Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware

Jed Lengyel*
Mark Reichert*
Bruce Donald**
Donald Greenberg***

TR 90-1122
May 1990

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Graduate Student, Program of Computer Graphics, Cornell University, Ithaca, NY 14853.
**Assistant and Director, Computer Science Robotics Laboratory, Department of Computer Science, 4130 Upson Hall, Cornell University, Ithaca, NY 14853.
***Director, Program of Computer Graphics, Jacob Gould Shurman Professor of Computer Graphics, Cornell University, Ithaca, NY 14853.

Proc. SIGGRAPH '90
Dallas, TX (August 5)

Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware

Jed Lengyel*
Mark Reichert*
Bruce R. Donald†
Donald P. Greenberg‡

Abstract

We present a real-time robot motion planner that is fast and complete to a resolution. The technique is guaranteed to find a path if one exists at the resolution, and all paths returned are safe. The planner can handle *any* polyhedral geometry of robot and obstacles, including disjoint and highly concave unions of polyhedra.

The planner uses standard graphics hardware to rasterize configuration space obstacles into a series of bitmap slices, and then uses dynamic programming to create a navigation function (a discrete vector-valued function) and to calculate paths in this rasterized space. The motion paths which the planner produces are minimal with respect to an L_1 (Manhattan) distance metric that includes rotation as well as translation.

Several examples are shown illustrating the competence of the planner at generating planar rotational and translational plans for complex two and three dimensional robots. Dynamic motion sequences, including complicated and non-obvious backtracking solutions, can be executed in real time.

1 Introduction

Motion planning has been regarded as a core algorithmic problem in computational robotics for many years, and many researchers have worked on finding better algorithmic solutions.[Yap85] However, despite the fact that all of the key elements in planning robot motion substantially overlap with computer graphics interests, the problem has not been presented as a computer graphics topic.

The classical formulation of the *Find-Path* or *Piano Mover's* problem is stated as follows: given an arbitrary rigid polyhedral object, P, and polyhedral environment, find a continuous collision-free path taking P from some initial configuration to a desired goal configuration.

*Graduate Student, Program of Computer Graphics, Cornell University, Ithaca, N.Y. 14853

†Assistant Professor and Director, Computer Science Robotics Laboratory, Department of Computer Science, 4130 Upson Hall, Cornell University, Ithaca, N.Y. 14853

‡Director, Program of Computer Graphics, Jacob Gould Shurman Professor of Computer Graphics, Cornell University, Ithaca, N.Y. 14853

We view this motion planning process as an algorithmic endeavor analogous to hidden surface removal in computer graphics. First, precise combinatorial solutions exist, but rasterized, approximating techniques (such as z-buffer algorithms) are faster and more effective. Second, such approximation algorithms can be provided with massively parallel, specialized hardware support. Instead of concentrating on more efficient, combinatorially exact algorithms, the end-user is more effectively served by (a) choosing good representations for the geometric constraints, (b) selecting local, isotropic geometric algorithms that are easily parallelizable, and (c) providing or using appropriate hardware support to make the algorithms run very fast.

Our algorithm is based on the configuration space representations that are due to Lozano-Pérez [LPW79], and we use a local, isotropic search algorithm [Don87][Don84] to obtain a very fast motion planning algorithm that runs on standard graphics hardware. Two parts of the local algorithm we present here are very similar to the work of [DT88]. The paths produced have minimal length with respect to an L_1 distance metric imposed on the rasterization which treats translational and rotational movements equally.

In addition to being simple, the algorithm is complete to a resolution (of the rasterization), and while inherently "local", does not suffer from local minima. Many other fast motion planning algorithms, in particular, potential field methods, get "stuck" in local minima and therefore cannot be effectively used to plan paths for complex, concave, or disconnected robots. While these other algorithms run well when the robot is (a) small and convex relative to the environment, or (b) when the space of solutions is "very dense", our planner may be more effective in more complicated cases.

For computer graphics applications, the visual impact of the robot motion is important. Since the motions are time-dependent and complex, real-time graphical playback is desirable to assess the motion sequence, including the complicated back-tracking paths which may be necessary to obtain a solution.

In Section 2 a brief overview of the historical approaches used by the robotics community is presented. Section 3 describes our algorithm in detail. Section 4 presents the results of the implementation for several specific cases. Section 5 presents the conclusions and topics for future research.

2 Background

The *Piano Mover's* problem has been approached in many ways. We present below a brief overview of the two major approaches used by the robotics community, the configuration space and potential field methods, and discuss the advantages and limitations of each method.

2.1 Configuration Space Methods

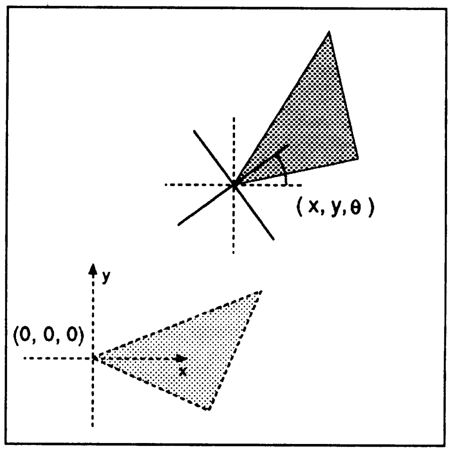

Figure 1: Planar Object.

