Planning Motion in Completely Deformable Environments[*]

Samuel Rodríguez[†]
sor8786@cs.tamu.edu

Jyh-Ming Lien[†]
neilien@cs.tamu.edu

Nancy M. Amato[†]
amato@cs.tamu.edu

***Abstract*— Though motion planning has been studied extensively for rigid and articulated robots, motion planning for deformable objects is an area that has received far less attention. In this paper we present a framework for planning paths in completely deformable, elastic environments. We apply a deformable model to the robot and obstacles in the environment and present a kinodynamic planning algorithm suited for this type of deformable motion planning. The planning algorithm is based on the Rapidly-Exploring Random Tree (RRT) path planning algorithm. To the best of our knowledge, this is the first work that plans paths in totally deformable environments.**

I. Introduction

The type of deformable motion planning addressed here involves planning paths in environments where the robot and obstacles are extremely flexible. Many situations arise where the environment can consist of elastically deformable objects. Examples of this type of planning could range from planning a path to move a mattress in a house or surgical applications. Motion planning for highly deformable objects is a problem that has received very little attention although it is important to be able to solve these problems.

Flexible representations are needed for problems that require robots to have large deformations and for problems that require not only the robot but the obstacles in the environment to be deformable. Deformable motion planning involves the planning of feasible paths for objects that can completely change their shape depending on the interaction with the environment. Examples of this include planning for elastic or air-filled objects, metal sheets or long flexible tubes. Moreover, for several cases, such as virtual surgery applications, it is important to be able to model environments that include both rigid objects, which could include bones and surgical instruments, and deformable objects, which could include soft tissue, flexible cartilage or flexible tubes.

Other applications of deformable motion planning include computer graphics and games. Motion of deformable objects in these areas is mostly generated using simulation because manually generating natural-looking deformation is extremely difficult and time consuming. On the other hand, it is also difficult to control the trajectory and the final resting location of a simulated deformable object. Naturally deformable motion planning can provide an intuitive and interactive tool to control the motion of deformable objects by allowing the users to set start and (intermediate) goal configurations.

[*]This research supported in part by NSF Grants EIA-0103742, ACR-0081510, ACR-0113971, CCR-0113974, ACI-0326350, and by the DOE. Rodriguez supported in part by a National Physical Sciences Consortium Fellowship.

[†]Parasol Lab., CS Dept., Texas A&M University.

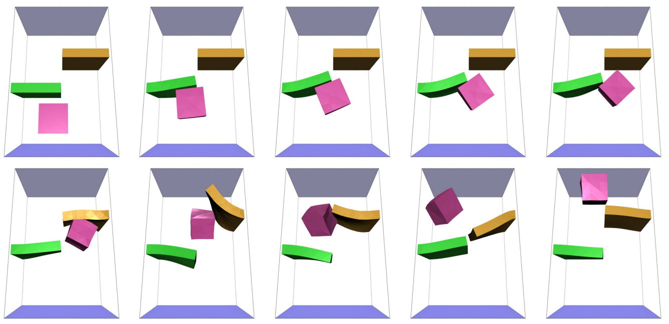


Fig. 1. Barriers Environment. Both the robot (the cube) and the obstacles (the plate barriers) in this environment are deformable. This image sequence is shown from left to right and from top to bottom.

Issues. One problem facing motion planning for deformable objects is having a (deformation) model that accurately reflects the physical properties of objects while maintaining the efficiency of the planner. As we have seen from the previously proposed methods, it is a difficult problem to maintain both properties. For example, a planner that uses a physically correct deformation model can be very slow [1] and a planner that uses only geometric deformations can result in unnatural motion [3]. The concept of visually plausible deformation has been used in planning motion for deformable objects to accelerate the speed of the planner [3], [5]. An aspect of visually plausible deformation that has not been addressed is how well a deformable model preserves the volume of an object undergoing deformation. Another issue that should be considered when selecting a deformable model is how the model handles varying material properties.

