Path Planning for Deformable Robots in Complex Environments

Russell Gayle*, Paul Segars†, Ming C. Lin*, and Dinesh Manocha*
*Department of Computer Science, University of North Carolina at Chapel Hill
†Departments of Radiology, Biomedical Engineering, and Environmental Health Sciences, Johns Hopkins University
http://gamma.cs.unc.edu/FlexiPlan/

***Abstract*— We present an algorithm for path planning for a flexible robot in complex environments. Our algorithm computes a collision free path by taking into account geometric and physical constraints, including obstacle avoidance, non-penetration constraint, volume preservation, surface tension, and energy minimization. We describe a new algorithm for collision detection between a deformable robot and fixed obstacles using graphics processors. We also present techniques to efficiently handle complex deformable models composed of tens of thousands of polygons and obtain significant performance improvement over previous approaches. Moreover, we demonstrate a practical application of our algorithm in performing path planning of catheters in liver chemoembolization.**

I. Introduction

Endoscopic manipulators for minimally invasive surgery, power assist suits for human-movement support, and flexible agents for entertainment are some examples of a growing number of "deformable robots" populating through many different applications. One of the major challenges in this area is controlling and planning the motion and behavior of these robots in simulated environments.

The problem of computing a collision free path for a robot through an environment has been extensively studied for decades. Practical path planning algorithms are known for rigid or articulated robots. In contrast, current planners for deformable robots are only capable of handling simple robots in small environments; these planners can take many hours to compute a collision free path. Motion planning for deformable robots introduces two major challenges. First, simulating physically plausible deformation for a robot is still considered a difficult problem in practice. In order to create any planning algorithm for flexible robots, we need to model the physical properties and mechanical constraints of the robots. The computational requirements of generating an accurate deformation using a continuum model can be rather high. The second challenge is fast and accurate collision detection between a deformable robot and surrounding obstacles. Current algorithms for collision detection between deformable models have a high overhead. Moreover, in many deformable planning scenarios, the free space of a robot becomes very constrained. The robot is often close to the obstacle boundary, leading to an extremely high number of proximity queries. As a result, collision detection is a major bottleneck in terms of developing efficient planners.

The driving application of our work is insertion of flexible catheters in human vessels for planning and guiding surgical procedures [8], [29]. Manipulation of catheters in small vessels frequently causes spasms, preventing adequate flow of fluids through the vessels. If the catheter has a cross-sectional area close to that of the vessel being injected, the size similarity will reduce fluid flow. Accurate path planning studies can help overcome this obstacle by becoming an integral part of preoperative surgical planning, i.e. choosing the size and properties of the catheter. However, accurate geometric models of the arteries and a deformable catheter consist of tens of thousands of primitives (e.g. polygons). Existing algorithms for motion planning and collision detection between deformable models are unable to handle models of such high combinatorial complexity.

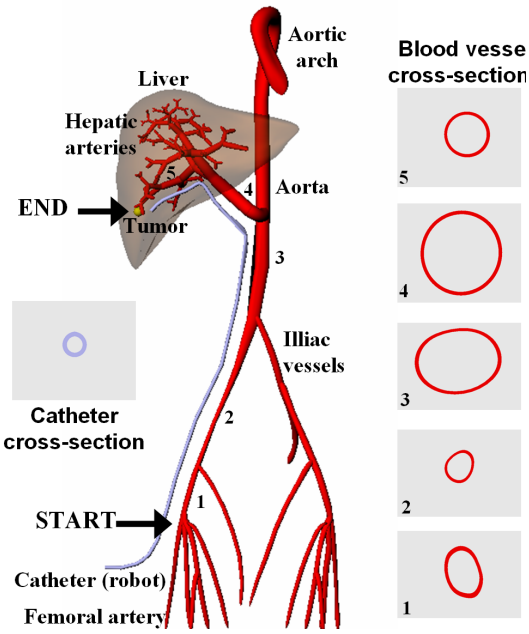

Fig. 1. **Path Planning of Catheters in Liver Chemoembolization:** The deformable catheter (robot), consisting of 10K triangles, is 1.35mm in diameter and approximately 1,000mm in length. The obstacles including the arteries and liver consist of more than 83K triangles. The diameter of the arteries varies in the range 2.5-6mm. Our goal is to compute a collision free path from the start to end configuration for the deformable catheter. The path computed by our motion planner is shown in Fig. 6.

Main Results: We present a novel path planning algorithm for a deformable robot in a complex environment. We treat the motion planning problem as a constrained dynamic sim-

ulation and then transform the planning problem into solving a list of constraints. The set of constraints include geometric constraints such as obstacle avoidance and non-penetration, as well as physical constraints such as volume preservation and energy minimization. We compute an estimated path using an approximate medial axis of the workspace and make appropriate adjustments and corrections to the estimated path using our constraint solver to compute a collision-free path.

Our planner checks for possible contacts between the robot and the obstacles during each simulation step. We present a new algorithm to detect collisions between a deformable model and a complex, stationary environment. We compute a potentially colliding set of overlapping primitives using *set-based* computations. Our algorithm uses 2.5D overlap tests between arbitrary objects and checks for the existence of a separating surface along a view direction. We use graphics processors (GPUs) to efficiently perform 2.5D overlap tests and we compute offsets and Minkowski sums to overcome image-precision errors. In practice, our collision detection algorithm is significantly faster when compared to prior approaches based on bounding volume hierarchies.

