Adv. Manuf. (2019) 7:64–72
https://doi.org/10.1007/s40436-018-00246-x

Tool-path generation for industrial robotic surface-based application

He Lyu[1] · Yue Liu[2] · Jiao-Yang Guo[2] · He-Ming Zhang[2] · Ze-Xiang Li[1]

Received: 19 May 2018 / Accepted: 19 December 2018 / Published online: 30 January 2019

Abstract Industrial robots are widely used in various applications such as machining, painting, and welding. There is a pressing need for a fast and straightforward robot programming method, especially for surface-based tasks. At present, these tasks are time-consuming and expensive, and it requires an experienced and skilled operator to program the robot for a specific task. Hence, it is essential to automate the tool-path generation in order to eliminate the manual planning. This challenging research has attracted great attention from both industry and academia. In this paper, a tool-path generation method based on a mesh model is introduced. The bounding box tree and kd-tree are adopted in the algorithm to derive the tool path. In addition, the algorithm is integrated into an offline robot programming system offering a comprehensive solution for robot modeling, simulation, as well as tool-path generation. Finally, a milling experiment is performed by creating tool paths on the surface thereby demonstrating the effectiveness of the system.

Keywords Industrial robot · Tool path generation · Simulation · Intelligent manufacturing

✉ He Lyu
hlv@ust.hk

[1] Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong, People's Republic of China

[2] Hong Kong University of Science and Technology, Shenzhen Research Institute, Shenzhen 518057, Guangdong Province, People's Republic of China

1 Introduction

In the present era of rapidly evolving automation, the manufacturing industries are compelled to constantly iterate and diversify the products through product innovations and flexible production lines to cope up with shortening product life cycles. At the same time, they are severely constrained by the shortage and high cost of skilled workers. In this challenging context, automation especially based on industrial robots is the best solution for enhanced productivity and flexibility. Nevertheless, programming of an industrial robotic system for a specific application is time-consuming and expensive. Particularly, surface-based operations such as painting, grinding, and milling are quite complex consuming excessive time and energy of workers.

Manual teaching is still a widely adopted method in robot programming. There have been many studies on this method to generate tool paths based on the contour of the workpiece [1, 2]. The teaching results largely depend on the knowledge and experience of the engineers on production facilities, equipment, processes, and tools. Trial and error methods need to be used for path planning, and the generated path is usually operator-dependent and error-prone. When it is necessary to consider performance standards, it is more difficult for engineers to find the path that can be used.

Although computer-aided tool-path planning such as CAD/CAM approach is often studied [3–5], it still caused a bottleneck for robotic applications.

Tool-path generation based on triangle mesh is a traditional research topic, and many methods have been proposed for this purpose. For example, mesh slicing is a widely used method for 3D printing [6]. The method is based on interception of the planes with mesh surface to obtain the tool path layer by layer. Therefore, the tool path

generated by this method is limited to only three dimensions. Iso-parametric tool-path generation is also a common method whereby the mesh surface is first fitted into a parametric surface, and then the tool path is generated by discretizing its parameters [7]. Iso-planner tool-path generation is another method widely used in commercial CAM software that obtains the path by finding the intersections of the cutting planes and the surface analytically either using surface and plane equations or by projecting lines onto the surfaces [8].

This paper presents a tool-path generation method based on triangle mesh. The tool path consists of the tool tip contact position and the tool direction. This study focused on the method of generating the contact position, involving several steps: construction of bounding box tree to represent the surface, hierarchy traversal to get the intersection pairs of triangles, triangle intersection to obtain the segments, and curve construction by connecting the segments. The algorithm provides a new method of intersection of two mesh surfaces. Compared with the mesh slicing algorithm, the cutting surface in this method is not only limited to planes but also to surfaces of any shape thereby facilitating generation of complicated tool paths. This method provides a robust mesh surface intersection algorithm compared with the analytical solution, and this can also be adopted in conventional tool-path generation algorithms.

The tool path needs to be translated into a robot language before executing a specific process that is completed by post-processing. The tool-path generation algorithm is integrated into an offline robot programming software framework with robot modeling and simulation to achieve a comprehensive solution for robotic applications.

The rest of this paper is organized as follows. Section 2 introduces tool-path generation algorithm based on triangle-mesh model. The building of bounding box tree, hierarchy traversal, and curve construction are included. In Sect. 3, the software system framework is introduced. Section 4 describes the experiments conducted to test the algorithm and the milling workcell that is set up to verify the tool path. Finally, the conclusions are presented in Sect. 5.