The workspaces studied in most previous work generally include only static and rigid obstacles. While problems with dynamic workspace have been addressed before, e.g., [8], [12], to the best of our knowledge, there is no previous work that plans paths in totally dynamic and deformable environments, such as under water environments or the internal body.

Our approach. As outlined above, there is a need for motion planning that can efficiently generate natural motion for deformable robots to maneuver among obstacles that can be either rigid or deformable themselves.

In this work, we address this problem and study motion planning problems in highly deformable workspaces. We represent deformable objects (robots and/or obstacles), as tetrahedral meshes and the dynamics of these tetrahedral meshes are modeled using a Finite Element Method [19] that includes volume preservation.

The proposed method plans a path for the robot by iteratively applying manipulation forces to the the robot until it reaches a configuration near the goal configuration. While we only apply forces to the robot, obstacles that are deformable will receive external forces through contact with the robot and other obstacles. Thus, each state of the problem space consists of the position and velocity of the robot and the obstacles as well. Figure 1 shows a sequence of images illustrating a solution path found by our planner in a highly deformable environment. As far as we are aware, this is the first work that plans paths in this type of environments.

Outline. This paper is organized as follows. In Section II we discuss related work. An overview of our method is described in Section III. The deformable model we use is described in Section IV, planning is discussed in Section V and finally we will close with results and discussion.

II. Related Work

There have been a number of deformable models developed. The goal of studying these models is to obtain a deformable model that can accurately reflect objects with varying properties. In this section we will give an overview of some deformable models that have been developed, mostly in computer graphics. We will then discuss some previous attempts at planning motion for deformable objects.

A. Deformable Models

Deformations which do not address physical properties include functional deformations (scale, bend, twist) [2] and free-form deformation [18]. Mass-spring and finite element methods (FEMs) are among the most commonly used strategies for building deformable objects with physical properties. Details of these methods are described below. While it is impossible for us to provide a complete review of work on modeling deformable objects here, several surveys are available on this topic [6], [17].

Mass-spring. Mass-spring systems are one of the most common forms of deformable models. In mass-spring systems, objects are modeled as interconnected points connected to each other by springs [6]. These springs can either be modeled as linear or nonlinear springs for varying material effects. This type of system often results in "stiff" equations causing stable numerical integration to be difficult.

FEM based systems. In recent years, several FEM-based deformation models have been developed [9], [16], [19]. Using an FEM-based system, objects are divided into a set of discrete geometric elements in order to approximate the volumetric structure of the object. Various deformation models can be applied to these meshes that are used with the finite element method. A strength of FEM based deformation is that they can handle large deformations.

B. Deformable Motion Planning

Motion planning for deformable objects is not new. The methods described below all require the environment to be static and are all extensions of the Probabilistic Roadmap Method (PRMs) [11], which sample and connect random configurations in the (reduced) configuration space of the robot. The f-PRM framework [1], [13] has been proposed for planning for models using physically correct deformations. f-PRM uses a variation of a mass-spring system that attempts volume preservation by minimizing the energy in a deformed state. 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.

A PRM-based method for path planning using geometric deformation is proposed in [3]. In this work a more efficient planner has been proposed by considering only geometric deformation that does not require volume preservation. The problem with this approach is that it could result in unrealistic deformation and hence, simulated paths that would not make sense in the real world.

Another attempt at motion planning for deformable robots in a static environment was presented in [5]. The planning was done using PRMs with a point robot and the deformation model was a variation of mass-spring systems. The scheme presented for volume preservation is not suitable for large deformations which would be apparent in situations where the robot is required to fold or bend dramatically. Analysis of the volume preservation scheme was not presented.