We have implemented and applied our planner to complex environments, including path planning of catheters for liver chemoembolization. Our planner can compute a collision-free path for a deformable robot in a complex environment consisting of tens of thousands of polygons in a few hours.

Organization: The rest of the paper is organized as follows. We give a brief overview of prior work on motion planning and collision detection for deformable models in Section 2. We give an overview of our approach in Section 3 and describe our planning algorithm in Section 4. In Section 5, we present a fast algorithm for collision detection between a deformable robot and stationary obstacles. We describe our implementation and its application to path planning of catheters for liver chemoembolization in Section 6.

II. Related Work

We briefly present an overview of prior research on planning of deformable robots and collision detection between flexible bodies.

A. Motion Planning for Deformable Robots

Most of the literature in robot motion planning has focused on robots with one or more rigid link. Some of the earlier work on deformable robots included specialized algorithms for bending pipes [26], cables [23] and metal sheets [24]. Holleman et al. [15] and Lamiraux et al. [18] presented a probabilistic planner capable of finding paths for a flexible surface patch by modeling the patch as a low degree Bèzier patch and used an approximate energy function to model deformation of the part. Guibas et al. [13] described a probabilistic algorithm for a surface patch by sampling the medial axis of the workspace. Anshelevich et al. [1] presented a path planning algorithm for simple volumes such as pipes and cables by using a mass-spring representation. Bayazit et al. [3] described a two-stage approach that initially computes an approximate path and then refines the path by applying geometric-based free-form deformation to the robot. Gayle et al. [9] a presented a motion planning algorithm for simple closed robots.

B. Collision Detection between Deformable Models

In this section, we give a brief overview of related work in collision detection between deformable models. The problem of collision detection has been extensively studied and some recent surveys are available in [5], [22], [27]. Some of the commonly used algorithms for collision detection are based on bounding volume hierarchies (BVHs). These hierarchies cull away portions of a model that are not in close proximity. Examples of such hierarchies include sphere-trees, AABB-trees, OBB-trees, k-DOP trees, etc. [5] and they are typically computed during preprocessing. Recently, algorithms have been proposed to lower the overhead of updating the hierarchy during every step of deformable simulation [16], [20], [28]

Many collision and proximity computation algorithms exploit the computational capabilities of graphics processors (GPUs) [2], [11], [14], [17]. Recent work also includes methods for self-collisions of deformable models [10]. Most of these algorithms involve no preprocessing, therefore applying to both rigid and deformable models. The GPU-based algorithms perform image-space computations and use the computation power of rasterization hardware to check for overlaps. However, a major issue with current GPU-based algorithms is limited accuracy due to image-space resolution, possibly resulting in missed collisions between small triangles due to sampling errors.

III. Overview

In this section, we give an overview of our planner. We introduce the notation used in the rest of the paper and present our framework to solve motion planning as a constrained dynamical system.

A. Modeling of Deformable Robots

The simplest physically-based deformable models are typically represented as mass-spring systems, where each object is modeled as a collection of point masses connected by springs in a lattice-like structure. In practice, mass-spring systems are easy to construct and can be simulated at interactive rates on current commodity hardware. More accurate physical models treat deformable objects as a continuum. One of the most commonly used continuous models is the finite element methods (FEM). The object is decomposed into elements joined at discrete node points and a function that solves the equilibrium equation is computed for each element. The computational requirements of FEM can be high (as a function of model complexity) and it is difficult to use them for complex models in real-time applications.

In this paper, we have chosen mass-spring systems for modeling a deformable robot. The main reasons for using this model is the overall runtime efficiency in representing a complex robot and the ease of implementation.

B. Notation

Let $\mathcal{R}$ be the deformable robot and $\mathcal{O}$ be the set of obstacles in the environment. The environment is composed of a set of obstacles $\prime = \{o_1, o_2, ...\}$ and the robot $\mathcal{R}$ is discretized and represented as a set of N masses m_i, each with varying positions over time t, $\forall i, 1 \leq i \leq N$. The masses are the connected by a set of M springs, r_j, $\forall j, 1 \leq j \leq M$. The areas enclosed by the springs is denoted by f_k. Along with each spring, a stress, $stress_j$, and a threshold value, δ_j, used to define material constraints, are stored.

Associated with each mass is a state vector $s_i(t) = (x_i(t), v_i(t))$ that represents its position and velocity at time t. The collection of position vectors, $X(t) = [x_1(t), x_2(t), \ldots, x_N(t)]$, represents the configuration of the robot at time t. Similarly, we can also define the state of the robot at time t as $S(t) = [s_1(t), s_2(t), \ldots, s_N(t)]$. As the robot $\mathcal{R}$ deforms when it comes into contact with an obstacle o_i in $\mathcal{O}$, the deformation can cause the total potential energy $E(X)$ of the system (i.e. the elastic solid of the robot) to change.

