Example Problem: A* on Grid

$f(n) = h(n) + g(n)$

heuristic cost L1 (MD)

+ path cost so far

	0	1	2	3	4	5	6	7	8	9
0	inf	inf	9	8	inf	inf	inf	inf	inf	inf
1	inf	9	8	7	6	5	inf	3	4	inf
2	inf	8	7	6	5	4	inf	2	3	inf
3	inf	7	6	5	4	3	inf	1	2	inf
4	inf	6	5	4	3	2	inf	0	1	inf
5	inf	7	6	5	inf	inf	inf	inf	2	inf
6	inf	8	7	6	5	4	3	2	3	inf
7	inf	9	8	7	6	5	4	3	4	inf
8	inf	10	9	8	7	6	5	4	5	inf
9	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

all MDs (for illustration)

equal distances
=> order of visits matters!!!

illustration of the total cost in each step

f(n) = h(n) + g(n) (heuristic cost (MD)+cost so far)

orange: visited node

yellow: visited neighbors

light green: CLOSED

	0	1	2	3	4	5	6	7	8	9
0	inf	inf			inf	inf	inf	inf	inf	inf
1	inf						inf			inf
2	inf						inf			inf
3	inf				4+1		inf			inf
4	inf			4+1	***3+0***	2+1	inf			inf
5	inf				inf	inf	inf	inf		inf
6	inf									inf
7	inf									inf
8	inf									inf
9	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

step1

	0	1	2	3	4	5	6	7	8	9
0	inf	inf			inf	inf	inf	inf	inf	inf
1	inf						inf			inf
2	inf						inf			inf
3	inf				4+1	3+2	inf			inf
4	inf			4+1	***3+0***	***2+1***	inf			inf
5	inf				inf	inf	inf	inf		inf
6	inf									inf
7	inf									inf
8	inf									inf
9	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

step2

	0	1	2	3	4	5	6	7	8	9
0	inf	inf			inf	inf	inf	inf	inf	inf
1	inf						inf			inf
2	inf					4+3	inf			inf
3	inf				4+1	***3+2***	inf			inf
4	inf			4+1	***3+0***	***2+1***	inf	0		inf
5	inf				inf	inf	inf	inf		inf
6	inf									inf
7	inf									inf
8	inf									inf
9	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

step3

	0	1	2	3	4	5	6	7	8	9
0	inf	inf			inf	inf	inf	inf	inf	inf
1	inf						inf			inf
2	inf				5+2	4+3	inf			inf
3	inf			5+2	***4+1***	3+2	inf			inf
4	inf			4+1	***3+0***	2+1	inf	0		inf
5	inf				inf	inf	inf	inf		inf
6	inf									inf
7	inf									inf
8	inf									inf
9	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

step4

	0	1	2	3	4	5	6	7	8	9
0	inf	inf			inf	inf	inf	inf	inf	inf
1	inf						inf			inf
2	inf				5+2	4+3	inf			inf
3	inf			5+2	4+1	3+2	inf			inf
4	inf		5+2	***4+1***	***3+0***	2+1	inf	0		inf
5	inf			5+2	inf	inf	inf	inf		inf
6	inf									inf
7	inf									inf
8	inf									inf
9	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

step5

	0	1	2	3	4	5	6	7	8	9
0	inf	inf			inf	inf	inf	inf	inf	inf
1	inf					5+4	inf			inf
2	inf				5+4	***4+3***	inf			inf
3	inf			5+2	4+1	***3+2***	inf			inf
4	inf		5+2	4+1	***3+0***	***2+1***	inf	0		inf
5	inf			5+2	inf	inf	inf	inf		inf
6	inf									inf
7	inf									inf
8	inf									inf
9	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

step6

...