III. Overview

The motion planning problems that we are interested in for this work can typically be stated as follows: given a workspace, the objective is to find a sequence of forces to move a deformable object (robot) from its start configuration to its goal configuration among a set of rigid or deformable obstacles without introducing unrealistic deformations. In addition to the *manipulation forces* generated by the motion planner, the motion planner has to deal with other forces that are defined by the system, e.g., gravity, and those that are introduced by deformations and interactions among robots and obstacles, e.g., volume preservation and collision response forces. Figure 2(a) illustrates 5 types of forces that can be applied to deformable robots and obstacles. It is these forces that bring the state of the system forward. Details of the deformable model and how these forces can be generated are discussed in Section IV.

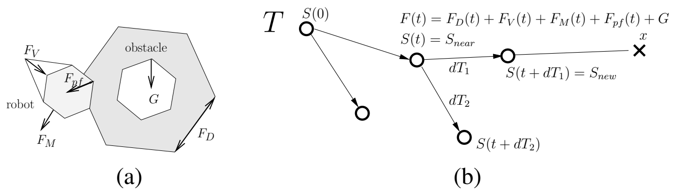


Fig. 2. (a) Forces that can be applied to both robots and obstacles. F_V, F_D, F_{pf}, F_M and G are volume and distance preservation, collision response, manipulation, and gravity forces, resp. F_M is always zero for obstacles. (b) Illustrations of how an RRT expands during planning. A state $S(t)$ is brought toward a randomly sampled point x by applying forces **F**(t) to robots and obstacles for a certain amount of time dT_1.

Because both robots and obstacles can be deformable in our system, a state $S(t)$ of the system at time t contains the *position* and *velocity* of all movable vertices. These states are organized in a tree $\mathcal{T}$ as shown in Figure 2(b). We say that a motion planning problem is solved when the tree $\mathcal{T}$ can be connected to the goal configuration. More precisely, $\mathcal{T}$ is constructed using the Rapidly-Exploring Random Tree (RRT) [14] approach by iteratively sampling a random position

x in workspace and manipulating the state with the robot closest to x towards x for a certain amount of time. Details of our planning method are discussed in Section V.

By kinodynamic planning using an RRT, we are able to accurately plan paths in these dynamically deformable environments. The PRM framework that has been used in all previous work, described in Section II-B, is generally not suitable for representing and handling the dynamic nature of deformable objects. This is because the current shape of the objects not only depends on the current state of the system but also depends on previous system states.

IV. The Deformable Model

The deformable model that we have chosen to work with comes from [19]. This is an FEM-based system that considers energies that the model uses in order to stabilize the system. Objects in the environment are represented in the form of tetrahedral meshes. More information for this model, collision detection [20] and response systems [7] will be described in the following section.

A. Model

The model [19] considers constraints on tetrahedral meshes. These constraints are of the form $C(\mathbf{p}_0, \ldots, \mathbf{p}_{n-1})$ where $\mathbf{p}_i$ are mass points in the tetrahedral mesh. Forces are derived from the energy constraints that are of the form

$$E(\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}) = \frac{1}{2} k C^2 \tag{1}$$

where k is the stiffness coefficient for the given constraint C. A force $\mathbf{F}^i$, for a given mass point $\mathbf{p}_i$ and constraint C, is derived from the energy constraint resulting in

$$\mathbf{F}^i(\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}) = -\frac{\partial}{\partial \mathbf{p}_i} E = -kC \frac{\partial C}{\partial \mathbf{p}_i} \tag{2}$$

At each time step the forces on each mass point are found and used for numerical integration, which is described in Section IV-B. Although three constraints were described in [19], we will only describe the distance and volume preserving forces used in our model. Surface area preserving forces are omitted since they are most useful for planar tetrahedral meshes, which we do not use.

Distance Preserving Forces. A distance preserving potential energy E_D is used to maintain distances between connected mass points in the tetrahedral mesh. The distance preserving potential energy uses the distance between neighboring mass points at a given time and the distance in the undeformed state to compute the potential energy at that time. This energy is given by

$$E_D(\mathbf{p}_i, \mathbf{p}_j) = \frac{1}{2} k_D \left(\frac{|\mathbf{p}_j - \mathbf{p}_i| - D_0}{D_0} \right)^2 \tag{3}$$

The purpose of this force is to restore the distances of deformed links to their original lengths. Damping, which greatly improves the numerical integration, is also used for the distance preserving forces, as described in [19].