Consider an object moving in the plane. The object's configuration can be uniquely described by its position in x and y and its orientation in θ. The *configuration space* for such an object is the set of all such possible configurations (x, y, θ) where $(x, y) \in R^2$ (the real plane) and $\theta \in S^1$ (the unit circle). The configuration space for an object moving in the plane is thus $R^2 \times S^1$ (Figure 1). (In robotics terminology, *configuration space* is often abbreviated to *c-space.*)

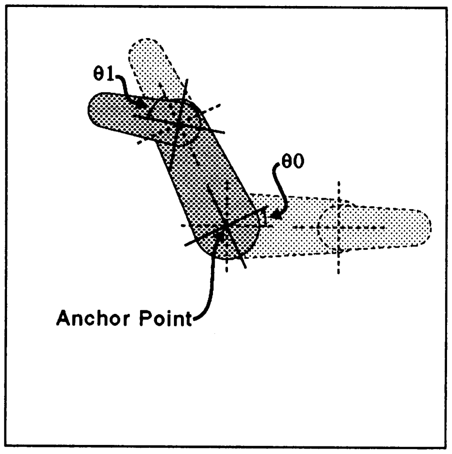

Figure 2: Two-link Arm.

As another example of a configuration space, consider a robot with a two-link arm in the plane anchored at a point. The angle of the base arm θ_0 and the angle of the free arm θ_1 completely determine the configuration of the robot (Figure 2). The configuration space is $S^1 \times S^1$, and a particular configuration is a point $(\theta_0, \theta_1) \in S^1 \times S^1$.

The problem of moving a complex robot through a physical environment can be transformed to the problem of moving a point through a c-space environment. The obstacle constraints in real space are encoded by "enlarged" obstacles in configuration space. If the reference point of the robot is outside of the enlarged c-space obstacles, then the robot itself is outside of the obstacles in the physical workspace. Lozano-Pérez generates the c-space obstacles using a *Minkowski sum* of the robot and the environment. In a Minkowski sum all the points of one set are offset by the points of another set (Figures 3 and 4). For sets represented by polygons, the Minkowski sum can be thought of as the "convolution" (set sum) of the obstacle polygons by the "negated" robot polygons. This "convolution" can be computed using a convex hull algorithm. This algorithm is suboptimal in the plane[1], and in the case where the generating polygons are convex, Lozano-Pérez gives a method for computing the Minkowski sum in linear ($O(n)$) time. The important ideas of *c-space slices* and *c-space slice projections*, which will be described in more detail later, were also introduced. For a more thorough introduction, we refer the reader to previous papers [Loz80] [LPW79] [Udu77].

Other researchers have used rasterizing techniques in c-space. For example, Lozano-Pérez [Loz87] used a representation of the obstacles derived in [Loz83],[Don87] and encoded them in a bitmap of the c-space. The search algorithm used was not local, isotropic, nor trivially parallelizable, and no hardware support was available.

Dorst and Trovato used a rasterized c-space approach to plan for a two-link arm[DT88]. They described the motion-planning problem in a differential geometry framework, with a metric topology imposed on configuration space and geodesics corresponding to optimal paths. They discretized the c-space and used cost wave propagation and gradient-following to find the optimal paths. The second two parts of the algorithm we present are very similar to their work. Dorst and Trovato do not, however, address the problem of generating the c-space obstacles (the first step in our algorithm.) We also emphasize the hardware context of the algorithm.

Donald described a motion planning algorithm that is a *provably-good approximation* algorithm. It is guaranteed to find a path if one exists at the given resolution and all paths it returns are safe. [See [Don87] for a formal definition of provably-good approximation algorithms.] He also used constraint equations to represent the c-space obstacles, imposed a grid on the c-space, and then used several local "experts" to guide the search through the c-space. This algorithm had the advantage of being complete to a resolution, but a software implementation ran slowly on a sequential machine[2]. We show how a variant of one part of this algorithm runs very fast on modern graphics hardware.

2.2 Potential Field Methods

Potential field methods were first pioneered by Khatib and Le Maitre [KLM78]. The obstacles were represented as zero level surfaces of scalar valued analytic functions, *i.e.* $f(x, y, z) = 0$. A potential field local to each obstacle, whose strength diminishes with the square of the distance from the obstacle, was generated. An arbitrary cutoff value, f_0, was assigned which corresponds to the distance at which the influence of the obstacle is no longer important (Figure 5). The potential field is mathematically described in the following form:

$$P(x, y, z) = \begin{cases} \alpha / f(x, y, z)^2 & f(x, y, z) \leq f_0 \\ 0 & f(x, y, z) > f_0 \end{cases}$$

A particle moving in accordance to Newton's laws in such a potential field will never hit the obstacle. Khatib observed that the sum of the gradients is the gradient of the sum; thus adding up the potential fields for many obstacles results in a single function under whose influence the particle cannot hit any obstacle.[Kod89] Since the real object to be maneuvered is not a point mass, Khatib identified a number of "distinguished points" on the object. The potentials of these distinguished points were linearly combined to

[1]Note that in general, calculating a convex hull requires time $O(n \log n)$.