Planning Problem Formulation: The planning problem for a deformable robot can be stated as follows:

Find a sequential set of robot configurations $X(t_i), \ldots, X(t_f)$

such that no $X(t_k)$ intersects any obstacle in $\mathcal{O}$ and $X(t_k)$ satisfies geometric and physical constraints of the robot while minimizing the total energy $E(X)$ of the entire system,

where $X(t_i)$ and $X(t_f)$ are the initial and final configuration of the robot and $t_i \leq t_k \leq t_f$.

C. Constraint-Based Motion Planning

We treat the motion planning problem as a boundary value problem. In particular, motion planning can be viewed as a dynamical system in which the initial and final configurations represent the boundary values and conditions [7]. By reformulating the motion planning problem as a constrained dynamics simulation, we transform the planning problem into solving a list of constraints, while minimizing the cost functions (e.g. total potential energy of the deformable robot). The planning problem can be solved by computing a sequence of intermediate states that link the boundary values and satisfy each of these constraints. However, prior approaches only work for rigid robots or simple deformable models. Their main limitations are:

1) Earlier algorithms can handle rather simple geometry with tens or hundreds of polygons. They do not scale well to complex environments due to simple collision handling.
2) The quality of the computed path may be poor because an initial guiding path was generated using random sampling with no path smoothing, resulting in unrealistic deformations.
3) The robot may deform in an unnatural way due to the lack of surface tension.

Next, we will describe our overall planning algorithm that overcomes these three problems.

IV. Planning for Deformable Robots

In this section, we present our planning algorithm for a complex, flexible robot.

A. Simulation Framework

The basic approach of our planning framework is to describe each robot as a dynamical system. This system is characterized by its state variables, stored in $S(t)$ for each time t. Let $X(t)$ be the configuration of the robot at some time t as defined in Section 3; then each constraint can be represented as a function of $X(t)$ as $C(X(t))$. The virtual force induced by each constraint $C(X(t))$ is simply:

$$f_c = \frac{-\partial E(C(X(t)))}{\partial X(t)}$$

where the energy function, $E(C(X(t)))$, is defined as

$$E(C(X(t))) = \frac{K_s}{2} C(X(t)) \cdot C(X(t))$$

and K_s is a generalized stiffness matrix [31].

The simulation steps from time t to time $t + h$ and updates the state of the robot, subject to the forces induced by the constraints using the following steps:

BEGIN

Get System State: Get $S(t)$ by concatenating $s_i(t)$, for all i.

Compute Constraint Forces: Sum up all virtual forces, $F_c(S(t)) = \sum_{j=1}^{l} F_j(S(t))$.

Update Robot State: Compute $S(t + h)$ from $S(t)$ subject to $F_c(S(t))$.

Increment Time: $t = t + h$

END

In this framework, the solution to the motion planning problem emerges as the sequence of states, $\{S(t_i), \ldots, S(t), S(t+h), \ldots, S(t+k*h), \ldots, S(t_f)\}$, where the robot is at its initial configuration at time t_i, and reaches the goal configuration at time t_f. The simulation runs until the robot reaches its goal configuration.

B. Robot Deformation

Update Robot State is computed at each simulation step by using a second-order ordinary differential equation (ODE):

$$M\ddot{X}(t) + C\dot{X}(t) + KX(t) = F_c(S^t) + F_e(S^t),$$

where M and C are diagonal matrices, and K is a banded matrix. The ith diagonal element of M is simply the value of each mass m_i and similarly the ith diagonal element of C is the dampening constant for the mass m_i. K is banded since it must represent spring forces which are functions of the

distance between two masses. $F_c(S(t))$ and $F_e(S(t))$ are $3N$-dimensional vectors representing the constraint and external forces acting upon each of the N masses. To help reduce numerical instability from stiff systems, we solve the ODE with a semi-implicit Verlet integration scheme. This solver requires only one additional force computation step, keeping the computational cost low.

Next, we must verify if the geometric and physical constraints are satisfied, subject to minimization of the total energy in the system. If not, then we perform the following steps:

1) Set the last valid milestone as the next destination
2) Back trace one step on the current roadmap
3) Find a new path from the last valid milestone to the goal configuration
4) Compute new constraint forces and solve the ODE, using the previous state of the robot R and F_e
5) Set the next robot state to be the new ODE solution

C. Constraints

We impose a number of known geometric, physical, and mechanical constraints suitable for the problem and to handle deformations [9]; each of which can be classified as either hard or soft.

Hard Constraints are those that absolutely *must* be enforced at each simulation step, such as the non-penetration constraint. The non-penetration constraint is enforced by computing the collision response between the flexible robot and nearby obstacles when contacts occur.

Soft Constraints serve as guides to encourage or influence the objects in the scene to behave in certain ways. These constraints, including goal seeking, obstacle avoidance, path following, volume preservation, and enforcement of surface tension, are simulated by using penalty forces. Angular constraints between adjacent edges are used to enforce surface tension.

D. Energy Minimization

The elastic deformation energy measures the amount of deformation. If the motion is simply a rigid transformation, meaning that it preserves the distances between all particles (no stretches), the energy must be zero.

Let $E(X)$ be the energy density function of an elastic solid undergoing deformation. The total energy is obtained by integrating $E(X)$ over the entire volume of the solid. We have chosen the energy function of a spring network that connects the neighboring nodes. The energy function can be written as:

$$E_s(X(t)) = \sum_j \frac{k}{2}(d_j - L_j)^2$$

where j is the index of a spring and L_j is the natural length of the spring and d_j is the distance between two masses x_i and x_k connected by the spring.

