpiece for simulation of three-axis NC milling processes.

The visualization methods for NC simulation are classified into three categories: the z-buffer method [6, 9], the contour display method [10], and the polygon display method [1]. The z-buffer method [6, 9] is very efficient for visualizing dexel-based objects. This method aligns the depth vector of the dexel co-ordinate system with the viewing vector of the screen co-ordinate system so that only the near face of each dexel is visible and each grid point of the dexel plane corresponds to a constant number of pixels on the display device. Because the colour index of the nearest dexel at each address is written directly to the frame buffer of the display device, this method enables real-time simulation that shows more than 10 frames per second. However, because dexels are aligned with the viewing vector, only the front and back views can be efficiently displayed; to display dexel-based objects in other viewing directions using this method, the entire dexel representation must

be reconstructed, which severely limits its application in manufacturing and engineering. In addition, the viewing direction is not usually optimal for verification. In three-axis milling, cutter direction is best for z-map direction, while an isometric view is preferred for visualization. Therefore, it is desirable to decouple the z-direction from the viewing direction.

To overcome the view dependency problem, the contour display method was introduced by Huang and Oliver [10]. This method generates contours that connect dexel faces (centre points) along constant x and y grid addresses. Thus, two sets of equally spaced, mutually orthogonal planar contours are displayed as piecewise-continuous closed-line loops to represent dexel-based objects. Because dexel density is typically quite high and line thickness is associated with the size of the object's screen-space bounding box relative to the view window, the display appears as a shaded image that may be viewed dynamically. Although the contour display method achieves dynamic viewing transformations of the milled part, the resultant shaded image is coarse and the gaps between contours are shown when users magnify the image. Therefore, to achieve a smoother rendering as well as dynamic viewing capability, it is desirable to incorporate an incremental triangulation algorithm to generate a polygonal mesh over the emerging workpiece.

The polygon rendering method generates polygons from the z-map or dexel models and displays them with the polygon rendering engine of a three-dimensional (3-D) graphics card. This method facilitates a view-independent smooth rendering of high quality. However, it is computationally expensive and requires an advanced 3-D graphics acceleration card for fast polygon rendering. Currently, advanced 3-D graphics cards can render from one to five million triangles per second. However, if the size of a workpiece is very large such as the stamping dies for automotive panels, even these cards cannot render the workpiece in real time. For example, if the die size is 500 mm x 500 mm and the grid size is 1 mm, 500,000 triangles should be rendered for each screen frame. For real time simulation, at least the screen should be refreshed 10 times per second. Thus, the graphics card must have a rendering capability of at least 5,000,000 polygons per second. Therefore, in order to enable real-time NC simulation using polygon rendering methods, it is necessary to develop a method to reduce the number of polygons without the rendering quality deteriorating. The objective of this paper is to develop such a mesh decimation algorithm.

The rest of this paper is as follows. Section 2 describes our new approach, called the local mesh decimation method, which is suitable for the workpiece model whose shape is changed locally for each tool movement. Section 3 introduces the overall procedure of our algorithm. Section 4 describes the mesh decimation algorithm for a region in detail. Section 5 contains case studies and their discussion. In Section 6, we present our conclusion.

OUR APPROACH

Let us observe the process for generating and rendering polygons from the z-map data of a workpiece. In general, rendering functions in a graphics library such as OpenGL are used to obtain a shaded image of an object. As these functions usually require a set of triangular polygons as input, a triangular mesh is constructed by generating two triangles for each grid of the z-map data. Because the number of triangles is usually huge, as shown in Fig. 1, it takes considerable time to render a shaded image, especially for the computers that are not equipped with an advanced graphics card. Therefore, if the triangles are reduced while maintaining the rendering quality, the rendering process can be accelerated. In the early stage of a rough cutting process, this method gives a great benefit, because as the large area of the workpiece is not machined it remains a simple geometry, such as a plane.

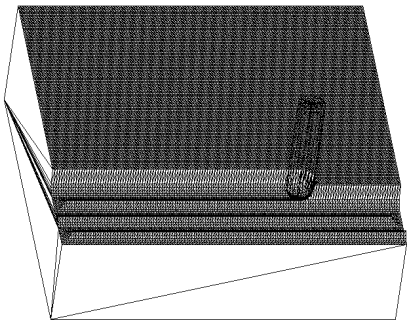


Fig. 1 A triangular mesh generated from the z-map data for a workpiece

In the area of computer graphics, a number of mesh decimation methods that reduce the number of triangles in a mesh have been proposed to improve interactive rendering performance, or reduce data storage and transmission requirements [11, 12]. They can be classified into three categories depending on the objects to be removed for mesh simplification: the vertex removal method, the edge collapse method, and the face removal method. The edge collapse method is the most popular of these.

