ELSEVIER

Computer-Aided Design 38 (2006) 378–391

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

Real-time, dynamic level-of-detail management for three-axis NC milling simulation

S.Q. Liu [a], S.K. Ong [b,*], Y.P. Chen [a], A.Y.C. Nee [b]

[a] *School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan, China*
[b] *Department of Mechanical Engineering, National University of Singapore, 9 Engineering Drive 1, Singapore, Singapore 117576*

Received 1 April 2005; accepted 13 November 2005

Abstract

The *Z*-buffer based NC milling simulation approach has been widely used in a number of commercial application softwares. However, this approach is essentially based on image and hence it has many drawbacks. Some polygon-based approaches have been presented to overcome these drawbacks, but these approaches are mostly not optimal. In order to achieve real-time simulation, the total number of polygons has to be reduced in these approaches, which results in poor image quality. Using those proposed level-of-detail mesh algorithms can reduce the number of polygons, but the majority of those algorithms are not suitable for dynamic real-time NC milling simulation. Therefore, it is necessary to develop an approach to reduce the number of polygons without much loss in image quality. An adaptive regular mesh decimation algorithm is proposed in this paper. It uses a quadtree to represent the milling surface and enabled flags are used to perform the optimization of the mesh, actual insertion and deletion operations of triangles do not exist. This algorithm can automatically adjust the polygon density to approximate the milling surface according to its changes in real-time, and numerical inspections of any area of the machined surface can be performed with full resolution.

Keywords: Mesh decimation; NC simulation; Real-time simulation; Polygon rendering; Levels of detail

1. Introduction

To prevent overcut and interference problems (refer to Appendix A) in NC machining, many computer-aided manufacturing (CAM) systems have been developed to verify and simulate these machining processes. In general, NC machining simulations can be classified into two categories: geometric simulation and kinetics simulation. Geometric simulation only considers the geometric information of the cutter and the workpiece and ignores the cutting parameters, cutting forces and other physical factors. Progress has been made in geometric simulation with the development of corresponding algorithms. There are several main approaches to implement the cutting operations of three-axis NC milling processes thus far, such as the direct Boolean subtraction between a tool path envelope and a solid object, the image-space Boolean subtraction between a cutter and a solid object, and the intersection between a cutter and the discrete vectors of a solid object, etc.

The direct Boolean subtraction approach is an exact and analytical approach. It directly performs the Boolean subtraction operation between a solid model and the volume swept by a cutter between two adjacent tool positions. The solid model of the workpiece is updated in the machining process. Many implementations of this approach are performed based on the CSG or B-Rep representation. Although this approach can provide accurate verification and error assessment, the computation cost is reported to be $O(N^4)$ [1], where N is the number of cutter movements. A complex NC machining program may consist of thousands of movements, which make the calculation intractable. In order to increase efficiency, a number of approximate approaches have been proposed. The computation cost of these approaches decreases to $O(N)$.

Van Hook [2] developed a real-time shaded display of a solid model being milled by a cutter that follows an NC path. This approach is accomplished with an extended *Z*-buffer data structure. The workpiece and the cutter are separated into many small rectangular bars extending along the *Z*-axis, which aligns the viewing direction, and each small rectangular bar corresponds to a pixel of the screen of a raster display device. A small rectangular bar and its data structure together are

* Corresponding author. Tel.: +65 6874 2222; fax: +65 6779 1459.
E-mail address: mpeongsk@nus.edu.sg (S.K. Ong).

0010-4485//$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2005.11.003

called a dexel (depth element). The main advantage of this approach is simple and highly efficient. The reason for this is: the cutting operations of this approach are attained through using Boolean set operations on the one-dimensional dexels, i.e. the nearest colour of the workpiece, cutter, or background is directly written into the frame buffer of the raster display device. Since this approach integrates the Boolean operation and display into one step, a very high update rate can be attained in the milling simulation. However, the machining result of this approach has no continuity, and cannot be zoomed in or out. Since it is an image-based approach, the machined workpiece cannot be rotated to any arbitrary angle for viewing, except that the simulation is performed again after each rotation, or several viewing directions are set beforehand and several simulations are performed simultaneously, or just a 180° rotation transformation is made using the characteristic of this algorithm. These limitations severely restrict its application in the manufacturing and engineering area. These problems were overcome through an extension of this approach by Huang and Oliver [3]. They presented a contour display approach that can achieve a dynamic viewing transformation of the machined workpiece. However, the resulting rendered image is coarse, and the gap between adjacent contours is apparent when the machined workpiece is enlarged. Therefore, to achieve the dynamic viewing capability and a smooth rendering result, the use of a polygonal mesh to represent a workpiece is introduced. This approach fully complies with the characteristics of the three-axis milling process.

Before the discussion of the polygonal mesh representation, we first take a look at the characteristics of a three-axis NC milling process: (1) only the upper surface of a workpiece is the milling surface; (2) a straight-line parallel to the axis of the milling tool has one and only one intersection point with the upper surface; and (3) the upper surface of the workpiece can be formed by a number of discrete vertices. In computer graphics, objects are usually represented using triangle meshes due to their simplicity. In addition, almost all graphics libraries provide graphic functions to render triangle meshes, and modern graphics acceleration cards provide the fast triangle mesh rendering function. Therefore, using triangle mesh representations can reduce the difficulty in devising the simulation software and speeds up the rendering.

In general implementations of triangle meshes, the upper surface of a workpiece is sub-divided into an even grid. Every four neighbouring vertices form a dexel. All vertices are connected to form triangles according to some rules and to be rendered later. During the milling simulation, the heights of the vertices are updated along the tool path, and the rendering of the upper surface is refreshed. Although this approach is computationally expensive and requires advanced 3D graphics acceleration cards to accelerate the rendering, fortunately, modern 3D acceleration graphic cards are very cheap and widely used in personal computers. However, if the workpiece is very large and the grid size is very small, even 3D graphics acceleration cards bite off more than they can chew. For instance, if a workpiece size is 600×600 mm and the dexel size is 1 mm, over 7,20,000 triangles (which include the triangles of other surfaces, and where each dexel is assumed to be sub-divided into two triangles) should be rendered for each frame. However, the screen refresh rate must be at least 10 frames per second (fps) to achieve real-time display. Thus, the rendering capability of a graphics card must be at least 7,200,000 triangles per second. This number of triangles has far exceeded the rendering capability of common graphic cards.

In order to achieve real-time NC milling simulation, there are two ways: one is to increase the dexel size to reduce the number of triangles and the other way is to develop a new approach that can perform real-time mesh decimation to decrease the number of triangles without much loss in image quality. The former is at the cost of poor image quality. The latter not only can achieve real-time simulation but also maintain highly rendered image quality. In addition, the resolution can be automatically increased when a local area is enlarged. A regular mesh decimation approach making full use of regular mesh characteristics is represented in this paper.

The rest of this paper is arranged as follows: the related work on levels of detail is presented in Section 2. Section 3 describes the motivation of this research. The details of our approach are elaborated in Section 4 which includes the basic idea, organization of nodes, storage of nodes, elimination of cracks, inserting nodes, deleting nodes, discretization of a workpiece and the steps in the NC milling simulation. In Section 5, we evaluate our approach with results of its use in the rough cutting simulation of a craftwork. Finally, the conclusion is given in Section 6.

2. Related work on levels of detail

Many applications have to deal with large polygonal surface models, such as digital terrain modelling and visual simulation. The number of polygons usually greatly exceeds the capabilities of the typical commercially available graphics hardware. To render these polygons in real-time, many methods using multi-resolution to approximate the polygonal surfaces have been proposed. Multiple surface models at varying levels of detail are first generated using these methods, and the applications will select an appropriate level-of-detail model to render.

A number of general algorithms have been proposed to produce simplified versions of the polygonal objects. Almost every simplification technique uses some variations or combinations of three basic polygon removal mechanisms, namely, sampling, refinement and decimation. Sampling algorithms [4–8] sample the geometry of an initial model with either the points on the surface of the model or the voxels superimposed on the model in a 3D grid. Refinement algorithms [9,10] first find a simple base mesh with the same topology as the input surface, and recursively sub-divide this base mesh, with more and more details being added into a certain local region of the base mesh during each phase. Decimation algorithms [11–14] iteratively remove vertices, edges (or vertex pairs) or faces from a mesh, and triangulate the resulting hole after each step. In summary, some of these algorithms maintain a good approximating quality of

the models but are slow, while other algorithms are exactly the opposite. Hence, they are not suitable for NC milling simulation.

Our implementation of the three-axis NC milling simulation also uses the height field that is widely used in terrain geometry. Hence, some algorithms for approximating terrain surfaces and other height fields using polygonal meshes are briefly introduced here. These algorithms usually attempt to approximate surfaces either with a given number of polygons, within a given geometric error metric, or in manner that preserves the key features of the surface. The methods to represent the surfaces can be classified into two categories: triangulated irregular networks (TINs) and regular grid representations.

