Probabilistic Roadmap Motion Planning for Deformable Objects*

O. Burchan Bayazit Jyh-Ming Lien Nancy M. Amato
Department of Computer Science
Texas A&M University
{burchanb,neilien,amato}@cs.tamu.edu

Abstract

In this paper, we investigate methods for motion planning for deformable robots. Our framework is based on a probabilistic roadmap planner. As with traditional motion planning, the planner's goal is to find a valid path for the robot. Unlike typical motion planning, the robot is allowed to change its shape (deform) to avoid collisions as it moves along the path. We propose a two-stage approach. First an 'approximate' path which might contain collisions is found. Next, we attempt to correct any collisions on this path by deforming the robot. We propose and analyze two methods for performing the deformations. Both techniques are inspired by physically correct behavior, but are more efficient than completely physically correct methods. Our approach can be applied in several domains, including flexible robots, computer modelling and animation, and biological simulations.

1 Introduction

Robot automation and motion planning have been inseparable since the very first robot. It is common practice to represent a robot as a set of rigid objects connected by joints. Although this representation covers many common situations, such as most industrial robots, there are many cases where a more flexible representation of the robot would be desirable.

Any problem that requires a robot to change its surface or shape is almost impossible for most existing planners. Yet such problems arise naturally in many common situations. Moving a long rod through corridors may require the rod to bend when passing around corners, or a robot pulling flexible sack may require the sack to deform when passing through narrow regions. There are many situations in which the other objects in the environment are most appropriately modeled as deformable objects. For example, in a surgical simulation, the environment contains rigid bones, somewhat deformable cartilage, and highly deformable tissue.

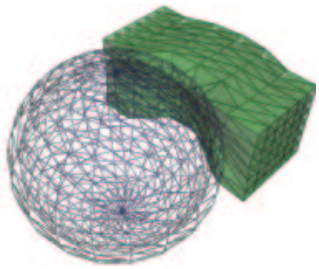


Figure 1: A brick deformed by a spherical obstacle.

Deformation, however, is not restricted to robotics. It is a commonly used operation in computer modeling and animation. Nevertheless, even for an expert, generating deformation animations, especially for complicated models, is always a very difficult and time-consuming task. Moreover, natural-looking deformed models usually require significant additional time for the animator to manually adjust the control points and parameters until the deformation 'looks right.'

Representing the deformable object with a very large number of dof (in the limit, one for each deformable point on its surface) could work in principle for most problems. However, this approach has the obvious disadvantage of increasing the dof to intractable levels and it still does not provide for a convenient way of enforcing physically correct behavior. Thus, finding an acceptable deformation within reasonable time limits is the objective of most algorithms. The acceptability criteria can be defined by constraints on the topology or physical properties which define the realism of the deformation. Unfortunately, physically-correct deformation and speed are usually mutually exclusive. Thus, a trade–off between the time required and the degree of realism required must be made. While a physically correct deformation is desirable, it might require several hours to produce. On the other hand, deformations which do not respect physical properties might be quickly computable, but produce unrealistic

*This research supported in part by NSF CAREER Award CCR-9624315, NSF Grants IIS-9619850, ACI-9872126, EIA-9975018, EIA-0103742, EIA-9805823, ACI-0113971, CCR-0113974, EIA-9810937, EIA-0079874, and by the Texas Higher Education Coordinating Board grant ARP-036327-017. Bayazit is supported in part by the Turkish Ministry of Education.

results.

Our goal is to find an acceptable deformation within a reasonable time limit (real-time in some cases). Our approach is based on the *probabilistic roadmap (*PRM*)* methods [1, 11, 12]. We concentrate on problems where the robot is deformable and the other objects are rigid. However, our method can easily be generalized to problems where everything is deformable. We first find a path in which the robot might collide with the obstacles. Such paths can be found by using a reduced-scale version of the robot during initial motion planning or by allowing the robot to penetrate an obstacle by some (fixed) amount. As a result, the initial path may contain collisions for the rigid version of the robot. After finding such a path, we identify the colliding portions and deform the robot in these regions to avoid collisions. We currently employ two different methods for producing the deformations. One is a hierarchical approach which first deforms the robot's bounding box, and then the robot itself, and the second is a more direct approach which directly deforms the robot's geometry (but not topology).

