Automatic Generation of High-level Contact State Space between 3D Curved Objects

Peng Tang and Jing Xiao

The International Journal of Robotics Research 2008; 27; 832
DOI: 10.1177/0278364908092463

Peng Tang
Jing Xiao
University of North Carolina
Charlotte, NC 28223, US
xiao@uncc.edu

Automatic Generation of High-level Contact State Space between 3D Curved Objects

Abstract

Information of high-level, topological contact states is useful and sometimes even necessary for a wide range of applications, from robotic tasks involving compliant motion to virtual prototyping and simulation. This paper addresses how to represent concisely and generate automatically graphs of contact states between 3D curved objects of a broad class, which may include smooth curved or planar surfaces. The approach is sound and complete if the step size of discretization is smaller than a finite threshold. By exploiting topological and geometrical constraints, it is also quite efficient. The implemented examples demonstrate the effectiveness of the approach.

KEY WORDS—Contact states, 3D curved objects, principal contacts, automatic generation, compliant motion

1. Introduction

Many robotic tasks involve objects in contact and compliant motion. Compliant motion is preferred in high-precision assembly operations to reduce uncertainty and is needed in tasks that require scribing, painting, grinding, polishing, contour following, object aligning and plotting in manipulation. For such tasks, it is often necessary to know not just the contact configurations between two objects but also the high-level, discrete, topological contact state shared by two or more contact configurations; this is more descriptive of the topological and physical characteristics of contact. For example, to polish a desk, the control strategy to keep the robotic tool contacting the desk top is usually different from that of keeping the tool contacting a side of the desk, and so on. The tool on the desk top can be considered a different contact state from the tool on the side of the desk. Each contact state describes a set of contact configurations of the tool with respect to the desk.

In general, contact states between two objects can be considered as partitioning the surface of configuration space obstacles (C-obstacles) (Lozano-Pérez 1983) of one object with respect to the other. Contact states and adjacency information can be captured by a contact state graph, where each node denotes a contact state and each arc links two adjacent contact states. Information of a contact state graph is usually needed for automatic assembly planning or control (Sturges and Laowattana 1995; McCarragher 1996; Pan and Schimmels 2003; Lefebvre et al. 2005). Such information is also required for general compliant motion planning and control.

Planning compliant motion means planning motion on the surface of C-obstacles. However, computing C-obstacles exactly in high-dimensional space remains a formidable task to date (Canny 1988). Most of the relevant work is limited to 3D C-obstacles (i.e. C-obstacles of planar objects) (Avnaim et al. 1988; Brost 1989; Rosell et al. 1997; Sacks and Bajaj 1998), and only a few approximations of C-obstacles of 3D polyhedra (Donald 1985; Joskowicz and Taylor 1996).

If a contact state graph is known, however, planning compliant motion can be greatly simplified as (1) a graph search problem to plan a sequence of contact state transitions in the contact state graph at the high level, and (2) planning motion compliant to a known contact state and the transition to a neighboring contact state at the low level, which is a *lower dimension* and *smaller scope* motion-planning problem. Indeed, a method for planning motion compliant with a known contact state was introduced for contacting polyhedral objects without requiring the construction of the C-obstacle in a 6D configuration space (Ji and Xiao 2001). The method takes advantage of known geometric features of the physical objects in the known contact state.

Such a two-level planning approach is also preferred to provide naturally the correspondences between contact configurations and higher level contact states that a compliant controller requires to execute a compliant motion plan

The International Journal of Robotics Research
Vol. 27, No. 7, July 2008, pp. 832–854
DOI: 10.1177/0278364908092463

Figures 18–24 appear in color online: http://ijr.sagepub.com

(Lefebvre 2003; Meeussen et al. 2005). To enable compliant control it is not sufficient to know only a path of contact configurations. The information of a sequence of high-level contact state transitions is necessary for designing proper compliant controllers (Meeussen et al. 2005). In other words, it is not possible to have a compliant control law applicable to all contact configurations. A practical way is to have a stratification of compliant control strategies based on different contact states and transitions, i.e. information of a contact state graph is necessary.

In haptic rendering and dynamic simulation (e.g. Ruspini and Khatib 1999; Luo and Xiao 2004), collision detection is inevitably subject to digital error. If object models are based on polygonal mesh approximation, which is very common, there are additional approximation errors. As a result, although multiple collisions are detected at the same time during simulation, not all of these collisions may be able to happen at the same time in reality. In other words, a false contact state may be identified. A false contact state leads to false force and dynamic effects, which should be prevented in high-fidelity simulation. Clearly, with a pre-determined graph of (valid) contact states, a simple search of the graph can rule out impossible contact states. Hence, information of a contact state graph is also needed.

For contacting polyhedral objects, it is common to describe or represent a contact state as a set of primitive contacts. Each primitive contact is defined in terms of a pair of contacting surface elements, which are faces, edges and vertices. One common representation (Lozano-Pérez 1983; Donald 1985) defines primitive contacts as point contacts in terms of vertex-edge contacts for 2D polygons, and vertex-face and edge-edge contacts for 3D polyhedra. Another representation (Xiao 1993) uses the notion of principal contacts as primitive contacts, where a principal contact can be a face contact, an edge contact or a point contact. Figure 1a shows different types of principal contacts (PCs) between two polyhedral objects. Figure 1b shows an example contact state described as a set of PCs. For convex curved objects, contact states are described similarly with each primitive contact defined in terms of a pair of contacting curved surface elements (Thompson II and Cohen 1999).

However, if non-convex curved surfaces or curves are present, there can be one or more than one contact region formed between the same pair of curved surface elements, resulting in different contact states with different contact constraints. Figure 2 shows an example. To resolve the ambiguity caused by such one-to-many mappings, one approach was to divide curved surface elements into so-called *curvature monotonic segments* (Luo et al. 2004; Tang and Xiao 2006a) so that between two curvature monotonic segments only one contact region can be formed (i.e. a one-to-one mapping). However, this approach of artificially dividing natural surface elements leads to a large number of contact states. A more concise approach is used (Tang and Xiao 2006b) to represent point contacts between strictly curved objects (i.e. without line segments in their surfaces).

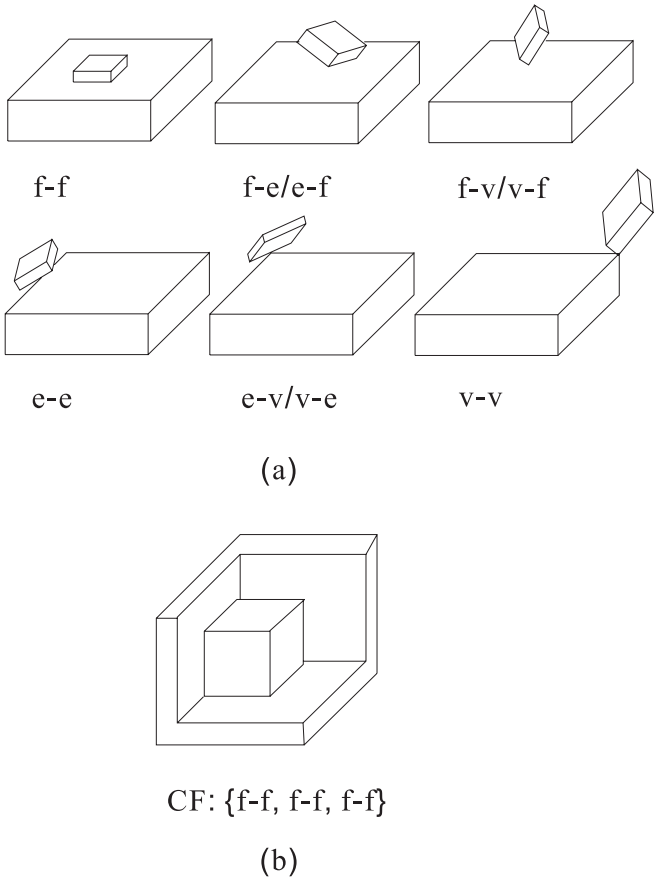


Fig. 1. Contact states between two polyhedral objects.

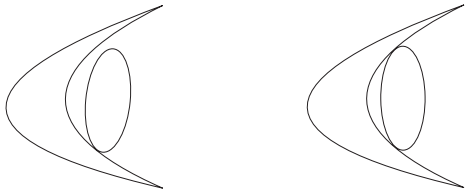


Fig. 2. Different numbers of contact regions can be formed between the same pair of contacting surfaces.

If ruled curved surfaces (i.e. curved surfaces that include line segments) are involved, there can be different types of contact regions between the same pair of surface elements. Figure 3 shows an example where there can be either a point contact or a line contact which have different contact constraints (or degrees of freedom). This type of ambiguity was not addressed before.

Under a suitable representation of contact states, automatic generation of a contact state graph is much desired. This is

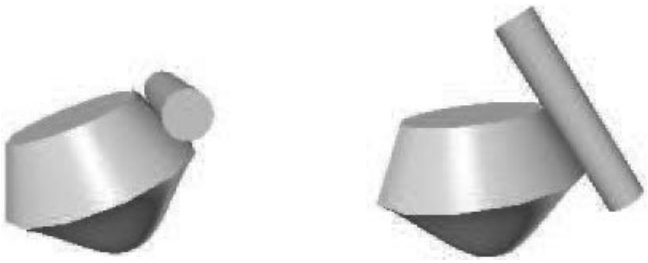


Fig. 3. Different types of contact regions can be formed between the same two contacting surfaces.

because building a contact state graph by hand is tedious for tasks of even simple geometry and is practically infeasible for cases involving (1) a large number of contact states and/or (2) contact states that are difficult to be visualized correctly (especially when curved objects are involved).

An early approach (Hirukawa et al. 1994) enumerated all possible contact states and connections between two convex polyhedra. A general approach was later developed to automatically generate graphs of contact states between polyhedral objects (Xiao and Ji 2001), where each contact state is represented as a set of principal contacts. The basic idea is to generate so-called goal-contact relaxation (GCR) graphs and then merge them, all done automatically. A GCR graph is grown from a seed contact state and includes its less-constrained neighboring contact states, their less-constrained neighboring contact states, and so on. However, a randomized strategy is sometimes used to find a valid neighboring transition. As such, the algorithm is not a complete algorithm. More recently, a method of automatically generating point-contact states between strictly curved objects was introduced (Tang and Xiao 2006b), but the issue of completeness has not been discussed.

In this paper, we study the problem not addressed by previous work: representing contact states and automatically generating contact state graphs between curved objects of a broad class (with curved, ruled or planar surfaces). In particular, we seek sound and complete algorithms for automatic generation of contact states between such general objects. We first describe the class of basic curved objects in Section 2. Next we introduce a concise representation of contact states between these objects that consist of point, line and planar types of contact regions in Section 3. Special neighboring relations between contact states are considered. In Section 4, we present the details of our approach to automatically generate contact state graphs and analyze the soundness and completeness of the approach. In Section 5, we describe some implementation results. In Section 6, we provide a discussion of complexity. We conclude the paper in Section 7.

2. Basic Curved Objects

Definition 2.1 (basic curved object): A *basic curved object* is a solid object in R^3 with its boundary consisting of smooth surface patches or closed smooth surfaces of finite size (such as spheres or ellipsoids) that satisfy the conditions:

1. each surface patch or closed surface can be described parametrically, referred to as *face*;
2. each face is bounded by curve segments or a closed curve of finite length that are planar, not self-intersected, and can be described parametrically (including line segments), referred to as *edges*;
3 each face can be curved or flat and has no self-intersection;
4. two adjacent faces are connected by either a common edge or a common point; and
5. the intersection point of three adjacent faces (i.e. the intersection point of two adjacent edges) or the contact point of two adjacent faces defines a *vertex*.

Many types of faces (surface patches) satisfy the above conditions, including those constructed by rotating or sweeping a planar parametric curved segment without self-penetration, quadric surface patches and non-degenerate parametric surface patches such as bicubic, Bezier and B-spline surface patches (Mortenson 1985). In fact, many man-made objects from CAD/CAM design satisfy the above conditions on their boundary surfaces. Figure 4 shows some different examples of applicable surface patches.

We do not consider objects with space curves (edges) in this paper, which can make contact regions more complex.

Definition 2.2 (surface element): Faces, edges and vertices as defined in Definition 2.1 are called *surface elements* of basic curved objects. Moreover, a face that is not closed is bounded by edges and vertices, and an edge that is not closed is bounded by vertices. Such edges and vertices are referred to as *bounding elements* of a face and an edge, respectively.

Definition 2.3 (adjacent elements): Two elements of a basic curved object are *adjacent elements* if one is the bounding element of the other.

For simplicity, we use f, e and v to denote face, edge and vertex, respectively, in the rest of the paper.

3. Contact States between Basic Curved Objects

We first analyze and describe the contact primitives between basic curved objects and then introduce a concise representation of contact states.