[2]Donald used a CADR (MIT-architecture) Lisp machine

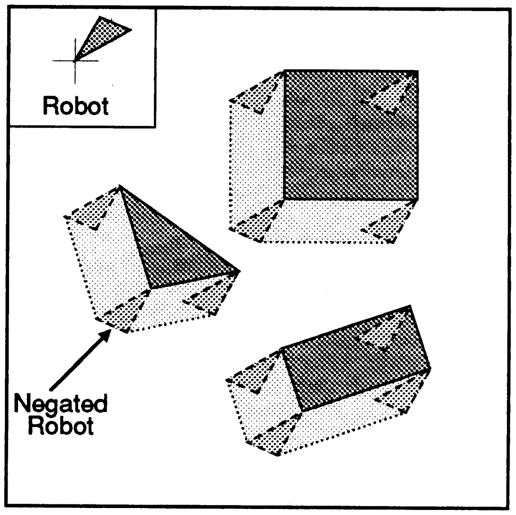

Figure 3: Minkowski Sum of Robot and Obstacles.

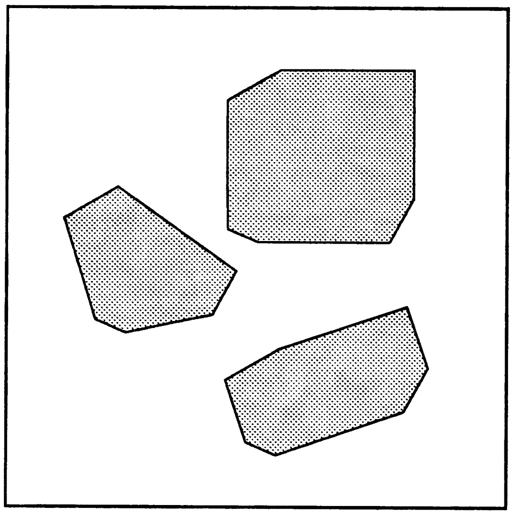

Figure 4: Resultant Polygons from Minkowski Sum.

produce a composite potential on the position and orientation of the rigid body.

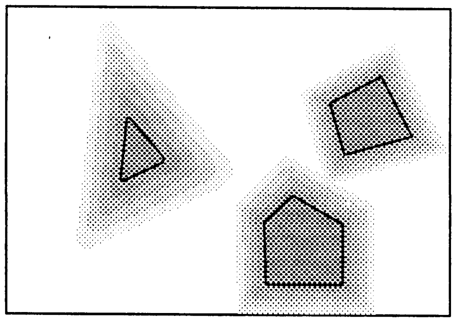

Figure 5: Potential field generated from obstacles. A higher density indicates a higher potential.

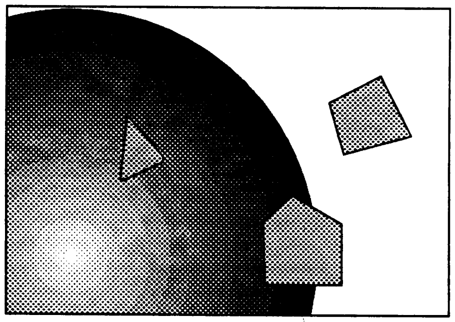

Figure 6: Bitmap potential field generated from goal. A higher density indicates a higher potential.

Potential field techniques have been successful for real-time obstacle avoidance in changing environments, but for motion planning, there are several limitations. One major problem is spurious local minima, especially for concave robots. To escape these local minima, one must resort to randomization techniques or other techniques as described below.

Barraquand and Latombe extended this idea and used bitmaps to represent the obstacles. They generated a separate potential field for a point robot starting from a goal position (Figure 6).[BL89] The potential field is not a function of the euclidean distance to the goal, but is instead a function of the path length to the goal (traveling *around* obstacles, and not "as the crow flies".)

They then used techniques similar to Khatib's to combine point potentials into a single potential field for a more complex body. Although the potential field for a point robot has no local minima, the combined potential field for a more complex robot may have many local minima due to "competition" of the distinguished points. To escape these minima, they described two possible techniques. Both of these techniques search for paths in c-space, with the potential field used as guide for the search.

Their first technique was to "fill" the local minima, which resulted in a planner that is complete to a given resolution. The algorithm we present is similar to this planner, since it also fills c-space and is complete to a resolution.

They suggested that local-minima-filling methods are effective only for low degrees of freedom (due to the memory requirements.)[3] To plan with higher degrees of freedom, they used randomization to get out of the local minima. This planner is capable of planning for robots with a great number of degrees of freedom (as demonstrated with multiple bars linked into chains), but is only probabilistically complete.

3 Algorithm

3.1 Overview

The motion planner presented below is algorithmically based on the grid search method used by Donald [Don87] with the configuration space approach of Lozano-Pérez [Loz80]. It consists of four separate modules.

[3]The memory requirements for the planner we present are very similar to those of Barraquand and Latombe's local-minima-filling technique. However, we are aware through personal communication with Tomás Lozano-Pérez, that a complete-to-a-resolution planner that uses techniques similar to the ones we present is being developed on a Connection Machine and appears promising for higher DOF cases.

- The first module allocates a voxel array representation of c-space and rapidly computes c-space obstacles in this array using standard graphics rasterization hardware.
- The second module calculates a c-space navigation function [Kod87] with a dynamic programming technique (expanding wavefront of solutions.) This navigation function is essentially a discrete vector-valued function which, when given the robot's current voxel location in c-space, returns the direction that the robot should move to decrease its distance to the goal.
- The third module determines the shortest path from any voxel start position to the goal if there is a viable solution. Since the navigation function gives the direction to move at every cell, this module can calculate the path quickly, and determine in constant time if a path exists.
- The fourth module produces a real-time kinematic simulation of the robot motion.

Each of these modules is described in more detail below.

3.2 Definitions

The following definitions due to [Loz80] are useful prior to the algorithmic explanations.