The paper is outlined as follows. We first discuss related work (Section 2). Section 3 gives an overview of our approach. Section 4 describes how we find 'approximate' paths for a deformable object, and we discuss our deformation methods in Section 6. Our experimental results are presented in Section 7 and Section 8 concludes our paper.

2 Related work

Deformable modeling and manipulation are essential in computer graphics. Deformable models are often used to design complicated objects. As in [9], these deformation methods can be classified into non-physical methods and those based on mechanic properties. Deformations which do not address physical properties include functional deformations (scale, bend, twist) [3] and free-form deformation [17]. Mass-spring and finite element methods are the most commonly used strategies for building deformable objects with physical properties. Deformation methods without constraints are very useful in interactive shape editing since they are fast and deform objects smoothly. However, physically–based deformations, which respond to applied external forces and constraints, are often required for simulation and animation, such as deformable tissue [7, 8], blood cells, creatures [14, 19], sand, mud, and snow [20], and cloth [10].

Although work on the motion planning problem has produced many practical planners for rigid robots, not as much work has been done in motion planning for deformable objects. The f-PRM framework [2, 13] has been proposed for planning for models using physically correct deformations. Due to expensive operations for solving mechanical models and generating collision detection data structures, their planner is not suitable for real-time use and has so far only been applied to simple objects, such as a sheet of metal or a pipe-like robot. In [6], deformed distance fields were used to control the amount of penetration between non-penetrating flexible bodies.

3 Overview

As mentioned in Section 1, our approach is based on the probabilistic roadmap (PRM) approach to motion planning [12]. Briefly, PRMs work by sampling points 'randomly' in C-space and retaining those that satisfy certain feasibility requirements (e.g., they must correspond to collision-free configurations of the movable object). Then, these points are connected to form a graph, or roadmap, using some simple planning method for connecting 'nearby' points. During query processing, paths connecting the start and goal configurations are extracted from the roadmap using standard graph search techniques.

Our approach is similar to a traditional PRM. In our case we modify the feasibility requirements used during the roadmap construction stage so that our roadmaps may contain paths having some collisions. Later, during query processing, if a path containing collisions is extracted from the roadmap, we attempt to deform the robot to avoid those collisions. An example of such a deformation is shown in Figure 1, where the brick robot is deformed to avoid a collision with a spherical obstacle. If we do not succeed, we remove the corresponding edge from the roadmap and search for a new path.

Motion Planning for Deformable Objects

```
1. Build a feasible roadmap
2. while (a valid path is not found)
3.     find a feasible path
4.     foreach path configuration
5.         if (there is collision)
6.             deform robot
7.             if (not deformable)
8.                 remove path segment from roadmap
9. endwhile
```

The way we have implemented this strategy is to use a rigid robot during roadmap construction, and a soft (deformable) robot during query processing. The key to our approach is to define appropriate feasibility metrics which enable one to construct useful roadmaps with rigid robots. The two feasibility tests we have studied thus far are:

1. A reduced-scale version of the rigid robot (in some nominal shape configuration) is not in collision in the sampled configuration.

2. An original-scale version of the rigid robot (in some nominal shape configuration) penetrates an obstacle only within some acceptable limit.

Clearly, both tests can accept colliding configurations. During roadmap generation, we use heuristics to assign edge weights to denote the difficulty of deforming that edge, and during query processing a minimum weight path is extracted.

This philosophy is similar to Fuzzy PRM [16], Lazy PRM [5], or Customizeble PRM [18] where the roadmap nodes and/or edges are not validated, or are only partially validated, during roadmap construction and are validated completely only at query time. These methods have been shown to greatly decrease the roadmap construction costs, while only slightly increasing the query costs. Moreover, as noted in [18], there are actually some other benefits to this approach, such as enabling the same roadmap to be used for different robots, or for queries with different requirements. Our approach is also related to a method we have successfully employed when incorporating haptically collected user-input with a PRM planner [4]. In this work, the user collects an 'approximate' path, which may contain collisions, and the planner tries to 'push' the colliding path segments to nearby free space. So, the planner deformed the path but not the robot, which is the opposite of the approach taken here.

