Generation of Point-contact State Space between Strictly Curved Objects*

Peng Tang
IMI Lab, Dept. of Computer Science
University of North Carolina - Charlotte
Charlotte, NC 28223, USA
ptang@uncc.edu

Jing Xiao
IMI Lab, Dept. of Computer Science
University of North Carolina - Charlotte
Charlotte, NC 28223, USA
xiao@uncc.edu

***Abstract*— Isolated point contacts are most common between two curved objects that do not have line segment on their surfaces. This paper addresses how to represent concisely and generate automatically graphs of topological contact states made of isolated contact points between such curved objects. Information of contact states is useful for a wide range of applications, from robotic tasks involving compliant motion to virtual prototyping, haptic rendering, and dynamic simulation. The approach has been implemented with an effective algorithm.**

***Index Terms*— point-contact states, 3-D curved objects, automatic generation, compliant motion**

I. Introduction

Many robotic tasks involve objects in contact and compliant motion and require the information of contact geometry. It is often necessary to know not only the contact configuration between two objects but also the high-level, discrete, topological contact state that is more descriptive of the topological and physical characteristics of contact shared by two or more contact configurations. 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 such a graph can be useful for automatic assembly planning or control [9], [13], [19], [15].

Information of contact states can facilitate planning and executing compliant motion in general. Planning compliant motion means planning motion on the surface of configuration space obstacles (C-obstacles) [10]. This poses a major problem because computing C-obstacles exactly in high-dimensional space remains a formidable task to date [4]. Most of the relevant work is limited to 3-D C-obstacles (i.e., C-obstacles of planar objects) [1], [3], [16], [18], and only a few concern approximation of C-obstacles of 3-D polyhedra [5], [7]. Planning compliant motion is greatly simplified with a known contact state graph. Here planning can be decomposed 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 compliant motion within a known contact state and a transition to a neighboring contact state at the low-level, which is a *lower dimension* and *smaller scope* motion planning problem [6].

In view of compliant motion control, it is not sufficient to know only a path of contact configurations because it is not possible to have a compliant control law applicable to all contact configurations, but rather it is practical to have a stratification of compliant control strategies based on different contact states and transitions [8], [14], i.e., information of a contact state graph is often necessary.

In haptic rendering and dynamic simulation (e.g., [12], [17]), collision detection is inevitably subject to digital error. As a result, when multiple collisions are detected as happened at the same time during simulation, actually 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 needed here.

There is considerable existing work concerning contact states between polyhedral objects. For contacting polyhedral objects, it is common to describe 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 [10], [5] 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 [23] 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. A general approach was developed to generate automatically graphs of contact states based on such a notion [24]. For convex curved objects, contact states are described similarly with each primitive contact defined in terms of a pair of contacting curved surface elements [22].

However, if non-convex curved surfaces or curves are present, between a pair of curved surface elements there can

*This work is supported by the U.S. National Science Foundation under grant IIS-#0328782.

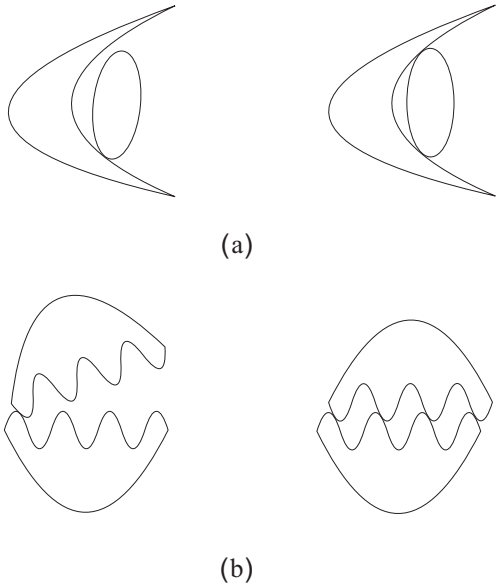

Fig. 1. Different numbers of contact regions can be formed between the same two contacting surfaces

be one or more than one contact region formed, resulting different contact states with different contact constraints. Figure 1 shows two examples. 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* [11], [20] so that between two curvature monotonic segments only one contact can be formed (i.e., a one-to-one mapping). However, such an approach of artificially dividing natural surfaces leads to a large number of contact states and does not work well for objects with space curves.

In this paper, we consider contact states consisting of isolated contact points between two arbitrary, *strictly curved* objects, i.e., those that have no line segment on their boundary surfaces, since such *point-contact states* are most common between strictly curved objects. We use a different approach to handle the one-to-many mapping problem illustrated in Figure 1 and represent point-contact states concisely without ambiguity (Section II). We next present an approach to generate graphs of such point-contact states automatically (Section III). We describe the implementation results in Section IV and provide a discussion of complexity in Section V. We conclude the paper in Section VI.

Note that in the rest of the paper, the term *curved objects* means strictly curved objects.