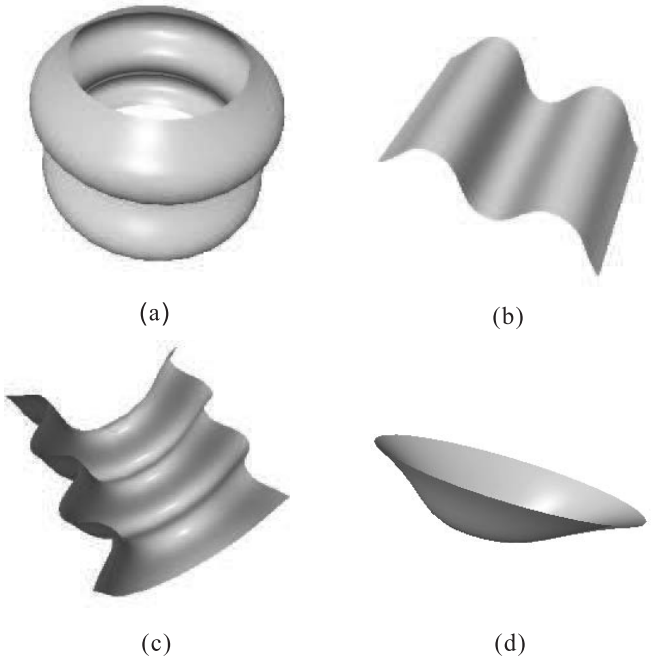


Fig. 4. Examples of different surface patch elements (faces), constructed by (a) rotating a curve segment; (b) sweeping a curve segment along a line; and (c) sweeping a curve segment along a curve segment. (d) An example of elliptic paraboloid (not a surface patch or revolution of sweeping).

3.1. Contact Regions between Two Surface Elements

We consider the following types of contact region between two surface elements:

- *point contact*: the contact region is an isolated point;
- *line contact*: the contact region is a continuous line segment; and
- *plane contact*: the contact region is a continuous set of points on a plane, which can be on a curve but not on a line.

A point contact can happen between many different surface elements, not necessarily involving a vertex, as shown in Figure 5. However, between two planar faces or one planar face and one straight line edge, no point contact can be formed. A line contact can happen between two ruled surfaces or a straight line edge and a ruled surface. Figure 6 shows two examples of line contacts. A plane contact can happen between two planar faces or between a planar face and a planar curved edge. Figure 7 shows two examples of plane contacts. Figure 7a is an example of a plane contact between two planar faces. Figure 7b is an example of a plane contact between a planar face and a planar curved edge.

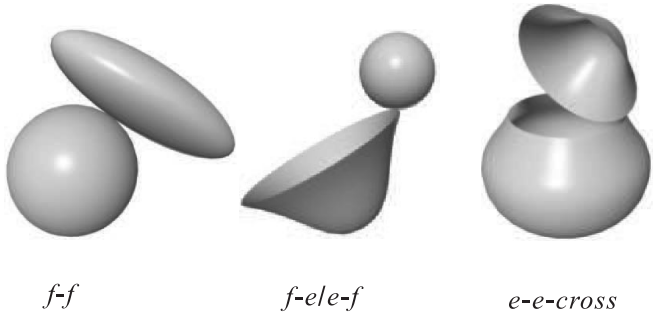


Fig. 5. Examples of different point contacts between two non-vertex surface elements.

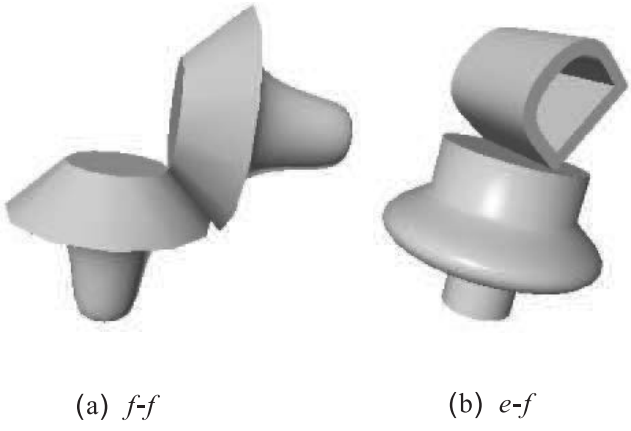


Fig. 6. Examples of line contacts.

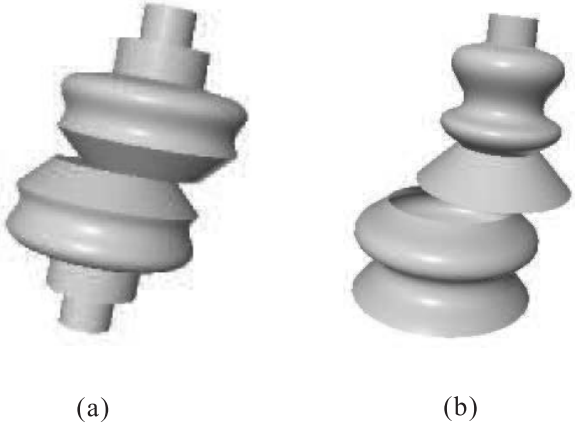
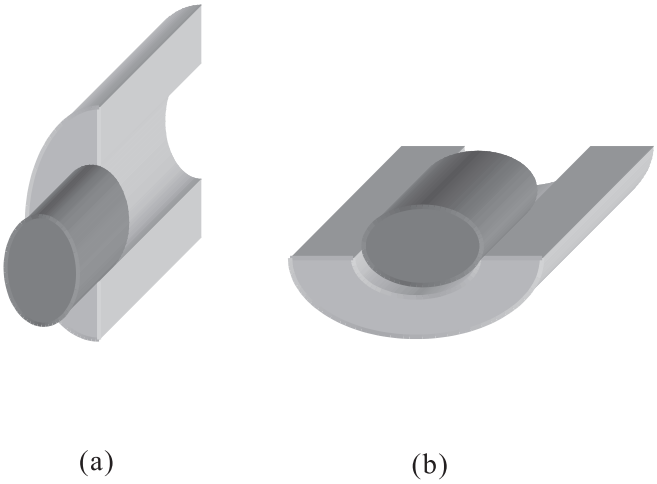


Fig. 7. Examples of plane contacts.

We do not consider the cases where two curved edges form a contact region that is also a curve or two curved faces form a contact region that is a non-flat surface region, because these cases hardly occur in practice. As shown in Figure 8, in or-



(a) (b)

Fig. 8. Contact cases that rarely occur in practice.

der for the contact region in case (a) to be a curve, the two contacting edges have to have identical portions in size and curvature that happen to be in contact. Similarly, the contact region in case Figure 8b can be a curved surface region only if the two faces have identical portions that happen to be in contact. Usually a point contact will only occur in case (a), and a line contact or multiple (isolated) point contacts will occur in case (b).

3.2. Contact Primitives between Two Surface Elements

We can extend the notion of principal contacts introduced between polyhedral objects (Xiao 1993) to describe a contact region between two surface elements of curved objects, taking into account contact types as follows.

Definition 3.1 (principal contact): A *principal contact* (PC) between two curved objects A and B describes a continuous contact region between a pair of surface elements (i.e. faces, edges and vertices) α_A and α_B that are not bounding elements of other pair(s) of contacting surface elements at a subset of the region. This is denoted

$$\text{PC} = \alpha_A \overset{\xi}{-} \alpha_B, \quad \text{where} \quad \xi \in \{\mathbf{P}, \mathbf{L}, \mathbf{PL}\},$$

and **P**, **L** and **PL** indicate the type of the contact region as point contact, line contact and plane contact, respectively.

Moreover, a PC between two faces is a *face tangential* contact because the two faces are tangent to each other. A PC between a face and an edge is an *edge tangential* contact but not a face tangential contact.

A PC between two edges is either an *e*-*e*-touch PC if the tangent lines of the two edges are collinear or an *e*-*e*-cross PC, otherwise. An *e*-*e*-touch PC can be either a point contact if one of the edges is curved or a line contact if both edges are straight-line edges. An *e*-*e*-cross PC is a point contact. An *e*-*e*-cross PC, *e*-*e*-touch PC or a PC involving a vertex is neither face tangential nor edge tangential because there is no face–face or face–edge pairs that are tangent to each other; these are referred to as *non-tangential* contacts.

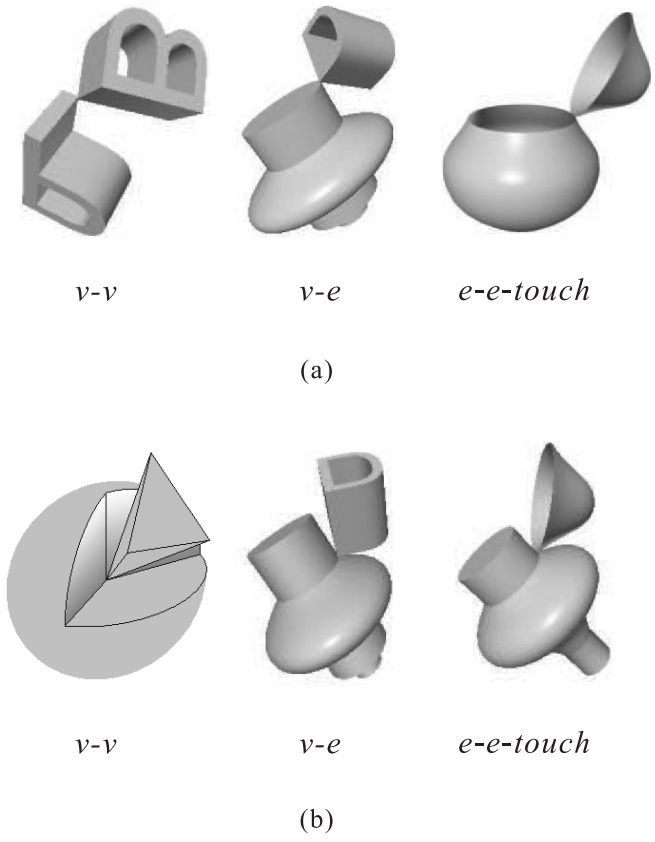


Fig. 9. PCs for both (a) degenerate and (b) non-degenerate cases.

Note that PCs of types v-v, v-e and *e*-*e*-touch have both *degenerate* cases, defined as extremely unstable contacts that rarely occur in practice. Note that this definition is different from what degeneracies usually mean in solid modeling (Edelsbrunner and Mucke 1990; Stroud 1990) and non-degenerate cases, as illustrated in Figure 9. We consider these PC types because of the non-degenerate cases.

We further define the *contact plane* and *contact line* for certain types of PCs as follows.

Definition 3.2 (contact plane and contact line): The *contact plane* of a PC involving at least one face is the plane tangent to the face and passing the contact region. For an edge-edge-cross PC, it is the plane determined by the two tangent lines of the two edges at the contact point. The *contact line* of a

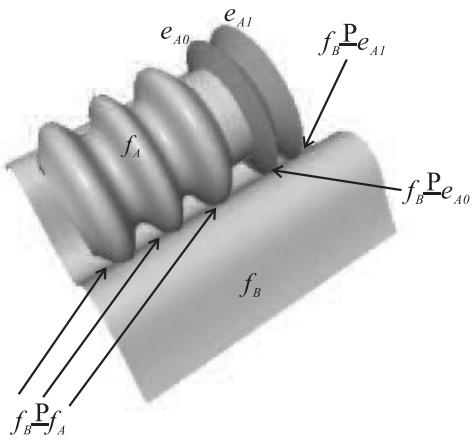


Fig. 10. A contact formation between two curved objects.

vertex-edge PC or an edge-edge-touch PC is the tangent line of the edge or both edges at the contact point.

Note that no single contact plane or contact line can be defined for a vertex-vertex PC.

3.3. Contact Formations between Curved Objects

As shown in Figures 2 and 3, there can be different numbers and types of PCs between two non-vertex surface elements of two contacting curved objects. Therefore, we characterize a general topological contact state between two curved objects by specifying the number and types of PCs involved, as follows.

Definition 3.3 (contact formation): A *contact formation*(CF) between two general curved objects is defined as the set of principal contacts (PCs) formed, where the same PC may be formed more than once, denoted

$$\mathrm{CF} = (\mathrm{PC}_1, n_1), (\mathrm{PC}_2, n_2), \ldots, (\mathrm{PC}_m, n_m),$$

where $n_i \in \mathbf{N}$, $\mathbf{N}$ is the set of positive integers $i = 1, \ldots, m$.

Moreover, the *cardinality* of a CF is denoted as:

$$\mathrm{card(CF)} = n_1 + n_2 + \ldots + n_m.$$

Figure 10 illustrates a contact formation between two curved objects.

3.4. Contact States and CF-Compliant Path

Given two objects A and B in a contact formation (CF), we denote the geometrical representation of a CF as the set of relative contact configurations of A with respect to B (as expressed by the homogeneous transformation matrix ${}^B T_A$) that satisfy the contact conditions of all the PCs in the CF at the same time.

In general, the set of contact configurations in the geometrical representation of a contact formation may consist of one or more connected regions of contact configurations, called *CF-connected* regions on the configuration space obstacle (i.e. C-obstacle) surface. Moving from one configuration to another within a CF-connected region does not need to change the CF. We therefore define a contact state as follows

Definition 3.4 (contact state): A single CF-connected region of contact configurations between two curved objects defines a *contact state* between the objects, represented by the CF and a representative contact configuration C in the region, denoted as a pair $< CF, C >$.

Definition 3.5 (CF-compliant motion): Within a contact state, from any contact configuration to any other, there exists a path as a continuous curve of contact configurations constrained by the CF, called a *CF-compliant motion*.

3.5. Neighboring Contact States and Contact Formations

Clearly, a C-obstacle surface (in the configuration space) is partitioned by contact states, and two adjacent contact states are called *neighboring contact states*. Alternatively, neighboring contact states can be defined in terms of neighboring transition motions.