Volume Preserving Forces. Volume preservation forces are computed in a manner similar to distance preservation forces. The energy constraint for each tetrahedron is given by Equation 4. V_0 is the initial volume of the tetrahedron being considered and the force computed will restore the volume of a deformed tetrahedron. This force will also work to restore the volume for inverted tetrahedra [19].

$$E_V(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k, \mathbf{p}_l) =$$

$$\frac{k_V}{2} \frac{\left(\frac{1}{6} (\mathbf{p}_j - \mathbf{p}_i) \cdot ((\mathbf{p}_k - \mathbf{p}_i) \times (\mathbf{p}_l - \mathbf{p}_i)) - V_0 \right)^2}{V_0^2}. \tag{4}$$

The forces presented to preserve distance and volume of a tetrahedral mesh will work on deformed tetrahedral meshes in order to restore the tetrahedral mesh to the original shape. By varying parameters k_D and k_V, various material properties can be simulated. The complexity of the tetrahedral meshes determines if the deformable model can be used in real-time.

B. Integration

We use Verlet Integration as our numerical integration scheme as described in [19]. This method was shown to be stable and fast. The speed comes from only having to compute the forces $\mathbf{F}(t)$ once per time step. $\mathbf{F}(t)$ is the sum of $\mathbf{F}_D(t)$, $\mathbf{F}_V(t)$ and any additional external forces which could include gravity, manipulation forces or collision forces.

C. Collision Detection and Response

Collision detection for deformable objects is particularly difficult since the size and shape of the objects are continuously changing [21]. In our motion planning library for deformable objects, we use the collision detection algorithm presented in [20]. This algorithm can quickly determine collisions for deformable objects relying on spatially dividing objects and restricting more expensive collision detection calls between objects that are in areas that are spatially close. This process is done at each time step of the simulation.

First in [20], vertices and tetrahedra are spatially hashed. Vertices are only spatially hashed to one hash cell by the vertex position. Tetrahedra can be hashed to multiple hash cells depending on which cells the axis-aligned bounding box (AABB) of the tetrahedron possibly overlaps. Intersection tests between vertices and tetrahedra can then be done using barycentric-coordinate tests. In this way vertices can be labeled as being either collision free or in collision.

The kind of collision response employed computes penalty forces which are based on the penetration depth of colliding vertices. Although we will give an overview of the system, the reader can refer to [7] for a more detailed description.

The first step assumes it has been determined whether or not a vertex is in collision. Based on these colliding vertices, intersecting edges can be found. Intersecting edges are edges that come from the tetrahedral mesh which contain one vertex that is in collision and one vertex that is not in collision. The intersecting edges are used to find the intersection point and a normal to the penetrated surface. Next, the penetration depth and direction of all intersecting vertices is computed. We use a linear response function for colliding vertices based on the penetration depth and direction.

V. Kinodynamic Planning

Using kinodynamic planning, we are interested in finding a path from some start state s to a goal state g. States in this framework can be considered to be of the form

$$\mathbf{S}(t) = \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{pmatrix},$$

where $\mathbf{x}(t)$ and $\mathbf{v}(t)$ are lists of the object's positions and velocities at time t. During the simulation, forces $\mathbf{F}(t)$, are used to manipulate the deformable object, restore the deformed object to its original shape and ensure the deformed objects are collision free. Manipulation forces are used to move the object through the environment. Forces used to restore the shape and the collision-free state of the deformable objects are $\mathbf{F}_D(t)$, $\mathbf{F}_V(t)$ and $\mathbf{F}_{pf}(t)$ as previously described.

