Deformable Proximity Queries and their Application in Mobile Manipulation Planning

M. Gissler and C. Dornhege and B. Nebel and M. Teschner

Computer Science Department, University of Freiburg, Germany

Abstract. We describe a proximity query algorithm for the exact minimum distance computation between arbitrarily shaped objects. Special characteristics of the Gilbert-Johnson-Keerthi (GJK) algorithm are employed in various stages of the algorithm. In the first stage, they are used to search for sub-mesh pairs whose convex hulls do not intersect. In the case of an intersection, they guide a recursive decomposition. Finally, they are used to derive lower and upper distance bounds in non-intersecting cases. These bounds are utilized in a spatial subdivision scheme to achieve a twofold culling of the domain. The algorithm does not depend on spatial or temporal coherence and is, thus, specifically suited to be applied to deformable objects. Furthermore, we describe its embedding into the geometrical part of a mobile manipulation planning system. Experiments show its usability in dynamic scenarios with deformable objects as well as in complex manipulation planning scenarios.

1 Introduction

Proximity queries play an important role in robot motion planning, dynamic simulation, haptic rendering, computer gaming, molecular modeling and other applications [1]. A plethora of papers has been published on different aspects of these queries in computational geometry and other research areas. Furthermore, many systems and libraries have been developed for performing different proximity queries. However, the attention to deformable proximity queries has been of moderate extent when compared to the many techniques capitalizing the special properties of rigid bodies. Regarding motion planning in robotics, distance queries have been used to accelerate the verification of execution paths [2]. If a deformable environment or flexible robots have to be considered in the planning process, an efficient distance computation algorithm is needed in the geometrical part of the motion planner that is capable of handling such environments.

Our contribution: We describe an approach for the computation of the minimum distance between arbitrarily shaped deformable objects. Objects are represented by triangulated surface meshes. We show how to use GJK for an adaptive decomposition of the meshes. GJK is employed on the sub-mesh pairs to find lower and upper bounds for the minimum distance. The bounds are used for a spatial subdivision scheme that only takes a small part of the domain into account to determine the exact minimum distance between the sub-meshes. The algorithm does not depend on spatial or temporal coherence. Thus, it is suitable

to be applied to deformable objects. We show the usability of the algorithm in a planning system for mobile manipulation. The system is able to find execution plans for complex tasks that require the replacement of objects to reach a specific goal or to take the deformability of the manipulable objects into account.

Organization: The rest of the paper is organized as follows. Section 2 surveys related work on proximity queries and manipulation planning. The proximity query algorithm is described in section 3. The embedding of the proximity queries into the manipulation planning framework is discussed in section 4. The paper concludes by presenting results for the proximity query technique and its application to manipulation problems in section 5.

2 Related work

Proximity queries. Proximity query algorithms find their application in many research areas such as computer graphics, physically-based simulation, animation, interactive virtual environments and robotics. They include the query for collision detection, distance computation or penetration depth. Extensive research has produced a variety of specialized algorithms. They may differ in the model representations they are able to process, the type of query they can answer or the specific properties of the environment. Excellent surveys can be found in [3–5]. Considering collision detection, many approaches exploit the properties of convex sets to be able to formulate a linear programming problem. Gilbert et al. propose an iterative method to compute the minimum distance between two convex polytopes using Minkowski differences and a support mapping [6]. In contrast, Lin and Canny [7] execute a local search over the Voronoi regions of convex objects to descend to the closest point pair. In dynamic environments, geometric and time coherence can be exploited to employ feature-tracking to improve the efficiency of the algorithms even more [7].The described techniques can be applied to non-convex objects, if the non-convex objects are either composed of [6, 7] or decomposed into [8] convex subparts. The algorithms then work on the convex subparts as usual. However, surface decomposition is a nontrivial and time consuming task and can only be considered as a preprocessing step when applied to undeformable objects.

Apart from the family of feature-tracking algorithms, there is the class of bounding volume hierarchies. For each object, a hierarchy of bounding volumes is computed that encloses the primitives of the object at successive levels of detail. Different types of bounding volumes have been investigated, such as spheres [9, 10], axis-aligned bounding boxes [11], k-DOPs [12] or oriented bounding boxes [13]. Further, various hierarchy-updating methods have been proposed [14].