Definition 3.6 (neighboring contact states, neighboring transition motion and neighboring CFs): Two different contact states $< CF_i, C_i >$ and $< CF_j, C_j >$, $i \neq j$ are called *neighboring contact states* if there exists a CF_i-compliant motion succeeded by a transition to a CF_j-compliant motion from C_i to C_j, and such a compliant motion from $< CF_i, C_i >$ to $< CF_j, C_j >$ is called a *neighboring transition motion*. Moreover, CF_i and CF_j are called *neighboring contact formations*.

A contact state space between two curved objects can be represented as a contact state graph $\mathcal{G}$ which includes nodes and arcs, where each node denotes a valid contact state $< CF, C >$, and each arc connects two neighboring contact states.

If two single-PC CFs $CF_i = \{(PC_i, 1)\}$ and $CF_j = \{(PC_j, 1)\}$ are neighboring contact formations, then PC_i and PC_j are called *neighboring PCs*. Neighboring PCs satisfy certain topological conditions, described as follows.

Theorem 3.1: If $PC_i = \alpha_{iA} \overset{\xi_i}{-} \alpha_{iB}$ and $PC_j = \alpha_{jA} \overset{\xi_j}{-} \alpha_{jB}$ are neighboring PCs, then one of the following is true.

1. If $\xi_i = \xi_j$, then either (i) $\alpha_{iA} = \alpha_{jA}$ and α_{iB} and α_{jB} are adjacent, or (ii) α_{iA} and α_{jA} are adjacent and $\alpha_{iB} = \alpha_{jB}$ or (iii) α_{iA} and α_{jA} are adjacent and α_{iB} and α_{jB} are adjacent.

2. If $\xi_i \neq \xi_j$, then either (i), (ii) or (iii) above is true or $\alpha_{iA} = \alpha_{jA}$ and $\alpha_{iB} = \alpha_{jB}$.

Proof: Suppose PC_i and PC_j are neighbors but neither condition in the theorem is held. If α_{iA} and α_{jA} are neither equal nor adjacent, changing the contact from PC_i and PC_j so that α_{iA} is changed to α_{jA} requires a compliant motion that has to go through another element α_{kA}, which results in another PC. This contradicts the fact that PC_i and PC_j are neighbors. ■

We now describe the topological conditions satisfied by two neighboring CFs.

Theorem 3.2: If two CFs CF_i and CF_j are neighboring CFs, and if card$(CF_j) \leq$ card(CF_i), then every PC in CF_j either belongs to CF_i or is a neighboring PC of a PC in CF_i.

Proof: We only need to prove that from a contact state in CF_i to a contact state in CF_j, a PC can be broken, changed to a neighboring PC compliantly or kept in the neighboring transition motion, but no new PC can be added. Suppose a new PC PC_j is added during the transition from CF_i to CF_j. A PC in CF_i, PC_k, has to be broken in order to have card$(CF_j) \leq$ card(CF_i). As breaking a PC requires only infinitesimal motion, but gaining a new PC requires a finite motion, breaking PC_k has to occur before the new PC_j is established. This implies that the transition has to go through at least another contact formation: either $CF_i - PC_k$ or $CF_j - PC_j$ before CF_j is reached, contradicting the fact that CF_i and CF_j are neighbors. ■

We can further distinguish two types of neighboring relations, as follows.

Definition 3.7 (locally-defined neighbor): If CF_i and CF_j are neighboring CFs, and card$(CF_j) \leq$ card(CF_i), we call CF_j a *locally-defined neighbor* (LN) CF of CF_i. Accordingly, the contact state $< CF_j, C_j >$ is a locally-defined neighboring contact state of the contact state $< CF_i, C_i >$.

Figure 11 shows an example, where CF_1 and CF_2 are two (mutually) LN CFs.

Definition 3.8 (globally-defined neighbor): If CF_i and CF_j are neighboring CFs, and card$(CF_i) >$ card(CF_j), we call CF_i a *globally-defined neighbor* (GN) CF CF_j. Accordingly, $< CF_i, C_i >$ is a globally-defined neighboring contact state of $< CF_j, C_j >$.

The reason we differentiate LNs and GNs is that given CF_i, according to Theorem 3.2, the topological information of its

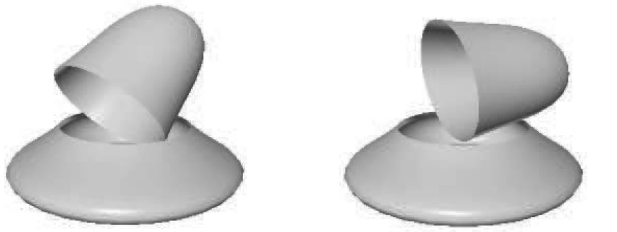


Fig. 11. An example of locally-defined neighbor (LN) CFs.

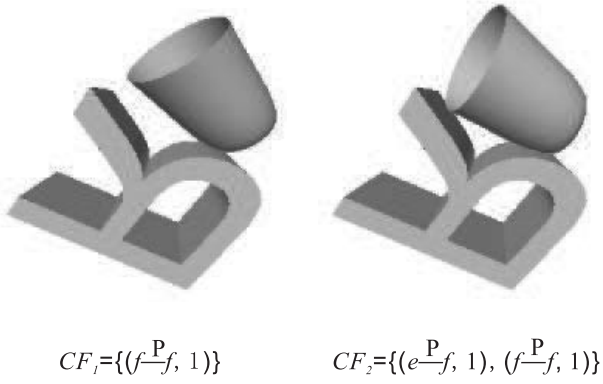


Fig. 12. Example of a locally-defined neighbor (LN) and a globally-defined neighbor (GN) CF.

LN CFs can be derived directly from the topological information of CF_i; i.e. from the PCs in CF_i, one can obtain the possible PCs of the LN CFs of CF_i. In the example shown in Figure 12, CF_1 is a LN CF of CF_2, which is derived from CF_2 by simply dropping the edge-face PC of CF_2. However, one cannot derive CF_2, which is a GN CF of CF_1, directly from CF_1.

Such a property means that given a contact state $< CF_s, C_s >$, the subsets of PCs in CF_s provide possible LN CFs of CF_s. In other words, possible LN CFs can be enumerated directly from CF_s, which can facilitate automatic generation of the corresponding LN contact states. We define the LN graph of $< CF_s, C_s >$ as an undirected graph consisting of $<CF_s, C_s>$, its LN contact states, their subsequent LN contact states and so on. Clearly the LN graph is finite.

On the other hand, it is difficult to enumerate all possible GN CFs of CF_s because there is no upperbound on the number of combinations of PCs (especially since the same PC can occur multiple times in a CF).

Fortunately, to obtain an entire contact state graph, it is sufficient to simply automatically generate the largest LN graphs and merge them, as explained in Section 4.

Algorithm 1 Generating LN Graph

Initialize the LN graph LNG with a single node containing $< CF_s, C_s >$
Initialize the FIFO queue *new* with a single node $< CF_s, C_s >$
while *new* is not empty **do**
  Remove node $< CF_i, C_i >$ from *new*
  Hypothesize topologically all possible LN CFs of CF_i
  for each hypothesized LN CF, CF_h, of CF_i **do**
    if LNG has no node with CF_h as the CF **then**
      if there exists a feasible neighboring transition motion from C_i in $< CF_i, C_i >$ to configuration C_h with CF_h **then**
        create a node for $< CF_h, C_h >$, link it to the node of $< CF_i, C_i >$ in LNG, and add it to the end of *new*
      end if
    else
      for each node $< CF_h, C_{h,k} >$ in LNG **do**
        if there is no link between it and $< CF_i, C_i >$ and there exists a feasible neighboring transition motion from C_i in $< CF_i, C_i >$ to $C_{h,k}$ in $< CF_h, C_{h,k} >$, **then**
          build a link between the node $< CF_h, C_{h,k} >$ and the node $< CF_i, C_i >$ in LNG
        end if
      end for
      if there exists a feasible neighboring transition motion from C_i to a different configuration $C_{h,*}$ ($C_{h,*} \neq C_{h,k}$, for all k), **then**
        create a node $< CF_h, C_{h,*} >$, link it to the node of $< CF_i, C_i >$ in LNG, and add it to the end of *new*
      end if
    end if
  end for
end while
Output LNG

4. Generation of Contact State Graphs between Curved Objects

Our approach is to generate special subgraphs of the contact state graph $\mathcal{G}$ automatically and to merge these subgraphs automatically. That may appear similar to the approach used in Xiao and Ji (2001) for polyhedral objects, but there are major differences as listed below.

1. First, the special subgraphs here are LN graphs, which are different from and less restrictive than the GCR graphs considered in Xiao and Ji (2001). Given a seed contact state $< CF_s, C_s >$, its LN graph is usually much larger than its GCR graph. Fewer LN graphs are needed for obtaining $\mathcal{G}$, especially by using seed CFs of locally maximum cardinality (i.e. number of PCs in a CF) which are much fewer than the seeds for GCR graphs (which are the locally most constrained CFs).

2. Second, curved objects are considered here, for which contact state representations are different and neighboring transition motions are more complex.

3. Lastly, our approach for generating LN graphs automatically is assured to be sound and complete under a finite resolution for discretization.

Starting from a seed contact state $< CF_s, C_s >$, the LN graph can be grown by repeatedly obtaining LN contact states until all the LN contact states have been generated in a breadth-first search. We explain the details in the following subsections.

4.1. Algorithm for Generating LN Graphs

Algorithm 1 for constructing an LN graph from a seed contact state $< CF_s, C_s >$ is outlined below.

Note that in Algorithm 1, if two or more contact states are generated under the same contact formation CF_h, it means that the geometrical representation of CF_h has two or more CF_h-connected regions in the configuration space, and $C_{h,k}$ is the representative contact configuration of the kth CF_h-connected region.

There are two key procedures in the algorithm.

1. From a known contact state $<CF_i, C_i>$, hypothesize its LN CFs based on the topological information of CF_i.

2. Determine if there exists a feasible neighboring transition motion from C_i under CF_i to some configuration C_j under a hypothesized LN CF CF_j. If so, $< CF_j, C_j >$ is a valid LN contact state of $< CF_i, C_i >$.

We explain both procedures in detail in the following subsections.

4.2. Hypothesizing Locally-defined Neighbor (LN) Contact Formations (CFs)

To obtain a hypothesized LN CF of a single-PC CF, $CF_i = \{(PC_i, 1)\}$, is to **change** $PC_i = \alpha_A \overset{\xi}{-} \alpha_B$ to one of its neighboring PCs. This involves changing either the type of contact ξ or a surface element α_A or α_B to an adjacent element. We can avoid hypothesizing topologically impossible PCs based on different kinds of topological properties of surface elements as detailed in Appendix A.

For a CF with multiple PCs, i.e. card$(CF) \geq 2$, we use the following action sets to hypothesize its possible LN CFs.

- Action set 1: **keep** some PCs and **change** some other PCs to their neighboring PCs.
- Action set 2: **keep** some PCs and **remove** some PCs.

Note that no **keep** action can be applied simultaneously to all PCs in the CF, and no **remove** action can be applied simultaneously to all PCs in the CF. Note also that the neighboring PCs should be topologically possible, according to Appendix A.

Once LN CFs are hypothesized, we next check if they are geometrically feasible by considering the relevant neighboring transition motions.

4.3. Neighboring Transition Motions

Between two neighboring contact states of two curved objects A and B, a neighboring transition motion can be one of the following four types of compliant motions of object A with respect to object B or certain combinations of these types:

- **slidingA**, where the contact points of A do not change but the contact points of B changes (Figure 13a);
- **slidingB**, where the contact points of B do not change but the contact points of A changes (Figure 13a);
- **pure rotation** about an axis through a contact point, which is usually either normal or tangent to the contact plane (if the contact plane exists) (Figure 13b); or
- **pure translation** along the contact plane of a line contact or plane contact (Figure 13b).

Note that neither **slidingA** nor **slidingB** are pure rotations or translations.

Neighboring transition motions can also be of the following types of combined motions:

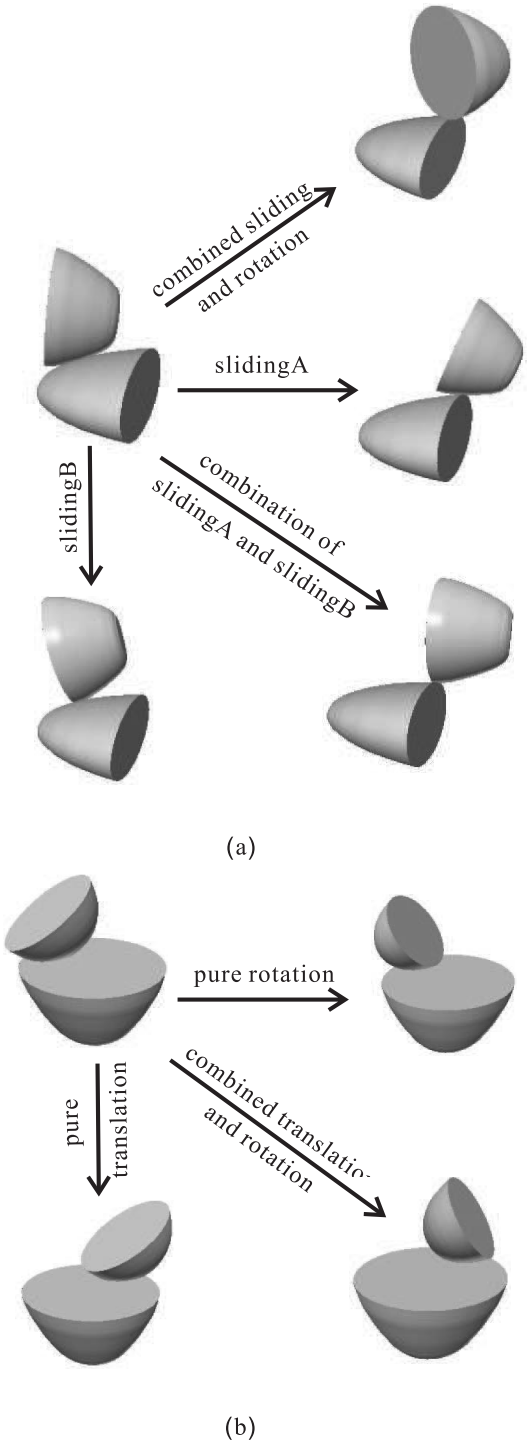


Fig. 13. Types of neighboring transition motions.