Alternative: L1 Voronoi on Regular Grid

max. clearance to the walls (instead of passing by walls)

compute via wave-front (aka bushfire method)

- propagate a wave front
  - from the obstacles boundaries away
  - in every step, update for every cell a pointer to the obstacle from which the wave front originated
- "collision" of 2 wave fronts from 2 different obstacles
  - collision point must be on the equidistant edge
  - hence on the GVD, i.e., mark this pixel as GVD part

Voronoi on Regular Grid

// initialize all occupied cells with 0, all unoccupied ones with infinity
$\forall\ (x, y)$ with $M[x][y] = 1$: $D[x][y] = 0$
$\forall\ (x, y)$ with $M[x][y] = 0$: $D[x][y] = \infty$
// put all occupied cells into queue Q
$\forall\ (x, y)$ with $M[x][y] = 1$: queue(Q,(x, y))
// take element from Q, update its distance, queue its neighbors if not yet visited
while(notempty(Q) {
    $(x, y) =$ dequeue(Q)
    visit$(x, y) =$ true
    $D[x][y] = \min\ (D[x][y], D[x+1][y]+1, D[x-1][y]+1, D[x][y+1]+1, D[x][y-1]+1)$
    if($\neg$ visit$(x+1, y)$): queue(Q,$(x+1, y)$)
    if($\neg$ visit$(x-1, y)$): queue(Q,$(x-1, y)$)
    if($\neg$ visit$(x, y+1)$): queue(Q,$(x, y+1)$)
    if($\neg$ visit$(x, y-1)$): queue(Q,$(x, y-1)$)
}

Voronoi on Regular Grid

black : obstacles

brightness : D[x][y]

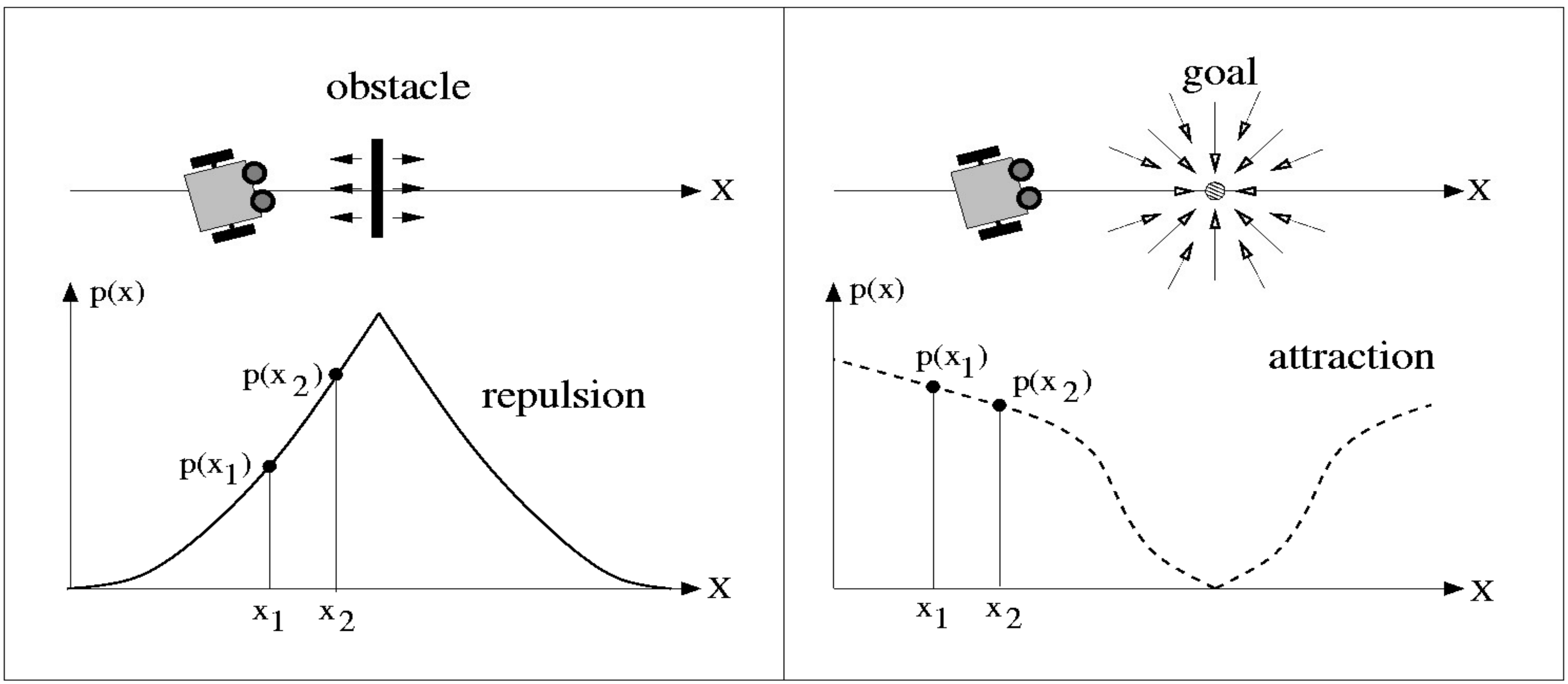
- the brighter,
- the higher the distance to the nearest obstacle