2 Tool-path generation

The tool-path generation method involves several key steps, as shown in Fig. 1. The algorithm cuts the target surface through a series of cutting surfaces. Accordingly, it finds the intersection of two surfaces by transforming them into a problem of mesh intersection as the surface is represented by mesh data. The bounding box tree is built to represent the surface in order to find the intersection pairs of the triangles in the mesh data. The segments derived by

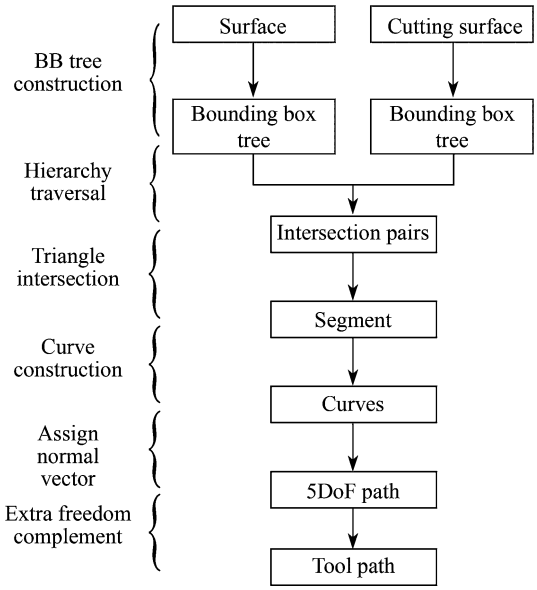


Fig. 1 Steps of tool-path generation algorithm

intersecting the triangle pairs are used to construct the curve, which is the result of the intersection of two surfaces. To generate a tool path, the tool position and orientation need to be determined. The intersection curves provide three-dimensional (3D) position information corresponding to the displacement of the tool tip. The information concerning the tool axis vector and the rotation around this vector also needs to be supplemented to obtain the complete constraints of the tool. In Fig. 1, the extra freedom complement transfers the 5DoF path into the 6DoF tool path for the robot.

2.1 Mesh object representation

The mesh surface is composed of a large number of triangular facets. Therefore, the intersection curve is composed of a series of intersections of the corresponding triangles. Accordingly, the algorithm is focused on finding the corresponding triangle pairs. Organizing the triangles in the mesh data into a hierarchical data structure is an effective way to determine the triangle pairs. The bounding volume (BV) and bounding volume hierarchy (BVH) are used in this method.

In our cases, BV is a closed volume that contains the union of triangles in the set, which are used to improve the efficiency of geometrical operations using their simple volumes. Unlike the original objects that are complex, the simple volumes help achieve cheap and fast overlap test (see Fig. 2); however, not all geometric objects serve as effective bounding volumes [9]. The desirable properties for bounding volumes include inexpensive intersection

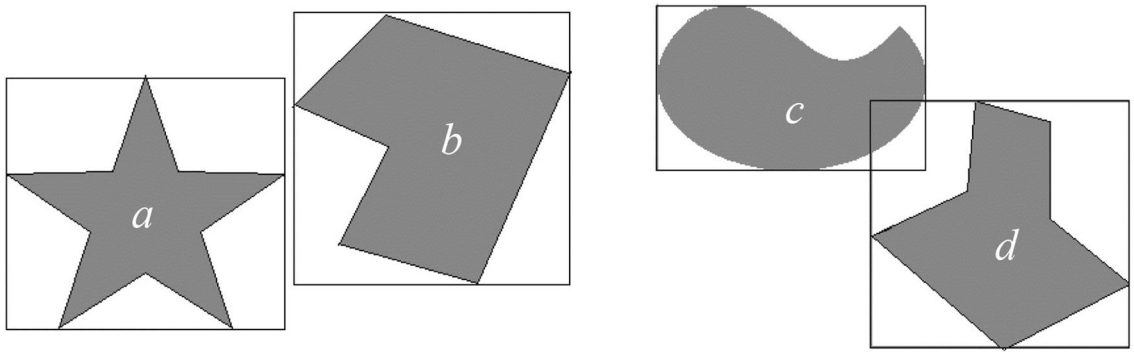


Fig. 2 Bounding volume example

tests, tight fitting, inexpensive to compute, easy to rotate and transform, and use of only a little memory.

Typical examples of bounding volumes, as shown in Fig. 3, include sphere, axis-aligned bounding box (AABB) [10], oriented bounding box (OBB) [11], and discrete oriented polytope (*k*-DOP) [12]. There exists a trade-off between the above properties.While choosing a bounding volume for a given application, various factors need to be considered: determining the cost of overlap, calculating the cost of computing the boundary volume of the object, the cost of updating it in case an object is moved or changed in shape or size, and the accuracy required for intersecting tests. As the bounding volume is mainly used to reduce the cost of intersection tests between primitives, it should encapsulate the primitives tightly, and the geometry used to enclose the volume should be as simple as possible.

The bounding volume helps speed up the computations through the multiple sets of geometric primitives. By arranging the bounding volume into a tree hierarchy known as BVH, the time complexity can be reduced significantly. Normally, the root BV encloses all the primitives of a model; the primitives in a parent BV are divided into several sub-BVs; and each BV encapsulates a separate partition of the primitives [13]. This process is then performed recursively, eventually resulting in a tree structure with a single BV at the top of the tree and each primitive wrapped in the BV as a leaf of the tree, as shown in Fig. 4. Typically, the BV in a leaf node contains one primitive. In some cases, several primitives may be placed at a leaf node, or several volumes may be used to contain a single primitive [14].