- a **combined slidingA and slidingB**, where the contact points of A and B both change (note that if the contact points of A and B change in equal displacements, the motion is in fact a rolling motion) (Figure 13a);

- a **combined sliding and rotation** motion, where a **slidingA** or a **slidingB** motion or a **combined sliding** is combined with a **pure rotation** (Figure 13a);
- a **combined translation and rotation** motion (Figure 13b).

In order to implement a neighboring transition motion, we need to set up a good moving task frame. The origin of the task frame should be at a contact point. For a PC with a contact plane, the $x-y$ plane is along the contact plane and the z axis is along the normal direction of the plane. Moreover, using the parametric representation of a face $s = s(u, v)$, either the x or the y axis can be along one parametric derivative vector s_u or s_v. For an e-e-cross PC, either the x or the y axis can be along the tangent line of one of the edges.

For a PC with a contact line the x axis is along the contact line. If the edge (or one of the edges) at the contact point is a curved edge, the z axis is on the plane formed by the contact line and that edge at the contact point, and is normal to the edge. In all other cases, the axes of the task frame can be set based on a neighboring PC that the transition is aimed at, which should have either a contact plane or a contact line.

Each **slidingA** and **slidingB** can be generally viewed as an integral of instantaneous **pure translation** ds combined with an instantaneous **pure rotation** $d\theta$, and the results can be implemented by a summation of digitized small motion steps. Each small motion step is implemented as a small translation Δs combined with a small rotation $\Delta\theta$. Specifically, by digitizing the u and v parameters of a face, we get a grid of points on the face. Each small sliding motion along the face is implemented as a small translation from one grid point p to an adjacent grid point q on the face followed by a small rotation along an axis through q on the tangent plane and orthogonal to the translation vector from p to q. The angle $\Delta\theta$ of the rotation is determined by the tangent plane T_1 at p and the tangent plane T_2 at q. A similar strategy can be used to implement a small sliding step along an edge.

Each combined motion can be similarly implemented as an integral (or summation) of small motion steps such that each small step consists of one small single-type motion δ_1 followed by another small single-type motion δ_2, etc.

4.4. Feasibility Check of Neighboring Transition Motions

Given a valid contact state $< CF_i, C_i >$ and a hypothesized LN CF CF_j between two curved objects A and B, to determine if there exists a valid LN contact state $< CF_j, C_j >$ of $< CF_i, C_i >$ is to check if there is a feasible neighboring transition motion from C_i to a configuration C_j of $< CF_j, C_j >$. Any neighboring transition may involve **remove**, **keep** or **change** one or more PCs of CF_i, based on Section 4.2. The **change** action is to change a PC to a neighboring PC.

The types of possible compliant motions to **keep** or maintain a principal contact $PC_i = \alpha^i_A \overset{\xi}{-} \alpha^i_B$ depends on the contact type ξ and the types of the surface elements of α^i_A and α^i_B, as presented in Appendix B.

The types of possible compliant motions to **change** a principal contact PC_i to a neighboring PC_j between objects A and B may consist of a **keep** motion of PC_i (i.e. object A's motion compliant to PC_i) followed by a transition motion to PC_j. The **keep** motion is to bring A to a configuration where the change of PC_i to PC_j can happen. This configuration is determined based on the contact types of PC_i and PC_j as well as which boundary element in PC_i is changed from PC_i to PC_j. Once A is in such a configuration, a rotation is needed for the transition motion between:

- a face- or edge-tangential contact and a non-tangential contact;
- a face-tangential contact and an edge-tangential contact; and
- two PCs sharing the same contacting elements but having different ξ types.

In general, if a neighboring transition from state $< CF_i, C_i >$ to a hypothesized LN state $< CF_j, C_j >$ requires changing some PCs while keeping or removing some other PCs, our strategy is to realize a **change** action from a PC PC_k to a desired neighboring PC $PC_{k'}$. If multiple PCs need to be changed to their neighbors, we prefer to pick such PC_k and $PC_{k'}$ that do not involve a face to have lower degrees of motion freedom. Once a pair of PC_k and $PC_{k'}$ are chosen, we construct a possible compliant motion of A to realize the **change** action from PC_k to $PC_{k'}$.

On the other hand, if a neighboring transition from state $< CF_i, C_i >$ to a hypothesized LN state $< CF_j, C_j >$ only requires keeping some PCs while removing other PCs, our strategy is to realize a **keep** action of one PC under a direction to remove another PC. If multiple PCs need to be kept, we prefer to pick a PC that does not involve a face or is not of e-e-cross type to have lower degrees of motion freedom. With the chosen PC and the direction to remove another PC, we construct a possible compliant motion of A to **keep** it.

If a compliant motion can be constructed without causing additional collisions and is able to **keep**, **change** or **remove** other PCs required by the transition, the motion is considered feasible and the hypothesized contact state is considered a valid LN contact state.

Sometimes a compliant motion can be immediately recognized as unfeasible even without construction. This happens when a chosen motion to **change** a PC is not a type of motion that can **keep** another PC also required by the neighboring transition. If such a motion is the only possible compliant motion, then the corresponding hypothesized LN CF is not valid. For example, the only type of motion that can **keep** a $v-v$

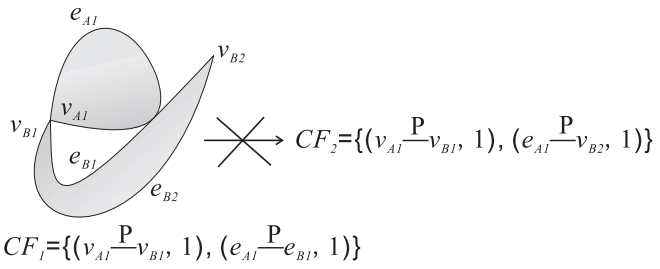


Fig. 14. An example of a hypothesized LN CF that is invalid.

PC is a pure rotation. As shown in Figure 14, a pure rotation is needed to keep the v-v type of PC from CF_1 to a hypothesized CF_2, but to change the e-e type of PC to the neighboring e-v type of PC requires a sliding motion. CF_2 is therefore not a valid LN CF.

If a neighboring transition motion is constrained to be a 1-DOF motion, there are only a finite number of possible neighboring transition motions. Our algorithm simply checks the feasibility of each possible neighboring transition motion one by one until either (1) a feasible motion is found to conclude that the corresponding LN CF and the LN contact state exists; or (2) no motion is feasible to conclude that the hypothesized LN CF is not valid and the LN contact state does not exist.

If a neighboring transition motion has two or more DOFs, there are an infinite number of possible neighboring transition motions and discretization is necessary. For example, in the case where neighboring transition is to remove a PC while keeping another PC that involves a face (i.e. a face-face, edge-face or vertex-face PC), there can be an infinite number of possible directions for the motion, but the motion itself can be infinitesimal and only needs a small step in practice. Our strategy is to discretize the directions and check the motion along each direction to see if a feasible small motion step exists or not.

Another case is where a neighboring transition motion has to have a finite length (more than a small step) and has more than one degree of freedom (DOF). For example, to change $CF_i = \{(f_A \overset{\mathbf{P}}{-} f_B, 1)\}$ (i.e. a CF with a single face-face contact) to $CF_j = \{(f_A \overset{\mathbf{P}}{-} e_B, 1)\}$, where e_B is an edge of face f_B, there can be an infinite number of possible CF_i-compliant motions to move A along f_B to reach (any point on) e_B before a rotation can be done to make the transition. For such cases, we have developed an *obstacle-tracing* algorithm to check if there exists a feasible motion (or not) as detailed below.

4.5. Obstacle-tracing Algorithm for Finding Neighboring Transition Motions

If a neighboring transition motion has more than one degree of freedom and has to be finite (i.e. not infinitesimal so that more than a small step is required), it involves moving an object A compliantly along a face of the other object B to reach an edge or vertex of the face. In such a case, there are an infinite number of possible neighboring transition motions. Without losing generality, consider moving A along the face f_B of B compliantly to reach an edge e_B of f_B for neighboring transition. Our goal is to either find a feasible motion if one exists or report that no feasible motion exists.

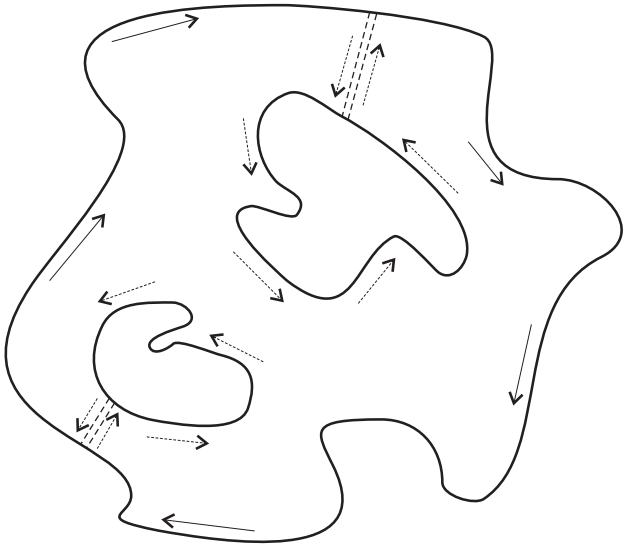


Fig. 15. Conversion of a multiply-connected region to a simply connected region.

The basic idea of our algorithm, called *obstacle-tracing*, applies to a simply-connected f_B. If f_B is a multiply-connected region, it can be pre-converted to a simply connected region by introducing pseudo edges, before the obstacle-tracing algorithm is applied. Figure 15 depicts an example of conversion to a simply connected region.

The algorithm moves object A compliantly on f_B towards the boundary edges of f_B until either an edge of f_B is reached or an obstacle is encountered. Then it makes A follow the boundary of f_B in one direction by either moving along boundary edges of f_B that A can reach without collision or tracing the extruding parts of obstacles that prevent A from reaching some boundary edges of f_B. In this way, A will either reach e_B (the desired edge of B) or travel in some loops. The algorithm searches a feasible motion by discretizing starting orientations of A and, for each starting orientation, moving A in digitized steps as described in Section 4.3. For each small step, it tries different possible moves of A and does collision checking for each move. We describe the algorithm in detail below.

First, a path τ as a sequence of curves on f_B, with parametric representation $f_B(u, v)$, is constructed in the following way: τ starts from the point (u_0, v_0) on f_B that A's task frame origin o_A contacts. It continues along the u axis (or the v axis)

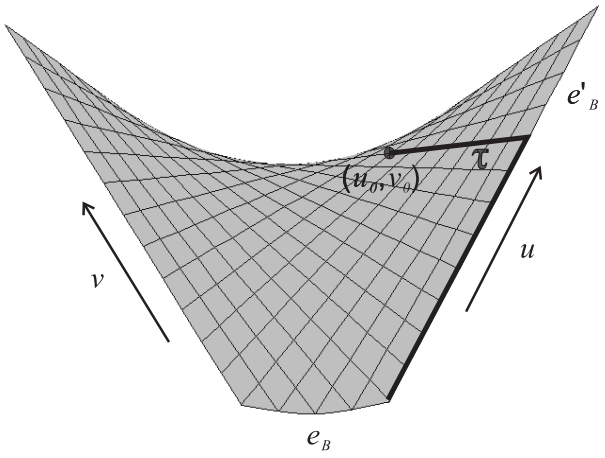


Fig. 16. An example path τ for a point contact.

Algorithm 2 Obstacle-tracing

```
1: Initialize τ, p ← (u0, v0), CA ← C0
2: repeat
3:    collision ← false
4:    (p, γ) ← FollowPath(p, τ, collision)
5:    if collision then
6:       γc ← γ {Remember the first collided surface element of B}
7:       m ← 1
8:       n ← 1
9:       repeat
10:         (p, γ) ← ObstacleTrace(p, γc, γ)
11:         collision ← false
12:         if A reaches an edge of fB then
13:            (p, γ) ← FollowEdge(p, collision, γc, γ)
14:         end if
15:      until eB is reached OR m = M OR n = N
16:   end if
17:   if eB is reached then
18:      solution ← true and go to 26
19:   else
20:      FindNewStart(u0, v0, CA)
21:   end if
22: until no new start configuration
23: if no new start configuration then
24:   solution ← false
25: end if
26: report solution
```

until A contacts a point on an edge e'_B of f_B (i.e. by updating only the u (or v) value). If $e'_B = e_B$, the path τ ends. If not, the path τ makes a turn to continue along e'_B until another edge (i.e. a vertex of another edge) is encountered by A, and so on, until e_B is reached. Figure 16 shows an example of a path of τ.