However, the mesh decimation for the workpiece for NC simulation has the following different characteristics from the existing mesh decimation methods:

- The workpiece shape is changed for each tool

movement command of the NC program, whereas the model for the existing mesh decimation algorithms does not vary.

- In general, the workpiece is rendered for each tool movement command of an NC program; however, the change of the workpiece between two screen frames is confined to a local area.
- We need to consider the trade-off relation between the rendering time and the computation time for mesh decimation. If the mesh for the workpiece is simplified, although the rendering time reduces, additional computation time for mesh simplification is required. If a very high performance 3-D graphics card and CPU are used or the number of triangles is relatively small, the rendering time without mesh decimation can be shorter than that with mesh decimation. Therefore, it is necessary to devise a mesh decimation algorithm that always guarantees faster rendering than the method without mesh decimation, independent of the hardware's performance, whilst preserving the rendering quality.

Considering the characteristics of the problem above, the two following methods are candidates for the mesh decimation method to accelerate the rendering of the dynamically varying workpiece.

[Method 1] Global Mesh Decimation

In this method, as illustrated in Fig. 2, mesh decimation is carried out globally for the workpiece for each tool movement. If the reduction rate is high, an improvement in rendering speed can be expected. However, if the reduction rate is low, this method may be slower than polygon rendering without decimation, because almost the same number of triangles may be rendered in spite of the mesh decimation. Additionally, this method is very inefficient, as the mesh decimation must be performed for the whole area, even though the change in the workpiece is confined to a small local area between two screen frames. Therefore, this method is inadequate to improve the rendering performance for NC simulation.

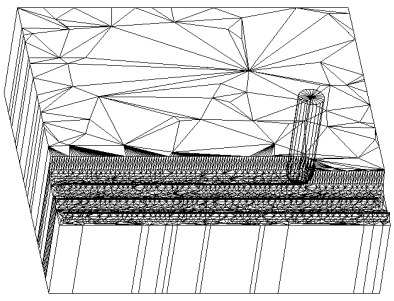


Fig. 2 Global mesh decimation for each tool movement

[Method 2] Local Mesh Decimation

In this method, as shown in Fig. 3, the workpiece is divided into regions, and meshes are generated and decimated for the regions. If the tool cuts a region, the decimated mesh for the region is discarded, and a new mesh for the region is generated from the z-map data and used for subsequent rendering. In this case, the decimated meshes and the non-decimated meshes exist simultaneously. Although it takes a long time for the first display, due to the mesh decimation, it subsequently always takes less time than rendering the non-decimated meshes for the whole area of the workpiece, as a lesser, or the same, number of triangles are displayed. This method is effective in the early stages of rough cutting processes. However, once the tool has swept the overall workpiece, the number of triangles becomes the same as that of the non-decimated case. Therefore, it is desirable to periodically perform mesh decimation for the changed regions. Even then, the elapsed time of this method is always less than that of the non-decimated method. The period can be optimized for a given NC program prior to graphical simulation. Whenever the de- cimation is carried out, the number of triangles is reduced again. Fig. 3 shows an example of this periodical local mesh decimation method. In this work, we adopt this method, and the detailed algorithm is described in the next section.

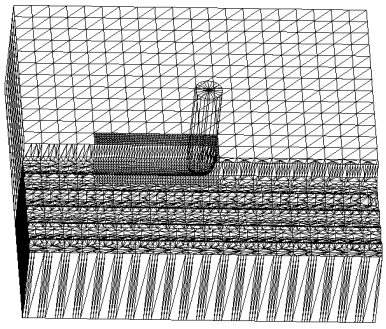


Fig. 3 Periodical local mesh decimation method

OVERVIEW OF THE LOCAL MESH DECIMATION ALGORITHM

The overall algorithm for the periodic local mesh decimation is as follows:

(Step 1) The z-map data of the workpiece are divided into small regions, and then meshes for each region are generated, decimated, and stored.

(Step 2) A tool movement command of the NC program is read and the z-map data are updated accordingly. The display mode of a region cut by the tool is set as the z-map rendering mode.