In the following sections we describe our algorithm in more detail. Our deformation methods are described in Section 6.

4 Roadmap Construction

As discussed in Section 3, our strategy is to build roadmaps which may include some colliding configurations. To increase the probability that we can in fact deform the colliding configurations to free configurations during the query phase, we place some feasibility requirements on the acceptable colliding configurations. We also attempt to weight our roadmap edges to denote the expected difficulty (cost and feasibility) of deforming that edge if necessary. That is, the weights are selected to denote the expected *deformation energy* required to deform the edge. Since this energy is not known *a priori*, we use heuristic methods to assign weights that are related to the parameters of the feasibility metrics used to define acceptable configurations.

In this section, we describe in detail the two strategies mentioned above for constructing the roadmap. In the first, roadmap nodes are accepted if the configuration corresponding to a reduced-scale version of the rigid robot is collision free, and in the second, the configuration is accepted if the amount of penetration by the original robot is less than some acceptable bound. In both cases our goal is to build a roadmap containing paths that are either already valid, or that can be made valid by deforming the robot.

4.1 Shrinkable Robots

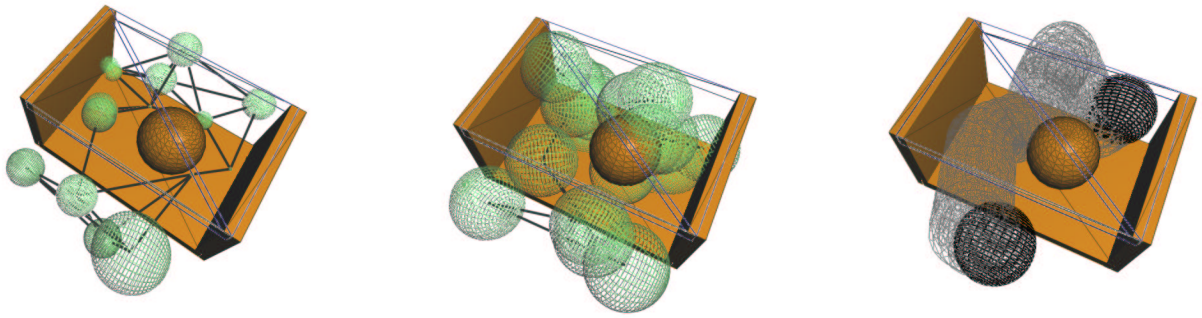
We would like to weight roadmap edges with some estimation of deformation energy. One heuristic estimate of this energy is the volume of the robot/obstacle intersection. While such intersection volumes are difficult to compute, the 'shrink' factor required to obtain a free robot in a particular configuration does offer some indication of the deformation energy and is at least loosely correlated with the intersection volume. Moreover, these shrink factors can be computed relatively inexpensively if we have pre-computed a set of scaled models, which can be viewed as a "uniformly deformed robot." The scaled robot models are constructed in a straightforward manner by scaling all vertices defining the robot model, and maintaining the robot topology. These scaled rigid models are used for collision detection in node generation and connection.

Node Generation and Connection. During node generation, for each node, we begin with the largest robot and progressively reduce its scale, while maintaining the configuration, until a collision free robot size is found. After the roadmap nodes have been generated, the roadmap is connected as with a traditional PRM. In particular, any of the normal local planners or connection methods could be employed [1]. An edge weight is set as the sum of the shrink factors of the two scaled models that define the (potential) edge's endpoints. (A larger (smaller) robot model has a smaller (larger) shrink factor.)

Figure 2(a) shows a roadmap generated with spherical robot. The variable sized roadmap nodes correspond to the shrink factors used to generate the roadmap nodes. One clearly sees the smaller nodes surrounding the obstacles, which should be avoided by the deformable object if possible.