Our *obstacle-tracing* algorithm is outlined in Algorithm 2. It starts by making A follow path τ. When a collision is detected, the motion is deviated to trace the obstacle boundary while avoiding it until an edge of f_B is reached or certain conditions are satisfied. If an edge is reached, the motion will continue along the edge when a collision may again occur so that the motion may again deviate to trace the obstacle boundary. A feasible neighboring transition motion is found if e_B can be reached eventually, and the algorithm returns 'solution is true'.

There are two conditions indicating that e_B cannot be reached by the current motion. One condition is that the first surface element γ_c of B which has collided is encountered again after A collides with some other elements of B, M times. Here the integer $M > 2$ depends on the characteristics of γ_c. It is set to make sure that after M times, some loops have been accomplished. The other condition is to deal with the case where A can only collide with γ_c and no other element of B. The large integer N depending on the size of γ_c is set to ensure that after N collisions of A with γ_c, some loops have been accomplished. If either condition is satisfied, the *obstacle-tracing* algorithm changes the starting orientation of A and again constructs a motion as described above. If no feasible motion is found for all possible starting orientations, the algorithm concludes that no feasible neighboring transition motion exists and returns 'solution is false'.

The function **FollowPath** checks if a feasible motion can be found by having A's task frame origin o_A following the path τ step by step: the motion starts from the point p on τ and calls a function **TryPossibleStepMotions** to see if there is a feasible step motion from p to the next point on τ. If such a step motion exists, the motion is made and the function **FollowPath** continues to check for the next step motion and so on until either: (1) e_B is reached, i.e. the end of τ is reached; or (2) there is a collision. If (1), it returns the end point of τ and sets the flag that e_B is reached to be true. If (2), it sets *collision* to be true and returns the collided surface element γ of B and the point from which the next step causes the collision.

The function **TryPossibleStepMotions** checks possible step motions to return a feasible one if it exists. A step motion is a **slidingA** motion with or without a rotation along an axis through o_A. In the case of neighboring transitions between a CF_i of a single PC $f_A - f_B$ and an LN CF CF_j of a single PC $f_A - e_B$ (or $f_A - v_B$), there are three possible step motions: a small **slidingA** and two small combined motions of **slidingA** and rotation about an axis through o_A and normal to f_B. The two combined motions involve clockwise and counter-clockwise rotations, respectively. Note that the rotation amount does not vary; it is a fixed small angle in one small step in either direction. If f_B is flat, **slidingA** becomes a translation.

In some other cases of neighboring transition, there can be an infinite number of possible step motions. For example, in the case of a neighboring transition between a CF_i of a single v-f PC and an LN CF_j of a single v-e PC, the possible

Algorithm 3 Procedure of ObstacleTrace

```
1: ObstacleTrace(p, γ_c, γ)
2: D_1 ← left (or right)
3: D_2 ← right (or left, the opposite of D_1)
4: γ_1 ← γ, γ_2 ← γ
5: repeat
6:    from p perform a trace step by turning D_1 to find the next point p_1
7:    if TryPossibleStepMotions(p, p_1) does not return a feasible step motion then
8:       from p turning D_1 again to find the next point p_2
9:       p_1 ← p_2
10:      γ_1 ← γ_2
11:      γ_2 ← the new collided surface element of B
12:   else
13:      p ← p_1
14:      from p perform an approach step by turning D_2 to find the next point p_1
15:      feasible ← true
16:      repeat
17:         if TryPossibleStepMotions(p, p_1) returns a feasible step motion AND an edge of f_B is not reached then
18:            p ← p_1
19:            from p perform another approach step without changing direction to find the next point p_1
20:            if TryPossibleStepMotions(p, p_1) returns a feasible step motion AND an edge of f_B is not reached then
21:               p ← p_1
22:               from p perform another approach step by turning D_2 to find the next point p_1
23:            else
24:               feasible ← false
25:               γ_1 ← γ_2
26:               γ_2 ← the collided surface element of B
27:            end if
28:         else
29:            feasible ← false
30:            γ_1 ← γ_2
31:            γ_2 ← the collided surface element of B
32:         end if
33:      until NOT feasible OR an edge of f_B is reached
34:      if γ_2 ≠ γ_1 AND γ_2 = γ_c then
35:         m ← m + 1 {γ_c is encountered again}
36:      else if m = 1 AND γ_2 = γ_c then
37:         n ← n + 1
38:      end if
39:   end if
40: until an edge of f_B is reached OR m = M OR n = N
41: return p, γ_2
```

step motions include: a small **slidingA** motion and an infinite number of small combined motions of **slidingA** and rotation as there are an infinite number of possible rotation axes through the contacting vertex o_A. In such a case, **TryPossibleStepMotions** discretize the space of rotation axes in order to discretize possible step motions into a finite set to find a feasible motion among them. If no feasible motion can be found, **TryPossibleStepMotions** reports that and returns the collided elements of B.

The function **FollowEdge** checks if a feasible motion can be found step by step to reach the desired e_B by following another edge, where the direction of **p** is the direction of motion. The feasible motion will be along a sequence of edges until e_B is reached. Note that A could change its contact points with an edge by sliding or translating in directions orthogonal to **p** during the edge following.

The function **ObstacleTrace** as shown in Algorithm 3 is called once a collision is detected. It finds a motion to trace

the obstacle by alternating **trace** steps to move sideway (in the outer loop) and **approach** steps to move towards the obstacle (in the inner loop). Note that two global variables m and n are incremented in this procedure. The algorithm terminates when either an edge of f_B is reached or some loops are formed, as indicated by $m = M$ or $n = N$ (Algorithm 2).

Figure 17 uses an example to illustrate how **ObstacleTrace** works. Figure 17a shows a contact state under a CF of a single *f-f* PC. Figure 17b shows the top view of the face f_B of object B, in which object A is viewed as a disc. The shadow area indicates the protrudent region of object B projected to f_B, which causes collision with object A. The path τ starts from (u_0, v_0) along v to reach e_B, the desired edge. The resulting path as a curve on f_B from **ObstacleTrace** is shown in thick lines until another edge e'_B is reached.

The procedure **FindNewStart** is called when the previous starting configuration of A cannot result in a feasible motion to reach e_B, even although A's orientation during the motion may also be adjusted step by step by **TryPossibleStepMotions**. The procedure **FindNewStart** finds a different starting orientation of A compliant to CF_i and, if necessary, a different starting point (u_0, v_0) of τ. Given the previous starting point (u_0, v_0) of τ that A's task frame origin o_A contacts, A's orientation can be changed by pure rotations about an axis (or axes) through (u_0, v_0) or **slidingB** in different directions and different amount as long as there is no collision.

As there can be an infinite number of starting orientations, **FindNewStart** systematically discretizes the possible directions (axes) and amounts of rotation so that the number of starting orientations form a finite set. If an orientation change by rotation or **slidingB** cannot be achieved at (u_0, v_0) due to collision, a one-step **slidingA** motion (or a translation for a flat f_B) in the opposite direction of τ, e.g. in the opposite direction of u, is conducted before the orientation change.

If the orientation change still cannot be achieved, a one-step **slidingA** motion sideways (i.e. along v if τ starts by along u) is conducted before the orientation change. This process of one-step back **slidingA** or one-step sideways **slidingA** followed by the orientation change is repeated until either (1) the orientation change is done or (2) a new collision is formed so that CF_i cannot be maintained. If (2) occurs, it means that the intended orientation change cannot happen. **FindNewStart** then proceeds to seek the next possible orientation change. Note that each sliding (or translation) motion will move o_A away from (u_0, v_0). In such a case, we either update A's task frame and make the point contacting (u_0, v_0) be the new origin o_A (in the case of **slidingB**) or update (u_0, v_0) (in the case of **slidingA**).

Alternatively, in order to apply to multiply-connected faces directly, our algorithm can be modified in a similar way to *Bug* algorithms (Lumelsky and Stepanov 1986; Kutulakos et al. 1993). In general, the path τ in our approach is similar to the M-line in *Bug* algorithms. In the basic *Bug* algorithms, the robot initially moves towards the goal until it encounters an obstacle; it then follows obstacle boundary. It leaves the obstacle boundary when some conditions are satisfied. In our approach, the **TryPossibleStepMotions** could be used to find the hit point as in *Bug* algorithms. Our **ObstacleTrace** procedure can be extended to include functionality of finding the leave point as in the *Bug* algorithms.

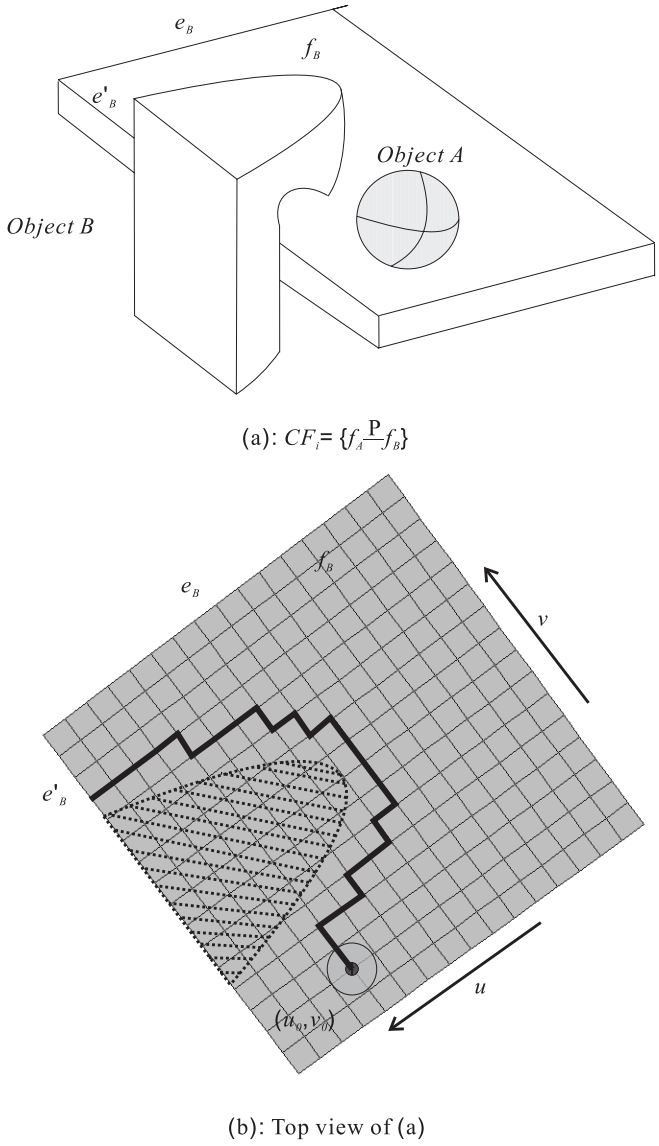


Fig. 17. Obstacle tracing example.

4.6. Soundness and Completeness

We now analyze our algorithm for generating LN graphs in terms of soundness and completeness. Recall that the algorithm generates an LN graph in a breadth-first search fashion, starting from a known seed contact state, and the generation of

each locally-defined (LN) contact state consists of hypothesis and test steps.

The first procedure, i.e. hypothesizing LN CFs, does not miss any topologically possible combinations of PCs in producing hypothesized LN CFs, and each hypothesized LN CF CF_j of a CF CF_i satisfies the requirements presented in Theorem 3.2 and Definition 3.7 for LN CFs. This procedure is therefore sound and complete.

We now examine the second procedure, i.e. checking if there exists a feasible neighboring transition motion from configuration C_i of a valid contact state $< CF_i, C_i >$ to a configuration C_j under a hypothesized LN CF CF_j. Here discretization is involved in a number of ways. Recall that a possible neighboring transition motion is constructed step by step as described in Section 4.3 to achieve compliant motion along a curved surface and, in each step, collision (other than the desired contact) is also checked.

In the cases where there are an infinite number of possible directions of one-step neighboring transition motions to remove some PCs, the directions are discretized and the step motion along each direction is checked until either a feasible motion is found or no motion is feasible. In the cases where the obstacle-tracing algorithm is used to find a feasible neighboring transition motion, if there is a solution under the discretizations of **FindNewStart** and **TryPossibleStepMotions**, the algorithm will find it; otherwise, it will report no solution.

We can show that there exists a finite threshold for the step size of discretization under which the second procedure is sound and complete. That is, it will find a feasible neighboring transition motion if one exists and it will discover and report correctly if no feasible motion exists. Let A and B be the two objects in contact.

Recall that a neighboring transition motion is a curve σ on the C-obstacle surface (of the configuration of A with respect to B) between two neighboring contact states $< CF_i, C_i >$ and $< CF_j, C_j >$. σ consists of a CF_i-compliant continuous segment σ_i and a CF_j-compliant continuous segment σ_j (Definitions 3.5 and 3.6). A feasible σ is one such that both σ_i and σ_j are free of collisions other than the contacts required by CF_i and CF_j respectively, i.e. both σ_i and σ_j are feasible.