Due to allowing variable spacing among vertices of the triangle meshes, TINs [15–17] need far fewer triangles than other methods to approximate a surface at any level of detail. However, creating TINs is usually time-consuming, and this hampers the use of TINs in real time applications.

Although regular grid representations usually generate many more polygons than TINs for a given level of detail and suffer from cracks, they allow easier construction of a multiple level-of-detail hierarchy. This is because the quadtree or binary trees are inherently suitable for creating this hierarchy. The simplicity of the hierarchy makes regular grid based algorithms more efficient than those based on TINs. The followings are the three classic algorithms based on regular grid representations.

Lindstrom [18] proposed a real time continuous level-of-detail terrain algorithm in 1996. Vertices are grouped into blocks, and sorted by their error value. The level of each block is first assessed according to a projected pixel error metric, and those vertices exceeding a given threshold of the projected pixel error are inserted. A binary tree is used to eliminate cracks between two adjacent blocks. Frame coherence is used to limit the number of vertices that are tested in each frame, and this speeds up the algorithm. The main disadvantage of this bottom-up strategy is that the pixel error has to be recalculated for all the vertices of the terrain data, and the computation for this is somewhat costly. In addition, it is theoretically possible to use geo-morphing to avoid vertex popping, but it is not implemented in the current version.

Duchaineau [19] proposed an algorithm to generate real time optimally adapting meshs (ROAM). ROAM uses a splitting diamond queue and a merging diamond queue to make progressive updates in a triangle mesh. The splitting and merging operations are based on triangle diamonds. The algorithm exploits much of coherence to maintain rather stable frame rates. ROAM uses an explicit binary tree somewhat similar to Lindstrom's binary vertex tree. However, Jonathan [20] pointed out that ROAM is very inefficient at high detail levels.

Röttger [21] also proposed an algorithm that uses a top–down strategy to create a triangulation and exploits geo-morphing to avoid vertex popping at virtually no additional cost. Using a top-down strategy, only a fraction of the entire data set needs to be visited in each frame. Röttger developed a calculation method to guarantee that the maximum difference between two adjacent nodes will never exceed one, and thus it is easy to eliminate cracks.

3. Motivation

For the ease of comparison between traditional dexel representation and our approach, a brief introduction to the traditional dexel representation is given here. As shown in Fig. 1, the workpiece is sub-divided into many even dexels. Each dexel is sub-divided into two triangles. The heights of the vertices of these dexels vary during the milling process simulation. This leads to the changes in the upper surface of the workpiece. This approach is easy to be implemented, but the rendered image quality is poor due to a large dexel size.

In order to overcome the drawback of the traditional dexel representation, an approach should be proposed to satisfy the following criteria:

(1) *Self-adaptation*. Different areas of the upper surface of a workpiece have different appearances. To reduce the number of polygons without much loss in the rendered image quality, the polygon densities of different areas should vary dynamically. Rough areas with high frequency data, such as convexities and concavities require a high resolution (high polygon density), and flat areas only require a low resolution (low polygon density).
(2) *Good trade-off*. Although a relatively small number of polygons would lead to a short rendering time, the mesh decimation requires extra computation time. Hence, a good trade-off between the decimation time and the number of polygons should be achieved.
(3) *Low sensitivity*. The upper surface is changed after each tool movement, but this change is only confined to the tool path. The changes of the heights of the local vertices, leading to a reconstruction of the upper surface of the workpiece, should not have a significant impact on the performance of the simulation.
(4) *Locality*. High frequency data, such as the brims of tool paths, convexities and concavities, should not have a widespread global effect on the complexity of the model. That is, high-density polygons should only concentrate on the areas around the high frequency data.
(5) *Fast indexing*. At any time, the data describing a mesh geometry should be directly and efficiently accessible, allowing fast spatial indexing of both the polygons and the vertices.

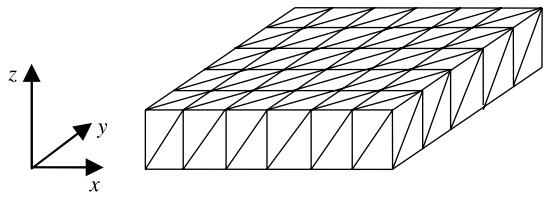

Fig. 1. Dexel representation of three-axis NC milling simulation.

A polygon-based level-of-detail algorithm that supports all these criteria is clearly important in the implementation of NC milling simulation that requires high frame rates and high visual fidelity, as well as fast and easy queries of the data of the milling surface.

Most approaches to the level-of-detail management fail to meet at least one of these criteria. In general, the level-of-detail models of irregular meshes that are widely used do not meet the second, third and fifth criteria. Generation of a moderate-sized irregular mesh requires extensive computational effort. Since multi-resolution irregular mesh representations are non-uniform in nature, the intersection between a cutter and a workpiece is difficult to be handled efficiently due to a lack of a spatial organization of the mesh polygons, such that they cannot have high performance for real-time simulation.

Since the traditional dexel representation mentioned above has a regular grid representation, it has the common drawbacks of regular grid representations where the polygonalization is seldom optimal, or even near optimal. In the dexel representation, large, flat areas of the upper surface require the same polygon density as small, rough areas. This results in a large number of polygons. This drawback can be overcome using the algorithm presented in this paper. Our approach satisfies all the above criteria. Some key features of the algorithm include: flexibility and efficiency resulting from a regular grid representation, localized polygon densities (Fig. 2), and a continuous level of details.

There are few explorations of level-of-detail management used in NC milling simulation. Lee and Lee [22] proposed a local mesh decimation algorithm for view-independent three-axis NC milling simulation. This algorithm can reduce the number of polygons, but it has some drawbacks. At the initial stage, it uses the dexel representation; however, after an overall cutting of the upper surface, the mesh becomes irregular, which results in the drawback of the level-of-detail model of an irregular mesh mentioned above.

Our implementation of the three-axis NC milling simulation uses the height field, and hence some techniques on terrain surface rendering are used in our algorithm, for example, using a quadtree to organize the data and achieve the adaptive representation. However, the height field of a workpiece is always varying during a machining process, and this is different from a terrain surface. Hence, certain surface rendering techniques cannot be used directly in our implementation. These techniques need to be modified according to the need of NC milling simulation, e.g. the data structure of the quadtree is different from other algorithms, and operations on a mesh are also not completely the same as terrain surface rendering, the preprocessing technique cannot be used, etc.

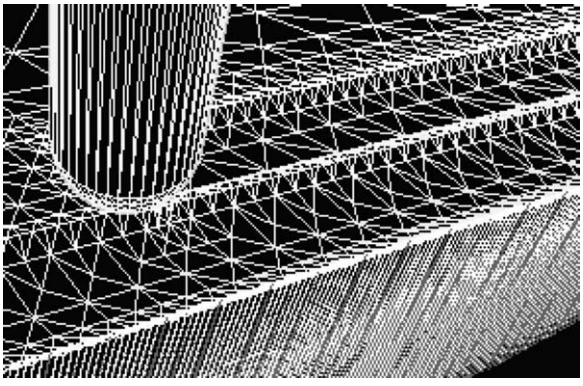

Fig. 2. Regular mesh decimation for three-axis NC milling simulation (partially enlarged detail).

4. Our approach

4.1. Basic idea

In order to implement the algorithm easily, a stipulation is given first: the $x-y$ projection of a workpiece is a square. Details are described in Section 4.8.

A quadtree is used to describe the level-of-detail model of the upper surface of a workpiece (Fig. 2). The logical structure of a quadtree is an inverted tree (Fig. 3). Each node of the quadtree has at most four child nodes, and each child node can also have its own child nodes. If a node does not have child nodes, this node is a leaf node. The node that represents the whole tree is a root node, and the others are intermediate nodes.

As shown in Fig. 3, the upper surface of a workpiece is first sub-divided into four even sub-squares, and each sub-square will be recursively sub-divided into four smaller sub-squares. Each sub-square corresponds to a node of the quadtree. Each node stores the data of a corresponding sub-square (Fig. 4), including the enabled flags which determine whether a sub-square should be sub-divided (inserted), fused (deleted) or optimized, i.e. which vertices are used to generate the output

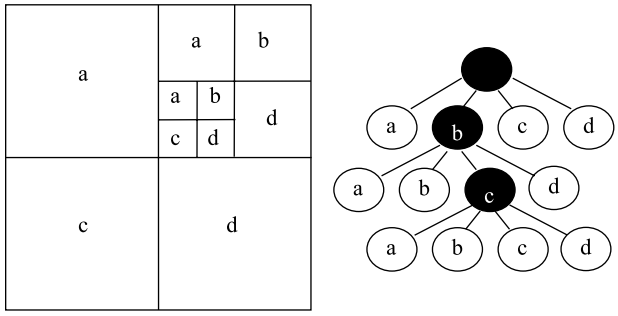

Fig. 3. Relationship between sub-squares and nodes of a quadtree.