A *c-space obstacle*, CO, is a forbidden region of c-space $R^2 \times S^1$.

A *slice*, $CO[\theta_1, \theta_2]$, of CO is CO restricted to an angular interval $[\theta_1, \theta_2]$, *i.e.* $CO \cap (R^2 \times [\theta_1, \theta_2])$.

A *slice projection* is the projection of $CO[\theta_1, \theta_2]$ onto the "plane" $R^2 \times \{\theta_0\}$ for $\theta_0 = (\theta_1 + \theta_2)/2$.

3.3 Generation of Configuration Space Representation

We calculate the configuration space obstacle polygons by taking the Minkowski sum of the obstacles and the rotated and negated robot as described by Lozano-Pérez [Loz80] (Figure 3). We then use graphics polygon-fill hardware to fill the configuration space obstacle polygons (Figure 4).

When the robot motion is restricted to two degrees of freedom (only translation in the plane) our discrete representation of c-space is a single bitmap with rasterized c-space obstacles. However, when rotation is allowed, the configuration space is represented by a set of bitmaps, where each bitmap is a slice projection representing the configuration space for the angular interval $[\theta_1, \theta_2]$.

While generating a representation of c-space at exactly one orientation is trivial, generation of a conservative, discrete, representation of c-space over some angular interval is more difficult. Our goal is to ensure that, in the bitmap slice representing c-space for the angular interval $[\theta_1, \theta_2]$, no cell is labeled as free if it has a c-space obstacle penetrating it at *any* orientation in $[\theta_1, \theta_2]$. To produce a discrete representation of c-space for the angular interval $[\theta_1, \theta_2]$, we generate n discrete representations of c-space at equally spaced orientations within the interval $[\theta_1, \theta_2]$. The angular increment $\theta = (\theta_2 - \theta_1)/n$. The bitmap slice representation for the angular interval $[\theta_1, \theta_2]$ is the union of all of the sub-intervals. Figure 7 is a pseudo-code implementation of this method.

Note that the bitmap slices are conservative. If any part of an obstacle penetrates a cell, then the whole cell is labeled with "obstacle." This discards some potential paths, but enforces the complete-to-a-resolution property of the planner. If two adjacent cells are labeled as being "free" of obstacles, then the path between them is free, since if there were any obstacle in the way, one or both of the cells would have been labeled with "obstacle," which would contradict the assumption of both cells being "free". Since movement is allowed only between "free" cells, every path returned is valid (collision-free.)

```
GENERATE C-SPACE()
 For theta = 0 to $2\pi$ by $2\pi/N$  ( N is # of theta slices)
   Foreach robotPoly in robot polygon list
     For t1 = theta - dtheta to theta + dtheta
       Rotate robotPoly by t1
       Foreach obstaclePoly in environment
         Generate the minkowski sum of
           robotPoly and obstaclePoly
         Fill minkowski polygon with
           obstacle color.
   Read filled polys from frame buffer
   Move bitmap into voxel array.
   Clear frame buffer.
 Return bitmap voxel array
```

Figure 7: Pseudo-code for Generation of Discretized C-Space

3.4 Calculation of Navigation Function

A dynamic programming technique is used to expand a wavefront of solutions from the goal, with a queue to keep track of the current wavefront.[DT88] Each element in the queue holds a position in the rasterized environment and a current length of the path from the goal position. An element is dequeued, its location in the environment is filled with the current length, and then all of its free space neighbors that have not yet been filled are put on the end of the queue with an incremented distance.

This algorithm is essentially isomorphic to the "Bumble Strategy" in Donald's 1984 algorithm [Don84] [Don87], which operated as follows: a search node N on a c-space grid is dequeued, and its c-space grid neighbors are generated. The reachable, unexplored, free-space neighbors are put on the end of the queue. Each new neighbor M contains a backpointer to N, and N's direction from M. This back pointer (and back-direction) corresponds precisely to our navigation function. Note the Bumble Strategy is simply a breadth-first-search (BFS) from the start (or goal.) We regard dynamic programming and BFS as "dual" algorithms in that BFS from the goal yields our dynamic programming algorithm.

```
FILL NAVIGATION FUNCTION(goal location)
 Enqueue goal location with distance = 0
 While queue is not empty
   Dequeue element F
   Label F's location with F's distance
   Enqueue all neighbors of F that are not
     obstacles and that have not yet been
     filled with distance = distance + 1
```

Figure 8: Pseudo-code for Navigation Function Fill

The initial element placed in the queue is the goal node with a path length of zero. The algorithm takes that node out of the queue, fills its location in the rasterized environment with zero, then queues up all of the neighboring nodes with a path length of one. The wavefront continues out like a "brush fire", spreading around the C-space obstacles, in a way similar to flood-fill or seed-fill algorithms used in computer paint programs[Pav81]. Each cell that can be reached from the goal is set just once (Figures 9 and 8.)

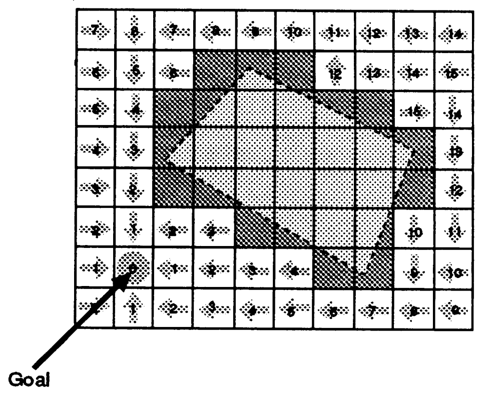

Figure 9: Calculation of Navigation Function.