Path-Planning Approaches

- Roadmap
- Cell decomposition
- **Potential field**

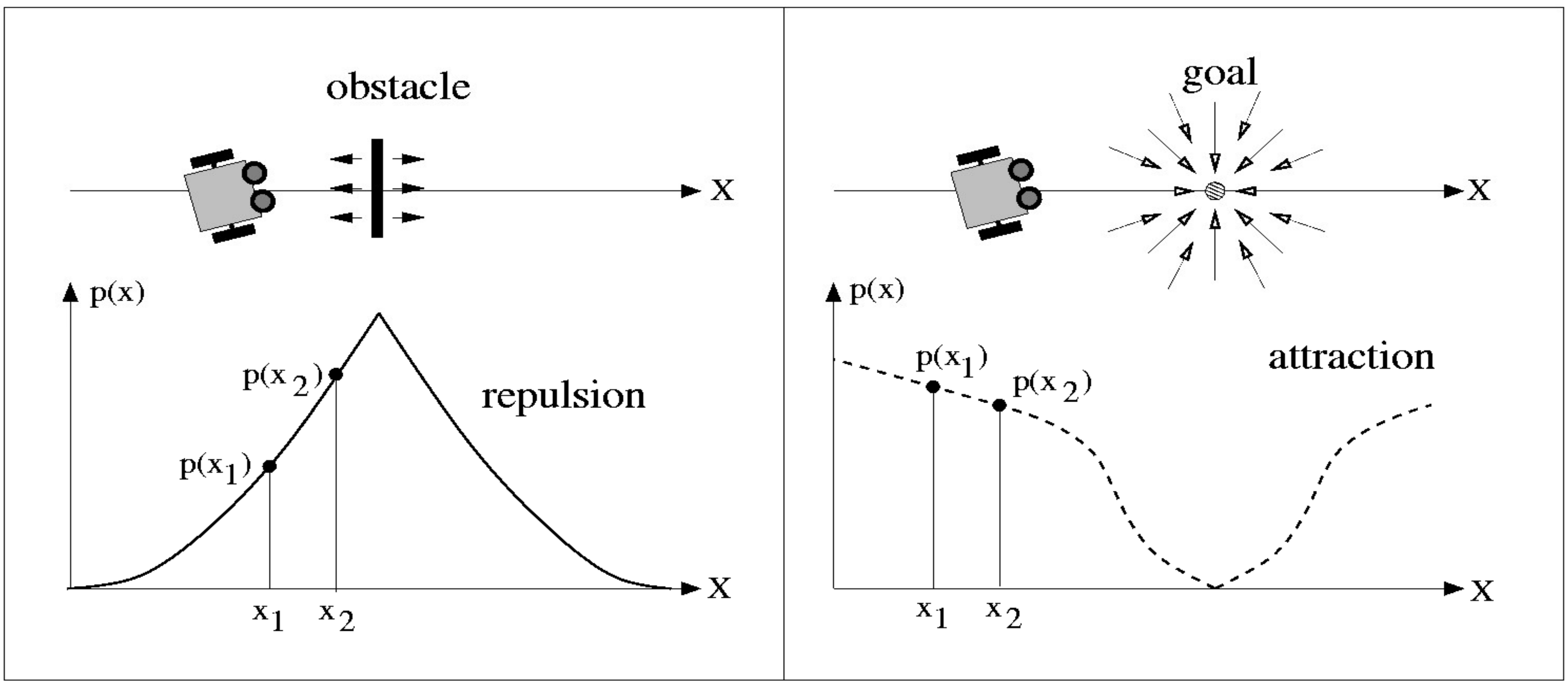
Potential Fields

- attraction and repulsion
- similar to physical fields (for example electric fields)
- agent follows gradients