Our general path planning algorithm for completely deformable environments is shown in Algorithm 1. The manipulation forces are used to move the object through the environment, thus exploring the environment. It is important to note that in line 12 we store the state of the deformed robot as well as the deformed state of any deformable obstacles in the environment. This makes the complexity of the planning much more difficult, but enable us to obtain paths that look much more physically realistic. We can avoid storing states of objects that are not deforming. These deformed states for the robot and obstacles are restored at each iteration which prevents discontinuities in the path.

The step size used at each iteration dT should not be confused with the time step, h, used during numerical integration. h is usually much smaller than dT which allows the manipulation force $\mathbf{F}_M(t)$ to be applied for many time steps. In this way, we can grow a tree from a start state s to a state that is within a certain distance of the goal state g.

Algorithm 1 Kinodynamic Planning for Deformable Environments

```
Require: a tree $\mathcal{T}$, total iterations $I$, minimum time step $T_{min}$, and
    maximum time step $T_{max}$
 1: for $i \in \{0, 1, \cdots, I\}$ do
 2:   $x$ = a random point in bounding box or the goal position
 3:   $S_{near}$ = Nearest neighbor to robot COM in $\mathcal{T}$ to $x$
 4:   $\mathbf{F}_M(t)$ = GetManipulationForce($S_{near}$,$x$)
 5:   Let $dT$ be a random time step, where $T_{min} < dT < T_{max}$
 6:   $S_{cur}$ = $S_{near}$
 7:   for $k \in \{0, 1, \cdots, dT\}$ do
 8:     $\mathbf{F}_{pf}(t)$ = Calculate Penetration Forces
 9:     $S_{new}$ = $S_{cur}$.UpdateState($\mathbf{F}_{pf}(t)$, $\mathbf{F}_M(t)$, $\mathbf{F}_D(t)$, $\mathbf{F}_V(t)$)
10:     $S_{cur}$ = $S_{new}$
11:   end for
12:   add $S_{cur}$ to $\mathcal{T}$ and add an edge between $S_{cur}$ and $S_{near}$
13: end for
14: return $\mathcal{T}$
```

A. Manipulation Forces

Manipulation forces are selected at random depending on the weight assigned by the user to that manipulation force. Selecting a manipulation force to apply for a given time dT corresponds to the function GetManipulationForce(S_{near},x) in line 4 in Algorithm 1. For example, we can assign higher weights to the manipulation forces we think will help more in the planning phase. In the following we describe the types of manipulation forces used during our planning.

Body manipulation forces are applied to each vertex in the tetrahedral mesh. The forces result in translating the robot by a given force vector. The direction of the force is given by the vector between the random position x in the environment and the current robot's center of mass (COM) in S_{near}. The magnitude of the body force applied is selected at random such that the force is less than a predefined maximum force. This type of force is useful in pushing the object to various locations in the environment and when changes in orientation are not needed to move through a given area.

Control Point manipulation forces are applied only to certain vertices. A control point vertex for a robot's tetrahedral mesh is selected at random. The control point, along with any other mesh vertices within a predefined radius, have a manipulation force applied to it that is generated in a way similar to the body forces. This type of force is useful in changing the orientation of the object as a whole. By applying a force to only certain vertices, the object will rotate causing the change in the body's orientation.

Interpolation can be used to get from one state in the tree to another state. Although in a general framework this force could be used to transition between any two states in a roadmap, we only use the interpolation forces to move from a state in the tree to the goal state g. This type of force will help to quickly get from the current configuration to a state near the goal state.

B. Query

During the query phase, the tree is searched for a robot state that is within a certain threshold of the goal state. Although our current implementation uses Euclidean distance between the center of mass of the states in the tree and the goal state, a more precise distance metric could be used. This might involve averaging the distances between each vertex of the states in question. When simulating the path, an interpolation force is applied at the end of the path to allow the robot to come to rest near the goal state.

VI. Details and Experimental Results

In this section, we will discuss some of the implementation details of our framework and present the performance and flexibility of the proposed motion planner for deformable objects under various situations.