II. Representation of Point-contact States between Curved Objects

We first describe the boundary elements of 3-D curved objects, then describe a point contact, and finally introduce a new representation to describe point-contact states between two such objects.

A. Boundary Elements of Curved Objects

A 3-D curved object can be characterized by the following: the boundary of such an object consists of smooth surfaces or surface patches that can be described parametrically, which are called *faces*; the intersection smooth curve segments of two faces are called *edges*; and the intersection points of two or more edges or the singular points of a surface are called *vertices*. Clearly, a smooth surface patch that is not closed is bounded by edges and vertices, and a smooth edge that is not closed is bounded by vertices. We call the faces, edges, and vertices of a curved object its *boundary elements*. For simplicity, we use *f*, *e* and *v* to denote *face*, *edge*, and *vertex* respectively.

B. Topological Point Contacts between Curved Objects

We define a topological *point contact* (PtC) between two 3-D curved objects A and B in terms of the pair of boundary elements α_A and α_B that form it, denoted as α_A-α_B. Moreover, a *f-f* type PtC is a *face tangential contact* between two faces, i.e., the two faces are tangent to each other. A *f-e* (or *e-f*) type PtC is an *edge tangential contact* between the edge and the face but is not a face tangential contact. A PtC between two edges is either an *e-e-cross* contact if the tangent lines of the two edges at the contact point are not collinear or an *e-e-touch* contact if the two tangent lines at the contact point are collinear. An *e-e-cross*, *e-e-touch*, or a *v*-* (or *-*v*) type PtC is neither a face tangential nor an edge tangential contact because there is no face-face or face-edge pairs that are tangential to each other; these are called *non-tangential contacts*.

Figure 2 shows an example for each type of topological point contacts between 3-D curved objects.

We further define the *contact plane* of *f-f*, *f-e* (or *e-f*), *f-v* (or *v-f*), and *e-e-cross* types of point contacts as follows:

- *f-f*: the contact plane is the plane tangent to both faces at the contact point.
- *f-e* or *f-v*: the contact plane is the tangent plane of the face at the contact point.
- *e-e-cross*: the contact plane is the plane determined by the two tangent lines of the two edges at the contact point.

We define the *contact line* of a *v-e* (*e-v*) or an *e-e-touch* type point contact as the tangent line of the edge (or edges) at the contact point. No single contact plane or contact line can be defined for a *v-v* type point contact.

C. Point-contact States between 3-D Curved Objects

As shown in Figure 1, there can be different number of point contacts between the same two non-vertex boundary elements of two curved objects. Therefore, we characterize a general topological contact state between two curved objects by revising the notion of contact formations originally defined for polyhedral objects [23] as follows: A point *contact*

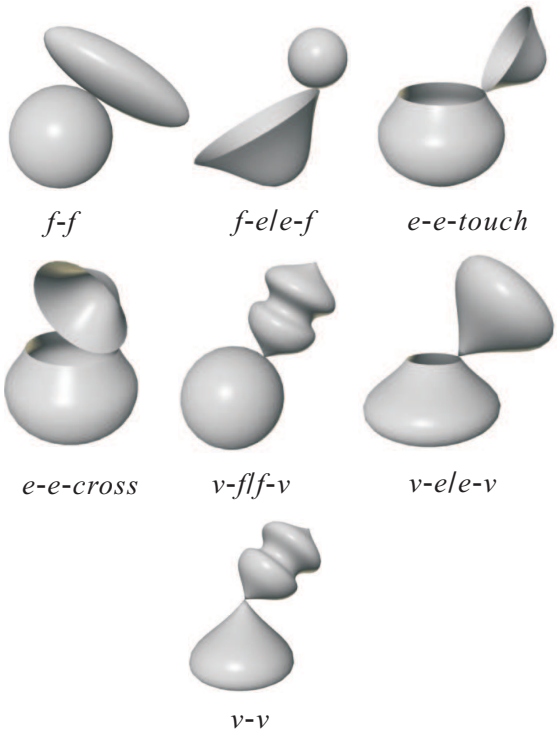

Fig. 2. The types of point contacts between two curved objects

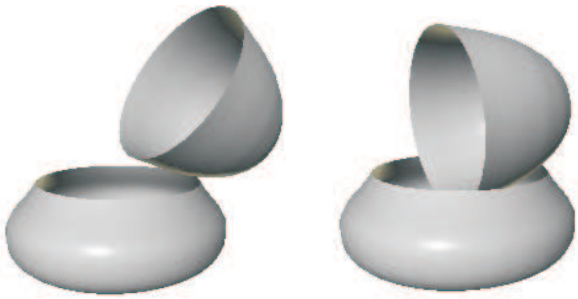

Fig. 3. Two point contact formations between two curved objects

formation(CF) between two curved objects is defined as the set of point contacts (PtCs) formed, where the same PtC may be formed more than once, denoted as:

CF = $\{(\text{PtC}_1, n_1), (\text{PtC}_2, n_2), \cdots, (\text{PtC}_m, n_m)\}$, where $n_i \in \mathbf{N}, \mathbf{N}$ is the set of positive integers, $i = 1 \cdot\cdot m$.