Potential Fields

- originally proposed for realtime obstacle avoidance
- close relation to behavior-oriented programming



Potential Field Generation

- potential field function *U(q)*
- artificial force field *F(q)*

$$F(q) = -\nabla U(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q) = \begin{bmatrix} \dfrac{\partial U}{\partial x} \\ \dfrac{\partial U}{\partial y} \end{bmatrix}$$

- set robot speed (v_x, v_y) proportional to the force $F(q)$

Attractive Potential Field

$q = [x, y]^T$ $q_{goal} = [x_{goal}, y_{goal}]^T$

e.g., parabolic function

representing the Euclidean distance to the goal

$$U_{att}(q) = k_{att} \cdot (q - q_{goal})^2$$

attracting force converges linearly towards 0 (goal)

$$\begin{aligned} F_{att}(q) &= -\nabla U_{att}(q) \\ &= k_{att} \cdot (q - q_{goal}) \end{aligned}$$

Repulsing Potential Field

idea: a barrier around all the obstacle

- strong if close to the obstacle
- no influence if far from the obstacle

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0} \right)^2 & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) \ge \rho_0 \end{cases}$$

- *p(q) =minimum distance to the object (to point q_{goal})*
- *ρ_0 = distance of influence of the object*

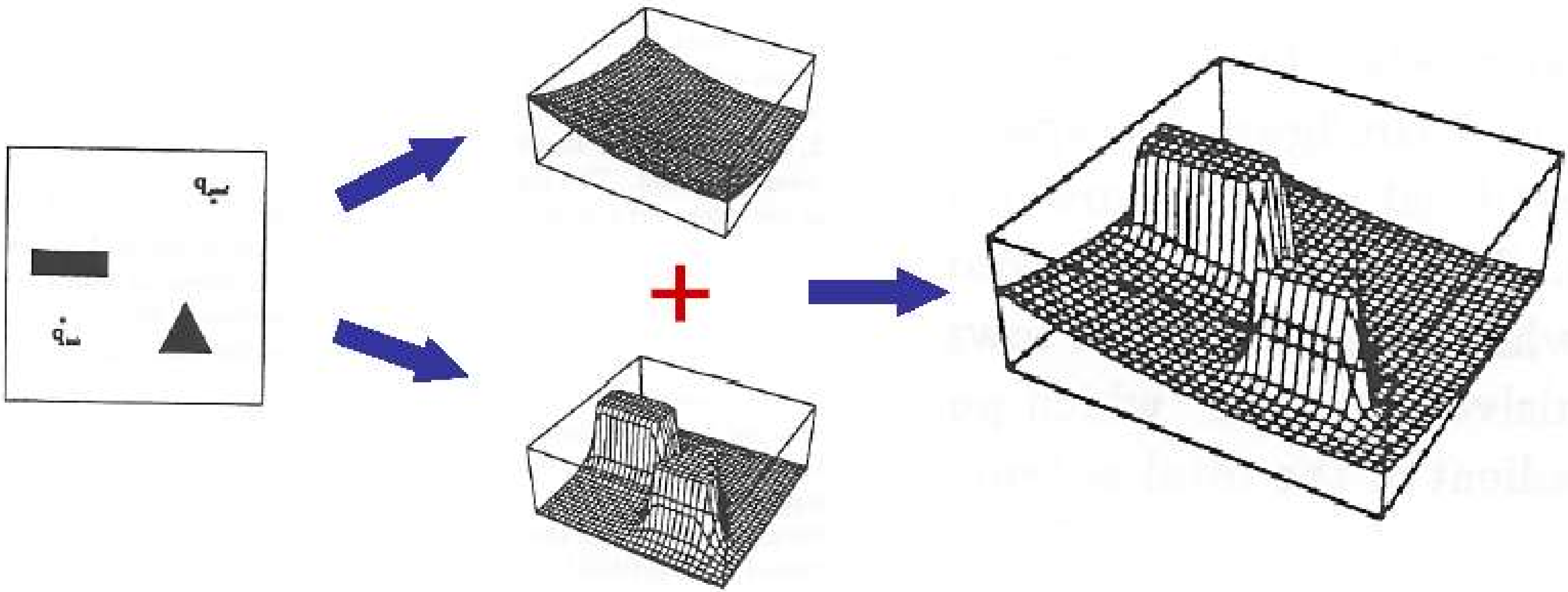
Force is positive or zero and *tends to infinity* as q gets closer to the object

$$F_{rep}(q) = -\nabla U_{rep}(q) = \begin{cases} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2(q)} \frac{q - q_{goal}}{\rho(q)} & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) \ge \rho_0 \end{cases}$$