A. Implementation Details

We generate a tetrahedral mesh of a given boundary represented geometry (i.e., polygonal mesh) by first using convex decomposition and then computing a Delaunay triangulation on each component of the decomposition [4], [10]. Although our base planner can efficiently handle an environment with 2500 tetrahedra, more interesting and realistic shapes can easily have tens of thousands of tetrahedra which become a bottleneck of the planner's efficiency.

In order to handle more complex geometries, we use *hierarchical deformation*. Hierarchical deformation has been shown to perform efficiently using bounding boxes for motion planning [3]. Instead of using bounding boxes, we use convex shells that can tightly approximate the target shape to produce more natural deformations. More precisely, we first approximate a shape using several disjoint convex shells [15]. Tetrahedral meshes are then generated from these convex shells. Deformations applied to the tetrahedral meshes are indirectly applied to the enclosed geometry using Free Form

Deformation (FFD) [18] (which computes the global coordinates of each vertex x of the geometry at every simulation time step using the Barycentric coordinates of x in x's enclosing tetrahedron). An illustration of this process is shown in Figure 3.

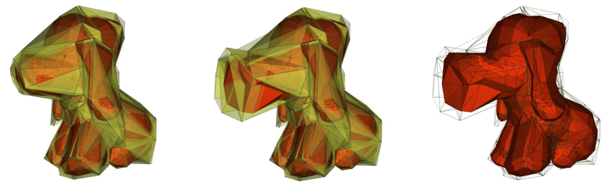


Fig. 3. Hierarchical deformation. First, convex shells are built from the input model. Next, a tetrahedral mesh is built from the convex shell. Then, the input model is bound to the tetrahedral mesh. Finally, deformations that are applied to the tetrahedral mesh can be indirectly applied to the input model.

B. Experiment Setup

We test our deformable planner with the environments shown in Figures 1, 6, 7, and a human body model. Using these examples, we will show that the proposed planner can efficiently handle large deformations, and highly dynamic situations in complex environments with a large number of tetrahedra and triangles and with both deformable and rigid obstacles. Details of these test sets are presented in Table I. All of our experimental results are obtained using a notebook computer with a 1.7 GHz CPU with 1 Gb memory. Animations and more detailed results are available at our website[‡].

TABLE I
Environment Properties and Experimental Results.

Environment names	**Obstacles types**	**Total Tetra**	**Sol. Iteration**	**Sol. Time** (*min*)
Barriers 1	deformable	673	500	10:34
Barriers 2	deformable	2343	1000	44:14
Windows	deformable	1710	2200	63:10
Falling Objects	deformable	396	1500	16:00
Body Parts	rigid/deformable	300	640	14:09

Total Tetra shows the number of tetrahedra in the environment including a robot and obstacles. **Sol. Iteration** and **Sol. Time** are the number of iterations to expand the tree and the time (in minutes) to solve the problem, resp.

C. Experiment 1: Barriers

Paths in the barriers environment result in large deformation; see Figure 1. The obstacles in the environment are the two plates positioned on opposite walls. The obstacles do not have an external gravity force applied to them and so the deformation occurs as a result of collision. The goal state of the robot (cube) is on the opposite side of the plates from the start state. The large deformation occurs as the robot maneuvers through the plates, in the process causing large deformations.

Barriers 1 environment is a simplified version of Barriers 2 environment. The simplification comes from Barriers 1 having less tetrahedra to represent the same obstacles. The actual workspace that the robot travels through is the same in both environments. Barriers 2 is one of the more computationally complex environments tested due to the number of tetrahedra. As shown in Figure 4, the volume of the objects in the environment remains relatively stable even during the times the obstacles undergo large deformation. We noticed similar results in the Barriers 1 environment with volume change less than three percent for all objects.