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

$$card(\text{CF}) = n_1 + n_2 + \cdots + n_m.$$

Figure 3 illustrates two point-contact formations between two curved objects.

The *geometrical representation* of a point CF denotes the set of relative contact configurations between the two curved objects that satisfy the contact conditions of all the PtCs in the CF at the same time. Generally, such a set may consist of one or more connected regions of contact configurations, called *CF-connected* regions in the configuration space. Within a CF-connected region, there exists a motion constrained by the CF from any contact configuration to any other one, called *CF-compliant* motion. In other words, there is no need to change the CF in moving from one configuration to another within a CF-connected region. Thus, we define a *point-contact state* between two curved objects as a single CF-connected region of configurations, represented by the point CF and a representative configuration in the region, denoted as a pair $<CF, C>$.

D. Locally-defined Neighbor (LN) and Globally-defined Neighbor (GN)

Two point-contact states $<CF_i, C_i>$ and $<CF_j, C_j>$ are called *neighboring point-contact states* if there exists a *neighboring transition motion* as a CF_i-compliant motion succeeded by a transition to a CF_j-compliant motion from C_i to C_j, and CF_i and CF_j are called *neighboring point-contact formations*.

If two single-point CFs, $CF_i = \{(PtC_i, 1)\}$ and $CF_j = \{(PtC_j, 1)\}$, where $PtC_i \neq PtC_j$, are neighboring point contact formations, then PtC_i and PtC_j are called *neighboring point contacts*. Moreover, two neighboring point contacts PtC_i $=\alpha_A^i$-α_B^i and PtC_j $=\alpha_A^j$-α_B^j satisfies one of the following conditions:

- α_A^i is α_A^j and α_B^i is adjacent to α_B^j,
- α_B^i is α_B^j and α_A^i is adjacent to α_A^j.

We can now further distinguish two kinds of neighboring point-contact states based on the topological information of the neighboring CFs. Given two neighboring point contact states $<CF_i, C_i>$ and $<CF_j, C_j>$, $<CF_j, C_j>$ is a *locally-defined neighbor* (LN) of $<CF_i, C_i>$ if the following are satisfied:

- $card(CF_j) \leq card(CF_i)$, and
- every PtC in CF_j either belongs to CF_i or is a neighboring PtC of a PtC in CF_i.

Accordingly, CF_j is called the LN CF of CF_i. If $card(CF_j) < card(CF_i)$, then CF_i is a *globally-defined neighbor* (GN) of CF_j, $<CF_i, C_i>$ is a GN point-contact state of $<CF_j, C_j>$, and $<CF_j, C_j>$ is a LN point-contact state of $<CF_i, C_i>$. Figure 3 gives an example, where CF_1 is a LN CF of CF_2, and CF_2 is a GN CF of CF_1.

The reason that we differentiate neighboring point-contact states into LNs and GNs is that given a CF, the topological information of its LNs can be derived directly from its own topological definition: from the PtCs in the CF, one can obtain the possible PtCs of the LNs of the CF. This is a very useful

property for automatic generation of point-contact states (see next section).

The point-contact state space (of the contacting objects) can be defined as a point-contact state graph $\mathcal{G}$, where each node denotes a valid point-contact state $<CF, C>$, and each link connects two neighboring point-contact states.

III. Generation of Point-Contact State Graphs

Our approach is to generate special subgraphs of the point-contact state graph $\mathcal{G}$ automatically and to merge these subgraphs automatically. This approach sounds similar to the one used for polyhedral objects [24], but the special subgraphs considered here are different, and the details of generation are different due to different subgraphs and distinct natures of curved objects.

Each special subgraph we generate is an undirected graph consisting of a seed point-contact state $<CF_s, C_s>$, its LN point-contact states, their subsequent LN point-contact states, and so on, which we call a *LN graph* of $<CF_s, C_s>$. Starting from the seed point-contact state $<CF_s, C_s>$, the LN graph can be grown by repeatedly obtaining LN point-contact states until all the LN point-contact states have been generated in a breadth-first search. In the loop for obtaining a LN point-contact state, there are two key steps:

(1) From a known point-contact state $<CF_i, C_i>$, hypothesize its LN CFs based on the topological information of CF_i, and

(2) determine if a hypothesized LN CF, CF_j, is valid or not by checking if there is a feasible neighboring transition motion from C_i under CF_i to some configuration C_j under CF_j. If such a feasible motion exists, then $<CF_j, C_j>$ is a valid LN point-contact state of $<CF_i, C_i>$.

We explain both steps below.

A. Hypothesizing Locally-defined Neighboring (LN) Point-Contact Formations (CFs)

For a CF with a single point contact, $CF_i = \{(PtC_i, 1)\}$, to obtain a hypothesized LN CF of CF_i is to **change** PtC_i to one of its neighboring PtCs.