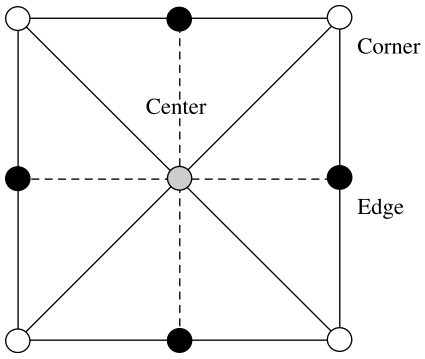

Fig. 4. Data of a node (sub-square) including enabled flags.

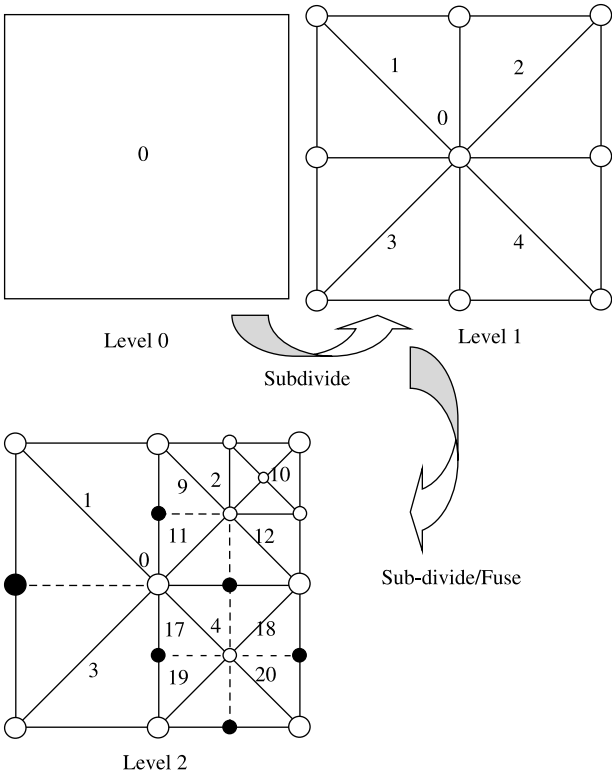

Fig. 5. Processes of sub-division, fusion or optimization.

triangle mesh. In Fig. 4, the gray vertex is a center vertex, the black vertices are edge vertices, the white vertices are corner vertices, and a node has nine vertices in total. Fig. 5 shows the process of sub-division and fusion. The numbers are the indices of the corresponding nodes (please refer to Section 4.2). The white vertices are the enabled vertices to generate the output triangle mesh. The black vertices are the disabled vertices. In the middle mesh in Fig. 5, the node 0 is sub-divided into four child nodes 1, 2, 3 and 4. In the Level 2 mesh in Fig. 5, the nodes 1 and 3 are fused; the nodes 2 and 4 are sub-divided, and then optimized. Dashed lines are optional in a level-of-detail mesh. Using the enabled flags of the sub-squares, self-adaptation can be achieved. Sub-division (insertion), fusion (deletion) or optimization is only needed to enable or disable certain vertices of the sub-squares instead of the actual corresponding operations on the nodes. This increases the efficiency of these operations on large-scale datasets, because allocating, freeing and searching mass data are not efficient. An evaluating criterion is used to determine which sub-squares should be further sub-divided, and which sub-squares should be fused or optimized.

4.2. Organization of nodes

The spatial organization of the nodes of a quadtree is a hierarchical *Z*-ordering indexing scheme as shown in Fig. 6. In Fig. 6, the numbering order is like the alpha *z*. Node i has four children: $4i+1$, $4i+2$, $4i+3$ and $4i+4$. The dashed arrows indicate the difference between the indices of two neighboring nodes. Positive number indicates that the index of the right

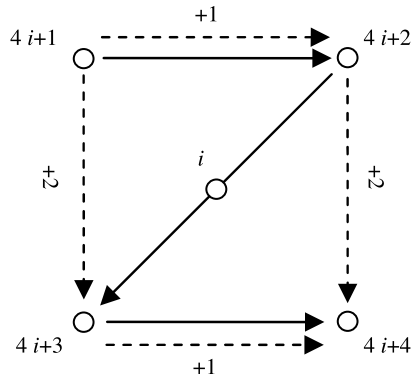

Fig. 6. *Z*-ordering indices of nodes.

node is greater than that of the left neighboring node, and the index of the bottom node is greater than that of the top neighboring node. Enquiring, inserting, deleting, optimizing or traversing operations on the nodes can be performed quickly using this spatial organization. Since the difference between the indices of two neighbouring brother nodes is a constant, which is one in the horizontal direction and two in the vertical direction, it is very easy to enquire the indices of the neighbouring nodes of a node.

Assumed that the depth of a complete quadtree is n ($n>0$), the quadtree has $\sum_{l=0}^{n-1} 4^l$ nodes, and the indices of these nodes are 0, 1, 2,…, $(4^n-1)/3-1$, as shown in Fig. 7. In Fig. 7, the depth of the quadtree is three. The circles denote the nodes of the quadtree. The numbers denote the indices of the quadtree. The left and top arrows denote the incremental directions of the indices. The positive numbers denote the difference between two neighboring vertices along the incremental directions. The dashed lines denote the grid model of the upper surface of a workpiece. Given a node with an index $i>0$, the indices of its

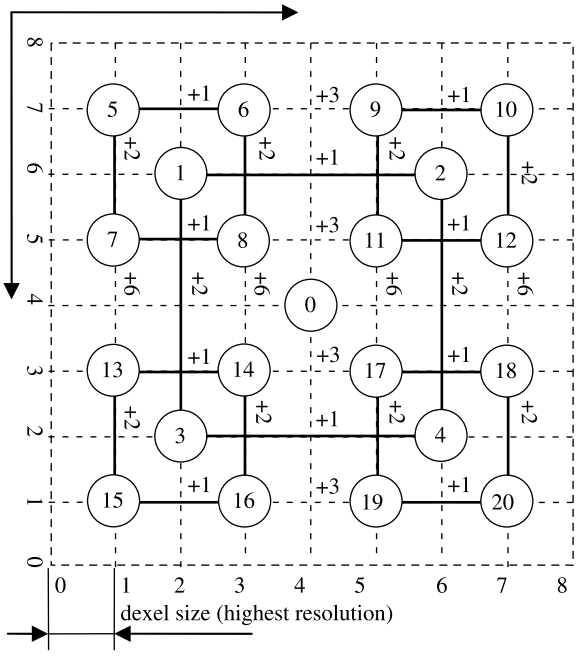

Fig. 7. A complete quadtree with *Z*-ordering spatial organization.

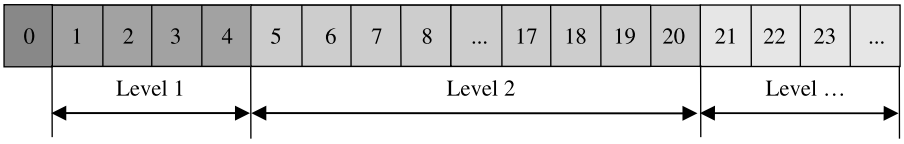

Fig. 8. A quadtree is stored as a one-dimensional array. The numbers are the indices of the nodes. 0 is the root node.

parent and children, and its level can be calculated using the following equations:

The index of its parent: $[(i-1)/4]$;
The indices of its four children: $4i+t$, where $t=1,2,3,4$;
The level of the node: $[\log_4(3i+1)]$.

In Fig. 7, the incremental difference in a same row or column forms an incremental vector of the indices. The incremental vector of the indices can be solved using the following iterative calculation:

(1) Horizontal incremental vector of indices:

$$\Delta_h(1) = [1],$$

$$\Delta_h(l) = \left[\Delta_h(l-1),\ \frac{2}{3}4^{l-1}+\frac{1}{3},\ \Delta_h(l-1)\right],$$

where l is the level number of nodes, $l=2,3,4,\ldots$

(2) Vertical incremental vector of indices:

$$\Delta_v(1) = 2\Delta_h(1) = [2],$$

$$\Delta_v(l) = 2\Delta_h(l),$$

where l is the level number of nodes, $l=2,3,4\ldots$

For instance (Fig. 7): $\Delta_h(1)=[1]$, $\Delta_v(1)=[2]$; $\Delta_h(2)=[1,3,1]$, $\Delta_v(2)=[2,6,2]$. The incremental vectors are calculated in the initial stage and used to quickly find the neighbouring nodes of a given node.

4.3. Storage of nodes

Some methods have been developed to store the quadtrees, but most of their goals are to reduce the memory requirement, i.e. to avoid data redundancy. In order to increase the speed of determining the indices of the nodes and implement the program conveniently, a one-dimensional array is used to store the nodes of the quadtree (Fig. 8), and a two-dimensional array (Fig. 7) is used to store the heights of the vertices of the nodes. Each node of a quadtree consists of three integer numbers, namely, two integer numbers for the address (x, y) of the corresponding two-dimensional array element, and one integer number for the enabled flags of the node, and other necessary data. Each element of the two-dimensional array stores the height of a corresponding vertex and the index of a node (while index exists). In Fig. 7, the leftmost numbers and the bottommost numbers are the addresses of vertices in the two-dimensional array. In this way, although data redundancy still exists to some degree, the data needed can be accessed easily and quickly.

