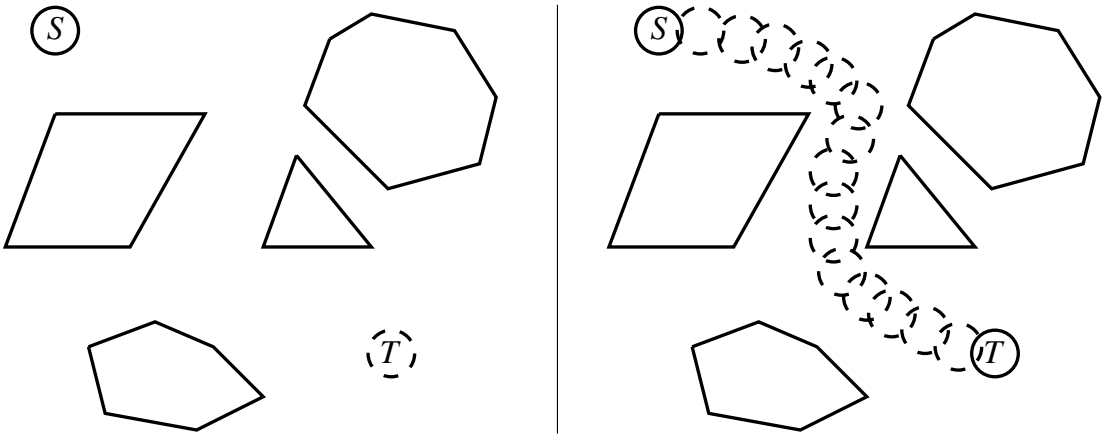

INPUT OUTPUT

17.14 Motion Planning

Input description: A polygonal-shaped robot starting in a given position s in a room containing polygonal obstacles, and a goal position t.

Problem description: Find the shortest route taking s to t without intersecting any obstacles.

Discussion: That motion planning is a complex problem is obvious to anyone who has tried to move a large piece of furniture into a small apartment. It also arises in systems for molecular docking. Many drugs are small molecules that act by binding to a given target model. Identifying which binding sites are accessible to a candidate drug is clearly an instance of motion planning. Plotting paths for mobile robots is another canonical motion-planning application.

Finally, motion planning provides a tool for computer animation. Given the set of object models and where they appear in scenes s_1 and s_2, a motion planning algorithm can construct a short sequence of intermediate motions to transform s_1 to s_2. These motions can serve to fill in the intermediate scenes between s_1 and s_2, with such scene interpolation greatly reducing the workload on the animator.

Many factors govern the complexity of motion planning problems:

- *Is your robot a point?* – For point robots, motion planning becomes finding the shortest path from s to t around the obstacles. This is also known as geometric shortest path. The most readily implementable approach constructs the *visibility graph* of the polygonal obstacles, plus the points s and t. This

visibility graph has a vertex for each obstacle vertex and an edge between two obstacle vertices iff they "see" each other without being blocked by some obstacle edge.

The visibility graph can be constructed by testing each of the $\binom{n}{2}$ vertex-pair edge candidates for intersection against each of the n obstacle edges, although faster algorithms are known. Assign each edge of this visibility graph with weight equal to its length. Then the shortest path from s to t can be found using Dijkstra's shortest-path algorithm (see Section 15.4 (page 489)) in time bounded by the time required to construct the visibility graph.

- *What motions can your robot perform?* – Motion planning becomes considerably more difficult when the robot becomes a polygon instead of a point. Now all of the corridors that we use must be wide enough to permit the robot to pass through.

  The algorithmic complexity depends upon the number of *degrees of freedom* that the robot can use to move. Is it free to rotate as well as to translate? Does the robot have links that are free to bend or to rotate independently, as in an arm with a hand? Each degree of freedom corresponds to a dimension in the search space of possible configurations. Additional freedom makes it more likely that a short path exists from start to goal, although it also becomes harder to find this path.

- *Can you simplify the shape of your robot?* – Motion planning algorithms tend to be complex and time-consuming. Anything you can do to simplify your environment is a win. In particular, consider replacing your robot in an enclosing disk. If there is a start-to-goal path for this disk, it defines such a path for the robot inside of it. Furthermore, any orientation of a disk is equivalent to any other orientation, so rotation provides no help in finding a path. All movements can thus be limited to the simpler case of translation.

- *Are motions limited to translation only?* – When rotation is not allowed, the *expanded obstacles* approach can be used to reduce the problem of polygonal motion planning to the previously-resolved case of a point robot. Pick a reference point on the robot, and replace each obstacle by its Minkowski sum with the robot polygon (see Section 17.16 (page 617)). This creates a larger, fatter obstacle, defined by the shadow traced as the robot walks a loop around the object while maintaining contact with it. Finding a path from the initial reference position to the goal amidst these fattened obstacles defines a legal path for the polygonal robot in the original environment.