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

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

B. Neighboring Transition Motions

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

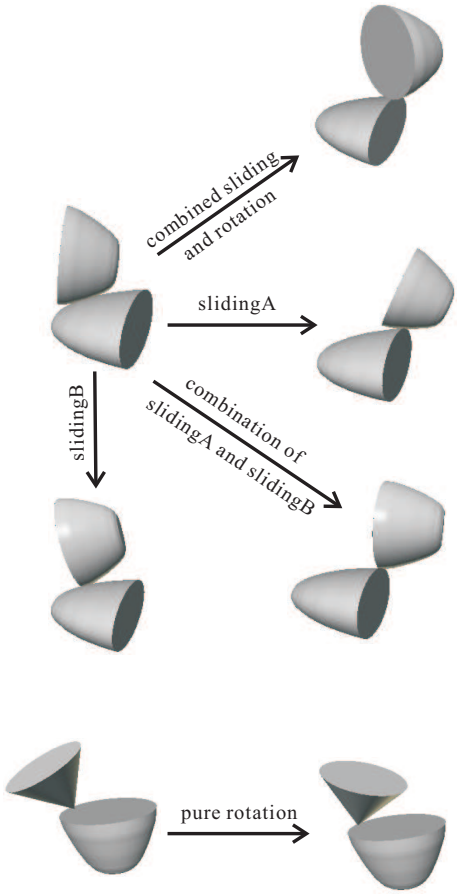

Fig. 4. Types of neighboring transition motions.

- **slidingA**, where the contact points of A do not change but the contact points of B changes (Figure 4);
- **slidingB**, where the contact points of B do not change but the contact points of A changes (Figure 4);
- **pure rotation** about an axis through a contact point, which is usually either on or normal to the contact plane (if the contact plane exists) (Figure 4);

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

Neighboring transition motions can also be of the following types of combined 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 4);
- a **combined sliding and rotation** motion, where a **slidingA** or a **slidingB** motion or a **combine sliding** is combined with a **pure rotation** (Figure 4);

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 PtC 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* PtC, either the x or the y axis can be along the tangent line of one of the edges.

For a PtC with a contact line, the x axis is along the contact line, 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 PtC 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, and 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 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. Similar strategy can be used to implement a small motion 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.

C. Checking the Feasibility of Neighboring Transition Motions

Given a valid point-contact state $<CF_i, C_i>$ and a hypothesized LN CF CF_j between two 3-D curved objects A and B, checking if there is a feasible neighboring transition motion from C_i to a configuration C_j of CF_j is to determine whether or not $<CF_j, C_j>$ is a valid LN CF of CF_i. Any neighboring transition may involve **remove**, **keep**, or **change** one or more PtCs of CF_i. The **change** action is to change a PtC to its neighboring PtCs.

The types of possible compliant motions (among the four types introduced in the previous subsection) to **change** a point-contact $PtC_i = \alpha_A^i$-α_B^i, to a neighboring point-contact, PtC_j, depends on the types of PtC_i and PtC_j as well as which boundary element in PtC_i is changed from PtC_i to PtC_j:

- Between (1) a non-tangential contact and a face or edge tangential contact, or (2) a face tangential contact and an edge tangential contact, a pure rotation is necessary for the transition. A sliding motion may also be needed.
- Between two non-tangential contacts, a sliding motion is necessary.
- If α_A^i has to be changed to an adjacent boundary element, a **slidingB** motion is needed. Otherwise, if α_B^i has to be changed to an adjacent boundary element, a **slidingA** motion is needed.

The types of possible compliant motions to **keep** or maintain a point-contact $PtC_i = \alpha_A^i$-α_B^i depends on the type of PtC_i:

- To keep a *v-v* PtC, the only possible motion is a **pure rotation**.
- To keep a *v-e/e-v* or *v-f/f-v* PtC, if α_A^i is a *v*, the possible motions are **pure rotation**, **slidingA**, or **combined slidingA and rotation** motions; else if α_B^i is a *v*, the possible motions are **pure rotation**, **slidingB**, or **combined slidingB and rotation** motions.
- To keep an *e-f/f-e* PtC, the possible motions are **pure rotation** (about the tangent line of the edge at the contact point), **slidingA**, **slidingB** combined **slidingA** and **slidingB**, and **combined sliding and rotation** motions. If α_A^i is an *e*, **slidingB** should be one dimensional along the edge, else if α_B^i is an *e*, **slidingA** should be one dimensional along the edge.
- To keep a *f-f* PtC, the possible motions are **slidingA**, **slidingB**, combined **slidingA** and **slidingB**, **pure rotation** about an axis normal to the tangent plane and **combined sliding and rotation** motions.
- To keep an *e-e-touch* PtC, the possible motions are one dimensional **slidingA**, **slidingB**, combined **slidingA** and **slidingB** along the edges, **pure rotation** about the tangent line and **combined sliding and rotation**.
- To keep an *e-e-cross* PtC, the possible motions are **slidingA**, **slidingB**, combined **slidingA** and **slidingB** and **pure rotation** about the contact normal.