Basically, we would like to compute X by solving

$$min \;\; E(X(t)) \text{ subject to } \nabla V(X(t)) \leq \epsilon.$$

Here, we relax the hard volume preservation constraint by allowing the change in volume to be less than a given tolerance ϵ. This problem can be solved by using a global constrained minimization technique. Our current implementation uses a local method that checks whether the internal pressure fluctuation is bounded and that the deformation at each edge e_i does not exceed certain pre-defined tolerance (i.e. $stress_i < \delta_i$) to achieve the same effects.

E. Guiding Path Generation

We use *estimated paths* to generate an initial approximation to the path. This idea has been used in motion planning [4], [6], [30]. In particular, we use a medial-axis based approach that computes an approximate medial-axis of the work space using voxelized methods and performs path smoothing. The estimated path tends to result in smoother final paths that maintain the farthest distance from nearby obstacles. However, this path may not be completely collision-free. The non-penetration constraint in our planning algorithm resolves any collision by either deforming the robot or adjusting the final path.

V. Collision Detection for a Deformable Robot

The running time of most practical motion planning algorithms is dominated by collision detection [21]. In the case of path planning for flexible or deformable robots, collision detection becomes a greater bottleneck due to the following reasons:

- The free space of a deformable robot is constrained, and in several configurations the boundary of the robot comes into close proximity of the obstacle boundary. This close proximity leads to a much higher number of potential contacts with the obstacles.
- Most prior collision detection algorithms are based on bounding volume hierarchies. As the robot deforms, the precomputed hierarchy needs to be updated to account for non-rigid motion. The cost of re-computing a hierarchy can be significantly higher for complex deformable models. Furthermore, the hierarchies may not be able to provide sufficient culling when the robot is in close proximity to the obstacles.

We present a new collision detection algorithm for a deformable robot undergoing motion among rigid obstacles. Our goal is to compute a small subset of potentially colliding primitives (e.g. triangles) and only perform exact interference tests among these primitives. Our algorithm is based on two main components:

1) **Reliable 2.5D overlap tests using GPUs:** Since the robot is close to the obstacle boundary, we perform a tighter overlap test by checking whether there exists any separating surface between the robot and the obstacles. We perform this test using the rasterization capabilities of the GPU. We also compute *Minkowski sums* of the robot and the environment with bounded spheres in order to overcome image-precision errors.
2) **Set-based computations:** In order to deal with a high number of colliding primitives, we compute sets of

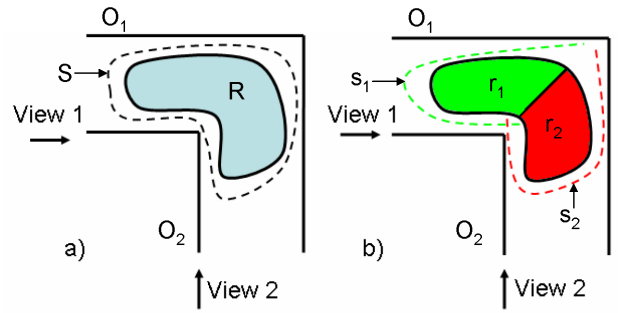

Fig. 2. This figure highlights the 2.5D overlap tests used for collision detection. The query checks whether there exists a separating surface along a view direction of depth complexity one. S has depth complexity more than one from View 1 as well as View 2. In the right image, s_1 has depth complexity one from View 1 and s_2 has depth complexity one from View 2. As a result, we use two 2.5D overlap tests to decide that R_1 and R_2 are not colliding with the obstacles (O_i).

potentially colliding primitives as opposed to computing each pair of overlapping primitives explicitly. The size of each set is at most $O(n)$, whereas the number of pairs in close proximity can be super-linear (or even $O(n^2)$ in the worst case) for a robot in a constrained free space.

A. Reliable 2.5D overlap tests using GPUs

The robot undergoes non-rigid deformation between successive steps of path planning. Instead of using BVHs, we check for overlaps between the robot and the obstacles using the rasterization capabilities of graphics processing units (GPUs). The GPUs are widely available on all commodity PCs and their computational capabilities are increasing at a rate exceeding Moore's law.

We perform visibility computations [11] between the objects on the GPUs to check whether $\mathcal{R}$ (robot) and $\mathcal{O}$ (obstacles) overlap. In particular, we choose a view direction, usually along an axis, and check whether $\mathcal{R}$ is fully visible with respect to $\mathcal{O}$ along that direction. If $\mathcal{R}$ is fully visible, then there exists a separating surface between $\mathcal{R}$ and $\mathcal{O}$ (see Fig. 2). Moreover, the separating surface needs to have a one-to-one mapping with a plane orthogonal to the viewing direction or depth complexity one along the view direction. We call this the *2.5D overlap* query; this provides a sufficient condition that the two primitives do not overlap. The 2.5D overlap test is significantly less conservative and more powerful as compared to earlier collision detection algorithms that check whether two bounding volumes (e.g. spheres, OBBs, etc.) overlap in 3D. For example, in Fig. 2(b) there exists a single separating surface between R_1 and the obstacles as well as R_2 and the obstacles. In this case, we can verify with two 2.5D queries that the robot does not overlap with the obstacles.