Potential Fields

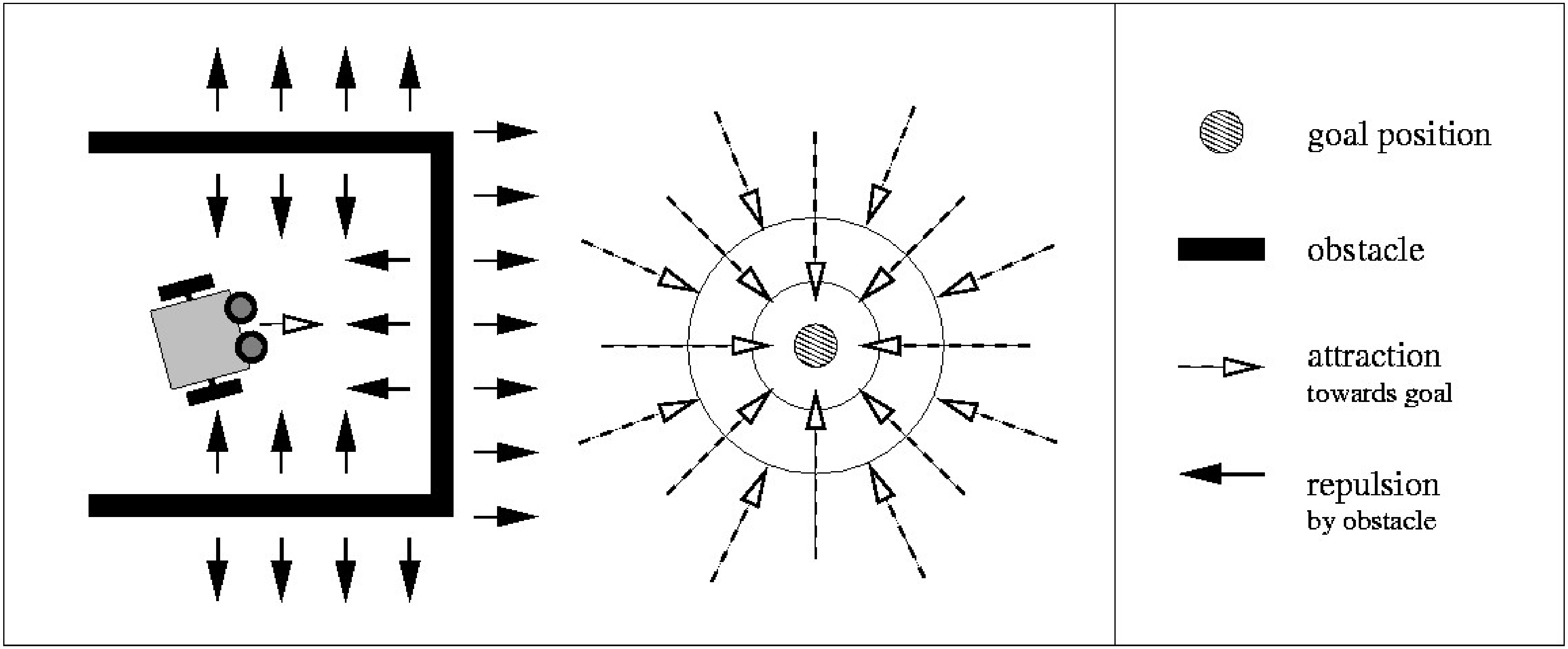
attractive and repulsive fields

- as fct‘s computed on the fly based on distance
- or precomputed in grids



Potential Fields

problems with local optima



option: combine with random walk

but better alternatives are possible...