4.2 Penetration

As discussed previously, another estimate of deformation energy is penetration depth. The deeper the robot penetrates inside an obstacle the harder it will be to deform the robot into a collision free shape. Unfortunately, penetration is hard to measure in most cases and is not provided by collision detection libraries (except for special cases such as convex objects



(a) (b) (c)

Figure 2: (a) A roadmap generated with shrinkable robots; roadmap configurations are shown in their accepted shruk size with wired surfaces. (b) A roadmap generated with penetrating robots. (c) The swept volume of a path found.

[15]).

Here, we propose using C-space penetration instead of workspace penetration. While C-space penetration is not easier to compute, we use a probabilistic approach to compute an estimate. Let c_0 be our initial configuration of interest. We generate n random C-space vectors which originate at c_0, and have random directions and lengths in some predetermined range (our allowable penetration depth). The choice of n affects the accuracy of the estimate. As n increases, the probability of finding a value close to the real penetration and the computation time both increase. In our experiments, the value of n is 20.Note that the size of a C-space vector depends on the distance metric selected and can be defined as $|c_i - c_0|$ where c_i is a configuration representing the ith vector. Adding these vectors to c_0 returns configurations within the allowable penetration distance. If any of them is collision free, then we accept configuration c_0. The minimum penetration depth found can be used to provide edge weights in the roadmap.

Node Generation and Connection. In this method, collision detection is replaced by a test for allowable penetration, i.e., if the penetration is small enough, a configuration which is in collision is accepted into the roadmap. The advantage of this approach is that we can use a standard PRM, including all node generation and connection methods. For example, the intermediate configurations that a local planner tests during the connection phase can be accepted based on their penetration. See Figure 2(b).

5 Query

While the roadmap was constructed using rigid bodies, a deformable version of the robot is used in the query phase. Since roadmap nodes and/or edges may be in collision, the query process must check them for collision and then, if collisions are found, call on some deformation method to try to deform the offending configurations into collision-free configurations. The query process below applies to roadmaps generated by either of the methods described above.

PATH QUERY

1. Test if robot can be deformed in start and goal.
2. Connect start and goal to roadmap (same Connected Component, CC).
3. Connected=false;
4. `while`(!Connected)
5. Find a path connecting start and goal.
6. Sort edges from largest weight to lowest.
7. `for` each edge
9. `if`(test failed) Delete this edge and break;
10. `endif`
11. `endfor`
12. Connected=true;
13. `endwhile`

In the query, the deformation test essentially replaces the collision detection test for traditional motion planning. Now, a configuration will be accepted if its deformation energy is less than some threshold, that can be user-defined to give the user some control over the deformations.

Because deformation is an expensive operation, we need a quick way to determine if a path edge cannot be deformed. To do this, we sort path edges according to their weights (which are estimates of deformation cost), and test the edges with highest weights first as they are the most likely to fail. If a test fails, the edge is deleted from the roadmap and the process repeats.

Figure 2(c) shows a path found by a query. Although the shortest path is on the right side of ball, our planner selected a longer path which had a smaller deformation energy.

6 Deformation

To construct huge numbers of deformable models on–line (at query time), we should have fast deformation methods. However, it is also important to consider the physical properties of the model. Although we consider physical properties, our approach is not physically complete, since our goal is to produce realistic *looking* deformations fast. For example, we do not attempt to precisely compute energy, as in [2].

Of the two objects participating in a deformation, one is the *deformable object* and the other is called the *deformer*. The deformer pushes (a portion of) the deformable object towards a collision-free state, and deformable object then changes shape according to these external forces. We implemented two deformation methods: bounding box deformation and geometric deformation.

6.1 Bounding-Box Deformation

The *bounding box deformation* deforms the given model hierarchically. First, the bounding box of the model is deformed, and then the model itself is deformed according to the deformation computed for the bounding box.

Our approach combines strategies of the ChainMail Deformation [7] and Free-Form Deformation (FFD) [17] to create a real-time deformation with reasonable physical properties. First, a bounding box of the model is deformed by ChainMail deformation, and then this model is deformed by FFD according shape of its bounding box. Figure 3 illustrate this process.