A main problem with a GPU-based overlap test is the underlying image precision used to perform visibility computations. In particular, the rasterization of $\mathcal{R}$ or $\mathcal{O}$ introduces many sampling errors, including projective errors and depth-buffer precision errors. In order to overcome these errors, we compute and render a bounding offset for each object. Let the dimension of square pixel used for orthographic projection be p. Moreover, let S_p represent a sphere of radius $\sqrt{3}p/2$ and $\mathcal{R}^{S_p}$ and $\mathcal{O}^{S_p}$ represent the Minkowski sum of $\mathcal{R}$ and $\mathcal{O}$ with S_p, respectively. In this case, we use the following lemma:

Lemma: *If $\mathcal{R}^{S_p}$ is fully visible with respect to $\mathcal{O}^{S_p}$ from any view direction under orthographic projection on a $2D$ discrete grid with pixel size p, than $\mathcal{R}$ and $\mathcal{S}$ do not overlap.*

We omit the proof due to space limitations. This lemma provides us with a sufficient condition that the robot and the obstacle do not overlap. The exact computation of the Minkowski sum of a primitive with a sphere corresponds to the offset of that primitive. The exact offset representation consists of non-linear spherical boundaries. Instead, we compute a bounding approximation of the offset. In case of obstacles, we decompose the boundary into triangles, edges and vertices. The offsets of each of these primitives are represented as swept sphere volumes: as rectangular swept-sphere (RSS), line swept-sphere (LSS) and point swept-sphere (PSS), respectively [19]. We precompute the swept spheres to enclose the obstacle primitives. Since the robot undergoes deformation, we dynamically compute a bounding OBB (oriented bounded box) for each triangle on the boundary. The cross-section of the OBB has the same plane as the triangle and the height of the OBB is equal to $\sqrt{3}p$. Moreover, we perform the 2.5D overlap test from a number of fixed directions (e.g. X, Y and Z axes) to check for the existence of a separating surface.

B. Set-based Computations

Our algorithm uses the concept of a potentially colliding set (PCS) of objects or primitives [11]. In this section, we present a specialized algorithm for a deforming robot among fixed obstacles. Given a collection of primitives, $\mathbf{P} = \{p_1, \ldots, p_n\}$, we initially insert all the primitives into a PCS. Next, we check whether p_i overlaps with the remaining objects: $\mathbf{P} - \{p_i\}$. If they do not overlap, we remove p_i from the PCS. Based on this property, we reduce the number of object pairs that need to be checked for exact collision. There are two main issues in using set-based computations for collision detection:

- *Set-based overlap tests:* We need the capability to perform overlap tests between two different sets of objects. In particular, we need a simple test to check that the objects in $\mathbf{S_1}$ do not overlap with objects in $\mathbf{S_2}$. We use the reliable 2.5D overlap test described above.
- *Set partitions:* A set of n objects has 2^n subsets and we cannot check every possible pair of subsets for overlap. Rather we want to perform almost linear number of set-based overlap tests.

We compute two sets for collision detection. These are the $\mathcal{R}$-set and the $\mathcal{O}$-set. The $\mathcal{R}$-set = $\{r_1, r_2, \ldots, r_m\}$ consists of all the polygonal or triangular primitives used to represent the robot. If the number of triangles in the robot is high, we group them into small clusters and each r_i represents a cluster of triangles. In the same manner, $\mathcal{O}$-set = $\{o_1, o_2, \ldots, o_n\}$ is a set of obstacles in the environment and we ensure that each obstacle o_i does not have a high polygon count.

We update the vertices of the robot based on the deformation and compute a new bounding OBB for each triangle on its

boundary. The set-based collision proceeds in two passes.

In the first pass, we compute $\mathcal{R}$-PCS and $\mathcal{O}$-PCS. In particular, $r_i \in \mathcal{R}$-PCS, if r_i does not overlap with all the obstacles in $\mathcal{O}$-set. Similarly, $o_i \in \mathcal{O}$-PCS, if o_i does not overlap with all the r_i's in $\mathcal{R}$-set. The $\mathcal{R}$-PCS is computed by performing 2.5D overlap test between each $\{r_i\}$ and $\mathcal{O}$-set. Similarly, $\mathcal{O}$-PCS is computed by performing 2.5D overlap tests between each $\{o_i\}$ and $\mathcal{R}$-set.

In the second pass, we perform set-based 2.5D overlap tests in a recursive manner. We represent $\mathcal{R}$-PCS = $\{\mathcal{R}_1, \mathcal{R}_2\}$, where $\mathcal{R}_1$ and $\mathcal{R}_2$ have approximately the same number of elements. Similarly we decompose $\mathcal{O}$-PCS = $\{\mathcal{O}_1, \mathcal{O}_2\}$. We perform 2.5D overlap tests between the following set combinations: ($\mathcal{R}_1$, $\mathcal{O}_1$), ($\mathcal{R}_1$, $\mathcal{O}_2$), ($\mathcal{R}_2$, $\mathcal{O}_1$) and ($\mathcal{R}_2$,$\mathcal{O}_2$). If none or only one of the 2.5D overlap test results in a separating surface, we terminate the recursion and perform exact collision checking between $\mathcal{R}$-PCS and $\mathcal{O}$-PCS. Otherwise, we remove either $\mathcal{R}_1$ or $\mathcal{R}_2$ from $\mathcal{R}$-PCS or remove $\mathcal{O}_1$ or $\mathcal{O}_2$ from $\mathcal{O}$-PCS. The set-based culling algorithm is applied recursively to the new PCS's.