Potential Fields: Navigation Function

- potential field with no local minima
  - for any point *p*
  - the navigation function *N*(*p*)
  - is the minimum cost to the goal
- use a *wavefront* algorithm
  - propagating from the goal to the current location
  - an active point updates costs of its 8 neighbors
  - a point becomes active if its cost decreases.
  - continue until robot’s position is reached

Potential Fields: Navigation Function

- potential field with no local minima
  - for any point *p*
  - the navigation function *N*(*p*)
  - is the minimum cost of a path to the goal
- use a *wavefront* algorithm
  - propagating from the go [illegible] rent location
  - an active poin [illegible] sts of its 8 neighbors
  - a [illegible] active if its cost decreases.
  - co [illegible] ue until robot's position is reached

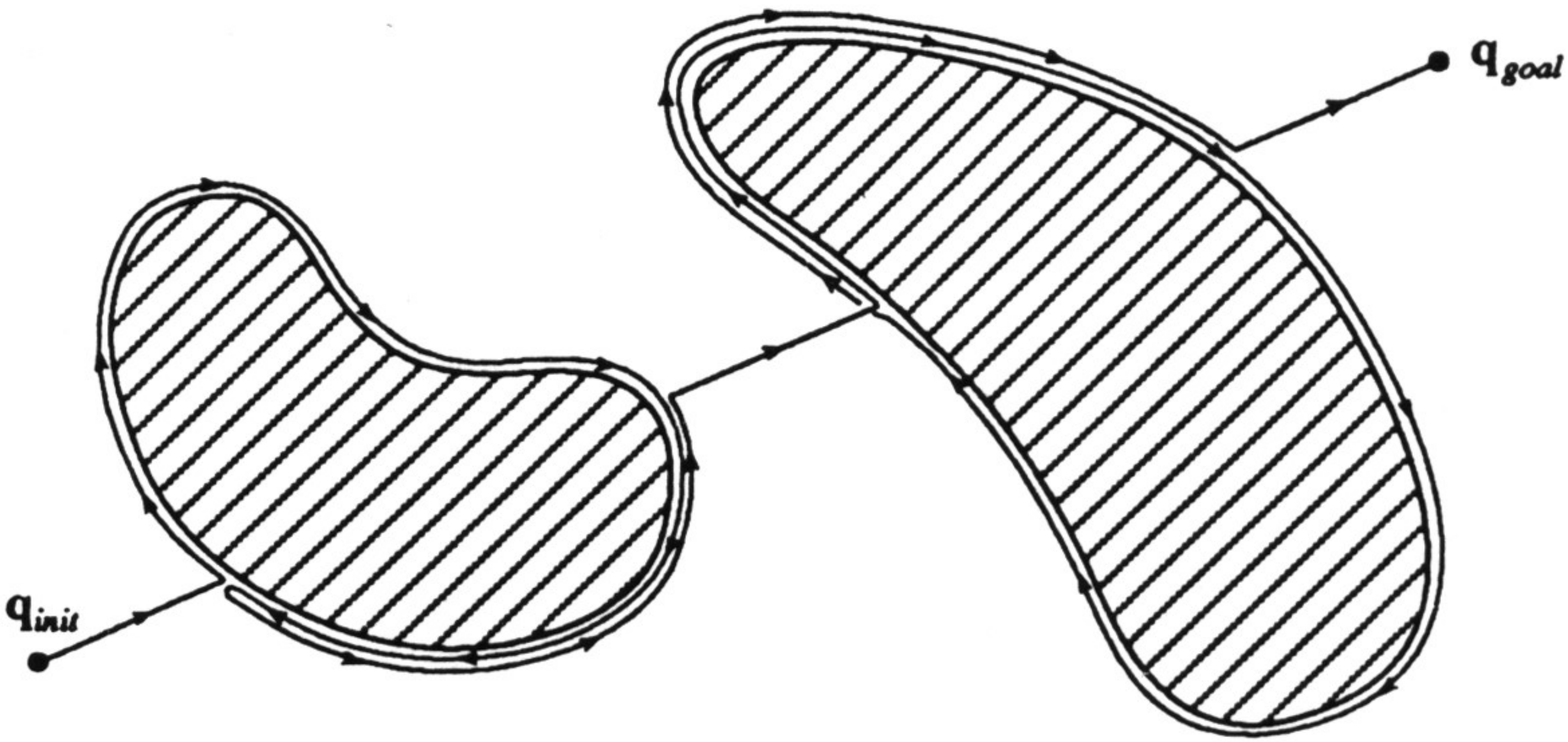
does this look familiar?!?!