[‡]http://parasol.tamu.edu/

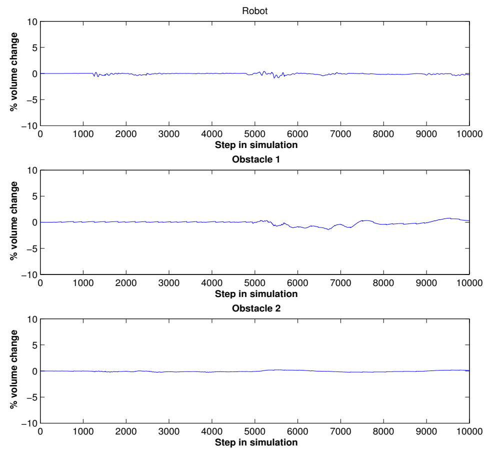


Fig. 4. Barriers Environment. Volume differences of robot and obstacles from their rest shape during path traversal.

D. Experiment 2: Windows

The robot in this environment has several ways to reach its goal. The environment consists of four chambers connected through passages of varying sizes. The passage way from the start chamber to the goal chamber is smaller than all of the other passage ways connecting neighboring chambers. Also, some points of the obstacle are held fixed during the simulation to allow the obstacle or passage-ways to maintain its same basic structure. During the simulation and planning phases, the passage ways have gravity as an external force. We allow the robot (sphere) to be much more flexible as it has to squeeze through each passage way from one chamber to another. The path obtained shows the robot taking a path that requires less deformation and so, is energetically more feasible.

The volume difference for the obstacle during the simulation of the path remains fairly constant. As shown in Figure 5, the robot, which is very flexible, usually maintains a volume difference that is less than six percent. However, there are times when the volume difference reaches ten percent.

E. Experiment 3: Falling Objects

In the falling objects environment, the objects are free to fall given gravity as an external force. The robot has to maneuver through the opening in the falling plate before it reaches the ground. The goal state of the robot (bunny) is above the plate. The other falling object in the environment is a dragon. This is a highly dynamic environment as the robot has to plan the path as objects are falling, making planning much more difficult.

The volume change for the falling plate remains close to constant throughout the simulation. Although the volumes for the robot and the other obstacle in the environment vary, they

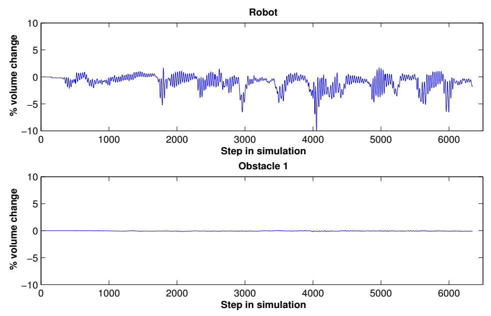


Fig. 5. Windows Environment. Volume differences of robot and obstacles from their rest shape during path traversal.

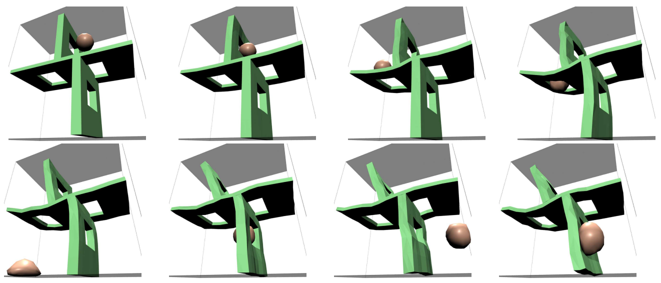
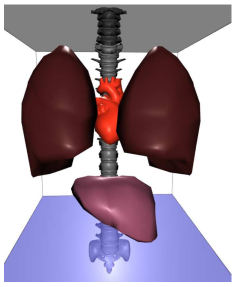


Fig. 6. Windows Environment. Both the robot (the sphere) and the obstacle (the wall with windows) in this environment are deformable. This image sequence is shown from left to right and from top to bottom.

remain within ten percent of their initial volume. This is a good result given the amount of collision and external forces applied to all objects.

