NAME

libpathplan – finds and smooths shortest paths

SYNOPSIS

```
#include <graphviz/pathplan.h>

typedef struct Pxy_t {
   double x, y;
} Pxy_t;

typedef struct Pxy_t Ppoint_t;
typedef struct Pxy_t Pvector_t;

typedef struct Ppoly_t {
   Ppoint_t *ps;
   int pn;
} Ppoly_t;

typedef Ppoly_t Ppolyline_t;

typedef struct Pedge_t {
   Ppoint_t a, b;
} Pedge_t;

typedef struct vconfig_s vconfig_t;

#define POLYID_NONE
#define POLYID_UNKNOWN

int Pshortestpath(Ppoly_t *boundary, Ppoint_t endpoints[2], Ppolyline_t *output_route);

vconfig_t *Pobsopen(Ppoly_t **obstacles, int n_obstacles);
int Pobspath(vconfig_t *config, Ppoint_t p0, int poly0, Ppoint_t p1, int poly1, Ppolyline_t *output_route);
void Pobsclose(vconfig_t *config);

int Proutespline (Pedge_t *barriers, int n_barriers, Ppolyline_t input_route, Pvector_t endpoint_slopes[2],
                  Ppolyline_t *output_route);

int Ppolybarriers(Ppoly_t **polys, int n_polys, Pedge_t **barriers, int *n_barriers);
```

DESCRIPTION

libpathplan provides functions for creating spline paths in the plane that are constrained by a polygonal boundary or obstacles to avoid. All polygons must be simple, but need not be convex.

int Pshortestpath(Ppoly_t *boundary, Ppoint_t endpoints[2], Ppolyline_t *output_route);

The function *Pshortestpath* finds a shortest path between two points in a simple polygon. The polygon is specified by a list of its vertices, in either clockwise or counterclockwise order. You pass endpoints interior to the polygon boundary. A shortest path connecting the points that remains in the polygon is returned in *output_route*. If either endpoint does not lie in the polygon, -1 is returned; otherwise, 0 is returned on success. The array of points in *output_route* is static to the library. It should not be freed, and should be used before another call to *Pshortestpath*.

vconfig_t *Pobsopen(Ppoly_t **obstacles, int n_obstacles);
Pobspath(vconfig_t *config, Ppoint_t p0, int poly0, Ppoint_t p1, int poly1, Ppolyline_t *output_route);
void Pobsclose(vconfig_t *config);

These functions find a shortest path between two points in the plane that contains polygonal obstacles (holes). *Pobsopen* creates a configuration (an opaque struct of type vconfig_t) on a set of obstacles. The

n_obstacles obstacles are given in the array *obstacles*; the points of each polygon should be in clockwise order. The function *Pobsclose* frees the data allocated in *Pobsopen*.

Pobspath finds a shortest path between the endpoints that remains outside the obstacles. If the endpoints are known to lie inside obstacles, poly0 or poly1 should be set to the index in the obstacles array. If an endpoint is definitely not inside an obstacle, then POLYID_NONE should be passed. If the caller has not checked, then POLYID_UNKNOWN should be passed, and the path library will perform the test. The resulting shortest path is returned in *output_route*. Note that this function does not provide for a boundary polygon. The array of points stored in *output_route* are allocated by the library, but should be freed by the user.

int Proutespline (Pedge_t *barriers, int n_barriers, Ppolyline_t input_route, Pvector_t endpoint_slopes[2], Ppolyline_t *output_route);

This function fits a cubic B-spline curve to a polyline path. The curve is constructed to avoid a set of *n_barriers* barrier line segments specified in the array *barriers*. If you start with polygonal obstacles, you can supply each polygon's edges as part of the barrier list. The polyline input_route provides a template for the final path; it is usually the output_route of one of the shortest path finders, but it can be any simple path that doesn't cross any barrier segment. The input also allows the specification of desired slopes at the endpoints via *endpoint_slopes*. These are specified as vectors. For example, to have an angle of *T* at an endpoing, one could use *(cos(T),sin(T))*. A vector (0,0) means unconstrained slope. The output is returned in *output_route* and consists of the control points of the B-spline. The function return 0 on success; a return value of -1 indicates failure. The array of points in *output_route* is static to the library. It should not be freed, and should be used before another call to *Proutespline*.

int Ppolybarriers(Ppoly_t **polys, int n_polys, Pedge_t **barriers, int *n_barriers);

This is a utility function that converts an input list of polygons into an output list of barrier segments. The array of points in *barriers* is static to the library. It should not be freed, and should be used before another call to *Ppolybarriers*. The function returns 1 on success.

BUGS

The function *Proutespline* does not guarantee that it will preserve the topology of the input path as regards the boundaries. For example, if some of the segments correspond to a small polygon, it may be possible that the final path has flipped over the obstacle.

AUTHORS

David Dobkin (dpd@cs.princeton.edu), Eleftherios Koutsofios (ek@research.att.com), Emden Gansner (erg@research.att.com).