Spatial subdivision is the third family of acceleration techniques for proximity queries. The simplest but also most efficient subdivision would be the use of a regular grid. Only primitives within the same grid cell are then queried for collision. This approach is best suited for n-body collision queries, since it only takes linear time to query the collisions between the n^2 object pairs. Furthermore, it is

well-suited for the detection of collisions and self-collisions between deformable objects [15]. An approach that combines the benefits of feature-tracking algorithms and spatial subdivision algorithms is described in [16] and [17].

In the recent years, graphics hardware has been used to accelerate various geometric computations such as collision detection [18, 19] or distance field computation [20]. Possible drawbacks of GPU-based approaches are that their accuracy is limited by the frame buffer resolution and the time for reading back the frame buffers. However, in [19] the amount of read-back is reduced with the introduction of occlusion queries for collision detection.

Manipulation planning. Solving the robotic planning problems in high-dimensional configuration spaces is often addressed using probabilistic roadmap (PRM) planners [21, 22]. Such planners randomly generate samples in the configuration space and attempt to connect each newly generated sample to one of the existing samples by means of shortest paths in configuration space. This procedure results in a connectivity graph that spans the configuration space. The sampled nodes and the path segments stored in the graph have to be tested for collision. The validation of a collision-free graph takes up most of the computation time in the construction of the PRM. Schwarzer et al. presented an approach to integrate distance computation algorithms in the PRM framework for a more efficient dynamic collision checking [2]. The problem of computing a measure of distance between two configurations of a rigid articulated model has also been addressed by Zhang et al. [23]. Furthermore, manipulation planning is addressed by building a so-called "manipulation graph". It consists of nodes representing viable grasps and placements. Nodes are connected by transit or transfer paths moving either the manipulator alone or together with a grasped object. Those paths are solved using the above-mentioned PRM planners [24, 25]. Integrating symbolic and manipulation planning has been studied in the past. Cambon et al. [26, 27] use the FF planner which they modified to integrate roadmap planning into the planner. However, they do not provide a general interface to the domain-independent planner. Therefore, we base our implementation on the work of Dornhege et al. [28] that presents a general domain-independent planner interface to geometric planning. A comprehensive survey on robot planning algorithms can be found in [29].

3 Proximity queries

In this section, the proximity query algorithm is described. It returns the minimum distance between pairs of arbitrarily shaped objects in three-dimensional space. The objects may be given as closed non-convex triangulated surface meshes. The algorithm can be divided into three stages. The first stage employs a variation of the Gilbert-Johnson-Keerthi algorithm (GJK) [6]. It determines the separation distance between the convex hulls of a pair of non-convex objects. Obviously, the points that define the separation distance lie on the convex hulls of the objects, but not necessarily on their surfaces. Thus, we obtain a lower distance bound for the exact separation distance. Furthermore, in GJK

the closest points on the convex hull are expressed by a combination of points on the actual surfaces, the support points. Thus, the closest pair of support points gives an upper distance bound (see the left side of figure 1). If the lower

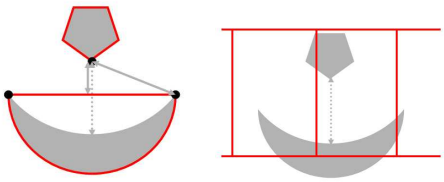

Fig. 1. Step one and two of the distance computation algorithm.
Left: Lower and upper distance bounds (gray lines) between the two shapes are derived from GJK. The lower bound is the minimum distance between the two convex hulls (red), the upper bound is the minimum distance between pairs of support points (black dots). The actual separation distance (dotted gray line) lies within these bounds.
Right: Twofold culling using spatial hashing: 1. Only the object parts inside the margins (horizontal red lines) are hashed. 2. Only primitives inside the same cell (red rectangle) are considered in the pair-wise primitive test.