While building a bounding volume hierarchy, the degree of the tree needs to be determined firstly where the degree refers to the maximum number of children a node can have. Theoretically, a tree can have any number of children at a node. A tree with a high degree tends to be shorter; however, such a case requires more work as each node is searched. There exists a trade-off between trees of high and low degrees. The binary tree is widely used by far, because it is easier to build, represent, and traverse than the other trees.

In the proposed method, AABB is selected to build a binary tree for the mesh. For a given set of data objects, the AABB is easy to compute, and it needs only a few bytes of storage. Besides, robust intersection tests are easy to implement with high efficiency. The bounding box tree construction algorithm is shown in Fig. 4. For each step, the algorithm requires to partition the model into two submodels as the children of the current node. There are several options in splitting the model: by objects mean, by spatial median, by bounding box centers, by objects median, etc. In Algorithm 1, the bounding box is split along its longest axis and at the center of the box, as shown in Fig. 5.

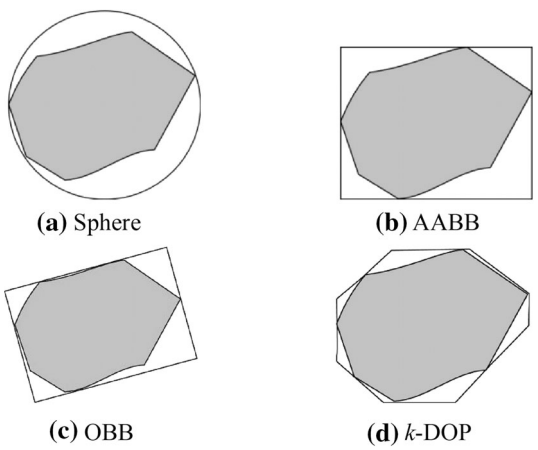


Fig. 3 Types of bounding volume

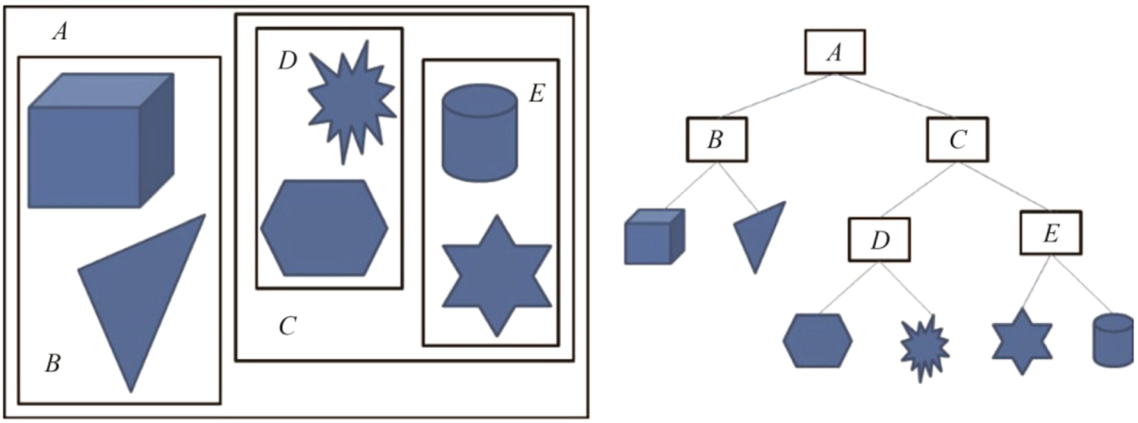


Fig. 4 A bounding volume hierarchy

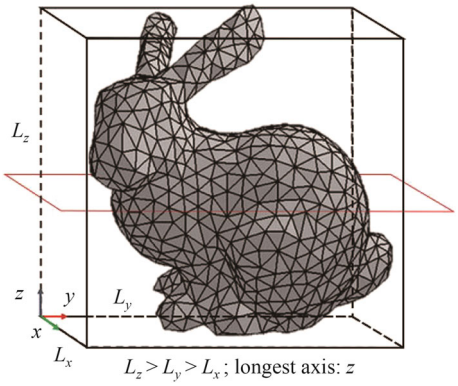


Fig. 5 Mesh partition method

Algorithm 1 Build bounding box tree

```
Input: Mesh data: M;
Output: Bounding box tree: bbtree
1:  function TreeBuild(meshdata)
2:    bbtree.bbox = makeBoudingBox(M)
3:    if M.indivisiable( ) then
4:      return
5:    end if
6:    M1 = SplitOnLongestAxis(M, 1)
7:    M2 = SplitOnLongestAxis(M, 2)
8:    bbtree.left = TreeBuild(M1)
9:    bbtree.right = TreeBuild(M2)
10: end function
```

2.2 Hierarchy traversal

The recursive traversal strategy is used to perform the intersection checks to find all the corresponding pairs of the intersecting triangles. The recursive algorithm is shown in Algorithm 2, and its input is the bounding box tree derived from Algorithm 1.Two trees (one for target surface and the other one for cutting surface) are traversed simultaneously in order to find all the intersection triangle pairs of both trees. The recursive strategy of traversal helps maintain the balance, and larger tree is traversed firstly for each step. Only when both inputs are leaves and the bounding boxes overlap, the two inputs are saved as an intersecting triangle pair. This is because the probability of these two triangles intersecting each other is high when the bounding boxes overlap.