(Step 3) Meshes for regions are rendered by calling the

rendering functions of a graphics library, such as OpenGL. If a region is set in the z-map rendering mode, a mesh is generated from the z co-ordinates of the grids in the region, and is passed to the rendering functions. Otherwise, the decimated mesh for the region is passed to the rendering functions.

(Step 4) If the current iteration is coincident with the decimation period, the meshes are generated and decimated for the regions in the z-map rendering mode, and then go to Step 2. Otherwise, go to Step 2 directly.

MESH DECIMATION FOR A REGION

Initially, the mesh decimation for a region is applied to all regions and subsequently to the changed regions periodically. Our algorithm for mesh decimation for each region is composed of four steps as follows:

(Step 1) A triangular mesh is created from the z-map data for a given region. The mesh is represented using the winged edge data structure in this work.

(Step 2) A vertex is selected in the mesh, and test if the vertex is removable.

(Step 3) If the vertex is removable, delete the vertex using the edge-collapse method in which the adjacency information of topological entities connected to the vertex is modified.

(Step 4) If all of the vertices are tested, the resultant decimated mesh is stored using an array of the vertices, which is passed to the rendering functions. Otherwise, go to Step 2

The detail of each step is described in the following sections.

(1) Construction of a Triangular Mesh for a Region from the Z-map Data

In this step, a triangular mesh represented in the winged edge data structure is generated from the z-map data for a region that is to be decimated. To build the mesh efficiently, the topological data of the mesh is filled in a predetermined manner. Data storage for winged edge data is allocated in the first step, and then used repeatedly until the termination of this program in order to avoid consuming time for memory allocation and de-allocation.

(2) Criteria for Vertex Removal

In this step, the system selects a vertex in the mesh and checks if the vertex is removable or not, using appropriate criteria. Two criteria are used in this work: the geometric error allowance and the topological validity. The error allowance prevents a large distortion of the workpiece's shape when a vertex is eliminated. The topological validity prevents any face from being flipped or degenerating to an edge when a vertex is eliminated. These two criteria are described in more detail in the following sections.

(2.1) Geometric Approximation Error

To estimate the approximation error, we adopt the method proposed by Schroeder et al. [11]. In this method, an average plane is constructed first using the normals $\mathbf{N}_i$, centres $\mathbf{P}_i$, and areas A_i of the triangles adjacent to the vertex $\mathbf{V}_j$. If $\mathbf{N}_{avg}$, $\mathbf{n}$, and $\mathbf{P}_{avg}$ denote the normal, the unit normal, and the centre point of the average plane, respectively, then

$$\mathbf{N}_{avg} = \Sigma A_i \mathbf{N}_i / \Sigma A_i,$$

$$\mathbf{n} = \mathbf{N}_{avg} / |\mathbf{N}_{avg}|,$$

$$\mathbf{P}_{avg} = \Sigma A_i \mathbf{P}_i / \Sigma A_i.$$

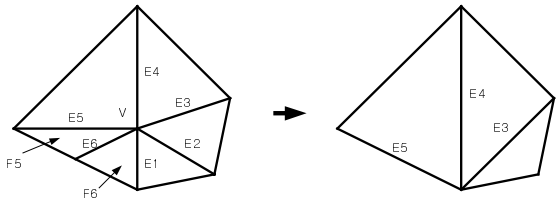
The distance from the vertex $\mathbf{V}_j$ to the average plane is then $d = |\mathbf{n} \bullet (\mathbf{V}_j - \mathbf{P}_{avg})|$, which is used as the local error. If the vertex $\mathbf{V}_i$ connected to $\mathbf{V}_j$ is deleted via an edge collapse, the vertex error e_j is distributed to $\mathbf{V}_i$ using $e_i = e_i + e_j$. Thus, the total error e_i is a combination of the local error, i.e., distance to plane, plus the accumulated error value at that vertex.

In order to determine whether the vertex is removable, the system calculates this total error. If the total error is within the error allowance, the vertex is determined to be removable. Otherwise, the vertex is retained.