distance bound is greater than zero, i.e. the convex hulls of the two objects do not overlap, the algorithm proceeds with stage two. It employs spatial hashing [15]. All surface triangles are hashed to the cells in the hash table. The cell size $\mathbf{c} = [x, y, z]^T$ of the hash grid is determined using the distance bounds found in the first stage, with dist_{upper} and dist_{lower} being the upper and lower bound, respectively. The grid is aligned to a local coordinate system, which has the z-axis parallel to the vector that connects the closest points on the convex hulls. We define the cell size along the different axes to be: $c_x = c_y := \sqrt{t_x^2 + t_y^2}$ and $c_z = 2 \cdot \text{dist}_{upper} - \text{dist}_{lower}$. Here, $\mathbf{t} = (t_x, t_y, t_z)^T$ is the vector that connects the support points for which $\|t\| = \text{dist}_{upper}$. Using this scheme, only triangles within the same cell and its neighbors can still contribute to the exact minimum distance. The distance for all other triangle pairs is guaranteed to be greater than the upper distance bound. They are efficiently culled away by the intrinsic properties of the subdivision scheme (see the right side of figure 1). If the convex hulls of the mesh pair overlap, the algorithm proceeds with stage three. In this stage, information computed by GJK is utilized to adaptively decompose the meshes into sub-meshes and pair-wise repeat the process in stage one recursively. In particular, GJK returns extremal points of the objects along a support vector. Planes orthogonal to this support vector and going through the extremal points divide the objects into sub-meshes (see figure 2). The overall minimum distance between the object pair is the minimum of the set of distances computed for all the sub-mesh pairs. In contrast to other approaches, the input data does not need to be pre-processed, i. e. no full surface decomposition is executed

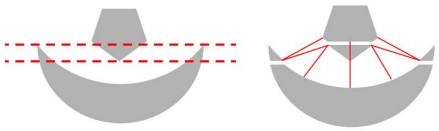

Fig. 2. Step three of the algorithm is invoked, if the convex hulls of the two shapes overlap.
Left: The objects are recursively divided into sub-meshes according to support planes (dashed red lines).
Right: The minimum of the set of separation distances (red lines) of the sub-mesh pairs gives the separation distance.

and no bounding volume hierarchies are constructed. Instead, decomposition of the surface meshes is only performed if it is required in the separation distance computation. This makes the approach suitable for the simulation of deformable objects.

4 System overview

In this section, we describe our framework for manipulation planning and the embedding of the proximity query algorithm into this framework. First of all, the manipulation problem is decomposed into a symbolic and a geometrical part. The symbolic planner allows for task specifications and domain descriptions to be given in high-level, human-like language, e. g. task specifications look like *on(box, table)* and domain descriptions like *pick-up(box)* or *put-down(box, table)*, respectively. On the symbolic level, the applicability of actions can be decided by evaluating the conditions of state variables. On the other hand, the geometric planner is used for constraint checking and effect calculation, i. e. the detection of collision free states and execution paths. Therefore, the geometric planner has access to a full domain description that represents the kinematics of the manipulator and a three-dimensional scene description. The decomposition serves to partition the complex manipulation problem into simpler planning problems. The interaction between the two parts is realized by external modules called semantic attachments [28]. They compute the valuations of state variables in the symbolic part by answering question like "Is there a collision-free way to move from point a to point b?" using the geometric part at run-time. Using the semantic attachments, the low-level geometric planner can provide information to the high-level symbolic planner *during* the planning process. However, it is only evoked when it seems relevant to the high-level planner. This is of particular importance, since the low-level planner performs the most time-consuming tasks, the proximity queries. The semantic attachments are implemented as probabilistic roadmaps (PRM) [22]. The roadmaps are connectivity graphs that provide

collision-free states and path segments in configuration space. The configuration space is given by the kinematics of the manipulator and the configurations of the objects.The verification of collision-free states and paths in workspace is performed using the proximity query algorithm described in section 3. Depending on the query, the exact distance is returned or a distance threshold is verified. Using distance queries instead of collision queries may seem to be slower in comparison, but only distance queries allow for a fast and safe path verification [2].