4.4. Eliminating cracks

When a quadtree is used to build a mesh model, cracks occur on the border of two neighbouring different resolution nodes. As shown in Fig. 9, Node N_1 has a higher resolution as compared to the neighbouring nodes N_2 and N_3, which makes the borders of (N_1, N_2) and (N_1, N_3) uncovered, forming holes which are cracks. In order to prevent cracks, some rules must be observed.

First, any combination of the four quadrants of a node can be sub-divided. When a quadrant is sub-divided, the quadrant is treated as a sub-square, and its centre vertex is enabled. For mesh consistency, the edge vertices of the parent square, which are the corner vertices of the quadrant, will have to be enabled as well (Fig. 10). Enabling a square implies the enabling of its centre vertex as well as those corner vertices. In Fig. 10, the white vertices are already known to be enabled, but the gray vertices must be enabled when the SE quadrant is to be sub-divided. The bold lines are added after sub-division.

Next, it can be observed that an edge vertex in a sub-square is shared with a neighbouring sub-square (except at the outside edges of the model). Hence, when an edge vertex is enabled, the neighbouring sub-square that shares this vertex must also

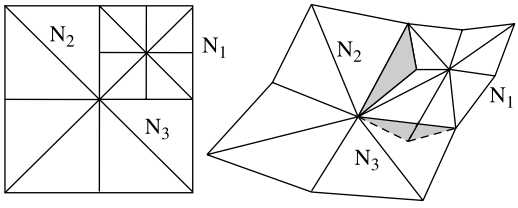

Fig. 9. Cracks on the border of two different resolution nodes are denoted in gray colour. Nodes (N_1, N_2) and (N_1, N_3) have different resolutions.

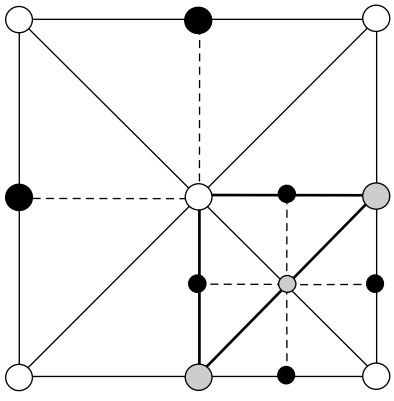

Fig. 10. Sub-dividing the SE quadrant of a square.

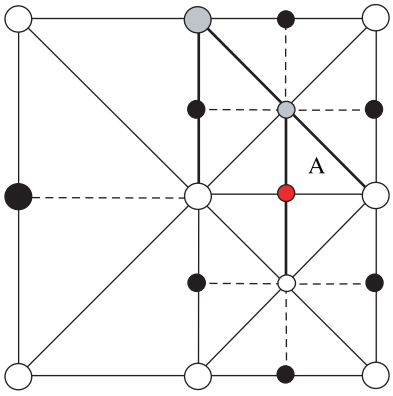

Fig. 11. Enabling the vertices of neighboring sub-square.

be enabled (Fig. 11). Enabling this neighbouring sub-square can in turn cause other vertices to be enabled, potentially propagating enabled flags through the quadtree. This propagation is necessary to ensure mesh consistency, but the propagation is usually confined in a local area. For example, in Fig. 11, while updating the SE quadrant, Vertex *A* has to be enabled. Since this vertex is also shared with the NE quadrant, that quadrant must also be enabled. Enabling the SE quadrant will in turn force the gray vertices to be enabled. The bold lines are added after all the affected vertices are updated.

4.5. Inserting nodes

After each tool movement, the heights of the vertices within the envelope of the tool path change. In order to maintain the quality of the rendered image of the upper surface of a workpiece, all the nodes intersecting with the tool path envelope will be inserted with higher resolution nodes. The insertion process is performed from the leaf nodes to the upper level nodes, i.e. first the leaf node, next the parent node, then the grandparent node and so on. For instance in Fig. 12, the gray rectangle denotes a corner vertex of Node A. B is the parent node of A, and C is the parent node of B. In this figure, a leaf node intersects with the tool path envelope, and the height of the NE

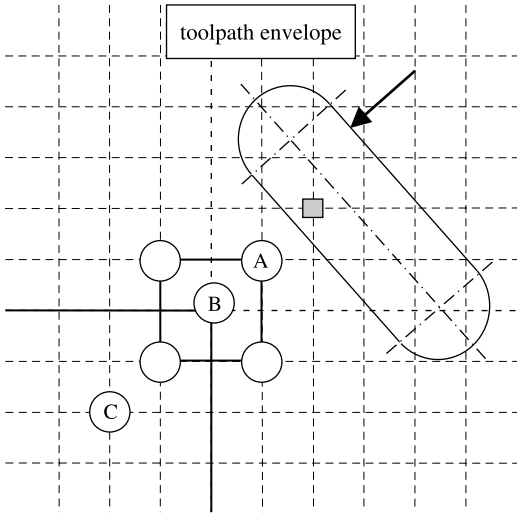

Fig. 12. Enabling a vertex will propagate this process.

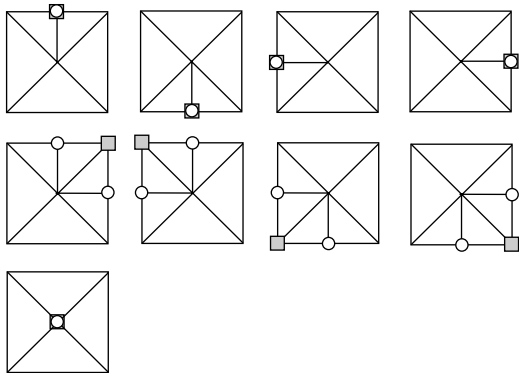

Fig. 13. Nine cases in enabling the vertices of leaf nodes. The gray rectangles denote the inserted vertices. The white circles are the vertices to be enabled.

corner vertex is changed. In order to ensure the quality of the rendered image and avoid cracks, the right edge vertex and the top edge vertex of the node A must be enabled. The enabled flags of the parent node B, the grandparent node C and so on must be modified correspondingly. Hence, the enabled flags of nodes A, B and C are affected by enabling the corner vertex. Fortunately, this propagation process is local, rather than global. Insertion operations can be divided into two categories: one is to enable the vertices of a node (Fig. 13); and the other is to insert a higher resolution node into its parent node (Fig. 14).

Since all the nodes, from the highest resolution to the lowest resolution, have been created in the initial stage of a quadtree, the real insertion operations do not exist but are replaced by the enquiring of the nodes and the enabling of vertices. In this way, the frequent memory allocation for the newly inserted nodes can be avoided.

4.6. Deleting nodes

4.6.1. Evaluating vertices

To obtain a good approximation of the grid model and achieve fast rendering, an evaluation criterion must be established to determine whether a vertex should be disabled, i.e. to determine whether the enabled flag of a vertex should be set to false. In general, when a machinist observes a milling process, he usually views the whole workpiece. In this case, the nodes are far from the viewpoint and only high-level nodes (large sub-squares) are needed to approximate the upper surface. After the completion of milling process, the machinist usually would like to zoom in the upper surface to examine a certain area. In this case, the nodes are near to the viewpoint, and the nodes located in this area should be sub-divided to give a higher resolution while the other nodes can be culled. In order to reduce the number of triangles to the utmost while maintaining high fidelity, the evaluating criterion should be

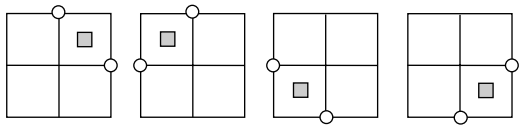

Fig. 14. Four cases in inserting nodes into non-leaf nodes. The gray rectangles denote the inserted nodes. The white circles are the vertices to be enabled.

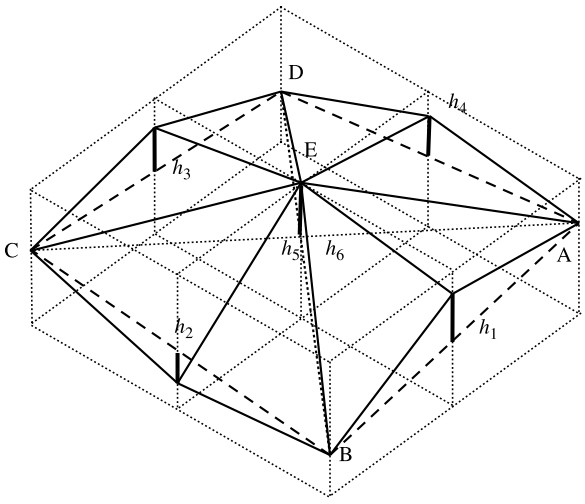

Fig. 15. Vertex interpolation errors Δh_1–Δh_6 denoted by bold line segments. The vertex interpolation error of center vertex is the maximum value of $\Delta h_5 = |h_E - \frac{(h_A+h_C)}{2}|$ and $\Delta h_6 = |h_E - \frac{(h_B+h_D)}{2}|$ where Δh_5 and Δh_6 are probably not equal.

determined based on two factors, namely, the vertex interpolation error (Fig. 15) and the view-dependent factor.