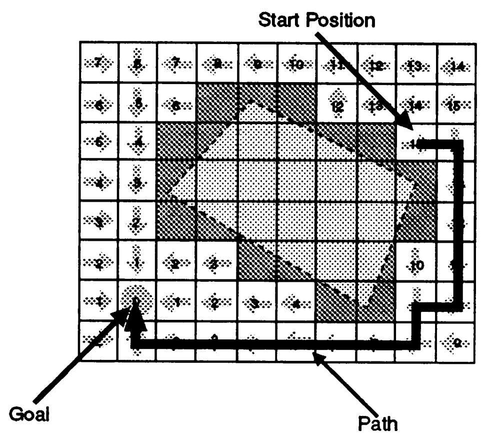

Figure 10: Path Generation.

Note that with the three-degree of freedom case (two translations plus rotation) that the fill expands upward and wraps-around in the theta direction as well as expanding in the x and y directions.

The running time is proportional to the number of free cells in the environment. Complicated scenes which consist of many obstacles have fewer free cells. Thus, paradoxically, more complicated scenes are quicker to fill![4]

3.5 Path Generation

Since the navigation function is generated by a breadth-first search through the rasterized configuration space, it yields the shortest path from the goal to any reachable position in the configuration space (where a "shortest path" is defined to be one passing through a minimal number of voxels.) [See [DT88] for other metrics.] To get closer to the goal from any start position, the robot moves to the lowest-numbered neighboring cell. This corresponds exactly to following the breadth-first search tree to the goal. (We call this process "surfing", since the robot is simply sliding down the "hills" of the potential function.)

If there are multiple cells which have the lowest number, any one can be picked, since each would correspond to a path with the same number of moves needed to reach the goal. Given a choice between a rotation (a move in the theta direction) or a translation (a move in the x or y direction), our algorithm selects the translation option to minimize rotation(Figures 10 and 11.)

By construction, every cell that can be reached from the goal has a neighbor with a lower number than itself (the cell which was queued to label it.) Thus, there are no local minima in the navigation function. The robot can just follow the bread-first search tree to the goal and not worry about getting stuck. The following of the search tree is fast, running in linear time with respect to path length. This path generation technique corresponds precisely to gradient following[DT88][Kod87].

3.6 Display Routines

A program has been implemented which uses standard graphics hardware acceleration to generate dynamic real-time playback of the robot motion, allowing the viewer's position to be interactively modified. Two sequential pre-processing steps are required, the rasterization of c-space and the computation of the navigation function. This discretized configuration space must be recomputed

[4]Also observed by [DT88].

```
GENERATE PATH(start location)
 let C = cell corresponding to start location
 let P = NULL
 if C was not reached by fill (label is blank)
   return CANNOT REACH GOAL
 else
   while C is not at goal
     add C to path P
     pick lowest numbered neighbor, L
     let C = L
   return P
```

Figure 11: Pseudo-code for Path Generation (Surfing)

for every change in the obstacle definitions or the robot geometry. The navigation function is recomputed only if there is a change in the goal position. Since the path generation is so fast, this can be computed on-the-fly, and the dynamic sequences of the robot motion can be displayed.

3.7 Use of Hardware

While our algorithm is the slowest possible on serial machines, it is very fast using parallel or specialized hardware.

Each of the above modules can benefit from use of specialized hardware. In our implementation only the first, the generation of the c-space obstacles, and the fourth, the kinematic simulation, use specialized graphics hardware. The second module, the flood-fill, uses a very local operation and is ideal for a distributed computation[BL89]. The third module, the gradient following or bread-first-search-tree following, is essentially a fast serial operation.

4 Examples

The algorithm was tested in several obstacle environments with robots and obstacles of varying shape and convexity. Experimental timings for each of these examples are presented in Table 1. Note that once the preprocessing steps for the configuration space and navigation function are complete, the path-generation algorithm runs almost instantaneously, and real-time motion display is possible.

Problem	*Figure*	*Size*	*C-space*	*Fill*	*Surf*	*Display*
Moving Planar Robot	13b	256x256x120	22.6	44.1	0.11	35.8
Backtracking Planar Robot	13e	256x256x120	*	*	0.11	50.3
Stuck Planar Robot	13f	256x256x120	*	*	0.01	0
Piano (Room1)	14a	192x192x180	26.0	50.2	0.05	27.1
Piano (Room2)	15a	92x92x90	8.1	8.1	0.04	7.2

Table 1: Table of experimental timings. Times are given in seconds. Processing and display was performed on a Hewlett Packard 835 with a Turbo-SRX graphics display. The precalculations whose times marked as "*" for Figures 13e and 13f were already performed for Figure 13b and did not need to be repeated.

4.1 Complex $R^2 \times S^1$

This example illustrates the motion of a complex, concave planar robot. Note the triangular peg obstacles in the upper left-hand corner (Figure 13b.) Since the search algorithm does not rely on a heuristic distance to the goal, backtracking paths are easily found (Figure 13e.) In this example, the robot is not able to turn around until it backtracks all the way to the right side (where the region is devoid of the pegs preventing the turning on the left side).