Incorporating the possibility to manipulate deformable objects extends the collision-free configuration space in the case of transfer paths - the paths, where the manipulator has grasped an object and moves it along. We require the manipulator to be collision-free. Only the grasped object may collide. If this collision can be resolved by deformation, the current state is verified to be collision-free. The deformation energy of an object is computed using a linear relation between the forces and the displacement of the tetrahedrons in the volume representation. If an object-specific threshold is passed, the state is not-collision-free. Besides the ability for deformable manipulable objects, navigation amongst deformable objects [30, 31] could also be realized with the tetrahedral data structure.

5 Experiments

We have staged a series of experiments to evaluate the distance computation approach as well as its application in the manipulation planning framework. In all experiments, the object representation is twofold. Surfaces are represented by triangular meshes to provide the input for the proximity query algorithm, whereas volumes are represented by tetrahedral meshes to provide the input for the force-displacement relation. All run-times were computed as average run-times on an Intel Core2Duo 6400 with 2 GB RAM using a single core. [1]

Proximity queries. The set of test scenarios for the evaluation of the distance computation approach includes (1) a pair of cows, (2) a pair of horses, (3) a stick and a dragon and (4) a pair of deforming teddies. The objects vary in shape and complexity. Scene complexity and timings are given in table 1. We compare

Table 1. Results for the test scenarios. The timings resemble the average distance computation time in milliseconds over 1000 consecutive frames.

Scenario	# of triangles	avg. [ms] our algorithm	avg. [ms] SWIFT
(4)	4400	67	1518
(3)	6000	90	1250
(1)	12000	680	1681
(2)	19800	762	2904

the timings with the ones gathered with the software package SWIFT++ [8].

[1] A video with five exemplary scenarios and plans can be found at: http://tinyurl.com/p5z82t.

SWIFT decomposes the surface of a non-convex object into convex pieces, which are stored in a bounding volume hierarchy (BVH). The query is then executed on the hierarchy of convex pieces. Using this data structures, distance queries can be answered very quickly. However, if the scene is considered to be unknown in each time step, surface decomposition and BVH generation has to be included in the total computation time. In comparison, our approach decomposes the objects into a tree of sub-meshes whose convex hulls do not overlap. This is more general when compared to a decomposition into convex pieces. However, it is also more adaptive with respect to the current scene configuration. Therefore, our algorithm achieves lower average computation times. Please note that SWIFT++ is optimized for the application in rigid body simulations. Therefore, the surface decomposition and the construction of the BVH can be executed as preprocessing steps. Thus, they are probably not optimized. Nevertheless, the timings indicate that the decomposition is less suitable for online computations in the context of deformable objects or for single-shot algorithms like the approach proposed here.

Manipulation planning. We demonstrate our manipulation planning framework on two synthetic test scenarios (see figures 3 and 4). The manipulator used in both scenarios consists of 2400 triangles and 2500 tetrahedrons. The first scenario consists of an additional three tables, with manipulable items placed on top. Triangles and tetrahedrons sum up to 2500 and 2600, respectively. In the second scenario, cubes are arranged to form a small narrow passage. The manipulator platform and a teddy are placed left and right of the passage. Triangles and tetrahedrons sum up to 3000 and 3500, respectively. The two scenarios demonstrate two different problems. In the tables scene, problems are formulated that place objects at the locations of other objects, forcing the planner to detect such situations and plan for them accordingly. The results shown in table 2 indicate that even multiple replacing of objects still results in reasonable runtimes.

Table 2. Results for the tables scene. The problem instances are separated in three classes: Simple pick-and-place tasks (Class I), problems that require the replacing of object to reach the goal configuration (Class II), and problems that require the replacing of multiple objects (Class III). Various tasks have been posed per class. Runtimes are given in seconds.

Class I	Runtime (s)
01	3.48 ± 1.23
02	6.08 ± 3.49
03	1.47 ± 0.12
04	3.77 ± 0.97
05	4.75 ± 2.36
06	5.27 ± 2.71
07	63.83 ± 7.67
08	5.66 ± 7.50
09	12.48 ± 14.74

Class II	Runtime (s)
01	24.32 ± 8.63
02	24.95 ± 9.25
03	91.87 ± 14.01
04	30.26 ± 9.74

Class III	Runtime (s)
01	37.33 ± 6.85
02	15.50 ± 2.52
03	78.55 ± 45.61

In the second scene, a problem is formulated that forces the geometric planner to take the deformability of the manipulable object into account. The execution path depicted in figure 4 required the deformation computation to be executed for 600 configurations. This adds an additional planning time of 100 ms per configuration on average.