Theorem 4.1: If there exists a feasible neighboring transition motion σ between $< CF_i, C_i >$ and $< CF_j, C_j >$, and if a CF_i-compliant motion has more than 1 DOF, then there exists at least a homotopy class of infinite number of CF_i-compliant motions that σ_i of σ belongs to, all leading to the feasible σ_j of σ. These homotopic CF_i-compliant curves form a continuous hyper-region S_i of a finite size in the patch of $< CF_i, C_i >$ on the C-obstacle (hyper)surface based on the Continuum Theory.

Proof: Suppose the theorem does not hold, then there is either only one feasible CF_i-compliant motion σ_i, or at least another feasible CF_i-compliant motion σ_i' that is not homotopic to σ_i but also meets σ_j at its starting point. This means that the region of $< CF_i, C_i >$ on the C-obstacle surface is a hypercurve (or a set of connected hypercurves), which has only 1 DOF (or independent variable). That contradicts the given condition that a CF_i-compliant motion has more than 1 DOF. ■

Based on the above theorem, as long as the discretized step size for each dimension is smaller than the size of S_i along that dimension, our procedure always finds a feasible σ_i if one exists. If a CF_i-compliant motion has only 1 DOF, there is only a finite set of possible feasible candidates. By checking each candidate one by one, a feasible σ_i can always be found if one exists. Thus, our procedure can find a feasible σ_i if one exists and correctly report no feasible σ_i if it cannot find one. We can prove a similar theorem for CF_j-compliant motion.

To determine the size of S_i along each dimension, we find the smallest non-zero range for each dimension by mating two objects A and B in the most limiting way, which usually involves contacting the most narrow concave region of one of the objects. We then try to move A along that dimension to find out the range of motion, which determines the smallest non-zero size of S_i along that dimension. This is practically doable with an interactive system, such as the system moving a virtual A via a haptic device to contact a virtual B (Chou and Xiao 2005).

In summary, our algorithm for generating LN graphs is both sound and complete under a finite step size. This can be found rather easily based on the given geometry of the two objects A and B, taking into account the tolerances between their elements when the two objects are in contact.

5. Implementation

We have implemented the general algorithm as described in Section 4 for automatic generation of an LN graph from a seed contact state between two basic curved objects A and B. The algorithm is implemented in Microsoft Visual C++ 6.0. We currently use the free OPCODE collision detection library (Terdiman Terdiman) for detecting collisions other than the desired contacts in feasibility checking of neighboring transition motions. Note that all we need to know here is whether a collision happens beyond the desired contacts and is not how a collision happens. Thus, a mesh-based detection package serves the purpose well. Other polygonal mesh-based collision detection packages could be used too. On the other hand, it should be emphasized that our approach generates exact contacts directly from contacting smooth surface features (represented parametrically) without polygonal mesh approximation.

Figures 18–20 show three examples where our algorithm has been applied. In Figure 18, objects are strictly curved objects without any planar region. In Figures 19 and 20, objects include some curved surfaces and planar surfaces. For all three examples, the step size for angular discretization is $\pi/24$. The step size for non-angular discretization is usually $1/25$ unit in

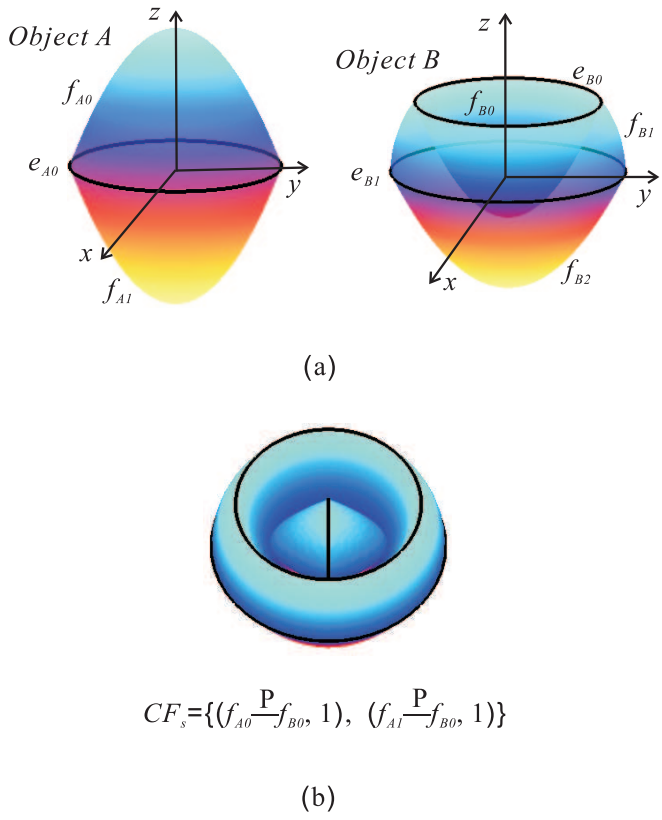


Fig. 18. Example 1.

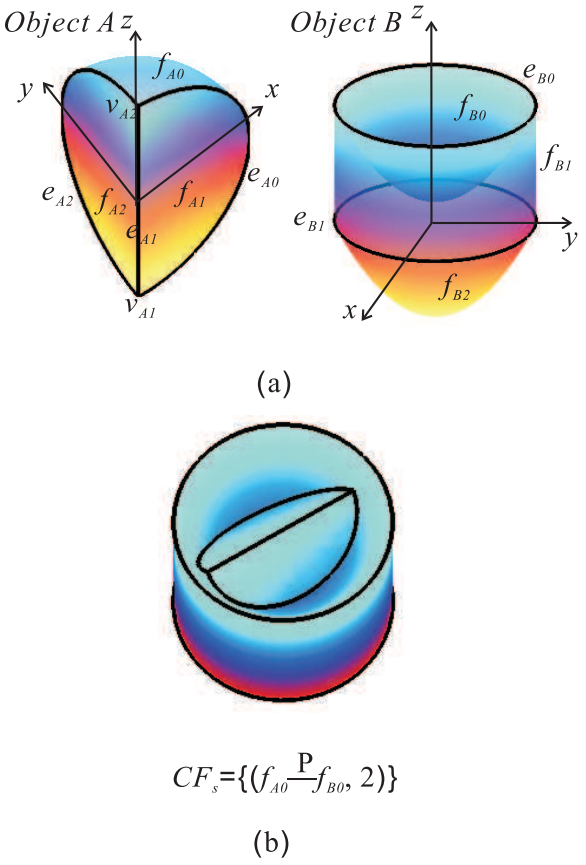


Fig. 19. Example 2.

the parametric domain and larger for contacts occurring on the outer surface of B.

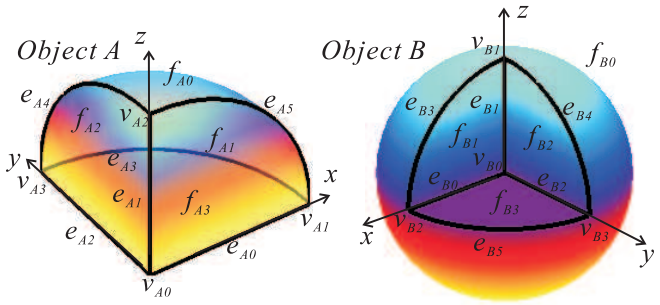
In Figure 18, object A has a surface consisting of two elliptic paraboloid faces: f_{A0} and f_{A1} which meet at edge e_{A0}. Object B includes three different faces: the upper part f_{B0} is an elliptic paraboloid concave face, the middle one f_{B1} is a part of a sphere and the lower part f_{B3} is an elliptic paraboloid face. It also includes two planar circular edges e_{B0} and e_{B1}. The parametric equations of the objects are in Appendix C.

In Figure 19, object A is a quarter of a solid ellipsoid. Its surface consists of three different smooth surface patches: an elliptic face f_{A0} and two planar faces f_{A1} and f_{A2}. Its surface also includes two curved edges e_{A0} and e_{A2}, one straight line edge e_{A1} and two vertices v_{A1} and v_{A2}. Object B is a curved object with three faces: the upper part f_{B0} is an elliptic paraboloid concave face, the middle part f_{B1} is a cylinder and the lower part f_{B2} is an elliptic paraboloid face. It also has two circular edges e_{B0} and e_{B1}. The corresponding parametric equations are in Appendix C.

In Figure 20, object A is the first quadrant of a solid sphere which includes one face f_{A0}, three planar faces f_{A1}, f_{A2} and f_{A3}, four vertices v_{A0}, v_{A1}, v_{A2} and v_{A3}, three curved edges e_{A3}, e_{A4} and e_{A5} and three straight line edges e_{A0}, e_{A1} and e_{A2}. Object B is a solid sphere with the first quadrant cut off. It includes one curved face f_{B0}, three planar faces f_{B1}, f_{B2} and f_{B3}, four vertices v_{B0}, v_{B1}, v_{B2} and v_{B3}, three curved edges e_{B3}, e_{B4} and e_{B5} and three straight line edges e_{B0}, e_{B1} and e_{B2}. The corresponding parametric equations can be found in Appendix C.

In order to have a clear observation, the objects are displayed with transparency and edges are drawn in solid lines. The seed contact states for the three examples are shown in Figures 18b, 19b and 20b respectively, with the contact formations labeled. In Figure 18b, the seed contact state shows a case where two point contacts are between two different pairs of surface elements. In Figure 18b, the seed contact state shows a case where two point contacts are between the same pair of surface elements. In Figure 19b, the seed contact state has three plane contacts among three pairs of planar faces.

For Example 1 in Figure 18, our algorithm has generated a LN graph of 18 valid nodes automatically from the seed contact state CS_s, which is the complete contact state graph for that example. Figure 21 displays some valid contact states generated. Figure 22 displays the entire contact state graph with 18 states. For Example 2 in Figure 23, our algorithm has generated a LN graph of 43 valid nodes including the seed from the



(a)

CF_s={(f_{A1} PL f_{B1}, 1), (f_{A2} PL f_{B2}, 1), (f_{A3} PL f_{B3}, 1)}

(b)

Fig. 20. Example 3.

seed contact state CS_s. This is the complete contact state graph for Example 2. Figure 23 displays some valid contact states generated for this example. For Example 3 in Figure 20, our algorithm has generated its LN graph consisting of 156 valid nodes (including the seed) automatically from CS_s. Figure 24 displays some valid contact states generated.

From these examples we can see that because a seed contact state is chosen to maximize the number of PCs, if there is one such contact state globally a complete contact state graph can be generated from the seed. This is a very nice property of an LN graph. If a complete contact state graph requires the merging of two or more LN graphs, the minimum number of necessary seed contact states (or LN graphs) is the number of local maxima states, i.e. states that have the most number of PCs compared to all neighboring states. A local maximum state usually involves concave elements in contact.

The program was executed on a Pentium 4, 2.8 GHz machine with 1024 MB RAM. The running time for Example 1 (in Figures 18 and 21) is 27.364 s. The running time for Example 2 (in Figures 19 and 23) is 21.484 s. The running time for Example 3 (in Figures 20 and 24) is 42.007 s.

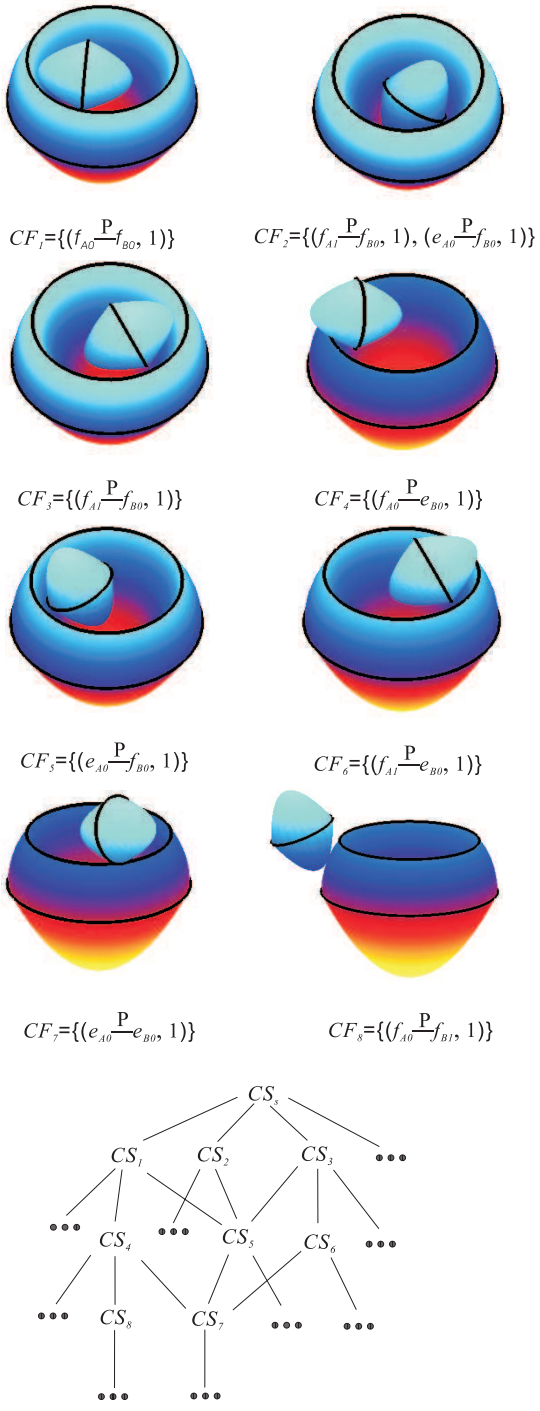


Fig. 21. Some contact states generated in Example 1.