If CF_i and CF_j involve a *face*, sometimes there can be an infinite number of possible neighboring transition motions. For example, let $CF_i = \{(f_A - f_B, 1)\}$, i.e., a CF with a single face-face PtC, and $CF_j = \{(f_A - e_B, 1)\}$ be a hypothesized LN CF, where e_B is an edge of face f_B. There can be an infinite number of possible paths of neighboring transition motions in the type of sliding A along f_B to reach (any point on) e_B. Our method in such a case is to enumerate possible paths until a feasible path is found.

In general, if a neighboring transition from state $<CF_i, C_i>$ to a hypothesized LN state $<CF_j, C_j>$ requires keeping some PtCs while changing or removing some other PtCs, our strategy is to try to realize a **keep** action of one PtC. If multiple PtCs need to be kept, we prefer to pick a PtC that does not involve a face or is not of *e-e-cross* type so that only a finite number of motions are possible to maintain it. Here, the preference is for speeding up the **keep** action when there are many options to do it. With the chosen PtC, we construct a possible compliant motion of A to maintain it (the possible types are listed above). If the motion is possible in all small (digital) steps without causing additional collisions

and is able to **change** or **remove** some other PtC(s) that the transition also require, the motion is considered feasible, and the hypothesized point-contact state is considered a valid LN state.

If no possible motion is feasible in any case, the hypothesized CF_j is discarded as not a valid LN CF of CF_i, and there is no corresponding LN point-contact state $<CF_j, C_j>$ of $<CF_i, C_i>$.

IV. Implementation

We have implemented the general algorithm as described in Section III for automatic generation of an LN graph from a seed point-contact state between two arbitrary, (strictly) curved objects A and B. The algorithm is implemented in Microsoft Visual C++ 6.0. We currently used the OPCODE collision detection library [21] for detecting collisions other than the desired point contacts in feasibility checking of neighboring transition motions. Note that since all we need to know here is whether a collision happens beyond the desired point contacts and is not how a collision happens, a mesh-based detection package serves the purpose well. Other mesh-based collision detection packages could be used too. On the other hand, it should be emphasized that our approach generate exact point contacts from contacting smooth surface features directly without mesh-based approximation.

Figure 5 and Figure 7 show two examples that our algorithm applied. In both examples, object B is the same, and its surface consists of three different smooth surface patches: the upper part is an elliptic paraboloid concave face, the middle one is a part of a convex sphere, the lower part is an elliptic paraboloid convex face, and the corresponding equations are:

$$\begin{aligned} &4x^2 + 4y^2 - 10.07047z = 15.75982, \\ &-1.56495 \le z \le 2.43505, \text{ and} \\ &x^2 + y^2 + z^2 = 16,\ 0 \le z \le 2.43505, \text{ and} \\ &x^2 + y^2 - 4z = 16,\ -4 \le z \le 0. \end{aligned}$$

In Figure 5, object A has an ellipsoid surface with the following equation:

$$225x^2 + 100y^2 + 36z^2 = 225$$

In Figure 7, object A has a surface consists of two elliptic paraboloid convex faces that meet on one edge, with the following equations:

$$8x^2 + 8y^2 - 9z = 18,\ -2 \le z \le 0, \text{ and}$$

$$8x^2 + 8y^2 + 9z = 18,\ 0 \le z \le 2$$

In order to have a clear observation, the objects are displayed with transparency, and edges are drawn in solid lines. e's and f's label the edges and faces of A and B. The seed point-contact states for the two examples are shown in Figure 5b and Figure 7b respectively, with the contact formations labeled. In Figure 5b, the seed point-contact state shows a

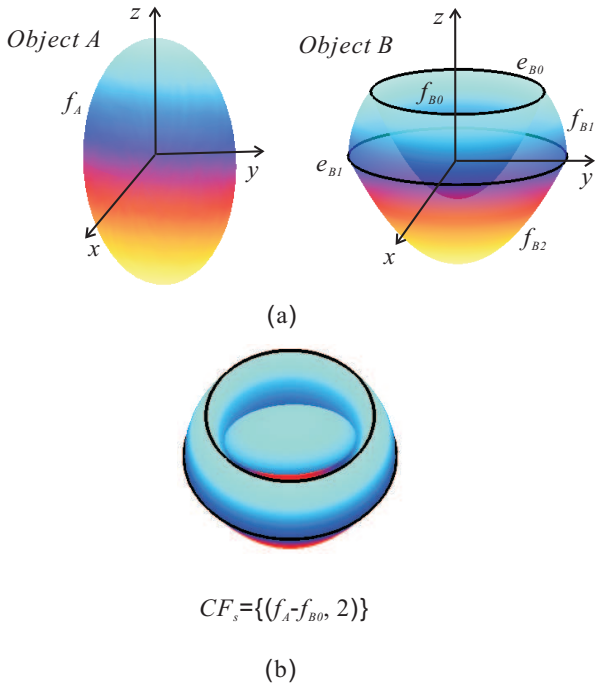

Fig. 5. Example 1

case where two point contacts are between the same pair of boundary elements.