As shown in Fig. 15, when a vertex is to be disabled, the shape of mesh is changed. The maximum change occurs at the position of the disabled vertex, shown by the bold line segments. The magnitude of this change is the difference between the true height of the vertex and the height of the mid-point of the dashed line segment. This magnitude is called the vertex interpolation error, denoted by Δh_i. As the vertex interpolation error increases, the probability for triangle fusion decreases. In other words, bumpy areas are rendered with small triangles while flat areas are rendered with large triangles.

The view-dependent factor also affects the criterion to disable a vertex. Many level-of-detail algorithms consider the direction from the viewpoint to the vertex. The justification is based on the screen-space geometric error. However, the screen-space geometric error is not a particularly good metric since it ignores texture perspective and depth buffering errors [23]. To minimize the projected pixel error, which is a good measure to improve the image quality, and allow for efficient computations, the L-norm is used to perform distance measurement instead of the screen-space geometric error. It results in a 3D view-space error proportional to the view distance. The evaluation criterion involves only three parameters, namely, an approximation of the viewpoint-vertex distance, the vertex interpolation error, and a detail threshold constant, given in Eq. (1)

$$l_i = \max(|\text{vertex}.x - \text{viewpoint}.x|, |\text{vertex}.y - \text{viewpoint}.y|, |\text{vertex}.z - \text{viewpoint}.z|)$$
$$f_i = \left(\frac{\Delta h_i}{l_i} \le \delta\right), \tag{1}$$

where vertex.x, vertex.y and vertex.z are the x, y and z coordinates of the vertex v_i, respectively; viewpoint.x, viewpoint.y and viewpoint.z are the x, y and z coordinates of the viewpoint, respectively; Δh_i is the interpolation error of the vertex v_i; δ is a detailed threshold constant. If f_i is true, the vertex v_i can be disabled.

4.6.2. Deletion operations

Deletion operations are similar to the insertion operations in some aspects. These deletion operations are accomplished through the enquiries of the nodes and disabling the vertices instead of actual deletion operations, and some enabled flags of the related nodes will be modified to avoid cracks. Deletion operations can be divided into two categories: (1) a node can be deleted entirely, i.e. all the four edge vertices and the centre vertex are disabled; and (2) at least one edge vertex is disabled. In the second case, a deletion operation is an optimization process and two adjacent triangles will be fused into one large triangle.

The deletion process is evaluated using the evaluation formula in Eq. (1). During the deletion process, each edge vertex of a node is first checked, and the deletion operations must observe the following rules:

(1) If an edge vertex has already been disabled, this vertex is not evaluated, and the next vertex will be checked.
(2) If an edge vertex has already been enabled and its evaluation is true, then this edge vertex is disabled.
(3) When four edge vertices have already been disabled, the centre vertex of the node will be checked; if the evaluation of this centre vertex is true, the centre vertex is disabled, and this node can be deleted.
(4) If an adjacent brother node of a deletable node can also be deleted, the corner vertex v shared by these two brothers will be enabled (Fig. 16). This shared vertex is also an edge vertex of their parent node. The enabled flags of the vertices of the adjacent nodes of their parent node also need modification to avoid cracks. This process will be propagated, but this modification process is usually confined to a local area.
(5) If both adjacent nodes A and B (Fig. 16) of the deletable node D can be deleted, the vertices of the third brother node C will be evaluated. If this third brother node can also be deleted, their parent node will be evaluated to determine if it can be deleted.

Deletion operations begin from the inserted leaf nodes which are stored in an array during the period of insertion

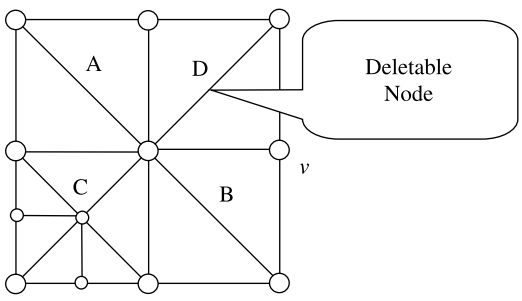

Fig. 16. Deletion operations. The deletable nodes D and A, B, C are brothers. Vertex v is the corner vertex of B and D, and it is also an edge vertex of their parent.

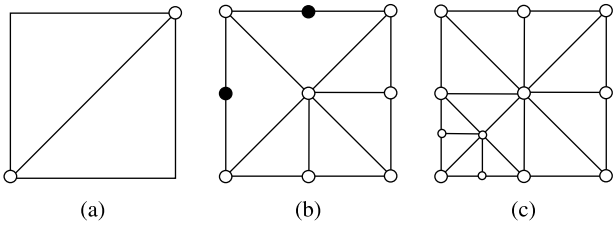

Fig. 17. Triangulation of nodes. Enabled vertices in white, disabled vertices in black.

operations, and deletion operations will be performed only when the inserted nodes have accumulated to a certain number.

4.7. Mesh rendering

When all the enabled flags of the vertices have been defined, the mesh can be constructed for rendering. There are three cases in triangulating a node. The triangulation process is accomplished by traversing a quadtree, and it is a post-order traversal process.

(1) If a node is deletable, this node can be sub-divided into two triangles (Fig. 17(a)).
(2) If a node is a leaf node, the triangulation is performed according to its enabled flags. A triangle fan with the centre vertex as the hub is made, and includes each enabled vertex in a clockwise order around outside (Fig. 17(b)).
(3) If a node is non-leaf node and at least one child node cannot be deleted, rules (1) and (2) are first applied to its child nodes recursively followed by itself (Fig. 17(c)).

4.8. Workpiece discretization

A workpiece is usually a rectangular solid (Fig. 18). However, a quadtree can only represent a discrete square. Hence, the workpiece needs to be standardized. The width w is multiplied with a scale constant f to make it equal to the length l ($w < l$).

After the standardization of the workpiece, it will be discretized. Since a quadtree has its own characteristics, the total number of rows and columns must be equal, and the discretization density cannot be selected arbitrarily. The discretization density must be equal to 2^{n-1}, where n is the depth of the quadtree. Assumed that the size of a model is $l \times l$ units, the resolution is $x_{res} = y_{res} = l/2^{n-1}$ units. In fact, the real resolution in the width direction is $y_{real} = y_{res}/f$ units.

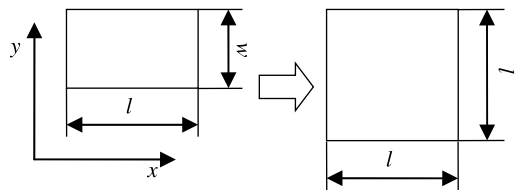

Fig. 18. Standardization of a workpiece.

4.9. Steps of NC milling simulation

The adaptive mesh decimation for three-axis NC milling simulation is described as follows:

Step 1. Standardize and discretize a workpiece, store the height of each vertex of the standardized workpiece in a two-dimensional array.

Step 2. Build the relationship between a quadtree and the two-dimensional array, and initialize them.

Step 3. Rebuild the model of the workpiece. When milling is in progress, the heights of vertices are changed, and extra operations are performed:

(1) Insert high-resolution nodes to the nodes that intersect with the tool path envelope (in fact, it is not necessary to solve the tool path envelope. An alternative method is used in our implementation. Please refer to Appendix B.), update the enabled flags of the affected vertices, and use a one-dimensional array to record the indices of the inserted leaf nodes.
(2) When the array is full, delete the nodes recorded in the one-dimensional array, and clear the array.
(3) According to the enabled flags, construct the triangle mesh for rendering.

Step 4. Repeat Step 3 until the milling simulation ends.

5. Experimental results

An example of the rough cutting of a craftwork was presented to analyze the number of remaining triangles, computation and rendering time and frame rates. Fig. 19 is the original picture used to generate the NC milling G codes. Fig. 20 shows the wire frame of the mesh based on dexel representation. Since the density of the vertices is too high, the mesh of the workpiece appears as if it had been rendered.

Fig. 19. Original picture used to generate the NC milling G codes.