F. Experiment 4: Human Body Model

This environment, shown on the right, is a simplified model of the human body consisting of a heart, lungs and a liver which are all deformable objects. The spine is modeled as a rigid body. In this environment, the robot is the heart and has to reach a goal state in front of the lungs and liver. The lungs and liver both have fixed points that prevent them from moving too much in the environment although these objects do deform. The deformation of these high resolution meshes is done using FFD as previously described.



We are able to obtain a path for the heart that slides between the lungs. The percent of volume change for all the deformable objects in this environment is typically within five percent. Given the interaction needed between the robot (heart) and obstacles (lungs, liver and spine), the amount of deformation obtained is reasonable.

VII. Conclusion

In this paper, we proposed a motion planning method that finds paths for deformable robots in completely deformable environments. To generate realistic motion, our deformable model considers volume preservation and our results show that the volumes of all deformable object (including robots and obstacles) are well preserved, within ten percent of their rest shape. The proposed planner is a tree-based planner that constructs a tree in the state space of the system until a state in the tree approaches the goal state. Our experimental results show that our planner can efficiently handle complex and dynamic environments with thousands of tetrahedra.

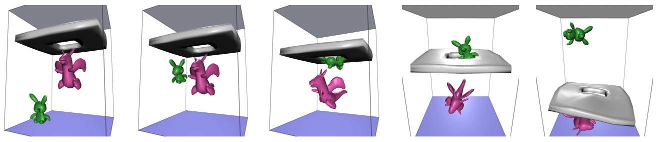


Fig. 7. Falling Objects Environment. Both the robot (the bunny) and the obstacle (the dragon and the wall with a hole) in this environment are deformable and free flying. This image sequence is shown from left to right.

References

[1] 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.

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

[3] O. B. Bayazit, J.-M. Lien, and N. M. Amato. Probabilistic roadmap motion planning for deformable objects. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2126–2133, May 2002.

[4] J.-D. Boissonnat. Automatic solid modeler for robotics applications. In *Robotics Research: Third International Symposium. MIT Press Series in Artificial Intelligence.*, pages 65–72, Cambridge, MA, 1986. MIT Press.

[5] R. Gayle, M. C. Lin, and D. Manocha. Constraint-based motion planning of deformable robots. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1058–1065, 2005.

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

[7] B. Heidelberger, M. Teschner, R. Keiser, M. Müller, and M. Gross. Consistent penetration depth estimation for deformable collision response. In *Proc. Vision, Modeling, Visualization (VMV)*, pages 339–346, 2004.

[8] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages SA1–SA18, 2000.

[9] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pages 131–140, 2004.

[10] B. Joe. Geompack. A software package for the generation of meshes using geometric algorithms. *Advances in Engineering Software and Workstations*, 13(5–6):325–331, Sept. 1991.

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

[12] R. Kindel, D. Hsu, J. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 537–543, 2000.

[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] S. M. LaValle and J. J. Kuffner. Rapidly-Exploring Random Trees: Progress and Prospects. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages SA45–SA59, 2000.

[15] J.-M. Lien and N. M. Amato. Approximate convex decomposition of polyhedra. Technical Report TR05-001, Parasol Lab, Dept. of Computer Science, Texas A&M University, Jan 2005.

[16] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler. Stable real-time deformations. In *Proc. ACM SIGGRAPH*, pages 49–54, 2002.

[17] A. Nealen, M. Muller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. In *Proc. Eurographics State-of-the-Art Report*, 2005.

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

[19] M. Teschner, B. Heidelberger, M. Müller, and M. Gross. A versatile and robust model for geometrically complex deformable solids. In *Proc. of Computer Graphics International (CGI)*, pages 312–319, 2004.

[20] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranerts, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling, Visualization (VMV)*, pages 47–54, 2003.

[21] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnetat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. In *Proc. Eurographics State-of-the-Art Report*, 2004.