The ChainMail deformation was created to model tissue in surgical simulations [7]. Briefly, for a 3D ChainMail Box, each vertex is connected to (at most) six neighboring vertices, and the maximum and minimum distances to these neighbors are set according to the material property (e.g., a rigid body will have maximum=minimum). The following shows the basic operations of the ChainMail deformation. In line 1, vertices are moved by user or deformers in our case.

ChainMail deformation
1. Put moved vertices in queue, Q.
2. while(Q is not empty)
3. vertex v = dequeue(Q)
4. for(all neighboring vertices, n, of v)
5. if(n violate constraints)
6. resolve constraints and enqueue(Q, n)

Free Form Deformation, or FFD [17], is a well known interactive modeling tool which uses affine transformations to map local coordinates to deformed global coordinates. The third step in Figure 3 shows a model deformed when control points on the FFD lattice are moved. While FFD is extremely fast, it is generally difficult to obtain physically-based deformations since it requires tweaking many control points.

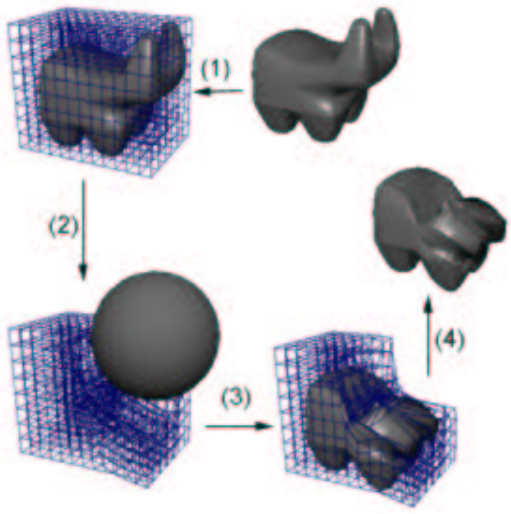


Figure 3: BBX deformation with 13*9*7 lattices. (1) Build voxel bounding box for a given model. (2) Bounding box deformed by ChainMail deformation. (3) FFD uses deformed bounding box to deform internal model. (4) Deformed model.

There are several benefits of this hybrid, hierarchical bounding box deformation. First, ChainMail provides physical properties, such as preservation of inter-vertex distances and elastic properties, and later FFD smoothly deforms the real model. One of the best aspects is that the motion planner can use only ChainMail, since it only requires the deformation energy, and utilize FFD during rendering. Thus, the motion planning cost is independent of the complexity of the model since only the ChainMail bounding box is deformed during planning. The display cost, which uses FFD, will still be dependent on the model complexity.

After the object is deformed, the deformation energy can be calculated as the summation of the displacement of each vertex from its initial position to its deformed position.

The **deformer** pushes the deformable object to a collision free condition. Since the deformable object is converted to a bounding box, the implementation of the deformer is quite simple. The center of the bounding box is the guide for pushing all points. First, we check if the center of the bounding box is inside or outside the deformer (it is always outside the obstacles if shrinkable robots are used, but may be internal for the penetration method). For each point on the surface of the bounding box, we shoot a ray to the center point. If the ray intersects the deformer, the point is pushed along the ray until no intersection exists.

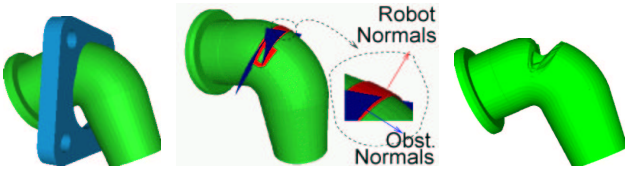
6.2 Geometric deformation

In this approach, we use the obstacle as the deformer, and directly deform the colliding portion of the robot. If we knew the colliding volumes of the robot and obstacle, it would be relatively easy to move the colliding part of the robot out of collision. Unfortunately, for the complex models we consider, collision detection routines only return the intersecting polygons of the models.