Algorithm 2 Hierarchy traversal

```
Input: bbtree: b1, b2;
Output: Intersection Pair: P
 1: function Traversal(b1, b2, P )
 2:       l1 = b1.indivisiable()
 3:       l2 = b2.indivisiable()
 4:       if l1&l2 then
 5:         if b1.overlap(b2) then
 6:           tmp = makepair(b1, b2)
 7:           P.push(tmp)
 8:         end if
 9:         return
10:       end if
11:       if ! b1.overlap(b2) then
12:        return
13:       end if
14:       if b1.size > b2.size then
15:         c1 = b1.left
16:         c2 = b1.right
17:          Traversal(c1, b2, P )
18:          Traversal(c2, b2, P )
19:       else
20:          c1 = b2.left
16:          c2 = b2.right
22:          Traversal(b1, c1, P )
23:          Traversal(b1, c2, P )
24:       end if
25:  end function
```

2.3 Intersection of triangles pair

The intersection of two triangles forming a line segment is a simple geometric phenomenon [15, 16]. The intersecting

strategy designed in this study is shown in Fig. 6. T_1 and T_2 are two triangles for intersection check. First, a ray L is obtained by intersecting the two planes where these triangles lie. Subsequently, two segments L_1 and L_2 are respectively generated by chipping the triangles with L. Finally, the intersection of these two line segments L_1 and L_2 forms the line segment L_3.

2.4 Curve construction

As described above, after the triangular intersection step, many segments without topology information are obtained. The proposed curve construction algorithm, shown in Algorithm 3, is intended to connect these segments into a curve.

Algorithm 3 Curve construction

```
Input: Segments
       S = (S1, S2, ···, Sk)
Output: Paths
       Path=(C1, C2, ··· Cn)
 1:  function CurveCstrct(S)
 2:    tree=kdTreeBuild(S)
 3:    while S.size()! = 0 do
 4:      seg=S.pop()
 5:      while true do
 6:        seg1=kdTresearch (seg)
 7:        if seg1=null||seg1 is searched() then
 8:          break
 9:        end if
10:        C.push(seg1)
11:      end while
12:      Path.push(C)
13:    end while
14:  return Path=(C1, C2, ··· Cn)
15:  end function
```

A kd-tree is constructed firstly by organizing the points of the segments in a k-dimensional space [17]. It is a binary search tree with a set of constraints imposed on it, which are very useful for range and the nearest neighbor searches. In terms of generation of tool paths, it usually deals only with 3D points; therefore, the constructed kd-tree is also 3D. Each level of a kd-tree splits all children along a specific dimension using a hyperplane that is perpendicular to the corresponding axis. After constructing a kd-tree, the segments can be connected recursively into a path by searching for the nearest points as shown in Fig. 7. The segments $S_i(P_{s_i,1}, P_{s_i,2})$ and $S_{i+1}(P_{s_{i+1},1}, P_{s_{i+1},2})$ are connected by the same points $P_{s_i,2}$ $(P_{s_{i+1},1})$ which can be easily searched through the kd-tree.

There are two types of paths as shown in Fig. 8. One is a closed polygon path, and the other one is a free-end path. For closed polygons, the curve construction algorithm ends when the kd-tree search result is an already traversed segment. For curves with free ends, the algorithm ends only when there is no nearest point within a given tolerance.

For a complete tool path, it is also usually necessary to determine the tool orientation. The tool direction consists of two parts, that is, the tool axis direction and the rotation of the tool axis around itself. In general, the tool orientation is determined based on the application. In the actual work, the tool direction generally adopts surface normal or along a certain fixed direction. As shown in Fig. 9, P and N give the displacement and axis of the tool, respectively. α is the rotation around N . The rotation N is irrelevant to the tasks to be finished, but it has a great effect on the manipulator configuration. The method in Ref. [18] is used to determine the extra freedom.

2.5 Adaptive mesh cutting

A series of cutting surfaces are selected to cut the target surface to create a tool path. In practice, the type and orientation of the cutting surface are defined in advance. The parallel plane and the cylindrical surface are the most common types as shown in Fig. 10, which generates two main path patterns, namely, raster and spiral, as shown in Fig. 11, respectively. The cut surface needs to be meshed before applying the above intersection algorithm.