If the search backwards from the goal (the navigation function) reached the start point, then a path exists from the start to the goal, otherwise one does not exist at the given resolution. With one array reference, it is known whether or not a path can be found from the start configuration to the goal. In Figure 13f, the robot's "head" is stuck between the peg and the wall, and thus the algorithm returns with the result that no path exists.

4.2 $R^2 \times S^1$ motion for 3D robots

Many three-dimensional robots have pieces which are quite distinct vertically. A piano, for example, has small legs and a large main body. By creating two classes of environmental objects, one class of objects which obstruct the legs, and the other which obstruct the body, the algorithm for planar problems can be extended by object space partitioning ($3\frac{1}{2}$D representation.)

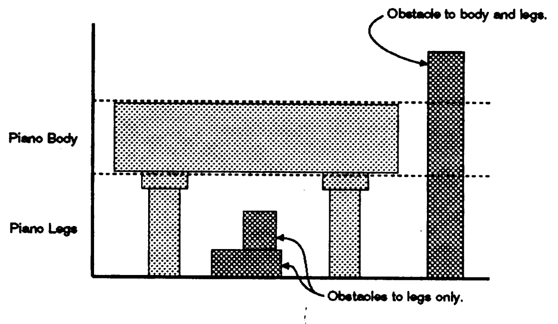

Figure 12: Piano Environment Partitioning.

The rasterization is performed as before, with the exception that the body is now convolved only with the body obstacles, and the legs only with the leg obstacles. The union of the leg and body c-space obstacles is used to create *one* bitmap. Since the leg obstacle is not convolved with the body when generating the c-space obstacles, the piano body can move over it, while the legs maneuver around it (Figure 14a.)

5 Discussion & Conclusion

We have presented a robot motion planner that is fast and, unlike other fast motion planners, is based entirely on complete and provably-good approximation algorithms. The algorithm is based on part of Donald's original algorithm [Don87], which was initially considered to be slow and ineffective for real-time motion planning, but when modified to run on current graphics hardware is actually quite fast.

The planner can handle *any* polyhedral geometry of robot and obstacles, including disjoint and highly concave unions of polyhedra.

The planner is very general and is guaranteed to find a path if one exists at the resolution. In constant ($O(1)$) time, it detects if a path exists from the start location to the goal, or between any two points through the goal (compare [DT88].)

The method is memory intensive, but for many problems the resolutions can be made much lower, especially in the rotational dimension. Storage can also be reduced by eliminating the navigation function value, and only storing a direction at each point, requiring as few as three bits per cell (for the 3DOF case.)

The method is resolution dependent. The higher the resolution, the closer the robot can squeeze by the obstacles. The lower the resolution, the lower the memory requirements.

We believe that we have demonstrated that provably good approximation algorithms for kinematic motion planning can be made to run very fast, if the algorithms are local, isotropic, and can take advantage of special purpose computer graphics hardware for tasks such as rasterization (poly-fill), flood-fill, and gradient-following.

We conjecture that the fastest solutions will involve algorithms similar to ours, that is, characterized by use of

1. Configuration space representations such as [LPW79].
2. Local, geometric, isotropic, parallelizable search algorithms such as [Don87].
3. Appropriate hardware support for geometric computation.

In robotics, and in animation, in addition to kinematic planning and simulation, one also desires to plan robot motions with full dynamics. In recent work, [CDRX88][DX89][DX90] have developed provably good approximation algorithms that generate motions that (1) avoid obstacles, (2) obey dynamic bounds on generalized forces and velocities, (3) respect full Lagrangian rigid body dynamics equations of motion, and (4) are provably close to optimal-time. These algorithms are also grid-based, and currently run very slowly on traditional architectures. We hope that by using techniques similar to those we propose in this paper, that these algorithms can be made to run quickly when modified to exploit appropriate hardware support.

We hope that this paper illustrates the substantial overlap in research areas between the graphics and robotics communities, and fosters collaborative efforts to create more innovative solutions.

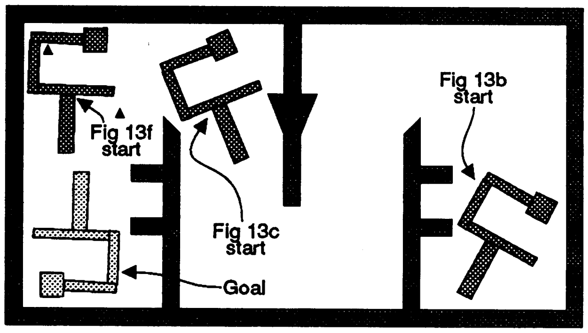

Figure 13a: Key to Robot Positions.

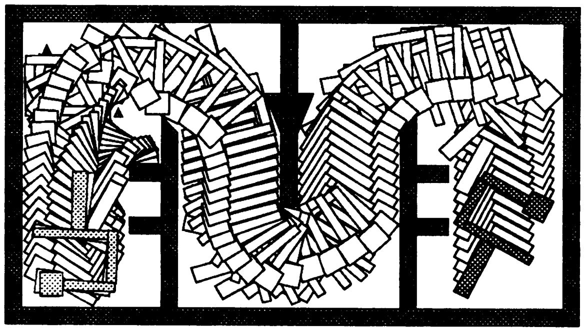

Figure 13b: Moving Planar Robot.