6 Conclusion

We have presented an algorithm for deformable proximity queries. It employs GJK to recursively find sub-mesh pairs whose convex hulls do not overlap. For such pairs, the minimum distance can be efficiently computed using spatial hashing. The overall minimum distance is governed by the minimum distance between the sub-mesh pairs. We have illustrated the efficiency and suitability of the algorithm with regard to deformable objects. Furthermore, we have described and demonstrated the application of the algorithm in a manipulation planning framework. Currently, we are investigating how to improve the runtimes of the framework. An optimized recursive decomposition would speed up the proximity query algorithm. Integration of geometric heuristics in the symbolic planning process could significantly reduce the amount of calls to the geometric planner.

7 Acknowledgments

This work has been supported by the German Research Foundation (DFG) under contract number SFB/TR-8. We also thank the reviewers for their helpful comments.

References

1. Lin, M., Gottschalk, S.: Collision detection between geometric models: a survey. In: Proc. of IMA Conference on Mathematics of Surfaces. (1998) 37–56
2. Schwarzer, F., Saha, M., Latombe, J.: Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. IEEE Transactions on Robotics and Automation **21(3)** (2005) 338–353
3. Lin, M.C., Manocha, D.: 35. In: Handbook of Discrete and Computational Geometry. CRC Press (2004) 787 – 806
4. Ericson, C.: Real-Time Collision Detection. Morgan Kaufmann (The Morgan Kaufmann Series in Interactive 3-D Technology) (2004)
5. Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.P., Faure, F., Magnenat-Thalmann, N., Strasser, W., Volino, P.: Collision detection for deformable objects. Computer Graphics Forum **24** (2005) 61 – 81
6. Gilbert, E., Johnson, D., Keerthi, S.: A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE Transactions on Robotics and Automation **4** (1988) 193–203
7. Lin, M., Canny, J.: A fast algorithm for incremental distance calculation. In: IEEE Int. Conf. on Robotics and Automation. (1991) 1008–1014

8. Ehmann, S., Lin, M.: Accurate and fast proximity queries between polyhedra using surface decomposition. Computer Graphics Forum (Proc. of Eurographics'2001) **20** (2001) 500–510
9. Quinlan, S.: Efficient distance computation between non-convex objects. IEEE Int. Conf. on Robotics and Automation **4** (1994) 3324–3329
10. Hubbard, P.: Approximating polyhedra with spheres for time-critical collision detection. ACM Transactions on Graphics **15** (1996) 179–210
11. van den Bergen, G.: Efficient collision detection of complex deformable models using AABB trees. J. Graphics Tools **2** (1997) 1–13
12. Klosowski, J., Held, M., Mitchell, J., Sowizral, H., Zikan, K.: Efficient collision detection using bounding volume hierarchies of k-DOPs. IEEE Transactions on Visualization and Computer Graphics **4** (1998) 21–36
13. Gottschalk, S., Lin, M., Manocha, D.: OBB-Tree: a hierarchical structure for rapid interference detection. In: SIGGRAPH '96: Proc. of the 23rd annual conference on Computer graphics and interactive techniques, ACM Press (1996) 171–180
14. Larsson, T., Akenine-Moeller, T.: Collision detection for continuously deforming bodies. In: Eurographics. (2001) 325 – 333
15. Teschner., M., Heidelberger, B., Mueller, M., Pomeranets, D., Gross, M.: Optimized spatial hashing for collision detection of deformable objects. In: Proc. Vision, Modeling, Visualization VMV'03, Munich, Germany. (2003) 47 – 54
16. Gissler, M., Frese, U., Teschner, M.: Exact distance computation for deformable objects. In: Proc. Computer Animation and Social Agents CASA'08. (2008) 47–54
17. Gissler, M., Teschner, M.: Adaptive surface decomposition for the distance computation of arbitrarily shaped objects. In: Proc. Vision, Modeling, Visualization VMV'08. (2008) 139–148
18. Knott, D., Pai, D.: CInDeR: Collision and interference detection in real-time using graphics hardware. In: Proc. of Graphics Interface. (2003) 73–80
19. Govindaraju, N., Redon, S., Lin, M., Manocha, D.: CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In: HWWS '03: Proc. of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware. (2003) 25–32
20. Sud, A., Govindaraju, N., Gayle, R., Kabul, I., Manocha, D.: Fast proximity computation among deformable models using discrete Voronoi diagrams. ACM Trans. Graph. **25** (2006) 1144–1153
21. Latombe, J.: Robot Motion Planning. Kluwer Academic Publishers (1991)
22. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation **12(4)** (1996) 566–580
23. Zhang, L., Kim, Y.J., Manocha, D.: C-dist: efficient distance computation for rigid and articulated models in configuration space. In: SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling, ACM Press (2007) 159–169
24. Alami, R., Laumond, J.P., Siméon, T.: Two manipulation planning algorithms. In: WAFR: Proceedings of the workshop on Algorithmic foundations of robotics, A. K. Peters, Ltd. (1995) 109–125
25. Simeon, T., Cortes, J., Laumond, J., Sahbani, A.: Manipulation planning with probabilistic roadmaps. The International Journal of Robotics Research **23** (2004) 729–746
26. Cambon, S., Gravot, F., Alami, R.: A robot task planner that merges symbolic and geometric reasoning. In: Proc. of ECAI. (2004) 895–899
27. Fabien Gravot, S.C., Alami, R.: aSyMov: A planner that deals with intricate symbolic and geometric problems. Springer Tracts in Advanced Robotics **15** (2005)

28. Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., Nebel, B.: Semantic attachments for domain-independent planning systems. In: Proc. of ICAPS. (2009) to appear.
29. LaValle, S.M.: Planning algorithms. Cambridge University Press (2006)
30. Rodriguez, S., Lien, J.M., Amato, N.: Planning motion in completely deformable environments. In: Proc. IEEE Int. Conf. on Robotics and Automation (ICRA). (2006) 2466–2471
31. Frank, B., Stachniss, C., Schmedding, R., Burgard, W., Teschner, M.: Real-world robot navigation amongst deformable obstacles. In: Proc. IEEE Int. Conf. on Robotics and Automation (ICRA). (2009) 1649–1654

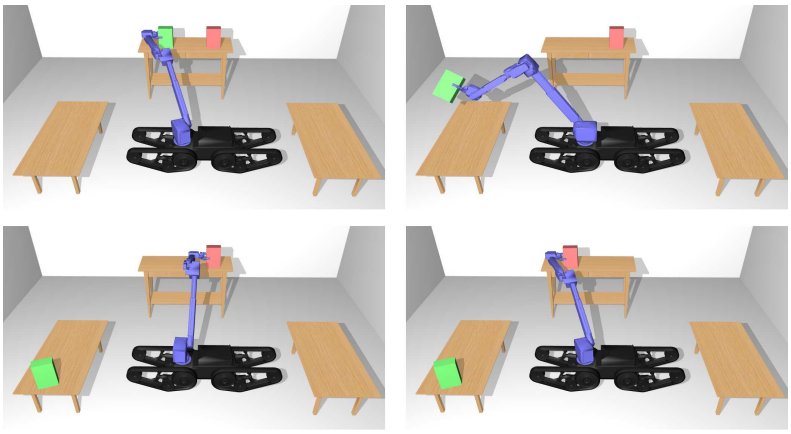

Fig. 3. An advanced pick-and-place task. The manipulator is required to pick up the red box and place it to where the green box is located. Therefore, it first has to pick up the green box and place it somewhere else (upper row) and then pick up and move the red box to the final position (lower row).

Fig. 4. A pick-and-place task applied to a deformable movable object. The teddy is picked up from behind the wall and moved trough the small narrow passage (left) to its final position above the table (right). An execution path can only be found, if deformability of the teddy is considered by the geometric planner.