The offset between the cutting surfaces determines the distance between tool paths. The offset is determined by the geometric information related to the current cutting surface and the target surface as shown in Fig. 12.

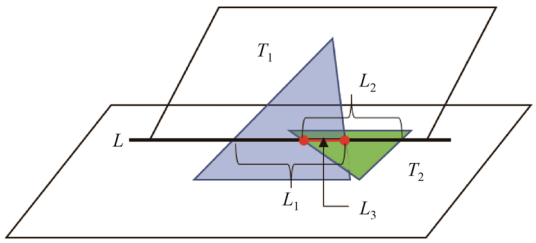


Fig. 6 Triangle intersection

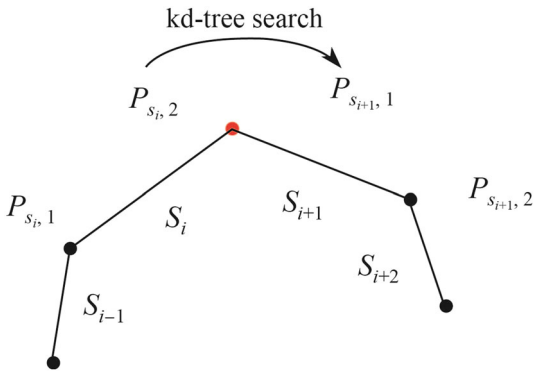


Fig. 7 kd-tree search for counter construction

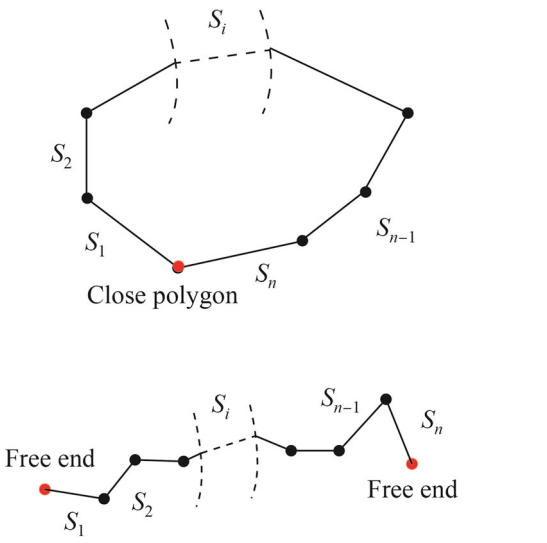


Fig. 8 Path types

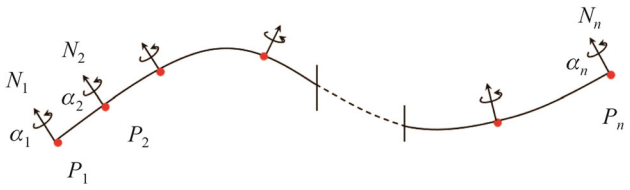


Fig. 9 Tool path for industrial manipulators

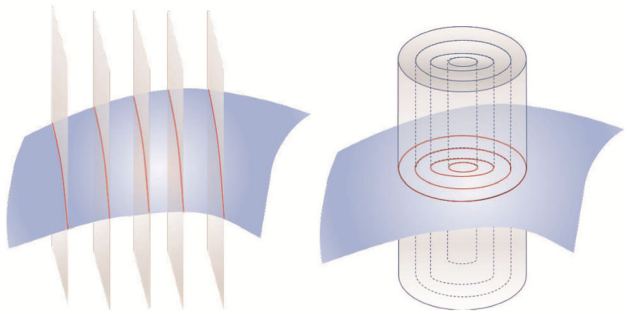


Fig. 10 Two typical methods for tool-path generation

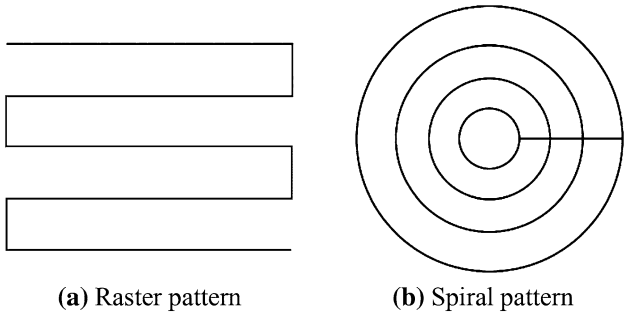


Fig. 11 Two typical patterns for tool path

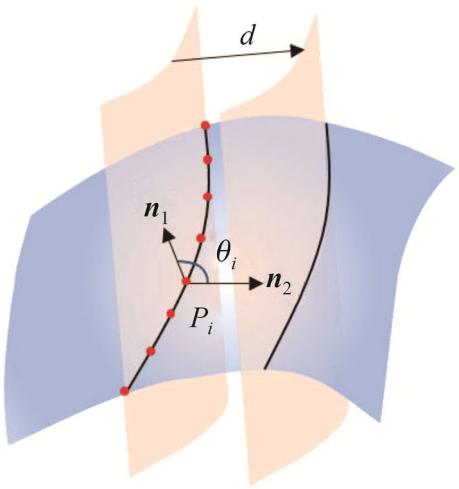


Fig. 12 Cutting surface offset

Equations (1) and (2) give the offset adaptively, where θ is the angle between $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$. The vectors $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ are the norm vectors of the cutting surface and the target surface, respectively. The offset mainly depends on θ as shown in Eq. (2), where θ is the average value of the angle between two surfaces which is obtained by integrating θ along the paths. d is offset between the cutting surface. ϵ is a constant small value. C is a constant value, and L is the given maximum steps.