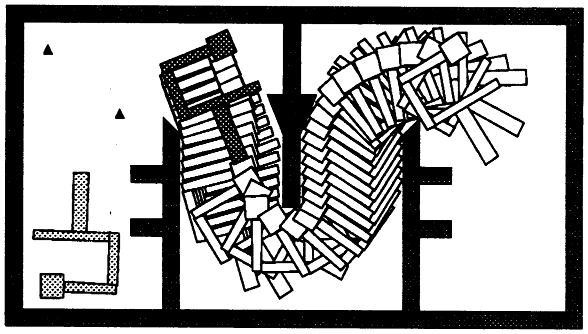

Figure 13c: First Leg of Backtracking.

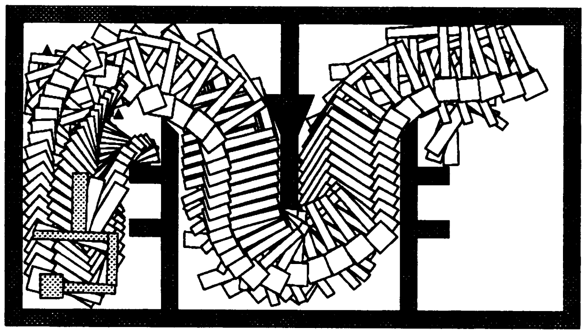

Figure 13d: Last Leg of Backtracking.

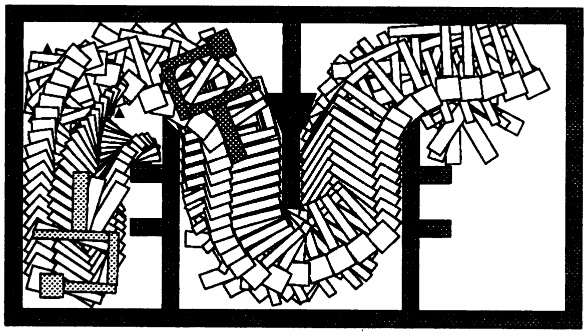

Figure 13e: Backtracking Planar Robot.

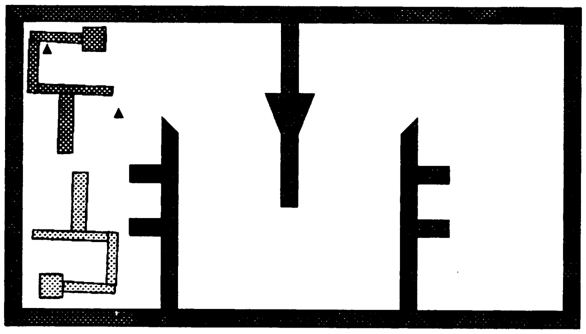

Figure 13f: Trapped Planar Robot.

13a is a key to the start and goal locations of the subsequent figures. In 13b, the robot follows the gradient of the potential function to the goal (which is the same as following the breadth-first search tree.)
In 13c, the robot first moves from the start to the right, since there is not enough room between the pegs on the left to reorient. The robot then has enough room to reorient (the right side is not blocked by the small pegs as on the left.) Finally, in 13d, the robot can squeeze by the small pegs, and obtain the goal position and orientation. 13e shows the entire motion of 13c and 13d. The algorithm is exactly the same as in 13b, simply following the breadth-first search tree. In 13f, no solution exists. Our planner detects this in constant $O(1)$ time.

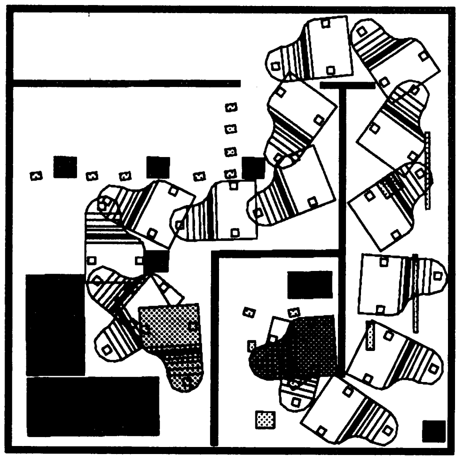

Figure 14a: Moving Piano.

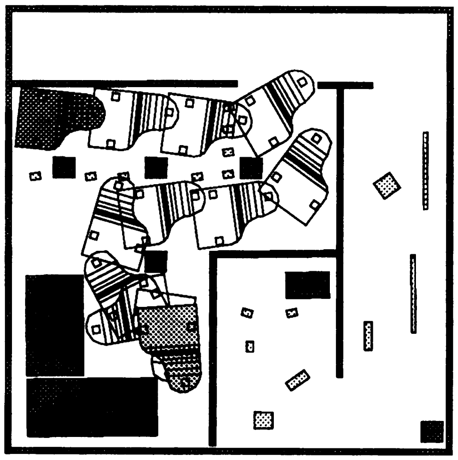

Figure 14b: Moving Piano from Alternate Start.

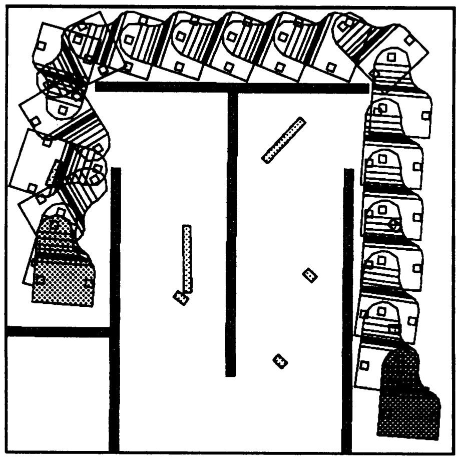

Figure 15a: Piano in Room 2.