Analysis: The running time of our set-based culling algorithm is bounded by $O(m \log n + n \log m)$. We assume that the cost of performing each 2.5D overlap test is constant. The first pass of the algorithm takes linear time. In the second pass, the algorithm takes linear time during the first iteration. During each successive iteration we reduce the number of objects in one of the PCS's by half and therefore, performing $O(m \log n + n \log m)$ 2.5D overlap tests in the worst case.

C. Exact Collision Detection

Given the potentially colliding sets, $\mathcal{R}$-PCS and $\mathcal{O}$-PCS, we perform exact tests between the primitives to check for collisions. If the number of primitives is small, we check all pairwise combinations. Otherwise, we compute a bounding box for each primitive of $\mathcal{R}$-PCS and $\mathcal{O}$-PCS. We perform pairwise overlap tests between the bounding boxes by projecting the bounding box along the X, Y and Z-axes and compute the overlapping intervals using insertion sort. If the projections of any bounding box pair overlaps along any axis, we explicitly check whether the corresponding 3D bounding boxes overlap and perform exact intersection tests between the primitives.

VI. Application and Results

We have implemented the algorithms described in this paper and tested them on a PC with a 2.8 GHz Pentium IV processor, 1 GB of main memory, and a NVidia GeForce FX 6800 card. We used of NVidia's occlusion query extension along with offsets to perform visibility queries for reliable 2.5D overlap tests.

A. Benchmark and Performance

In order to test the effectiveness of our algorithm, we have used it for two scenarios:

- **Serial Walls.** This scenario is based on a Parasol Benchmark [12] in which a stick-like robot must navigate through a series of walls with holes (shown in Fig. 3). We have extended the benchmark by changing the robot into a soft-body sphere with 1280 polygons and the walls are represented using 18K polygons. The sphere's diameter is set to be larger than the holes, but small enough so that the robot's material constraints allow it to fit through, forcing it to deform to reach its goal configuration. It takes about 16 minutes to compute a collision-free path.
- **Liver Chemoembolization.** This scenario demonstrates the ability of our planner to work in a realistic complex environment. We attempt to plan the path of a tube-like cylinder, called a catheter, through a set of arteries in order to mimic the catheterization process in liver chemoembolization. More details are given below.

Scenario	Robot Complexity (tris)	Obstacle Complexity (tris)	Constraint update (s)	2.5D Overlap Test (s)	Exact Triangle Intersection Test (s)	Solve System (s)	Total Time (s)
Walls	1280	18432	0.0071	0.0232	0.0252	0.0323	0.0878
Catheter	10080	80086	0.0159	0.1596	0.0062	0.0227	0.204

Fig. 4. This table gives a breakdown of the average time step for each scenario. Constraint update refers to the time spent in computing each constraint for the given configuration. The 2.5D overlap test along with the exact triangle intersection test are the two stages of the collision detection algorithm. The Solve-System time is that spent in solving the motion equations during each step.

We highlight the performance of our algorithm in these environments in the table and graph. Fig. 4 shows a detailed breakdown of the average time spent in various stages of a simulation step. Constraint update is the time required to process the list of constraints. The collision detection and response phase consists of both the 2.5D overlap tests along with the exact triangle-triangle intersection tests. Finally, we highlight the time spent in solving the motion equations and the total time spent on a time step. One curious result is that more time was spent on exact test in the Walls scenario even though it is a simpler environment. This result is due to greater culling effectiveness in the catheterization case.

Fig. 5 compares the effectiveness of the 2.5D overlap tests as a function of the scene complexity. To measure the performance of our new collision detection algorithm, we varied the scene complexity of the Walls environment and ran our algorithm with and without the GPU-based 2.5D overlap tests. The case of no overlap test reduces to solely using the bounding volume hierarchies. In the graph, we see a greater speedup in collision detection as the obstacle complexity increases.

B. Path Planning of Catheters in Liver Chemoembolization

We use our path planning algorithm as a guidance tool for a catheterization procedure, specifically chemoembolization of liver tumors. Liver chemoembolization involves the injection of chemotherapy drugs directly into the hepatic artery that supplies a tumor. The procedure takes advantage of the fact that liver tumors obtain their blood supply exclusively from the branches of the hepatic artery. Under X-ray guidance, a small tube or *catheter*, is inserted into the femoral artery and is then advanced into the selected liver artery supplying the