Our algorithm continuously moves the colliding polygons of the robot until they are outside the obstacle. The most challenging part of this algorithm is to decide which direction to move intersecting polygons. An intuitive approach would be to move the robot polygons in the direction of the obstacle polygons' outward normals. Unfortunately, there are some important cases where this approach does not produce the desired result. For example, if most of the robot is on the 'other' side of the obstacle from the colliding polygons, we would prefer to deform the robot in the opposite direction. To address this, we employ a sampling strategy to identify the correct direction. If we select directions randomly, we expect that directions toward the larger external portions of the robot should become collision free faster than other directions. To improve efficiency, we restrict our sampling to the following directions: (i) normals of colliding polygons on the robot, (ii) normals of the colliding polygons (or average of several colliding neighbors) on the obstacle, (iii) some uniformly selected directions, and (iv) the inverse of the directions in (i-iii).

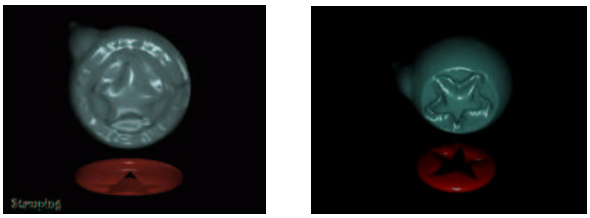
Figure 4 shows an example. The robot is the elbow shaped object and the obstacle is the hole shown in a colliding configuration in Figure 4(a). Some colliding polygons are shown in 4(b). After we chose a correct direction (from the types i-iv mentioned above), we move the robot's colliding polygons in that direction until we reach the collision free shape shown in 4(c).

If the robot does not fit in a narrow passage, then this method will fail to find a pushing direction unless we use a reduced-scale version of the robot. When the smaller robot is free, the colliding polygons of the original robot are moved towards the corresponding polygons on the smaller robot. Another problem with this method is that it can be distracted by remote collisions that occur when deforming some colliding region of the robot. To avoid this, we define a neighborhood function which eliminates the influence of the remote collisions. Finally, this method can produce unrealistic looking deformations because the resulting polygons may be quite large. In some cases we may even have topological changes. To address such issues, we added a constraint which states that none of the deformed edges can be shorter or longer than some percentage of the original edge. After the deformation a smoothing operation is applied to enforce this constraint. However, after smoothing we may end up with a shape that is in collision. If this occurs, we simply continue deforming, and then smoothing, until a valid deformation or the maximum number of iterations is reached.



(a) (b) (c)

Figure 4: (a) Colliding configuration where robot is elbow and obstacle is hole, (b) intersecting polygons of robot and obstacle and normals to try, (c) deformed version.



(a) (b)

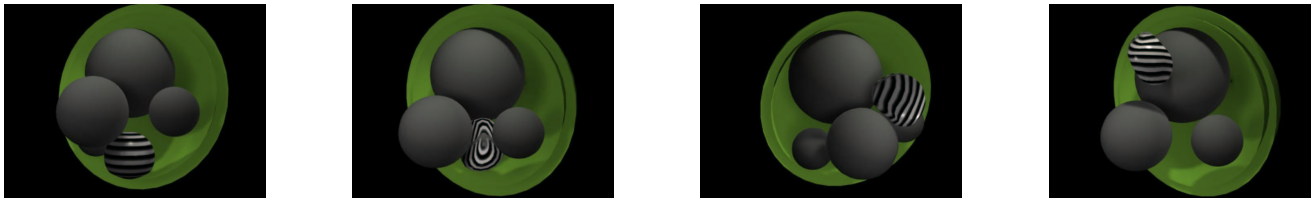
Figure 5: Snapshots of the stamping environment; (a) box deformation, (b) geometric deformation.

7 Experiments

In this section we report on experiments designed to study the shrinkable robot and penetration techniques for roadmap construction (Section 4) and the bounding box (BBox) and geometric deformation techniques (Section 6). All experiments were implemented on a PC with a Athlon 1.33 Mhz CPU and 256MB RAM.