- *Are the obstacles known in advance?* – We have assumed that the robot starts out with a map of its environment. But this can't be true, say, in applications where the obstacles move. There are two approaches to solving motion-planning problems without a map. The first approach explores the

environment, building a map of what has been seen, and then uses this map to plan a path to the goal. A simpler strategy proceeds like a sightless man with a compass. Walk in the direction towards the goal until progress is blocked by an obstacle, and then trace a path along the obstacle until the robot is again free to proceed directly towards the goal. Unfortunately, this will fail in environments of sufficient complexity.

The most practical approach to general motion planning involves randomly sampling the *configuration space* of the robot. The configuration space defines the set of legal positions for the robot using one dimension for each degree of freedom. A planar robot capable of translation and rotation has three degrees of freedom, namely the x- and y-coordinates of a reference point on the robot and the angle θ relative to this point. Certain points in this space represent legal positions, while others intersect obstacles.

Construct a set of legal configuration-space points by random sampling. For each pair of points p_1 and p_2, decide whether there exists a direct, nonintersecting path between them. This defines a graph with vertices for each legal point and edges for each such traversable pair. Motion planning now reduces to finding a direct path from the initial/final position to some vertex in the graph, and then solving a shortest-path problem between the two vertices.

There are many ways to enhance this basic technique, such as adding additional vertices to regions of particular interest. Building such a road map provides a nice, clean approach to solving problems that would otherwise get very messy.

Implementations: The *Motion Planning Toolkit* (MPK) is a C++ library and toolkit for developing single- and multi-robot motion planners. It includes SBL, a fast single-query probabilistic roadmap path planner, and is available at *http://robotics.stanford.edu/∼mitul/mpk/*.

The University of North Carolina GAMMA group has produced several efficient collision detection libraries (not really motion planning) of which *SWIFT++* [EL01] is the most recent member of this family. It can detect intersection, compute approximate/exact distances between objects, and determine object-pair contacts in scenes composed of rigid polyhedral models. See *http://www.cs.unc.edu/∼geom/collide/* for pointers to all of these libraries.

The computational geometry library CGAL (*www.cgal.org*) contains many algorithms related to motion planning including visibility graph construction and Minkowski sums. O'Rourke [O'R01] gives a toy implementation of an algorithm to plot motion for a two-jointed robot arm in the plane. See Section 19.1.10 (page 662).

Notes: Latombe's book [Lat91] describes practical approaches to motion planning, including the random sampling method described above. Two other worthy books on motion planning are available freely on line, by LaValle [LaV06] (*http://planning.cs.uiuc.edu/*) and Laumond [Lau98] (*http://www.laas.fr/∼jpl/book.html*).

Motion planning was originally studied by Schwartz and Sharir as the "piano mover's problem." Their solution constructs the complete free space of robot positions that do not intersect obstacles, and then finds the shortest path within the proper connected component. These free space descriptions are very complicated, involving arrangements of higher-degree algebraic surfaces. The fundamental papers on the piano mover's problem appear in [HSS87], with [Sha04] a survey of current results.

The best general result for this free-space approach to motion planning is due to Canny [Can87], who showed that any problem with d degrees of freedom can be solved in $O(n^d \lg n)$, although faster algorithms exist for special cases of the general motion-planning problem. The expanded obstacle approach to motion planning is due to Lozano-Perez and Wesley [LPW79]. The heuristic, sightless man's approach to motion planning discussed previously has been studied by Lumelski [LS87].

The time complexity of algorithms based on the free-space approach to motion planning depends intimately on the combinatorial complexity of the arrangement of surfaces defining the free space. Algorithms for maintaining arrangements are presented in Section 17.15 (page 614). Davenport-Schinzel sequences often arise in the analysis of such arrangements. Sharir and Agarwal [SA95] provide a comprehensive treatment of Davenport-Schinzel sequences and their relevance to motion planning.

The visibility graph of n line segments with E pairs of visible vertices can be constructed in $O(n \lg n + E)$ time [GM91, PV96], which is optimal. Hershberger and Suri [HS99] have an $O(n \lg n)$ algorithm for finding shortest paths for point-robots with polygonal obstacles. Chew [Che85] provides an $O(n^2 \lg n)$ for finding shortest paths for a disk-robot in such a scene.

Related Problems: Shortest path (see page 489), Minkowski sum (see page 617).