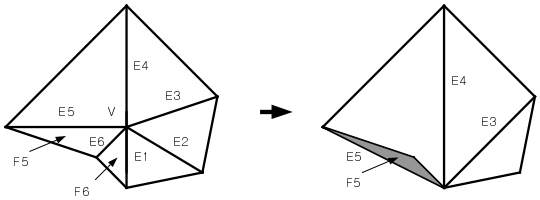
(2.2) Topological Validity

The edge collapse method is adopted to remove vertices in this paper. That is, one of the edges connected to the vertex to be removed is deleted together with the vertex. The Euler operators [13] are introduced for the implementation of the edge collapse. During the edge collapse process, the validity of a mesh may be violated. According to our algorithm, in order to delete a vertex V and the edge E_1 shown in Fig. 4(a), two KEFs and one KEV are applied: KEF(E_2,F_2), KEF(E_6, F_6), and KEV(V, E_1). As a result, the face F_5 degenerates to a line when the edge E_1 is collapsed. On the other hand, in the case of Fig. 4(b), the face F_5 is flipped when the edge E_1 is collapsed. Thus, it is necessary to check these cases before performing the edge collapse operation. In our algorithm, the shortest edge is chosen as a candidate for edge collapse initially. Then, the criterion for topological validity is examined. If the resulting mesh is predicted to be invalid, another edge is chosen as a candidate. This process is repeated until the criterion is satisfied. If all of the edges connected to the vertex do not satisfy this criterion, the

vertex is not removed.



(a)



(b)

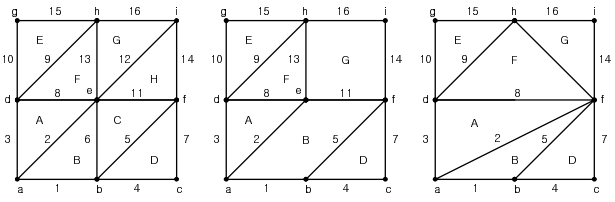
Fig. 4 Illegal topology change caused by edge collapse:
(a) generation of a degenerate face;
(b) generation of a flipped face.

(3) Vertex Removal by Edge Collapse

Once a pair of the vertex and edge to be removed is determined, the topological data related with them should be modified. In our work, the edge collapse operation is implemented using the Euler operators. To facilitate the implementation, vertices are classified as interior or boundary vertices, and the algorithms devised accordingly.

(3.1) Deletion of an Interior Vertex

If a vertex to be removed is located inside the mesh, two KEFs and one KEV are applied sequentially for an edge collapse. As illustrated in Fig. 5, if a pair of the vertex **e** and the edge **11** is selected for an edge collapse, a KEF is applied to the edges **6** and **12** first, and then a KEV is applied to the edge **11**.

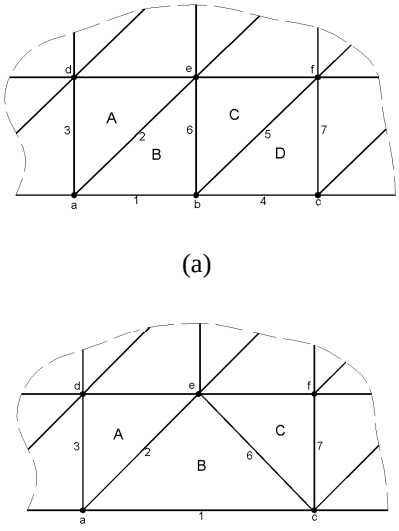


(a) (b) (c)

Fig. 5 Edge collapse procedure: (a) initial shape; (b) result of two KEFs; (c) result of a KEV.

(3.2) Deletion of a Boundary Vertex

The edge collapse algorithm for a boundary vertex is similar to that for an interior vertex. However, there are different points in the method as to how an edge to be collapsed is selected and the method of applying KEFs to that edge. For a boundary vertex, only two boundary edges can be candidates for an edge collapse, whereas for interior vertices, all of the edges connected to the vertex can be candidates for an edge collapse. For example, in Fig. 6, only the edges **1** and **4** can be candidates. A KEF and a KEV are executed to collapse an edge on the mesh boundary. In Fig. 6, the edge **4** is selected as an edge to be collapsed. Then, the face **D** and the edge **5** are first deleted by a KEF, and then a KEV deletes the vertex **b** and the edge **4**.



(a)

(b)

Fig. 6 Boundary edge collapse:
(a) initial shape; (b) result of a KEF and a KEV.