$$\bar{\theta} = \frac{\left\| \sum_1^n \theta \right\|}{n}, \tag{1}$$

$$d = \begin{cases} \dfrac{L}{C}, & \bar{\theta} \leqslant \epsilon \\ \dfrac{2\bar{\theta}L}{\pi}, & \text{otherwise.} \end{cases} \tag{2}$$

3 CAD based interface programs

The above algorithm is integrated into a software framework (RSCAM) (see Fig. 13), a system that integrates geometric modeling, analysis, and viewing capabilities. It employs a mature CAD kernel as a geometry engine and adopts the component-based architecture shown in Fig. 14 [19].

In addition to extracting the surface information from CAD model and generating the tool path for different applications, the system also supports robot kinematic simulation by integrating kinematic library and related algorithms [20].

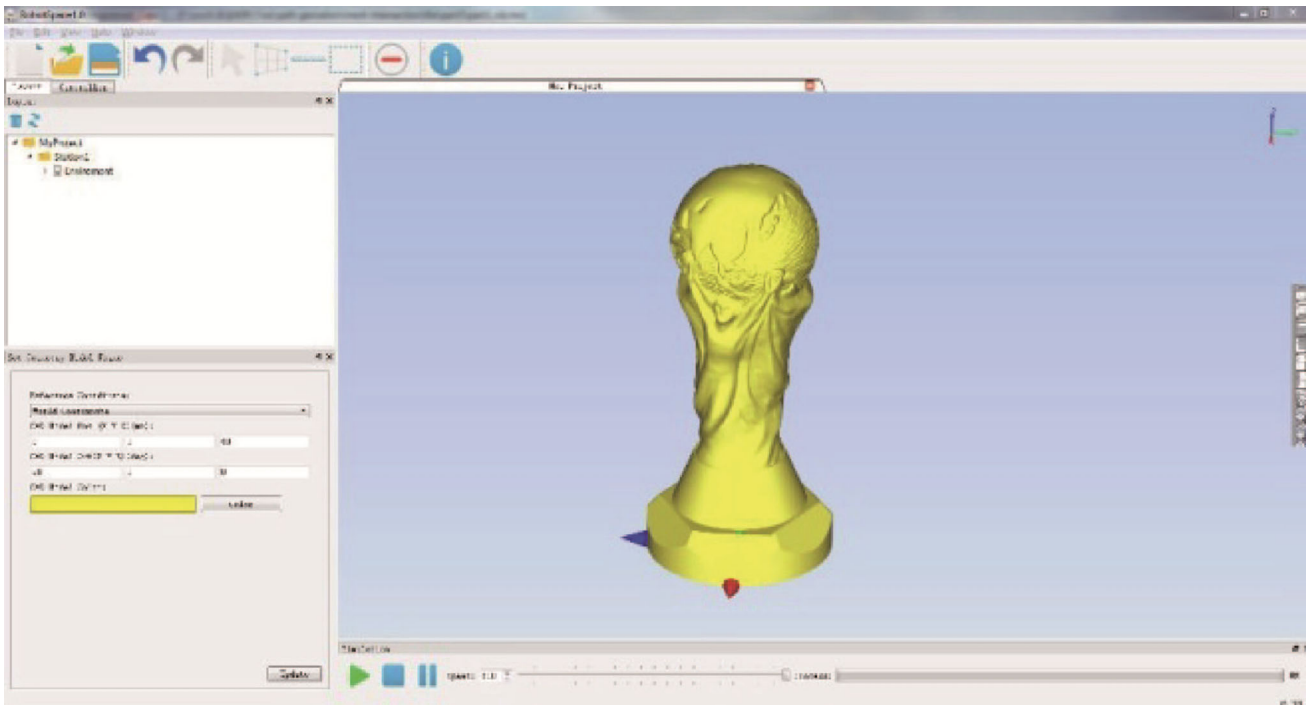


Fig. 13 Software UI

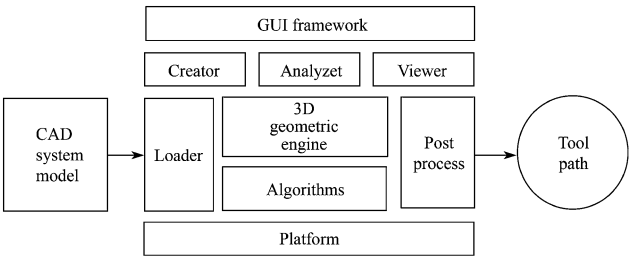


Fig. 14 Software framework

4 Experiment

4.1 An intersection test

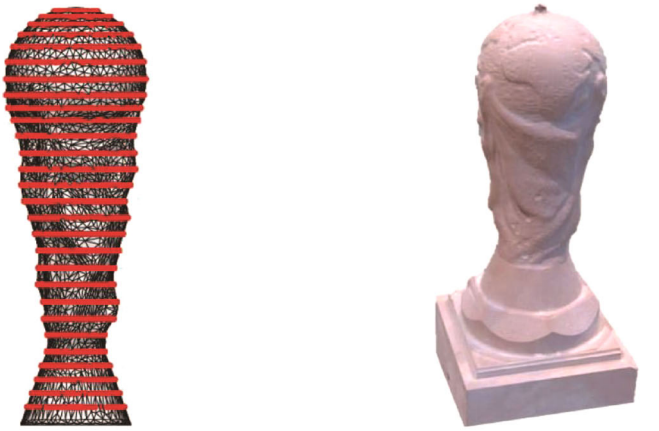
A World Cup model represented by mesh is selected as the experiment object to test the proposed algorithm by intersecting it with a hemisphere surface as shown in Figs. 15a and b. Two bounding box trees containing the model and the cutting surface are built. Figure 15c gives the bounding box tree of the World Cup model. Axis-aligned bounding box is utilized in this binary tree, and each mesh triangle is contained in a leaf of this tree. The World Cup and hemisphere contain 67 254 and 11 396 triangles; therefore, the two corresponding bounding box trees also have the same number of leaves.

Then, by hierarchy traversal, those overlapped the bounding box pairs from two trees are detected as shown in Fig. 16. The red and the green triangles are from the World Cup model and half sphere, respectively. From Fig. 16, it is noticed that some triangle pairs do not intersect with each other; however, it does not have effect on the final intersecting curve generation. The reason behind these non-intersecting pairs is that their bounding boxes overlap with each other. For this experiment, we obtain a total of 4 653 triangle pairs, among which 910 pairs are intersecting with each other. Then, the triangle-intersecting test applied on these pairs generates 910 segments. Using Algorithm 3, these segments can finally be constructed into a curve in Fig. 16.