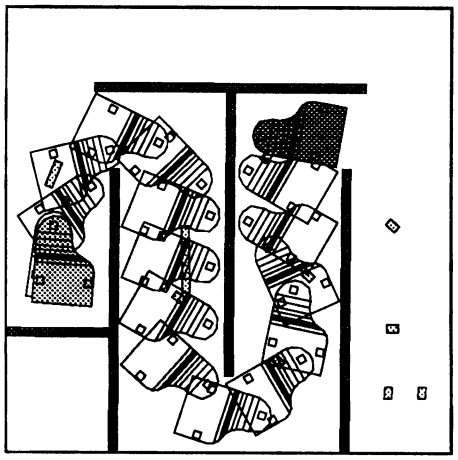

Figure 15b: Piano in Room 2 with Alternate Start.

Figure 16: Moving Piano in 3D.

6 Acknowledgements

The authors gratefully acknowledge the National Science Foundation whose two research grants entitled "Interactive Input and Display Techniques" (DCR8203929) and "Visualization for Scientific Computing" (ACS8715478) help support the Program of Computer Graphics. Support for Bruce Donald's robotics research is provided in part by the National Science Foundation under grant No. IRI-8802390 and by a Presidential Young Investigator award, and in part by the Mathematical Sciences Institute. Jed Lengyel is supported by a National Science Foundation Graduate Fellowship. Our interactive graphics computing environment has been substantially enhanced by generous equipment donations from Hewlett-Packard and Digital Equipment Corporation. Thanks to Robotics CS661, and Bruce's teaching assistants Pat Xavier and Russell Brown, for introducing us to the field of robotics (and for assigning Homework 1, which lead directly to this work.) Thanks to Len Wanger for the piano model, to Emil Ghinger for photographic work, to Tim O'Connor for keeping our computers alive and well, to Fran and Ellen for keeping everything else alive and well, to Jim Ferwerda for breathing life into the video system and to Dave Baraff for taking time to show us how to use it. Thanks to Roy Hall for proof-reading drafts of this paper. Special thanks to Wendy, Michele, Rachael, and Iris for their unending patience while this paper was being written.

References

[BL89] Barraquand, J. and J. Latombe. *Robot Motion Planning: A Distributed Representation Approach*, Report No. STAN-CS-89-1257, Stanford University, Department of Computer Science, May 1989.

[CDRX88] Canny, J., B. Donald, J. Reif, and P. Xavier. "On the Complexity of Kinodynamic Planning," in 29^{th} Symposium on the Foundations of Computer Science, White Plains NY., 1988.

[Don84] Donald, B. R. *Motion Planning with Six Degrees of Freedom*, Report No. MIT AI-TR 791, MIT, Artificial Intelligence Laboratory, 1984.

[Don87] Donald, B. R. "A Search Algorithm for Motion Planning with Six Degrees of Freedom," *Artificial Intelligence*, 31, 1987, pages 295–353.

[DT88] Dorst, L. and K. Trovato. "Optimal path planning by cost wave propagation in metric configuration space," *Proceedings of SPIE-The International Society for Optical Engineering*, 1007, November 1988, pages 186–197.

[DX89] Donald, B. and P. Xavier. "A Provably Good Approximation Algorithm for Optimal-Time Trajectory Planning," in IEEE Int. Conf. On Robotics and Automation, Scottsdale, AZ, 1989.

[DX90] Donald, B. and P. Xavier. "Provably Good Approximation Algorithms for Optimal Kinodynamic Planning for Cartesian Robots and Open Chain Manipulators," in Proceedings of the ACM Symposium on Computational Geometry, Berkeley, CA, 1990.

[KLM78] Khatib, O. and J. Le Maitre. "Dynamic Control of Manipulators Operating in a Complex Environment," *Proceedings Third International CISM-IFToMM Symposium*, September 1978, pages 267–282.

[Kod87] Koditschek, D. E. "Exact Robot Navigation by Means of Potential Functions: Some Topological Considerations," *IEEE International Conference on Robotics and Automation*, March 1987.

[Kod89] Koditschek, D. E. "Planning and Control via Potential Functions," in Lozano-Pérez, T. and O. Khatib, editors, *Robotics Review I*, MIT Press, 1989, pages 349–367.

[Loz80] Lozano-Pérez, T. *Spatial Planning: A Configuration Space Approach*, A.I. Memo No. 605, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, December 1980.

[Loz83] Lozano-Pérez, T. "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, C-32, 1983, pages 108–120.

[Loz87] Lozano-Pérez, T. "A Simple Motion Planning Algorithm for General Robot Manipulator," *IEEE Journal of Robotics and Automation*, RA-3(3), 1987, pages 224–238.

[LPW79] Lozano-Pérez, T. and M. A. Wesley. "An Algorithm for Planning Collison-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, 22, 1979, pages 560–570.

[Pav81] Pavlidis, T. "Contour Filling in Raster Graphics," *Proceedings of SIGGRAPH'81 (Dallas, Texas, August 3–7, 1981)*, 1981, pages 29–36.

[Udu77] Udupa, S. *Collision Detection and Avoidance in Computer Controlled Manipulators*, PhD dissertation, Department of Electrical Engineering, California Institute of Technology, Pasadena, California, 1977.

[Yap85] Yap, C. K. "Algorithmic Motion Planning," in Schwartz, J. T. and C. K. Yap, editors, *Advances in Robotics*, Lawrence Erlbaum Associates, 1985.