CASE STUDY

As a case study, we measured the numbers of the rendered faces and the computation times for rendering the workpiece in the early stage of the rough cutting process for the core insert of an injection mould, shown in Fig. 1. The PC used for this experiment is equipped with a Pentium-II 350MHz CPU and an ATI RAGE IIC graphics card that does not have an OpenGL accelerator. The size of the workpiece is 140 x 140 mm, and the grid size of the z-map data is 1 mm. The cutting tool is a ball-end mill whose diameter is 11 mm and length is 50 mm. The NC program used in this experiment has 255

blocks of tool movement commands.

(1) Rendering Time Related to Region Size

When the decimation period and error allowance is fixed, the variation of the rendering time is illustrated in Fig. 7 as the region size varies from 5 x 5 to 30 x 30. In this figure, **s** and **e** denote the decimation period and the error allowance, respectively. The assignment **s = -1** means that the mesh decimation is carried out for all of the regions initially, and subsequently no decimation is performed. As shown in the graphs in Fig. 7, the rendering time does not increase monotonously as the region size increases. Therefore, the optimization of the region size for a given NC program and a workpiece can be a future research topic.

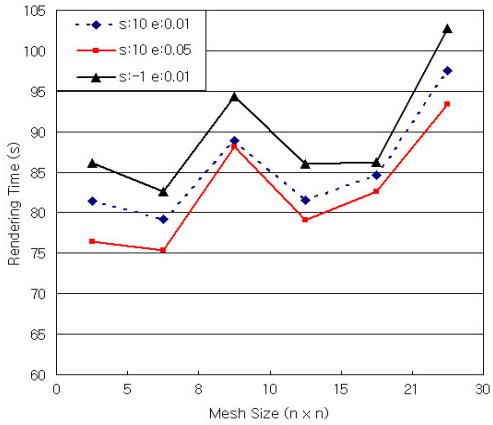


Fig. 7 Relationship between the rendering time and the region size.

(2) Reduction Rate and Rendering Time for Varying Error Allowance

Let us assume that the region size is fixed at 8 x 8, and the decimation period is set to 20 tool movement commands. Then, as the error allowance increases from 0.01 to 0.1 mm, the total number of the rendered triangles and the rendering time decrease as shown in Fig. 8. If the error allowance is zero, triangles are generated from the z-map data directly and no mesh decimation is performed. The y-axis values denote the ratios with respect to the value of the case in which the error allowance is zero. The larger is the error allowance, the smaller are the number of triangles and the computation time.

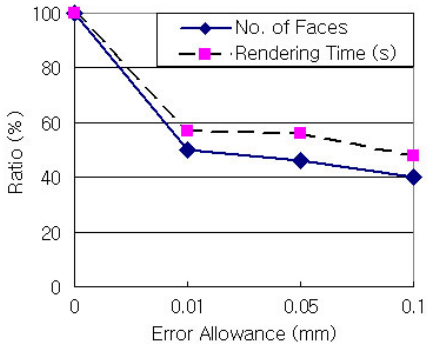


Fig. 8 Number of faces rendered and the rendering time vs. error allowances.

Fig. 9 and Fig. 10 show the graphs for the number of faces and the rendering time as the simulation proceeds. As shown in Fig. 9, the graphs for the number of triangles have a sawtooth appearance, as the number of faces is reduced whenever the mesh decimation is performed. As shown in Fig. 10, the rendering time increases stepwise, and peaks appear periodically due to the computation time for mesh decimation. To clarify the trends, straight lines are added to the graphs in Fig. 10.

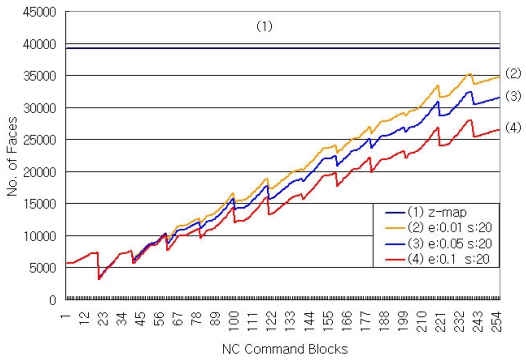


Fig. 9 Numbers of faces displayed for each frame during the rough cutting simulation.