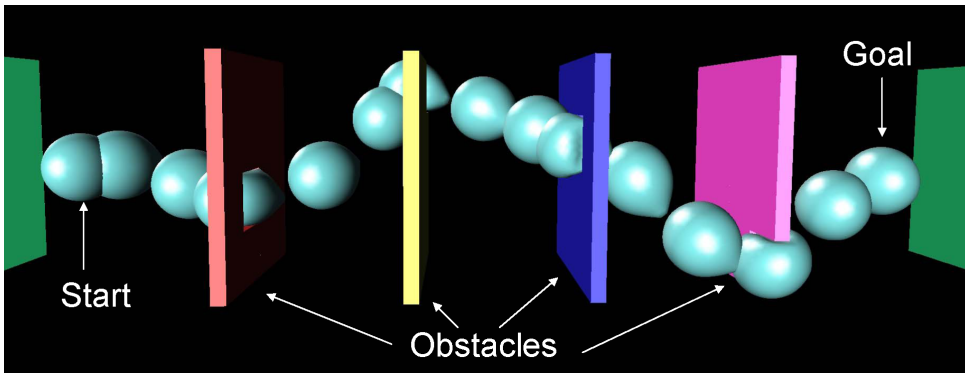

Fig. 3. **Spherical robot through Walls:** This scenario consists of a robot (deformable sphere with 1280 polygons) moving through six walls ($18,432$ polygons) with small holes. The robot is larger than the holes and needs to deform to generate a collision-free path the initial configuration to the final configuration.

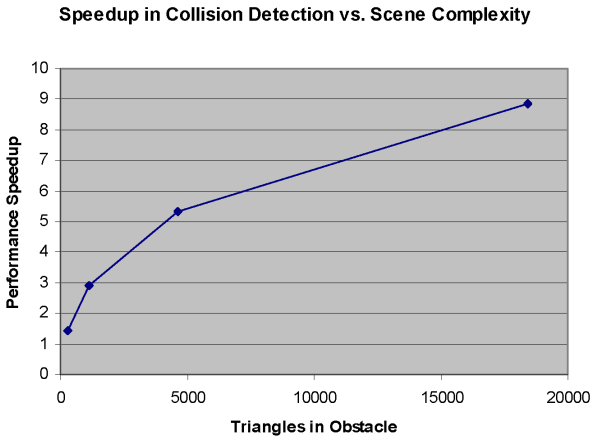

Fig. 5. This graph highlights the speedups obtained by utilizing the 2.5D overlap tests in our collision detection algorithm, as we increase the polygonal complexity of our scene. We observe nearly an order of magnitude improvement in complex scenes over prior algorithms based on bounding volume hierarchies.

tumor. Chemotherapy drugs, followed by embolizing agents, are then injected through the catheter into the liver tumor.

During this procedure, careful manipulation of catheters is essential [8], [29]. Manipulation of catheters in small vessels frequently causes spasms, which prevent adequate flow to carry the chemoembolization material into the tumor. Another problem may arise if the catheter has a cross-sectional area close to that of the vessel being injected. In this case, the size similarity will also reduce fluid flow and increase the risk of reflux of chemoembolization material into other arteries. Accurate planning studies can help to overcome these difficulties. Preoperatively, path planning can be used as part of surgical planning techniques to help choose the size and properties of the catheter used. We used a geometric model of the catheter and arteries shown in Fig. 1 (along with their relative dimensions). The catheter is modeled using $10,080$ triangles. The model of the arteries consist of $70,006$ triangles and the liver is represented using $12,459$ triangles. The start and end configuration of the catheter are shown in the same figure.

We used our motion planning algorithm to compute a path for a flexible catheter to a specific hepatic artery that is supplying a tumor inside the liver. The 3D models of the liver and the blood vessels, that make up the environment, were obtained from the 4D NCAT phantom [25]. The flexible, snake-like, catheter was modeled as a cylinder with a length of 100 cm and a diameter of 1.35 mm. Figure 6 shows a 3D rendering of the models used in this study with the starting (insertion of the catheter) and ending (tumor supplying vessel) locations marked.

Despite the scenario's complexity, our planner was able to successfully plan a path for this problem. A breakdown of the step time averages is given in Fig. 4. As the table shows, a large portion of the computation time is spent done in the collision detection phase (more than 80%). Further optimizations in GPU-based 2.5D overlap tests would improve the performance of the overall planner.

We use a large number of material constraints and are able to generate fairly smooth deformations throughout the simulation (as shown in Fig. 6). An additional path smoothing step further helps to improve the quality of the deformation.

VII. Conclusion and Future Work

We present a new algorithm for computing a collision-free path for a deformable robot in a complex static environment. We generate an initial path for a robot based on the approximate medial axis of the workspace and probabilistic roadmap planner. We present a novel collision detection algorithm to check for overlaps between the deforming robot and the obstacles. We have applied our planner to different configurations, including path planning of catheters in liver chemoembolization. The initial results are very promising.

There are several directions for future research. We plan to develop more physically accurate algorithms based on FEM and combine them with multiresolution techniques to accelerate overall performance. We would also like to handle scenarios where the obstacles are not rigid or stationary and can deform as well, e.g. guiding flexible tubes among deformable organs. We also plan to validate the results of our planning algorithms for catheterization procedure on clinical trials.