4.2 Tool-path generation

In this section, the proposed algorithm is applied to a World Cup model to generate a milling tool path for an industrial robot. In this case, a series of parallel planes are used to cut the model. These planes are perpendicular to the z-axis, and the cutting step adapts to the geometric information using the proposed algorithm. The result shown in Fig. 15d obtained by setting $C = 10$ and $L = 5$ shows the effect of the adaptive information. For real milling applications, L is usually assigned a fairly small value, depending on the milling requirements. As can be seen from Fig. 15d, the tool path is adapted to the surface slope. For example, at the top of the model, the cutting step is lower than the rest.

(a) World Cup mesh model **(b)** Intersection test **(c)** BB-tree of World Cup model



(d) Tool path on the World Cup **(e)** Milling result of the World Cup

Fig. 15 World Cup mesh model

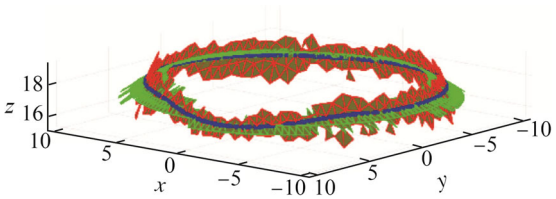


Fig. 16 Intersecting curve

4.3 Hardware workcell

The workstation platform, designed to mill a World Cup model, consisted of Kawasaki RS020N robots [21] with Marvie Controller [22], a worktable with a fixture, a spindle tool, and a workpiece. RS020N robot is a 6-DoF robot with parameters as shown in Table 1. The controller supports three types of motion including linear motion (MOVL), arc motion (MOVC), and joint motion (MOVJ). The minimal control accuracy of the controller is 1 mm. In our experiment, the milling path is performed by translating the tool paths into robot control files employing the linear motion. The spindle tool is a ball end mill with a diameter of 6 mm. The material of the workpiece is wood; therefore, the cutting force is not too large for the 20 kg payload robot. The algorithm generates a tool path and then converts it into a robot program through post-processing to control the robot and thereby perform the program. Finally, the World Cup model is obtained as shown in Fig. 15e.