For the example in Figure 5, our algorithm has generated, from the seed contact state CS_s, a LN graph of 6 valid states including the seed. Figure 6 displays the LN graph. The graph is in fact the complete point-contact state graph for this example. For the example in Figure 7, from CS_s, our algorithm has generated its LN graph consisting of 18 valid nodes automatically, which is also the complete point-contact state graph for that example. Figure 8 displays some valid point-contact states generated.

From both examples we can see that because a seed point-contact state is chosen to maximize the number of point contacts, and because there is one such state globally in each case, a complete point-contact state graph is generated from the seed. This is a very nice property of an LN graph[1]. If a complete point-contact state graph requires the merge of two or more LN graphs, the minimum number of necessary seed point-contact states (or LN graphs) is the number of local maxima states, i.e., states that have the most number of point contacts comparing to all neighboring states. A local maximum state usually involves concave elements in contact.

The program is executed on a Pentium 4, 2.8Ghz machine with 1024MB RAM. The running time for Example 1 (in Figure 5 and Figure 6) is 11.272 seconds. The running time Example 2 (in Figure 7 and Figure 8) is 27.364 seconds.

[1]From the same seed state, an LN graph is larger than a goal-contact relaxation (GCR) graph [24].

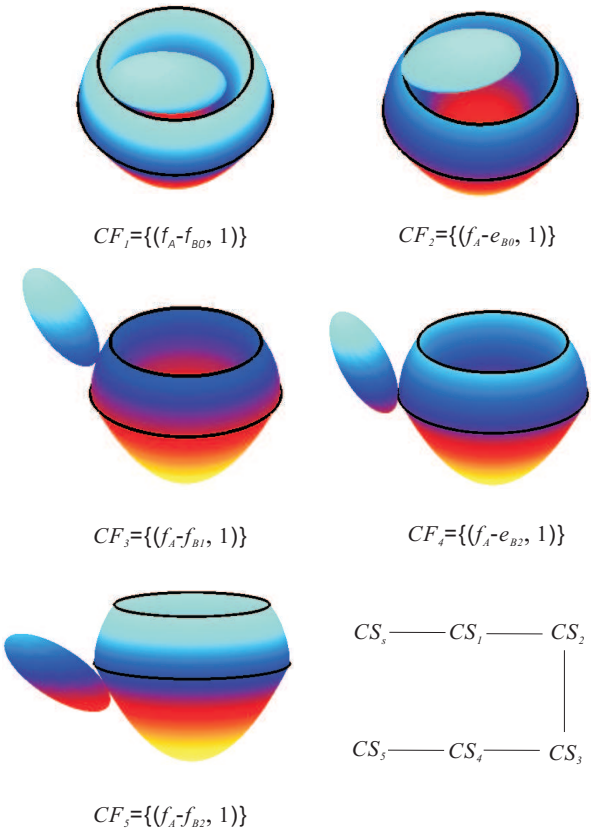

Fig. 6. Point-contact states generated in Example 1

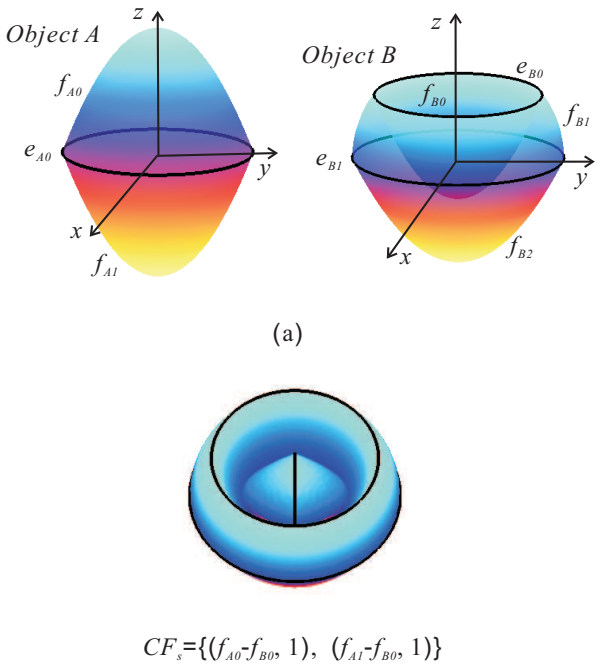

Fig. 7. Example 2

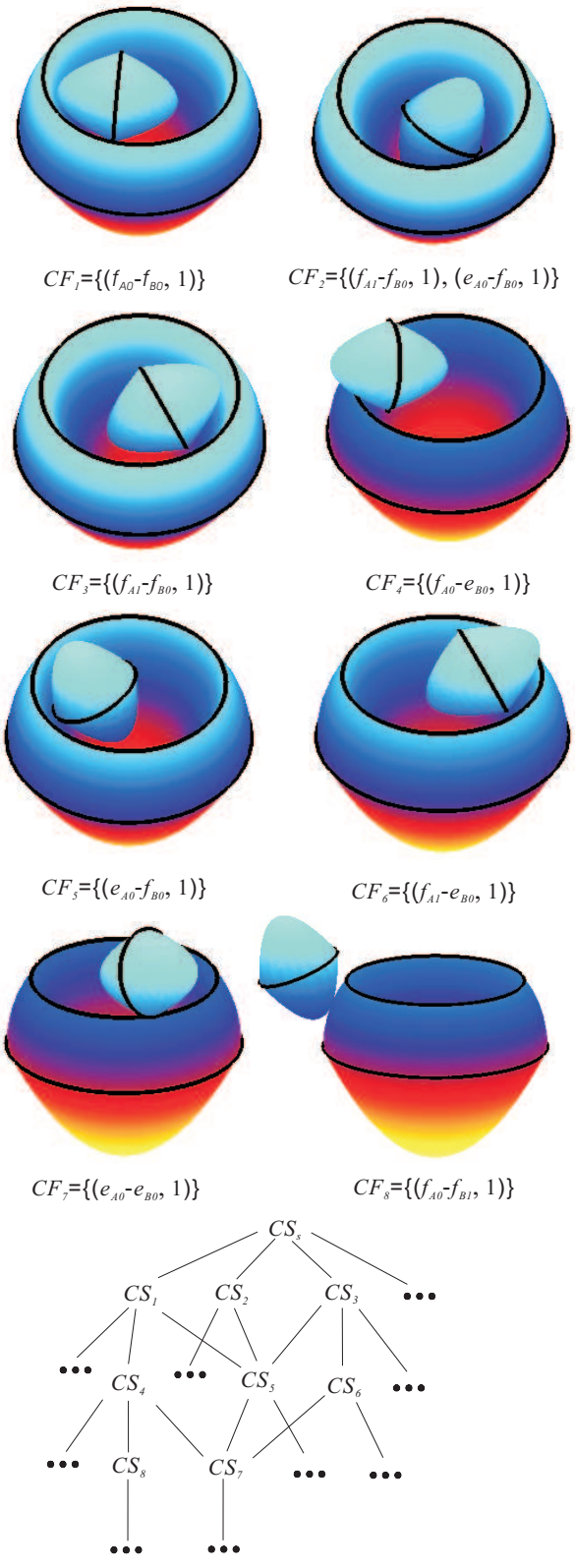

Fig. 8. Some point-contact states generated in Example 2

V. Discussion of Complexity

The complexity of the generation algorithm for a LN graph mainly depends on the total number of hypothesized nodes in the LN graph, the number of collision checks in constructing a neighboring transition motion, and the time for each collision detection query during the process of feasibility checking.