6. Discussion of Complexity

The complexity of our algorithm for generating a LN graph mainly depends on the total number of hypothesized nodes in the LN graph the number of collision checks in constructing

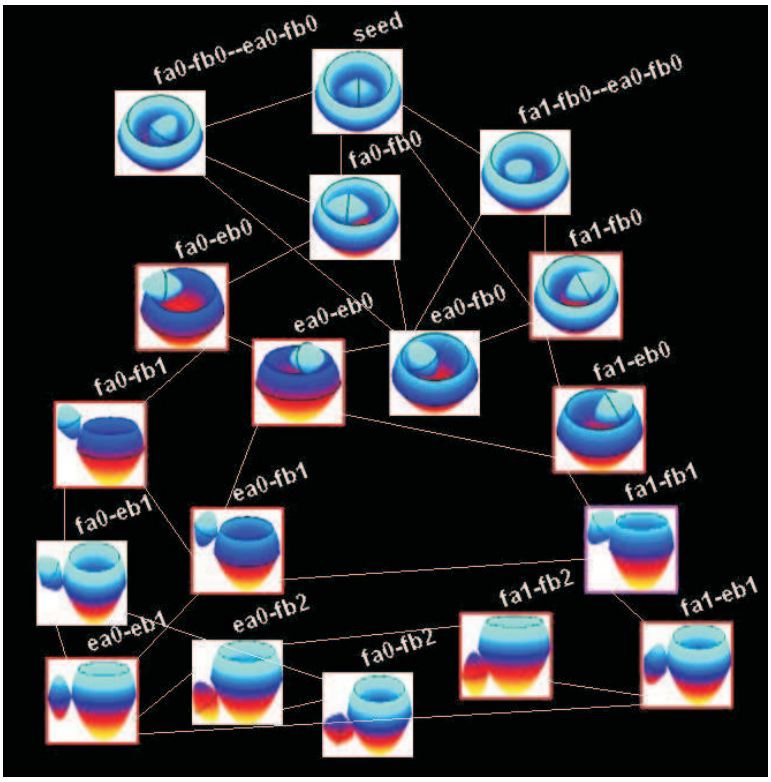


Fig. 22. The contact state graph for Example 1.

neighboring transition motions, and the time for each collision detection query.

Let N_A and N_B be the total number of surface elements of two contacting curved objects A and B, respectively. For a pair of elements α_A and α_B, there are up to three different contact types (i.e. point, line and plane contacts). Thus, an upper bound on the total number of different single-PC CFs is $3N_AN_B$. In a multi-PC CF, there could be the same PC for multiple times, depending on the geometric characteristics of the contacting elements of the PC related to convexity and concavity. Therefore, the greatest upper bound on the total number of hypothesized contact states with no more than three PCs is:

$$9N_AN_B + \begin{pmatrix} 3N_AN_B \\ 2 \end{pmatrix} + \begin{pmatrix} 3N_AN_B \\ 3 \end{pmatrix}$$

which is of $O(N_A^3N_B^3)$. In practice, the actual number of hypothesized contact states is much smaller, with topologically impossible cases eliminated according to Appendix A. For the three examples presented in the previous section, the actual numbers of hypothesized contact states are 57, 212 and 3163 respectively. These numbers are 5.3%, 2.9% and 0.0093% of the greatest upper bounds respectively. Note also that, as shown in the previous section, the actual number of valid contact states generated for these examples are 18, 43 and 156 respectively, which represent 31%, 20% and 5% of hypothesized states, respectively. The more combinations of PCs, the more drastic is the reduction of numbers in percentage.

Note also that for $m > 3$, an m-PC CF is likely to contain redundancy in contact constraints because an object can usually be immobilized by three contact points. Therefore, an LN CF of a m-PC CF ($m > 3$) is usually a two-PC CF or a single-PC CF.

7. Conclusions

We have presented a general and concise representation of contact states between two basic curved objects and provided a systematic approach for automatic generation of such contact state space as contact state graphs. The approach is sound and complete if the step size of discretization is smaller than a finite threshold. By exploiting topological and geometrical constraints, it is also quite efficient.

Automatic generation of contact state graphs between curved objects is not only highly desirable (because it is tedious to generate such states manually), but also necessary since many contact states cannot be accurately visualized easily. Unlike contact states between polyhedral objects, it is often

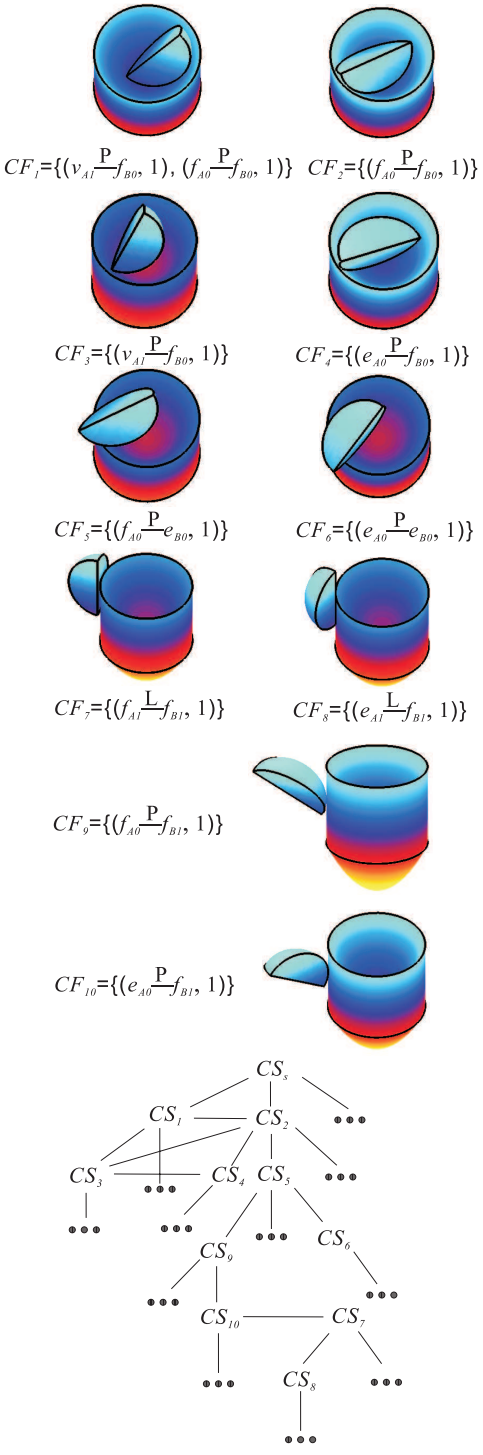


Fig. 23. Some contact states generated in Example 2.

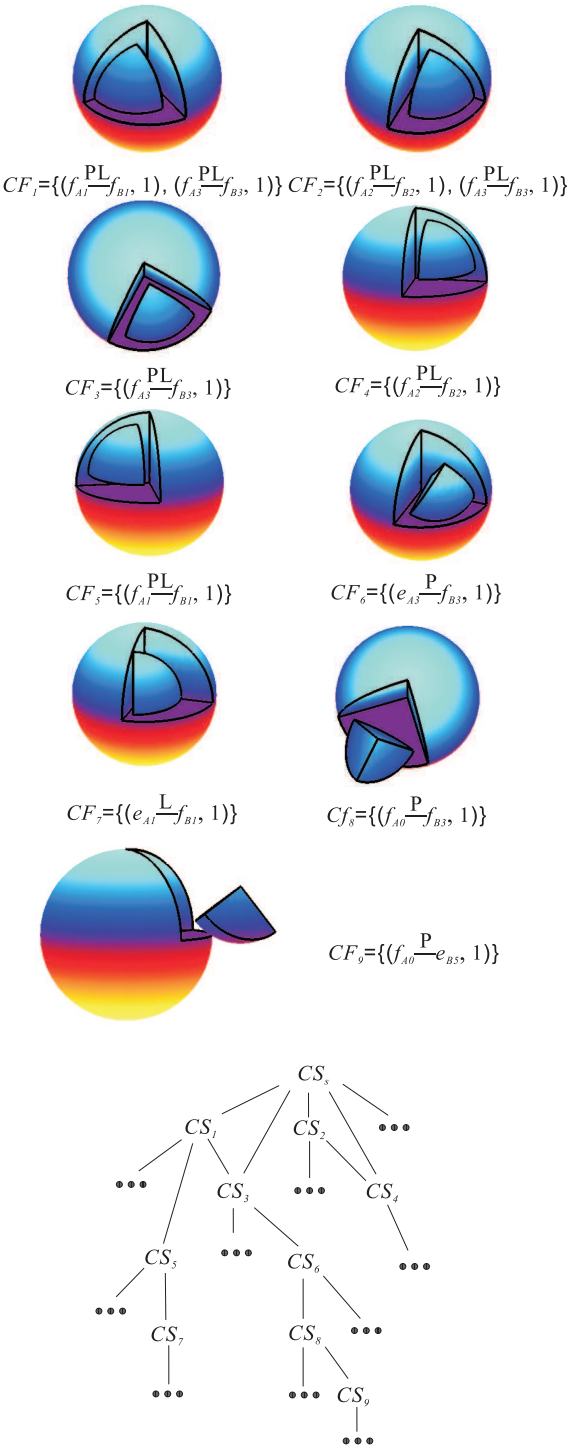


Fig. 24. Some contact states generated in Example 3.

much less obvious to human eyes whether a contact state is actually possible or not between two curved objects. On the other hand, there are more curved objects than polyhedral objects in the real world.

As a future step, our work can be extended to address contact states between an even broader class of curved objects, including those with space curves (edges).

Acknowledgements

This work is supported by the US National Science Foundation under grant No. IIS-#0328782.

Appendix A: Ruling Out Topologically Impossible Principal Contacts

Given a pair of surface elements only certain types of principal contacts (PCs) are possible and, in some cases, no PC is possible – all depending on the topological properties of the surface elements.

Based on different properties, we can differentiate surface elements into different types. First, we say that a curved face is *convex* if all its outward normals point away from each other, *concave* if all its outward normals point toward each other or is *convex-concave* if it contains both convex and concave segments. We say that two faces of an edge form a *valley* if the edge's neighboring points on the faces have normals pointing toward each other; otherwise, they form a *hump*.

We further define the different types of vertices and edges:

- *convex edge*: an edge that is a convex curve or a line segment and whose faces form a hump;
- *convex vertex*: a vertex such that every pair of adjacent faces form a hump, or it is a singular point on a convex (part of a) face;
- *concave edge*: an edge that is a concave curve or a line segment and whose faces form a valley;
- *concave vertex*: a vertex such that every pair of adjacent faces form a valley, or it is a singular point on a concave (part of a) face;
- *convex-edge in valley*: an edge that is a convex curve and whose faces form a valley;
- *concave-edge on hump*: an edge that is a concave curve and its faces form a hump; and
- *convex-concave vertex*: a vertex with both valleys and humps formed between its adjacent faces.

Figure 25 shows some examples of different kinds of faces, vertices and edges.

We now can use simple rules to eliminate topologically impossible PCs:

- no PC with a vertex can be a line or plane contact;
- no PC with two planar faces can be a point contact;
- no PC with a straight-line edge and a planar face can be a point or plane contact;

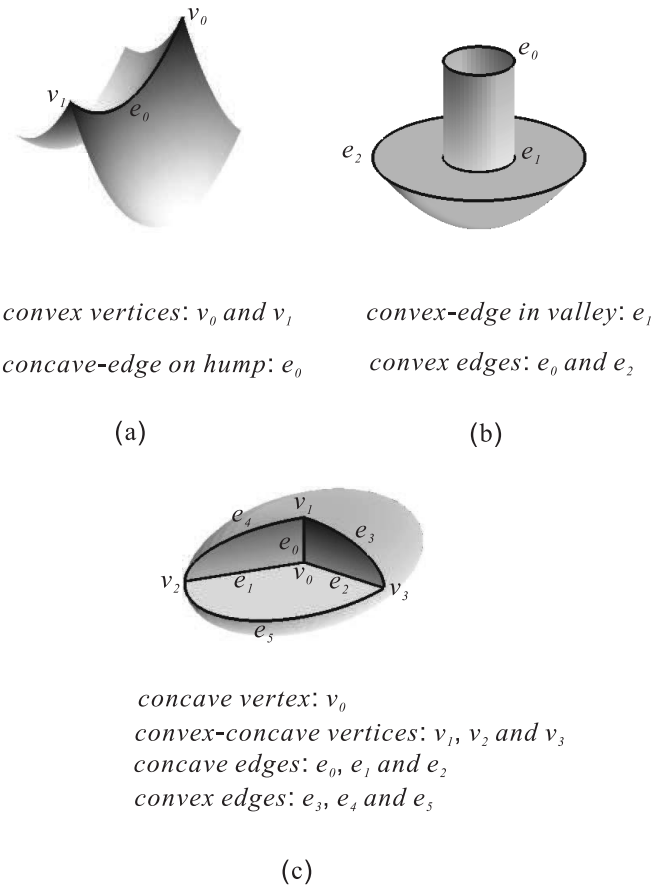


Fig. 25. Different types of vertices and edges.