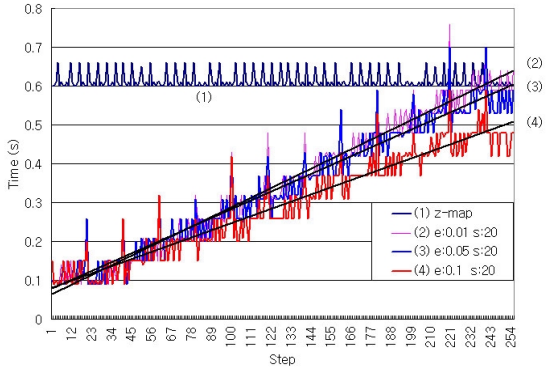


Fig. 10 The variations of the rendering time of each screen frame for different error allowances.

(3) Relation between Rendering Time and Reduction Period

When the region size is 8 x 8 and the error allowance is fixed at 0.05, the total number of faces rendered and the rendering time for the whole simulation are measured with changing decimation period. The result is illustrated in Fig. 11.

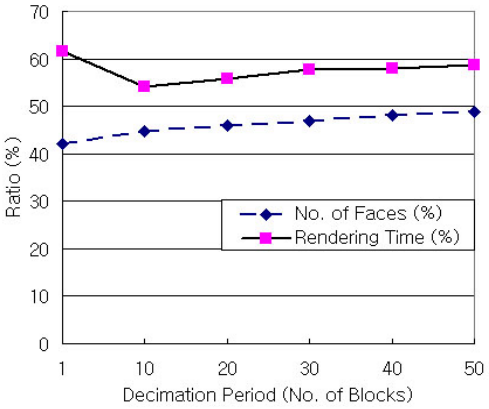


Fig. 11 The number of faces and the rendering times vs. the decimation period.

As the decimation period increases, the number of faces rendered increases. However, if the period is too short, the rendering time increases. The reason is because the increment of the computation time due to the mesh decimation is larger than the decrement of the rendering time due to the reduction of the triangles. The optimal period is chiefly dependent on the NC program, region size, and hardware performance. Thus, the optimization of the period also can be a research topic and remains for future work.

CONCLUSION

In this paper, we propose a local mesh decimation method to enhance the rendering speed for real-time three-axis NC milling simulation. In this approach, the mesh decimation is confined to the regions modified by the tool. Thus, this approach is always faster than the existing polygon rendering methods without mesh decimation, and always enables faster simulation, independent of the hardware performance. However, further work to complete our approach remains as follows. First, the region size and the decimation period for a given NC program and a workpiece may be optimized prior to graphical simulation. Second, although the polygon rendering method is view-independent, it would be convenient for users to provide a capability to calculate the error allowance for mesh decimation from the current viewing parameters.

REFERENCES

[1] B. K. Choi and R. B. Jerard, Sculptured Surface Machining, Kluwer Academic Publishers, 1998.

[2] H. B. Voelcker and W. A. Hunt, “The role of solid modeling in machining process modeling and NC verification”, SAE Technical Paper 810195, 1981.

[3] R. Fridshal, K. P. Cheng, D. Duncan and W. Zucker, “Numerical control part program verification system”, Proc. Conf. CAD/CAM Technology in Mechanical Engineering, MIT Press, 1982, 236–254.

[4] J. R. Woodwark and A. F. Wallis, “Creating large solid models for NC tool-path verification”, Proc. CAD–84 Conf., Brighton, UK, Butterworths, 1984, 455–460.

[5] R. O. Anderson, “Detecting and eliminating collisions in NC machining”, Computer-Aided Design, 10(4), 1978, 231–237.

[6] T. Van Hook, "Real-time shaded NC milling display", Proceedings of SIGGRAPH '86 in Computer Graphics, 20(4), 1986, 15–20.

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

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

[9] W. P. Wang and K. K. Wang, “Real-time verification of multiaxis NC programs with raster graphics”, Proceedings of the IEEE Int’l Conf. On Robotics and Automation, San Francisco, 7-10 April, pp. 1986, 166–171.

[10] Y. Huang and J. H. Oliver, "Integrated simulation, error assessment, and tool path correction for five-axis NC milling", Journal of Manufacturing Systems, 14(5), 1995, 331-344.

[11] W. J. Schroeder, J. A. Zerge and W. E. Lorensen, "Decimation of triangle meshes", Proceedings of SIGGRAPH '92 in Computer Graphics, 26(2), 1992, 65-70.

[12] P. Cignoni, C. Montani and R. Scopigno, "A comparison of mesh simplification algorithms", Computers & Graphics, 22(1), 1998, 37-54.

[13] K. Lee, Principles of CAD/CAM/CAE Systems, Addison Wesley, 1999.