For two contacting 3-D curved objects A and B, we use N_A and N_B to represent the total number of boundary elements of A and B respectively, and we use p to indicate the maximum number of valid contact formations between a single pair of elements α_A and α_B (which could be single-PtC and multiple-PtC CFs). Note that p depends on the geometric characteristics of α_A and α_B related to convexity and concavity. An upper bound on the total number of hypothesized point-contact states with no more than three PtCs is:

$$pN_AN_B + p^2 \begin{pmatrix} 2 \\ N_AN_B \end{pmatrix} + p^3 \begin{pmatrix} 3 \\ N_AN_B \end{pmatrix}$$

which is of $O(N_A^3 N_B^3 p^3)$. In practice, the actual number of hypothesized point-contact states is much, much smaller. Note that for $m > 3$, an m-PtC CF is likely to contain redundancy in contact constraints because an object can be usually immobilized by three contact points. Therefore, an LN CF of a m-PtC CF ($m > 3$) is usually a two-PtC CF or a single-PtC CF.

Now, for a neighboring transition motion, if it takes on average k collision checks (among all step motions), and since the time for each collision detection query is T_{opcode} by OPCODE[2], its average time cost is kT_{opcode}.

VI. Conclusions

In this paper, we have provided a general representation of point-contact states between two 3-D curved objects and a systematic algorithm for automatic generation of such point-contact state space as point-contact state graphs. Automatic generation of point-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 point-contact states cannot be imagined or drawn easily. Unlike contact states between polyhedral objects, it is often much less obvious to human eyes whether a point-contact state is actually possible or not between two arbitrary 3-D curved objects.

We plan to continue this research by extending the approach to handle more general and complicated objects with more types of contact states, such as objects with both curved and flat surfaces.

[2]From [21], for all contacts, OPCODE provides $O(1)$ queries most of the time. In the worst case, it will not be over $O(nlogn)$, where n is the number of mesh triangles [2], [21].