- no PC with two edges can be a plane contact;
- no PC with a curved edge can be a line contact;
- no PC with a non-ruled face can be a line contact;
- no PC can be formed between two concave surface elements;
- no PC can be formed between a convex-in-valley edge or concave edge and a face;
- no PC can be formed between a concave vertex and a face or an edge;
- no PC can be formed between a concave-on-hump edge and a concave face/edge; and
- no PC can be formed between a concave face and a planar face or a straight-line edge.

Appendix B: Possible Compliant Motions to Maintain a Principal Contact

The types of possible compliant motions to **keep** or maintain a principal contact $PC_i = \alpha^i_A \overset{\xi}{-} \alpha^i_B$ depends on the contact type ξ and the types of the boundary elements of α^i_A and α^i_B, as follows.

- To keep a *v-v* point PC, the only possible motion is a **pure rotation**.
- To keep a *v-e/v-f* point PC, if α^i_B is a curved edge or non-ruled surface, the possible motions are **pure rotation**, **slidingA** motion, or **combined slidingA and rotation**. If α^i_B is a straight-line edge or a ruled surface, additional possible motions are **pure translation** and **combined rotation and translation**.
- To keep a *e-v/f-v* point PC, the possible motions are similar to the case of a *v-e/v-f* PC but the edge/face in this case is α^i_A, and **slidingA** above becomes **slidingB**.
- To keep an *e-f* point PC, if the edge e is curved and the face f is a non-ruled surface, the possible motions are **pure rotation** (about the tangent line of the edge or about the normal line at the contact point), **slidingA**, **slidingB** along the edge (i.e. one-dimensional), **combined slidingA and slidingB**, and **combined sliding and rotation** motions; if the edge e is a straight line edge or the face f is a ruled surface, then additional possible motions are **pure translation** and **combined translation and rotation**.
- To keep a *f-e* point PC, the possible motions are similar to the above for case *e-f*, except that **slidingA** is along the edge.
- To keep a *f-f* point PC where no face is a ruled surface, the possible motions are **pure rotation** about the contact normal, **slidingA**, **slidingB**, **combined slidingA and slidingB**, and **combined sliding and rotation** motions.
- To keep a *f-f* point PC where at least one face is a ruled surface but no face is planar, the possible motions are **pure rotation** about the contact normal, **pure translation**, **slidingA**, **slidingB**, **combined slidingA and slidingB**, **combined sliding and rotation**, and **combined translation and rotation** motions.
- To keep a *f-f* point PC where one face is a planar, the possible motions are **pure rotation** about the contact normal, **pure translation**, **slidingA** if α^i_A is the planar face, else **slidingB**, **combined translation and rotation** and **combined sliding and rotation**.
- To keep an *e-e-touch* point PC where both edges are curved, the possible motions **pure rotation** about the tangent line, **slidingA** along the edge of B, **slidingB** along the edge of A, **combined slidingA and slidingB**, and **combined sliding and rotation**.
- To keep an *e-e-touch* point PC where one edge is curved and the other is a straight line edge, the possible motions are **pure rotation** about the straight line edge, **pure translation** along the straight line edge, **combined translation and rotation**, **slidingA** if the edge of B is curved, else **slidingB** and **combined sliding and rotation**.
- To keep an *e-e-touch* point PC where both edges are straight line edges, the possible motions are **pure rotation** about the straight line(s), **pure translation** along the straight line(s) and **combined translation and rotation**.
- To keep an *e-e-cross* point PC where both edges are curved, in addition to the motions possible for an *e-e-touch* point PC where both edges are curved, **pure rotation** about the contact normal is also possible.
- To keep an *e-e-cross* point PC where one edge is curved and the other is a straight line edge, in addition to the motions possible for an *e-e-touch* point PC where one edge is curved and the other is a straight line edge, **pure rotation** about the contact normal and a combined translation, rotation and sliding motion are possible.
- To keep an *e-e-cross* point PC where both edges are straight line edges, the possible motions are **pure rotation** about the straight line(s) or about the contact normal, **pure translation** and **combined translation and rotation**.
- To keep an *e-f/f-e* line PC where the edge e is straight line and the face f is planar, the possible motions are **pure rotation** (about the edge or about a contact normal), **pure translation** and **combined translation and rotation**.
- To keep an *e-f/f-e* line PC where the edge e is straight line and the face f is a ruled surface, the possible motions are **pure rotation** about the edge, **pure translation** along the edge, **combined translation and rotation**, **slidingA** if α^i_A is the edge e else **slidingB** and **combined sliding and rotation** motions.
- To keep an *f-f* plane contact, the possible motions are **pure rotation**, **pure translation** and **combined translation and rotation** motions.

Appendix C: Parametric Equations of Example Objects

Object A (Example 1, Figure 18)

$$f_{A0}: x = 1.5v\cos u,\ y = 1.5v\sin u,\ z = 2 - 2v^2,$$

$$f_{A1}: x = 1.5v\cos u,\ y = 1.5v\sin u,\ z = 2v^2 - 2$$

where $u \in [0, 2\pi], v \in [0, 1]$.

Object B (Example 1, Figure 18)

$$f_{B0}: x = 3.1734v\cos u,\ y = 3.1734v\sin u,$$

$$z = 4v^2 - 1.56495$$

where $u \in [0, 2\pi], v \in [0, 1]$,

$$f_{B1}: x = 4\cos u\sin v,\ y = 4\sin u\sin v,\ z = 4\cos v$$

where $u \in [0, 2\pi], v \in \left[\frac{7}{24}\pi, \frac{1}{2}\pi\right]$,

$$f_{B2}: x = 4v\cos u,\ y = 4v\sin u,\ z = 4v^2 - 4$$

where $u \in [0, 2\pi], v \in [0, 1]$.

Object A (Example 2, Figure 19)

$$f_{A0}: x = 2\cos u\sin v,\ y = 2\sin u\sin v,\ z = 2.5\cos v$$

where $u \in \left[0, \frac{\pi}{2}\right], v \in [0, \pi]$,

$$f_{A1}: x = v\sin u,\ y = 0,\ z = 1.25v\cos u$$

where $u \in [0, \pi], v \in [0, 2]$,

$$f_{A2}: x = 0,\ y = v\cos u,\ z = 1.25v\sin u$$

where $u \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], v \in [0, 2]$.

Object B (Example 2, Figure 19)

$$f_{B0}: x = 3v\cos u,\ y = 3v\sin u,\ z = 3v^2 + 2$$

where $u \in [0, 2\pi], v \in [0, 1]$,

$$f_{B1}: x = 3\cos u,\ y = 3\sin u,\ z = v$$

where $u \in [0, 2\pi], v \in [0, 5]$,

$$f_{B2}: x = 3v\cos u,\ y = 3v\sin u,\ z = -3v^2$$

where $u \in [0, 2\pi], v \in [0, 1]$.

Object A (Example 3, Figure 20)

$$f_{A0}: x = 2\cos u\sin v,\ y = 2\sin u\sin v,\ z = 2\cos v$$

where $u \in \left[0, \frac{\pi}{2}\right], v \in \left[0, \frac{\pi}{2}\right]$

and

$$f_{A1}: x = v\cos u,\ y = 0,\ z = v\sin u,$$

$$f_{A2}: x = 0,\ y = v\cos u,\ z = v\sin u,$$

$$f_{A3}: x = v\cos u,\ y = v\sin u,\ z = 0$$

where $u \in \left[0, \frac{\pi}{2}\right], v \in [0, 2]$.

Object B (Example 3, Figure 20)

Upper $f_{B0}: x = 3\cos u\sin v,\ y = 3\sin u\sin v,$

$z = 3\cos v$

where $u \in \left[0, \frac{3\pi}{2}\right], v \in \left[0, \frac{\pi}{2}\right]$,

Lower $f_{B0}: x = 3\cos u\sin v,\ y = 3\sin u\sin v,$

$z = -3\cos v$

where $u \in [0, 2\pi], v \in \left[0, \frac{\pi}{2}\right]$,

$$f_{B1}: x = v\cos u,\ y = 0,\ z = v\sin u,$$

$$f_{B2}: x = 0,\ y = -v\cos u,\ z = v\sin u,$$

$$f_{B3}: x = v\cos u,\ y = -v\sin u,\ z = 0$$

where $u \in \left[0, \frac{\pi}{2}\right], v \in [0, 3]$.

References

Avnaim, F., Boissonnat, J. D., and Faverjon, B. (1988). A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. In *Proceedings of IEEE International Conference on Robotics & Automation*, pp. 1656–1661.

Brost, R. (1989). Computing metric and topological properties of configuration-space obstacles. In *Proceedings of IEEE International Conference on Robotics & Automation*, pp. 170–176.

Canny, J. (1988). *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA.

Chou, W. and Xiao, J. (2005). An interactive approach to determining complex contact states. In *Proceedings of 2005 IEEE Symposium on Assembly & Task Planning*, pp. 106–111.

Donald, B. (1985). On motion planning with six degrees of freedoms: Solving the intersection problems in configuration space. In *Proceedings of IEEE International Conference on Robotics & Automation*, pp. 536–541.

Edelsbrunner, H. and Mucke, E. (1990). Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics,* 9(1): 66–104.

Hirukawa, H., Paperguy, Y., and Mastui, T. (1994). A motion planning algorithm for convex polyhedra in contact under translation and rotation. In *Proceedings of IEEE International Conference on Robotics & Automation*, pp. 3020–3027.

Ji, X. and Xiao, J. (2001). Planning motion compliant to complex contact states. *International Journal of Robotics Research,* 20(6): 446–465.

Joskowicz, L. and Taylor, R. H. (1996). Interference-free insertion of a solid body into a cavity: An algorithm and a medical application. *International Journal of Robotics Research,* 15(3): 211–229.

Kutulakos, K. N., Lumelsky, V. J., and Dyer, C. R. (1993). Vision-guided exploration: A step toward general motion planning in three dimensions. In *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 289–296.

Lefebvre, T. (2003). *Contact Modeling, Parameter Identification and Task Planning for Autonomous Compliant Motion using Elementary Contacts*. Ph. D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium.

Lefebvre, T., Xiao, J., Bruyninckx, H., and Gersem, G. D. (2005). Active compliant motion: A survey. *Advanced Robotics,* 19(5): 479–500.

Lozano-Pérez, T. (1983). Spatial planning: A configuration space approach. *IEEE Transactions on Computing,* 32(2): 108–120.

Lumelsky, V. J. and Stepanov, A. A. (1986). Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Transactions on Automatic Control,* 31(11): 1058–1063.

Luo, Q., Staffetti, E., and Xiao, J. (2004). Representation of contact states between free-form objects. pp. 3589–3595.

Luo, Q. and Xiao, J. (2004). Physically accurate haptic rendering with dynamic effects. *IEEE Computer Graphics and Applications (Special Issue: Touch-Enabled Interfaces),* 24: 60–69.

McCarragher, B. J. (1996). Task primitives for the discrete event modeling and control of 6-DOF assembly tasks. *IEEE Transactions on Robotics and Automation,* 12(2): 280–289.

Meeussen, W., De Schutter, J., Bruyninckx, H., Xiao, J., and Staffetti, E. (2005). Integration of planning and execution in force controlled compliant motion. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1217–1222.

Mortenson, M. E. (1985). *Geometric Modeling*. John Wiley & Sons Inc, New York, NY.

Pan, F. and Schimmels, J. M. (2003). Efficient contact state graph generation for assembly applications. In *Proceedings of IEEE International Conference on Robotics & Automation*, pp. 2591–2598.

Rosell, J., Basañez, L., and Suárez, R. (1997). Determining compliant motions for planar assembly tasks in the presence of friction. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots & Systems*, pp. 946–951.

Ruspini, D. and Khatib, O. (1999). Collision/contact models for dynamic simulation and haptic interaction. In *Proceedings of 9th International Symposium on Robotics Research*, pp. 185–194.

Sacks, E. and Bajaj, C. (1998). Sliced configuration spaces for curved planar bodies. *International Journal of Robotics Research,* 17(6): 639–651.

Stroud, I. (1990). Modeling with degenerate objects. *Computer-Aided Design,* 22(6): 344–351.

Sturges, R. H. and Laowattana, S. (1995). Fine motion planning through constraint network analysis. In *Proceedings of International Conference on Assembly and Task Planning*, pp. 160–170.

Tang, P. and Xiao, J. (2006a). Automatic generation of contact state graphs based on curvature monotonic segmentation. In *Proceedings of IEEE International Conference on Robotics & Automation*, pp. 2633–2640.

Tang, P. and Xiao, J. (2006b). Generation of point-contact state space between strictly curved objects. In *Proceedings of Robotics: Science and Systems Conference*, pp. 31–38.

Terdiman, P. http://www.codercorner.com/opcode.htm.

Thompson II, T. V. and Cohen, E. (1999). Direct haptic rendering of complex trimmed nurbs models. *ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems*.

Xiao, J. (1993). Automatic determination of topological contacts in the presence of sensing uncertainties. In *Proceedings of IEEE International Conference on Robotics & Automation*, pp. 65–70.

Xiao, J. and Ji, X. (2001). On automatic generation of high-level contact state space. *International Journal of Robotics Research,* 20(7): 584–606.