5 Conclusions

In this paper, a tool-path generation method based on a mesh CAD model is proposed. The method is implemented by six steps: bounding box tree construction, hierarchy

Table 1 RS020N robot parameter

Payload/kg	Horizontal reach ability/mm	Vertical reach ability/mm	Repeatability/mm	Maximum speed/ ($mm \cdot s^{-1}$)
20	1 725	3 078	0.05	11 500

traversal, triangle intersection, curve construction, assigning normal vector, and complementing extra freedom. Finally, the algorithm is integrated into the RSCAM system to provide a friendly interface and visualization. The algorithm is tested by intersecting the World Cup model with a hemisphere model. The results show that the method is effective. The parallel plane is then used to cut the model, obtain the tool path, and execute it on the milling work cell.

In summary, the main contribution of this paper consists of two parts. Firstly, the paper proposes an algorithm to calculate the intersection of two mesh surfaces for tool-path generation. Then, a system (RSCAM) is developed by integrating the tool-path generation algorithm with a CAD engine to provide a complete solution for robotic surface-based applications. With RSCAM, programming time is reduced significantly compared with traditional manual teaching.

Acknowledgements Funding was provided by Research Grants Council, University Grants Committee (Grant No. 16205915) and Innovation and Technology Commission (HK) (Grant No. TS/216/17FP).

References

1. Lee S, Li C, Kim D et al (2009) The direct teaching and playback method for robotic deburring system using the adaptive force-control. In: IEEE international symposium on assembly and manufacturing, 17–20 Nov 2009, Seoul, South Korea, pp 235–241
2. Kim HJ, Back J, Song JB (2009) Direct teaching and playback algorithm for peg-in-hole task using impedance control. J Inst Control Robot Syst 15(5):538–542
3. Asakawa N, Toda K, Takeuchi Y (2002) Automation of chamfering by an industrial robot; for the case of hole on free-curved surface. Robot Comput Integr Manuf 18(5–6):379–385
4. Nagata F, Kusumoto Y, Fujimoto Y et al (2007) Robotic sanding system for new designed furniture with free-formed surface. Robot Comput Integr Manuf 23(4):371–379
5. Buckmaster DJ, Newman WS, Somes SD (2008) Compliant motion control for robust robotic surface finishing. In: World congress on intelligent control and automation, 25–27 June 2008, Chongqing, China, pp 559–564
6. Minetto R, Volpato N, Stolfi J et al (2017) An optimal algorithm for 3D triangle mesh slicing. Comput Aided Des 92:1–10
7. Sun YW, Guo DM, Jia ZY et al (2006) Iso-parametric tool path generation from triangular meshes for free-form surface machining. Int J Adv Manuf Technol 28(7–8):721–726
8. Ding S, Mannan M, Poo AN et al (2003) Adaptive iso-planar tool path generation for machining of free-form surfaces. Comput Aided Des 35(2):141–153
9. Ericson C (2004) Real-time collision detection. CRC Press, Boca Raton
10. Bergen GVD (1997) Efficient collision detection of complex deformable models using AABB trees. J Graph Tools 2(4):1–13
11. Gottschalk S, Lin MC, Manocha D (1996) OBB tree: a hierarchical structure for rapid interference detection. In: Proceedings of the 23rd annual conference on computer graphics and interactive techniques, New Orleans, LA, USA, 4–9 August, pp 171–180
12. Klosowski JT, Held M, Mitchell JS et al (1998) Efficient collision detection using bounding volume hierarchies of *k*-dops. IEEE Trans Vis Comput Graph 4(1):21–36
13. Larsen E, Gottschalk S, Lin MC et al (2000) Fast distance queries with rectangular swept sphere volumes. Proc IEEE Int Conf Robot Autom 4:3719–3726
14. Quinlan S (1994) Efficient distance computation between non-convex objects. In: Proceedings of the IEEE international conference on robotics and automation, 8–13 May, San Diego, USA, pp 3324–3329
15. Tropp O, Tal A, Shimshoni I (2006) A fast triangle to triangle intersection test for collision detection. Comput Anim Virtual Worlds 17(5):527–535
16. Sabharwal CL, Leopold JL, McGeehan D (2013) Triangle-triangle intersection determination and classification to support qualitative spatial reasoning. Polibits 48:13–22
17. Wald I, Havran V (2006) On building fast kd-trees for ray tracing, and on doing that in O(N log N). In: IEEE symposium on interactive ray tracing, Salt Lake City, USA, 18–20 Sept, pp 61–69
18. Lyu H, Song X, Dai D et al (2017) Tool path interpolation and redundancy optimization of manipulator. In: The 13th IEEE conference on automation science and engineering (CASE), 20–23 Aug, Xi'an, China, pp 770–775
19. The 3D modeling and visualization platform. http://www.anycad.net/. Accessed 30 March 2018
20. Murray RM, Li ZX, Sastry SS et al (1994) A mathematical introduction to robotic manipulation. Chemical Rubber Company Press, Boca Raton
21. RS020N product detail. https://robotics.kawasaki.com.cn/cn1/products/robots/small-medium-payloads/RS020N/. Accessed 2 Sept 2018
22. Marvie controller product detail. http://www.googoltech.com.cn/product/mcp/marvie/142/. Accessed 2 Sept 2018