References

[1] F. Avnaim, J. D. Boissonnat, B. Faverjon, "A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacl es," *Proc. IEEE Int. Conf. Robotics & Automation*, pp. 1656-1661, April 1988.

[2] G. van den Bergen, "Collision Detection in Interactive 3D Environments," *Morgan Kaufmann Publishers*, 2004.

[3] R. Brost, "Computing Metric and Topological Properties of Configuration- Space Obstacles", *Proc. IEEE Int. Conf. Robotics & Automation*, pp. 170-176, May 1989.

[4] J.F. Canny, "The Complexity of Robot Motion Planning," *MIT Press*, 1988.

[5] B. Donald, "On Motion Planning with Six Degrees of Freedoms: Solving the Intersection Problems in Configuration Space," *IEEE Int. Conf. Robotics & Automation*,pp. 536-541, 1985.

[6] X. Ji and J. Xiao, "Planning Motion Compliant to Complex Contact States," *International Journal of Robotics Research*, 20(6):446-465, June 2001.

[7] L. Joskowicz, R. H. Taylor, "Interference-Free Insertion of a Solid Body Into a Cavity: An Algorithm and a Medical Application," *Int. J. Robotics Res.*, 15(3):211-229, June 1996.

[8] T. Lefebvre, "Contact Modeling, Parameter Identification and Task Planning for Autonomous Compliant Motion using Elementary Contacts," Ph.D. Thesis, Katholieke Universiteit Leuven, Leuven, Belgium, May 2003.

[9] T. Lefebvre, J. Xiao, H. Bruyninckx, G. De Gersem, "Active Compliant Motion: A Survey," *Advanced Robotics*, 19(5):479-500, July 2005.

[10] T. Lozano-Pérez, "Spatial Planning: A Configuration Space Approach," *IEEE Trans. Comput.*, C-32(2):108-120, 1983.

[11] Q. Luo, E. Staffetti, and J. Xiao, "Representation of Contact States between Free-Form Objects," *IEEE Int. Conf. Robotics & Automation*, pp. 3589-3595, New Orleans, April 2004.

[12] Q. Luo and J. Xiao, "Physically Accurate Haptic Rendering with Dynamic Effects," *IEEE Computer Graphics and Applications, Special Issue - Touch-Enabled Interfaces*, Nov/Dec. 2004.

[13] B. J. McCarragher, "Task Primitives for the Discrete Event Modeling and Control of 6-DOF Assembly Tasks," *IEEE Trans. Robotics and Automation*, 12(2):280-289, April 1996.

[14] W. Meeussen, J. De Schutter, H. Bruyninckx, J. Xiao, and E. Staffetti, "Integration of Planning and Execution in Force Controlled Compliant Motion," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Alberta, Canada, August 2005.

[15] F. Pan and J.M. Schimmels, "Efficient Contact State Graph Generation for Assembly Applications," *IEEE Int. Conf. Robotics & Automation*, pp. 2591-2598, Taipei, Taiwan, September 2003.

[16] J. Rosell, L. Basañez, and R. Suárez, "Determining Compliant Motions for Planar Assembly Tasks in the Presence of Friction," *Proc. IEEE/RSJ Int. Conf. on Intell. Robots & Sys.*, pp. 946-951, 1997.

[17] D. Ruspini and O. Khatib, "Collision/Contact Models for Dynamic Simulation and Haptic Interaction," *Proc. ninth Int. Symp. Robotics Research*, pp. 185-194, Oct. 1999.

[18] E. Sacks, C. Bajaj, "Sliced Configuration Spaces for Curved Planar Bodies," *Int. J. Robotics Res.*, 17(6):639-651, June 1998.

[19] R. H. Sturges, and S. Laowattana, "Fine Motion Planning through Constraint Network Analysis," *IEEE Int. Conf. Assembly and Task Planning*, pp. 160-170, Pittsburgh, August 1995.

[20] P. Tang and J. Xiao, "Automatic Generation of Contact State Graphs based on Curvature Monotonic Segmentation," to appear in *IEEE Int. Conf. Robotics & Automation*, Orlando, May 2006.

[21] P. Terdiman, *http://www.codercorner.com/Opcode.htm.*.

[22] T.V. Thompson II and E. Cohen, "Direct Haptic Rendering of Complex Trimmed Nurbs Models," *In Proceedings of Symposium on Haptic Interfaces, ASME, 1999.*

[23] J. Xiao, "Automatic Determination of Topological Contacts in the Presence of Sensing Uncertainties," *IEEE Int. Conf. Robotics & Automation*, pp. 65-70, Atlanta, May 1993.

[24] J. Xiao and X. Ji, "On Automatic Generation of High-level Contact State Space," *International Journal of Robotics Research*, (and its first multi-media extension issue http://www.ijrr.org/), 20(7):584-606, July 2001.