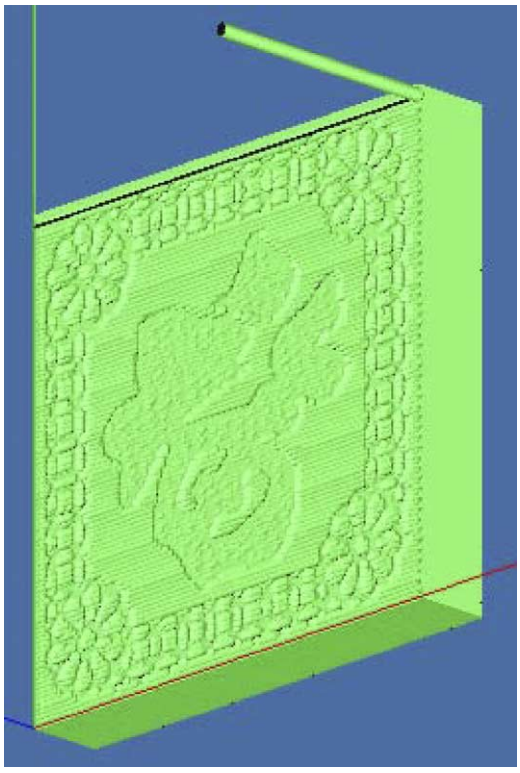

Fig. 20. Wire frame of the mesh using dexel representation.

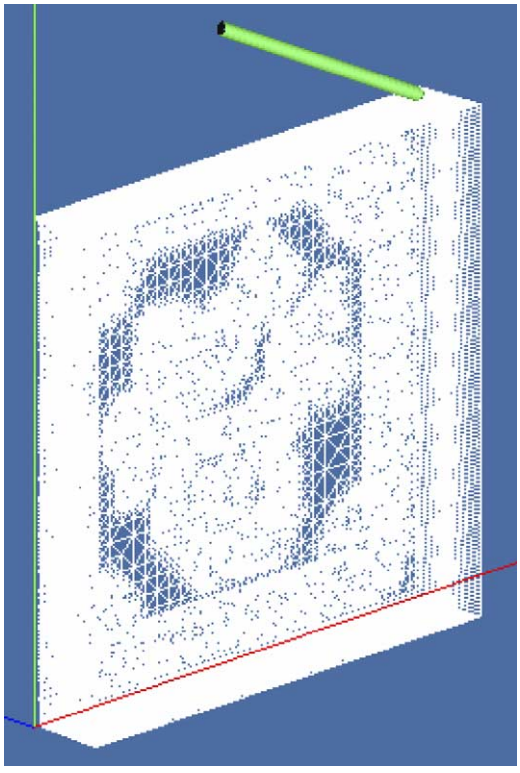

Fig. 21. Wire frame of the mesh using the adaptive LOD representation.

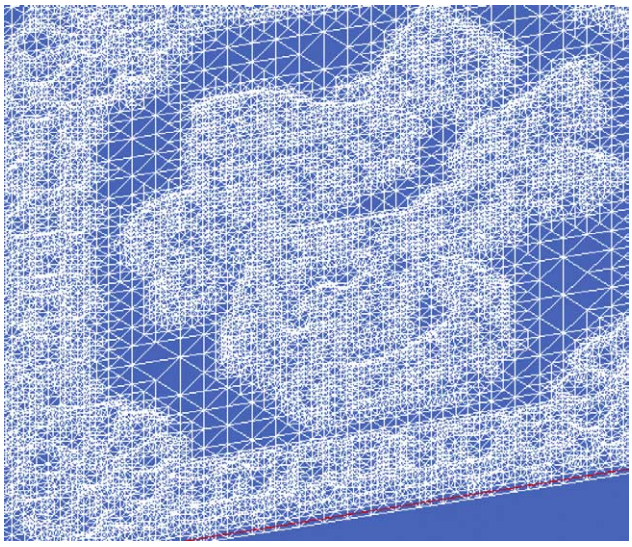

Fig. 22. Wire frame of the upper surface using the adaptive LOD representation. All the other surfaces have been removed.

Figs. 21 and 22 show the wire frames of the mesh based on the adaptive level-of-detail representation proposed in this paper. In order to view the details of the mesh clearly, only the wire frame of the enlarged upper surface is shown in Fig. 22, and all other surfaces have been removed.

The initial resolution of the mesh was 512×512 units after the workpiece standardization. The personal computer used for this experiment was equipped with an Intel Pentium 4 Willamette 1.72 GHz processor, an Intel 82845G graphics controller and a 512 MB memory stick. The radius of the ball-end cutter was six units. The NC milling program had a total of 7700 blocks of tool movement commands. All the data were sampled at no. 7000 block of tool movement commands except for special declaration.

The number of remaining triangle was first examined as a function of the threshold δ. The workpiece was completely accommodated in the field of view frustum, and was not rotated and moved during the milling operation. The size of the one-dimensional array to store the inserted leaf nodes for later

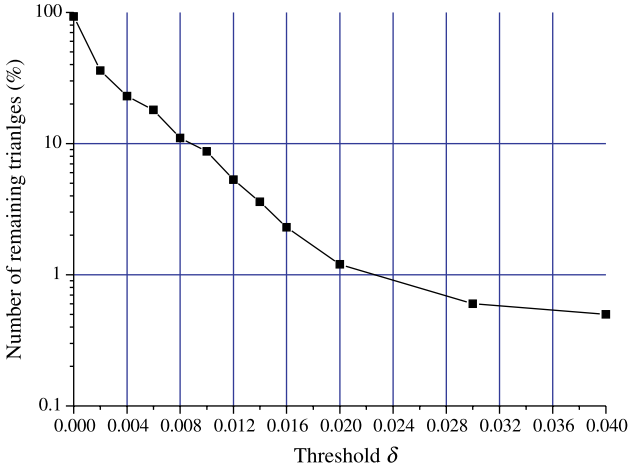

Fig. 23. The number of remaining triangles as a function of δ.

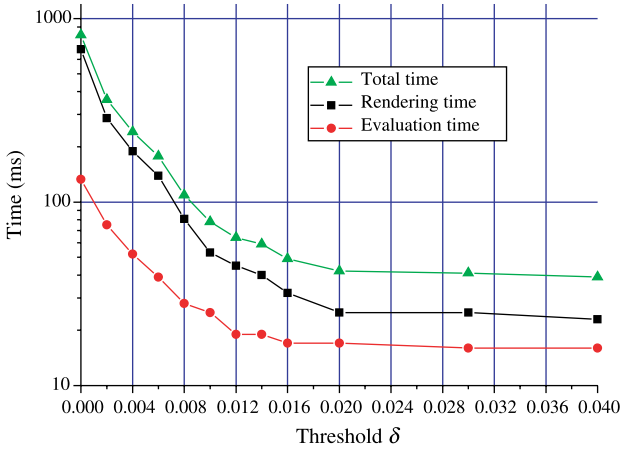

Fig. 24. Rendering and evaluation times and their sum as functions of δ.

deletion was 1000. Fig. 23 shows the ratio of triangle reduction as a function of the threshold δ. The curve is drawn on a logarithmic scale (vertical axis). As the threshold increases from 0 to 0.04, the total number of triangles rendered decreases, i.e. the remaining triangles decreases after simplification. At $\delta=0$, only coplanar triangles are fused. As the threshold increases, the curve flattens out.

Fig. 24 demonstrates the variation of the evaluation and rendering times as the threshold δ increases. Evaluation time is the total time to evaluate the vertices of the nodes affected by the tool movement to determine whether these vertices should be disabled and change the states of corresponding enabled flags (refer to Section 4.6). Comparing the curve of the rendering time with the curve of the evaluation time, it can be seen that the evaluation time is only a small fraction of the rendering time.

Fig. 25 shows how the frame rate varies as the simulation proceeds. Analyzing the trend of the curve, a frame rate of approximately 15 fps was sustained eventually. The variation of the computation time and the number of triangles has resulted in the fluctuation of the curve.

In addition, the rendering frame rate of the adaptive level-of-detail mesh is compared with the traditional dexel grid.

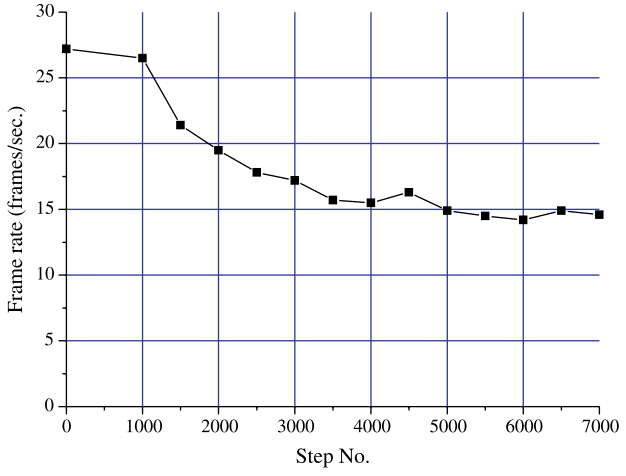

Fig. 25. The frame rate of NC milling simulation as a function of step number ($\delta=0.01$).

When threshold $\delta=0.01$, the frame rate of the former basically sustains 15 fps. However, the frame rate of the dexel grid only has 1.2 fps.