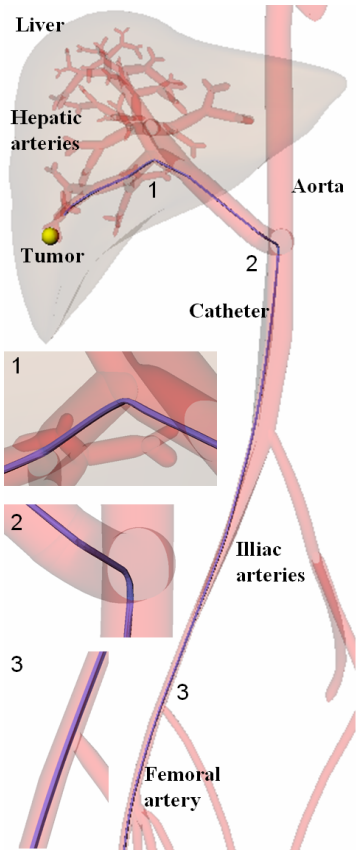

Fig. 6. **Path Planning of Catherers in Liver Chemoembolization:** We highlight the collision-free computed by our algorithm for the catheter shown in Fig. 1. We show the overall path from the start to the end configuration in the rightmost image. The left images highlight the zoomed portions of the path, showing bends and deformations.

We would also like to explore new applications of our planners in virtual prototyping, engineering design, and other applications. Our collision detection algorithm only checks for collisions between the robot and the obstacles and we would like to handle self-collisions in the future.

VIII. Acknowledgment

We wish to thank the many anonymous reviewers for their useful feedback which has helped to improve this work. This work was also supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0118743, 0429583, and 0404088, ONR Contract N00014-01–0496, DARPA/RDECOM Contract N61339-04-C-0043, Intel Corporation, and the DOE HPCS Fellowship.

References

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

[2] G. Baciu and S. Wong. Image-based techniques in a hybrid collision detector. *IEEE Trans. on Visualization and Computer Graphics*, 2002.

[3] O. B. Bayazit, H. Lien, and N. Amato. Probabilistic roadmap motion planning for deformable objects. *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2002.

[4] H. Choset and J. Burdick. Sensor based planning: The hierarchical generalized voronoi graph. *Workshop on Algorithmic Foundations of Robotics*, 1996.

[5] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2004.

[6] M. Foskey, M. Garber, M. Lin, and D. Manocha. A voronoi-based hybrid planner. *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001.

[7] M. Garber and M. Lin. Constraint-based motion planning using voronoi diagrams. *Proc. Fifth International Workshop on Algorithmic Foundations of Robotics*, 2002.

[8] J. Gates, G. Hartnell, K. Stuart, and M. Clouse. Chemoembolization of hepatic neoplasms: Safety, complications, and when to worry. *Radiographics*, 19:399–414, 1996.

[9] R. Gayle, M. Lin, and D. Manocha. Constraint based motion planning of deformable robots. *IEEE Conf. on Robotics and Automation*, 2005.

[10] N. Govindaraju, D. Knott, N. Jain, I. Kabal, R. Tamstorf, R. Gayle, M. Lin, and D. Manocha. Collision detection between deformable models using chromatic decomposition. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)*, 2005.

[11] N. Govindaraju, S. Redon, M. Lin, and D. Manocha. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 25–32, 2003.

[12] P. R. Group. Motion planning benchmarks. http://parasol.tamu.edu/groups/amatogroup/benchmarks/mp, 2003.

[13] L. Guibas, C. Holleman, and L. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proc. of IROS*, 1999.

[14] B. Heidelberger, M. Teschner, and M. Gross. Real-time volumetric intersections of deforming objects. *Proc. of Vision, Modeling and Visualization*, pages 461–468, 2003.

[15] C. Holleman, L. Kavraki, and J. Warren. Planning paths for a flexible surface patch. *IEEE Int. Conf. Robot. Autom. (ICRA)*, 1998.

[16] D. L. James and D. K. Pai. BD-Tree: Output-sensitive collision detection for reduced deformable models. *Proc. of ACM SIGGRAPH*, pages 393–398, 2004.

[17] D. Knott and D. K. Pai. CInDeR: Collision and interference detection in real-time using graphics hardware. *Proc. of Graphics Interface*, pages 73–80, 2003.

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

[19] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, University of North Carolina, 1999.

[20] T. Larsson and T. Akenine-Möller. Collision detection for continuously deforming bodies. *Eurographics*, pages 325–333, 2001.

[21] J. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, pages 1119–1128, 1999.

[22] M. Lin and D. Manocha. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*, 2003.

[23] H. Nakagaki and K. Kitagaki. Study of deformation tasks of a flexible wire. *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1997.

[24] W. Ngugen and J. Mills. Multi-robot control for flexible fixtureless assembly of flexible sheet metal auto parts. *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 2340–2345, 1996.

[25] W. P. Segars. *Development of a new dynamic NURBS-based cardiac-torso (NCAT) Phantom*. PhD thesis, University of North Carolina at Chapel Hill, 2001.

[26] D. Sun, X. Shi, and Y. Liu. Modeling and cooperation of two-arm robotic system manipulating a deformable object. *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 2346–2351, 1996.

[27] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 19(1):61–81, 2005.

[28] G. van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[29] T. van Walsum and M. Viergever. Deformable b-splines for catheter simulation. Technical report, Image Science Institute, Utrecht University, 1998.

[30] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. *IEEE Conference on Robotics and Automation*, pages 1024–1031, 1999.

[31] A. Witkin and D. Baraff. *Physically Based Modeling: Principles and Practice*. ACM Press, 1997. Course Notes of ACM SIGGRAPH.