Our experiments investigate the performance of our algorithm in three situations requiring diverse deformations. In the **sliding** experiment, the letters "DSMFT" simulate multiple robots following a path passing very close to an obstacle; the path is found for one letter and then applied to the others. In the **narrow passage** experiment (Figure 6), a deformable sphere (the robot) passes through a narrow passage bounded by four different-sized rigid spheres (obstacles) in a bounding sphere (front part not shown). The **stamping** experiment (Figure 5) enables us to see the effect of pushing the teapot (robot) against a complex seal (the obstacle).

Our first set of experiments studies the BBox and geometric deformation techniques. As reported in Table 1, both deformation techniques were applied to



(a) Bounding Box Deformation

(b) Geometric Deformation

Figure 6: Snapshots of a path in the narrow environment; (a) box deformation, (b) geometric deformation.

paths extracted from roadmaps constructed using the shrinkable robot and the penetration methods. Snapshots of the deformed robots in the stamping and narrow enviroments are shown in Figures 5 and 6, respectively. Snaphsots for the sliding environment and movies for all three environments can be found at http://www.cs.tamu.edu/faculty/amato/dsmft/. Our results show the BBox deformation is several hundred times faster than the geometric deformation. This is due to the high cost of the search for the pushing directions and because collision detection costs are higher than in the BBox deformation where only the bounding box is tested for collision (as opposed to the entire model in the geometric deformation). The BBox method also produces smoother deformations, which is as expected because ChainMail directly propagates a deformation towards the neighboring points, whereas the geometric deformation propagates deformations recursively (i.e., indirectly); this difference in the methods can be mitigated by a good neighborhood function, perhaps determined experimentally for each environment. We note that the average cost of the deformations varies with the complexity of the environment and the deformations; in the BBox deformation, this is also affected by the number of lattice points in the bounding box.

Our second set of experiments studies the effects of the parameters for the roadmap generation techniques in the narrow passage environment (the spheres). For the shrinkable robot method, roadmaps were constructed using two and six scaled robots (models). For the penetration method, we consider two penetration depths, $15r$ (small) and $25r$ (large), where r is the environmental resolution set for collision detection. As previously discussed, these parameters should approximate the deformation energy. The results in Table 2 show that both the shrink factor and the penetration value are good indications of deformation energy. As expected, if a larger number of reduced-scale robots is used, or a smaller penetration is allowed, the resulting roadmap will be more expensive to compute, but will contain paths that are easier to deform. We also note that most (99% to 100%) of the query time is spent on deformation processing. (The geometric deformation did not find a solution within acceptable time limits for the roadmap generated by the two scaled robots.)

Although the main disadvantage of the geometric deformation is speed, it was also observed to sometimes alter the topology of the model. This could be solved by including self-collision checks as the object is deformed. Nevertheless, this method is suitable for some types of situations, e.g., narrow passage problems where the robot must assume the shape of the surrounding environment.

8 Conclusion and Future Work

This paper considers motion planning for deformable objects. We have suggested two different methods for constructing and querying roadmaps, and have presented two deformation techniques that can be applied to the resulting path. This approach decouples motion planning and deformation, and enables hybrid methods utilizing multiple methods in a plug and play fashion. Our future work includes optimizing the geometric deformation method and applying our algorithm to particle systems where several robots deform independently.

Roadmap and Deformation Statistics

Environment	Roadmap Generation				Average Deformatation Times(s)	
	Shrunk Robot		Penetration			
	#CC	Total Time (s) (gen.+con.)	#CC	Total Time(s) (gen.+con.)	Bounding Box	Geometric
Sliding	65	5.4(2.7+2.7)	16	16.5(8.4+8.1)	0.04	13.3
Narrow	87	16.2(0.5+15.7)	35	116.9(25.9+91)	0.40	86
Stamping	9	5.2(0.1+5.1)	1	124.2(50.54+73.77)	0.86	536

Table 1: Roadmap and Deformation statistics. Total time includes node generation time and connection time for 1000 nodes. We use OBPRM for Sliding and PRM for Narrow and Stamping. Deformation time includes time for deformation, time for relaxation (for BBox only), and time for energy calculation. The average deformation time is for the successful deformations. All times are in seconds.