6. Conclusion

In this paper, an adaptive regular mesh decimation algorithm that can achieve high frame rates for three-axis NC milling simulation is proposed. In summary, the advantages of the algorithm proposed in this paper are as follows:

- Since the total number of triangles for rendering has been reduced heavily and the triangle density can be adjusted, this algorithm can achieve optimization for fast rendering.
- As a direct relationship between the single threshold δ and the number of triangles to be rendered exists, this algorithm provides the capabilities for sustaining consistent frame rates.
- Using the algorithm proposed in this paper, any area of the machined workpiece can be inspected with full resolution, such as an overcut, undercut or error assessment, etc. (refer to Appendix A). However, it is difficult to achieve these inspections using image-based algorithms.
- During a milling process simulation, the workpiece can be translated, rotated, zoomed in/out, or measured, and this is especially important for virtual reality or augmented reality applications. These advantages are not possessed by the image-based algorithms that are widely used in many commercial application softwares. In a virtual reality or augmented reality environment, the machinist will observe the milling process from different directions, and the distance between the machinist and the workpiece is always varying. Using an image-based algorithm would require re-computation if the viewing direction has been changed, and this will hamper the use of these image-based algorithms in the virtual reality or augmented reality environments.

Since a change of view point does not need to be considered, sometimes it is more convenient to use a view-independent algorithm. Although the method of evaluating the vertices (Section 4.6.1) is view-dependent in our algorithm, it is easy to change this method into a view-independent evaluation method, using the interpolation error Δh as the unique evaluation parameter.

Appendix A. Brief introduction of overcut and undercut

From an engineering point of view, a machinist is more concerned about the numerical inspections, i.e. the overcut, undercut and error assessment between the machined workpiece and the master model. Corresponding values should be provided to analyze if these cases have happened. Related work can be found in [24]. Although these topics are not the main focus of this paper, these inspections can be easily implemented using the data structures presented in this paper. Therefore, the overcut, undercut, error assessment are discussed briefly. The full resolution model will be used for inspection.

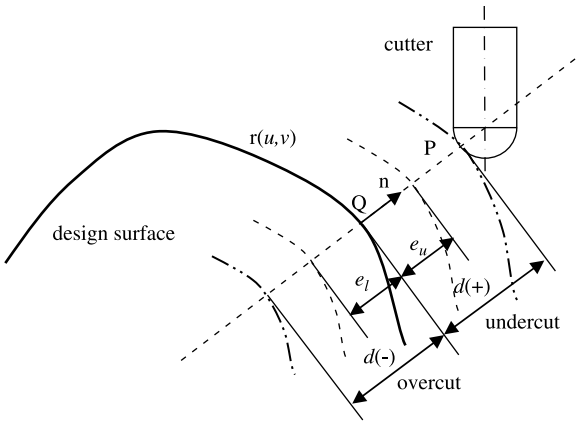

Fig. A1. Numerical inspections: overcut and undercut between the machined workpiece and the master model.

As shown in Fig. A1, the directed distance d (called the local error) is the distance from point Q of the design surface $r = r(u,v)((u,v) \in D)$ to vertex P of the machined workpiece, where Q is the point closest to P. Assumed that the range of mismachining tolerance is $[e_l, e_u]$, the rules for determination of overcut and undercut are:

(1) If $d < e_l$, then an overcut exists in this vertex.
(2) If $d > e_u$, then an undercut exists in this vertex.
(3) If $e_l \le d \le e_u$, then this vertex meets the design technical equirement.

High-light colours can be used to indicate the regions of overcut and undercut, such as a red colour representing an overcut and a green colour representing an undercut.

The key point to verify the NC milling result is to calculate the directed distance d. Generally, the design surface is a continuous smooth surface, so the directed distance d can be calculated using numerical methods. Since the vector $\vec{QP}$ is vertical to the tangent plane of the surface $\mathbf{r}$ at the point Q, the following nonlinear equations can be obtained:

$$\vec{QP} \cdot \frac{\partial \mathbf{r}}{\partial u} = 0, \ \vec{QP} \cdot \frac{\partial \mathbf{r}}{\partial v} = 0$$

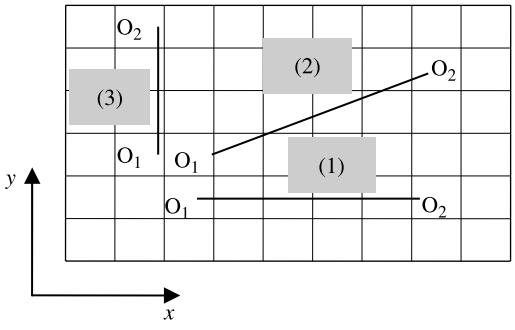

Fig. B1. Three cases of straight-line interpolation instruction G01.

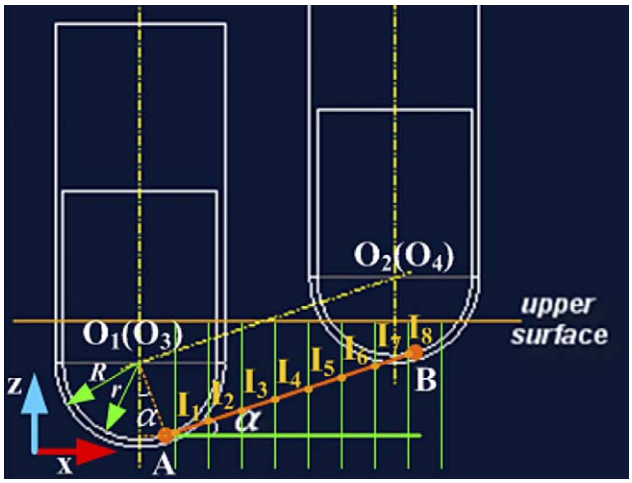

Fig. B2. Front view of a ball-end cutter.

Using the Newton–Raphson iteration algorithm, the coordinate of point Q can be obtained and the directed distance d can be determined immediately. Using the directed distance d, the mean error, maximum error and standard deviation can be determined too.

Appendix B. Brief introduction of the alternative method for tool path envelope

Since using the tool path envelope to perform Boolean operations is a very slow method, and hence an alternative approach is used in our implementation. A brief introduction is given here. As shown in Fig. B1, the tool path is from O_1 to O_2, where O_1 and O_2 are the centres of a ball-end cutter. On the projection plane $x-y$, there are three cases of straight-line interpolation instruction G01: (1) O_1O_2 is parallel to X-axis; (2)

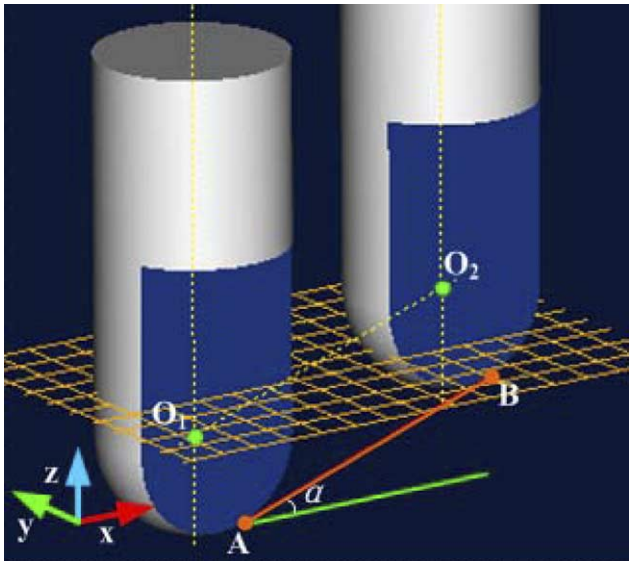

Fig. B3. The shaded result of a ball-end cutter.

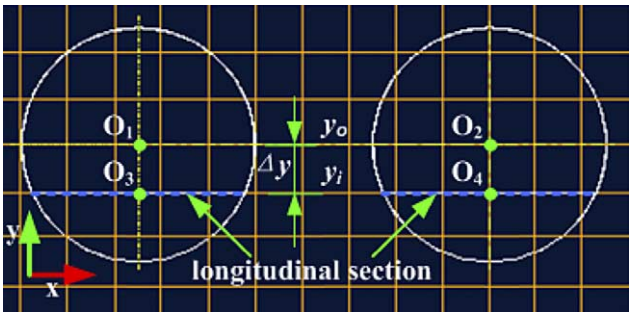

Fig. B4. Top view of a ball-end cutter.

O_1O_2 is neither parallel to X- nor Y-axis; and (3) O_1O_2 is parallel to Y-axis. Case (1) is taken as an example to illustrate the alternative approach.

Fig. B2 is the front view of a ball-end cutter, Fig. B3 is the shaded result, and Fig. B4 is the top view. In these figures, O_1 and O_2 are the centres of the ball-end cutter, while O_3 and O_4 are the centres of the half circle on the longitudinal section. In Fig. B2, when the tool moves from $O_1(x_1, y_1, z_1)$ to $O_2(x_2, y_2, z_2)$, where z_1 is likely not equal to z_2, the heights of the vertices falling into the tool path envelope will be changed.

For example, in Fig. B4, the heights of the vertices of row y_i are determined using the half circle on the longitudinal section. Since the distance Δy between the parallel lines O_1O_2 and O_3O_4 is known, the radius r of the half circle on the longitudinal section can be solved immediately. In Fig. B2, the line segment AB is tangent to the start and end of the half circles, A and B are tangent points, respectively. $AB//O_3O_4$, O_1 and O_3 are coaxial, and O_2 and O_4 are also coaxial. Hence, the slope rates $k_{AB} = k_{O_3O_4} = k_{O_1O_2}$. Since the coordinate of O_3 can be solved, the height of A can be obtained immediately. Next, the height of I_1 can be determined immediately using the slope rate k_{AB}. For those vertices between A and B, their new positions are $I_1, I_2, I_3, \ldots, I_8$. The X-direction intervals Δx of every two adjacent vertices ($I_1, I_2, I_3, \ldots, I_8$) are equal, hence $h_{I_{j+1}} = h_{I_j} + k_{AB}\Delta x \quad (j = 1, 2, \ldots, 7)$, where h_{I_j} is the height of vertex I_j. Obviously, the heights of the vertices $I_2, I_3, \ldots, I_8$ can be easily obtained using this incremental equation, and hence the computation effort is greatly reduced. For those vertices on the left side of I_1 and on the right side of I_8, their heights are the heights of the corresponding points of the half circle on the longitudinal section.

Computation of Case (2) is slightly complex, but the kernel is also incremental computation. For Case (3) of the straight-line interpolation instruction G01, the method to solve the new heights of the vertices is completely the same as in Case (1).

For circular interpolation instructions G02 and G03, one method is to approximate an arc with many line segments; another method is to use incremental method to determine the next position of the cutter, and copy the heights of the corresponding vertices in the previous position, or change the heights using an incremental equation.

References

[1] Chappel IT. The use of vectors to simulate material removed by numerically controlled milling. Comput Aided Des 1983;5(3):156–8.
[2] Hook TV. Real time shaded NC milling display. Comput Graph (Proc ACM SIGGRAPH 86) 1986;20(4):15–20.
[3] Huang Y, Oliver JH. Integrated simulation, error assessment, and tool path correction for five-axis NC milling. J Manufact Syst 1995;14(5): 331–44.
[4] Turk G. Re-tiling polygonal surfaces. Comput Graph (Proc ACM SIGGRAPH 92) 1992;26(2):55–64.
[5] Rossignac J, Borrel P. Multi-resolution 3D approximations for rendering complex scenes. Geometric modeling in computer graphics. Berlin: Springer; 1993. p. 455–65.
[6] Low K, Tan T. Model simplification using vertex-clustering. Symposium on interactive 3D graphics. Proceedings of ACM SIGGRAPH 97; 1997.
[7] Lindstrom P. Out-of-core simplification of large polygonal models. Comput Graph (Proc SIGGRAPH 2000) 2000;34:259–62.
[8] Hoppe H, DeRose T, Duchamp T, Mc-Donald J, Stuetzle W. Mesh optimization. Comput Graph (Proc SIGGRAPH 93). 1993:19–26.
[9] DeHaemer Jr M, Zyda M. Simplification of objects rendered by polygonal approximations. Comput Graph 1991;15(2):175–84.
[10] Eck M, DeRose T, Duchamp T, Hoppe H, Lounsbery M, Stuetzle W. Multiresolution analysis of arbitrary meshes. In: Proceedings of SIGGRAPH; 1995. p. 173–82.
[11] Soucy M, Laurendeau D. Multi-resolution surface modeling from multiple range views. In: Conference on computer vision and pattern recognition; 1992. p. 348–53.
[12] Soucy M, Laurendeau D. Multiresolution surface modeling based on hierarchical triangulation. Comput Vis Image Understand 1996;63(1): 1–14.
[13] Garland M, Heckbert PS. Surface simplification using quadric error metrics. Comput Graph (Proc ACM SIGGRAPH 92) 1992;209–16.
[14] Hoppe H. Progressive meshes. Comput Graph (Proc SIGGRAPH 96) 1996;30:99–108.
[15] Fowler RJ, Little JJ. Automatic extraction of irregular network digital terrain models. Comput Graph (Proc SIGGRAPH 79) 1979;13(2): 199–207.
[16] Garland M, Heckbert PS. Fast polygonal approximation of terrains and height fields. Technical report CMU-CS-95-181. CS Department, CarnegieMellon University; 1995. http://graphics.cs.uiuc.edu/~garland/papers/scape.pdf
[17] De Floriani L, Puppo E. Hierarchical triangulation for multiresolution surface description. ACM Trans Graph 1996;14(4):363–411.
[18] Lindstrom P, Koller D, Ribarsky W, Hodges LF, Faust N, Turner G. Real-time continuous level of detail rendering of height fields. Comput Graph (Proc SIGGRAPH 96) 1996;109–18.
[19] Duchaineauy M, Wolinsky M. ROAMing terrain: real-time optimally adapting meshes. IEEE visualization '97 proceedings; 1997; p. 81–8. http://www.llnl.gov/graphics/ROAM
[20] Jonathan B. Terrain rendering at high levels of detail. Game developers' conference, San Jose, California, USA; 2000. http://www.bolt-action.com/papers/gdc2000_final.zip
[21] Röttger S, Heidrich W, Slasallek P, Seidel HP. Real-time generation of continuous levels of detail for height fields. Winter school in computer graphics (WSCG Proceedings 98); 1998. http://www.cs.ubc.ca/~heidrich/Papers/WSCG.98.pdf
[22] Lee SH, Lee KS. Local mesh decimation algorithm for view-independent three-axis NC milling simulation. Int J Adv Manuf Technol 2002;19: 579–86.
[23] Ulrich T. Continuous LOD terrain meshing using adaptive quadtrees; 2000. http://www.gamasutra.com/features/20000228/ultrich_pfv.htm
[24] Jerard RB, Drysdale RL, Hauck K, Schaudt B, Magewick J. Methods for detecting errors in numerically controlled machining of sculptured surfaces. IEEE Comput Graph Appl 1989;9(1):26–39.

S.Q. Liu is currently pursuing the PhD in Electro-Mechanical Engineering from Huazhong University of Science and Technology (HUST), China. He received his BS with honors in Mechanical Engineering from HUST in 1996. He was a senior IT researcher in the Centre of Computer Integrated Manufacturing System at Huabao Air-Condition Factory from 1996 to 1998. He joined the renovation project of material flow control system for Murata Manufacturing Co. Ltd., Japan, independently and successfully improved the task scheduling routine of stereoscopic warehouses and automated guided vehicles. The renovation team received the first-class award of technical achievement from Foshan government, China. From 1998 to 2000, he was an engineer in the Research and Development Department at Huabao Electronics Corporation, independently designed an automatic control system for a production line of electric water heater, and developed two series of monolithic-processor-control water heater. He invented a new valve for electric water heater, and has received the national patent. His research interests include computational geometry, CAD/CAM, feature modeling, and virtual reality applications in manufacturing.

Y.P. Chen is currently a professor of manufacturing automation, School of Mechanical Science and Engineering, Huazhong University of Science and Technology, China, since 1995. He received his BS and MS degrees from Shanghai Jiao Tong University in 1982 and 1984 respectively, and his PhD from Huazhong University of Science and Technology, China, in 1990. From 1991 to 1992, he worked as a visiting research fellow at University of Birmingham. From 1998 to 1999, he was a visiting scientist at National University of Singapore. His research interests include numerical control system, computer vision, intelligent manufacturing systems, virtual and augmented reality applications in manufacturing. He has published 3 books and over 200 papers in refereed journals and conference presentations. He has served as a reviewer for several international journals.

S.K. Ong has been an associate professor in the Manufacturing Division, Department of Mechanical Engineering, at the National University of Singapore since 1996. Her research interests are intelligent and distributed manufacturing systems, life cycle engineering, environment impact assessment, and virtual and augmented reality applications in manufacturing. She has attended and made presentations in many international conference. She has published two books, and over 80 international refereed journals and conference papers. She is the recipient of the Norman Dudley Award in 2002 for the best paper published in the International Journal of Production Research. In November 2003, she was selected to receive the 2004 M. Eugene Merchant Outstanding Young Manufacturing Engineer Award from the Society of Manufacturing Engineers. She is a corresponding member of CIRP. She has served on several international conference committees, invited reviewer of many journals in manufacturing and is an assistant editor of a book series on Manufacturing Systems and Technology published by World Scientific.

A.Y.C. Nee is a professor of manufacturing engineering, Department of Mechanical Engineering, National University of Singapore since 1989. He received his PhD and DEng from UMIST in 1973 and 2002 respectively. His research interest is in computer applications to tool, die, fixture design and planning, intelligent and distributed manufacturing systems, virtual and augmented reality applications in manufacturing. He has published 5 books and over 450 papers in refereed journals and conference presentations. He currently held regional editorship, department editorship, associate editorship and member of editorial board of 15 international journals in the field of manufacturing engineering. He is an active member of CIRP and an elected Fellow of the Society of Manufacturing Engineers, both in 1990.