Comparison of Roadmap Generation and Deformation Techniques

Generation Method	Roadmap Generation		Bounding Box				Geometric			
			Time(s)				Time(min)			
	#CC	Time(s)	Query	Deform	#Deform	#Failed	Query	Deform	#Deform	#Failed
Shrunk (2 models)	22	2.8	619	617	1403	25	N/A	N/A	N/A	N/A
Shrunk (6 models)	21	4.3	82	81	199	0	322	322	212	1
Penetration (large)	3	14.9	144	144	328	4	514	514	344	3
Penetration (small)	14	25.6	75	75	162	2	255	255	174	1

Table 2: Comparison of roadmap generation (350 nodes) and deformation methods for the narrow passage environment. The table shows query and total deformation times as well as number of deformations and the number of the failed deformations. These values are for different roadmap generation methods; using 2 shrunk robots and 6 shrunk robots, enabling small and large penetration. Deform time is total deformation time in the query phase. The times for Bounding Box Deformation are in seconds while the times for Geometric deformation are in minutes.

References

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 155–168, 1998.

[2] E. Anshelevich, S. Owens, F. Lamiraux, and L. Kavraki. Deformable volumes in path planning applications. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2290–2295, 2000.

[3] A. H. Barr. Global and local deformations of solid primitives. In *Proc. ACM SIGGRAPH*, pages 21–30, 1984.

[4] O. B. Bayazit, G. Song, and N. M. Amato. Enhancing randomized motion planners: Exploring with haptic hints. *Autonomous Robots, Special Issue on Personal Robotics*, 10(2):163–174, 2001. Preliminary version appeared in ICRA 2000, pp. 529–536.

[5] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 521–528, 2000.

[6] S. Fisher and M. C. Lin. Deformed distance fields for simulation of non-penetrationg flexible bodies. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2001.

[7] S. Gibson. 3D Chainmail: A fast algorithm for deforming volumetric objects. In *Proc. Symp. Interactive 3D Graphics*, pages 149–154, 1997.

[8] S. Gibson. Using linked volumes to model object collision, deformation cutting, carving, and joining. In *IEEE Visualization and Computer Graphics*, pages 169–177, 1999.

[9] S. Gibson and B. Mirtich. A survey of deformable modeling in computer grapics. In *Technical Report TR-97-19,MERL*, 1997.

[10] D. House, R. DeVaul, and D. Breen. Towards simulating cloth dynamics using interacting particles. *International Journal of Clothing Science and Technology*, 8(3):75–94, 1996.

[11] D. Hsu, L. Kavraki, J-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, 1998.

[12] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[13] F. Lamiraux and L. Kavraki. Planning paths for elastic objects under manipulation constraints. *The International Journal of Robotics Research*, 20(3):188–208, 2001.

[14] G. S. P. Miller. The motion dynamics of snakes and worms. In *Proc. ACM SIGGRAPH*, pages 169–177, 1988.

[15] B. Mirtich. V-clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electric Research Lab, Cambridge, MA, 1997.

[16] C. L. Nielsen and L. E. Kavraki. A two level fuzzy prm for manipulation planning. Technical Report TR2000-365, Computer Science, Rice University, Houston, TX, 2000.

[17] T. Sederberg and S. Parry. Free-form deformation of solid gemetic models. In *Proc. ACM SIGGRAPH*, pages 151–160, 1986.

[18] G. Song, , S.L. Miller, and N. M. Amato. Customizing PRM roadmaps at query time. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1500–1505, 2001.

[19] X. Tu and D. Terzopoulos. Artifical fishes: physics, locomotion, perception, behavior. In *Proc. ACM SIGGRAPH*, pages 43–50, 1994.

[20] R.W. Sumner R. W., J.F. Brien, and J.K. Hodgins. Animating sand, mud, and snow. In *Computer Graphics Forum*, 1999.