Bug Algorithm

- follow straight line segment to the goal
- hit an obstacle => follow boundary
- when returning to the hitting point
  - follow the boundary
  - to the point on the boundary
  - that is on the line segment and closest to the goal
- then resume the straight line segment path

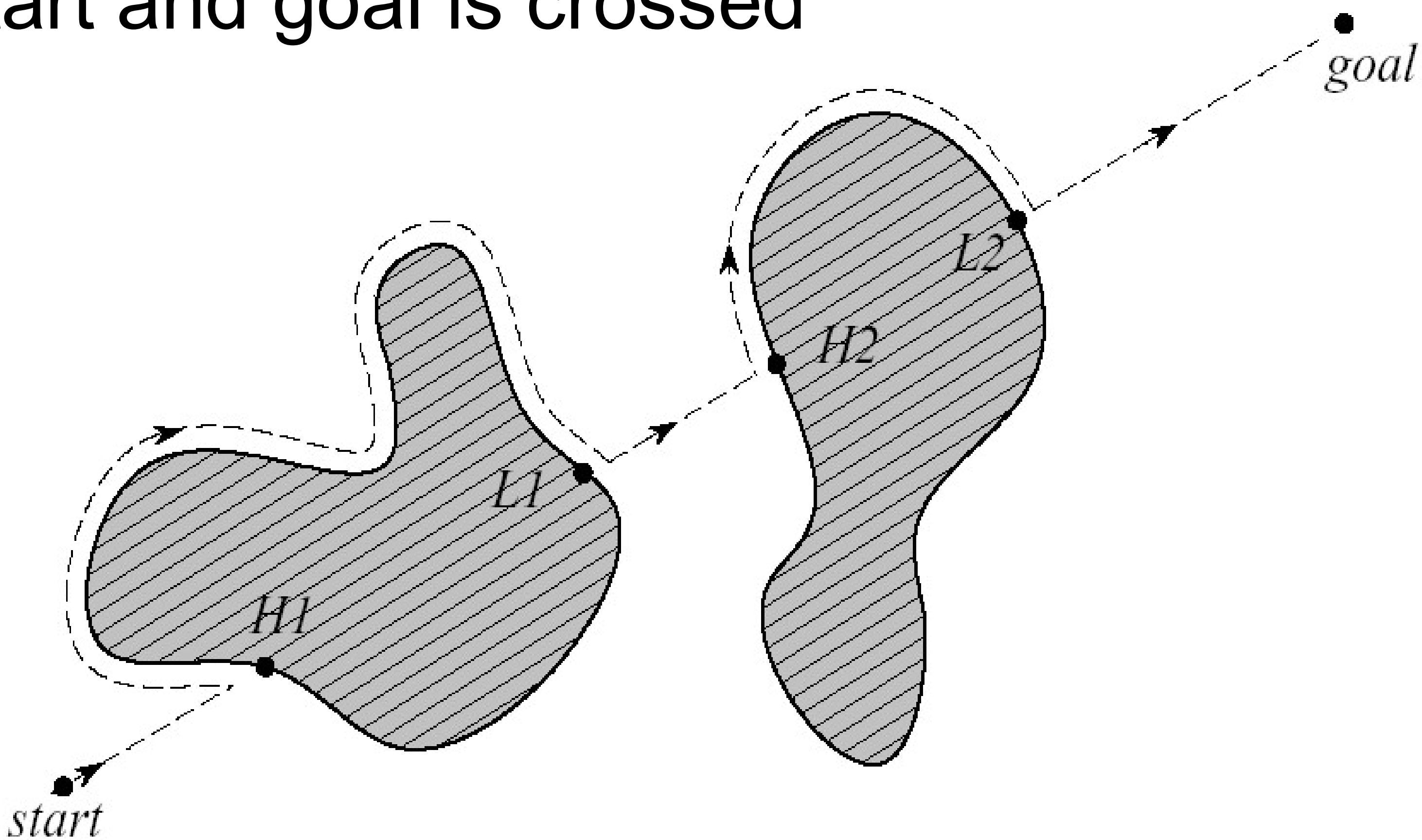
Bug Algorithm

- reactive attraction to goal
- but always finds a path (if it exists)



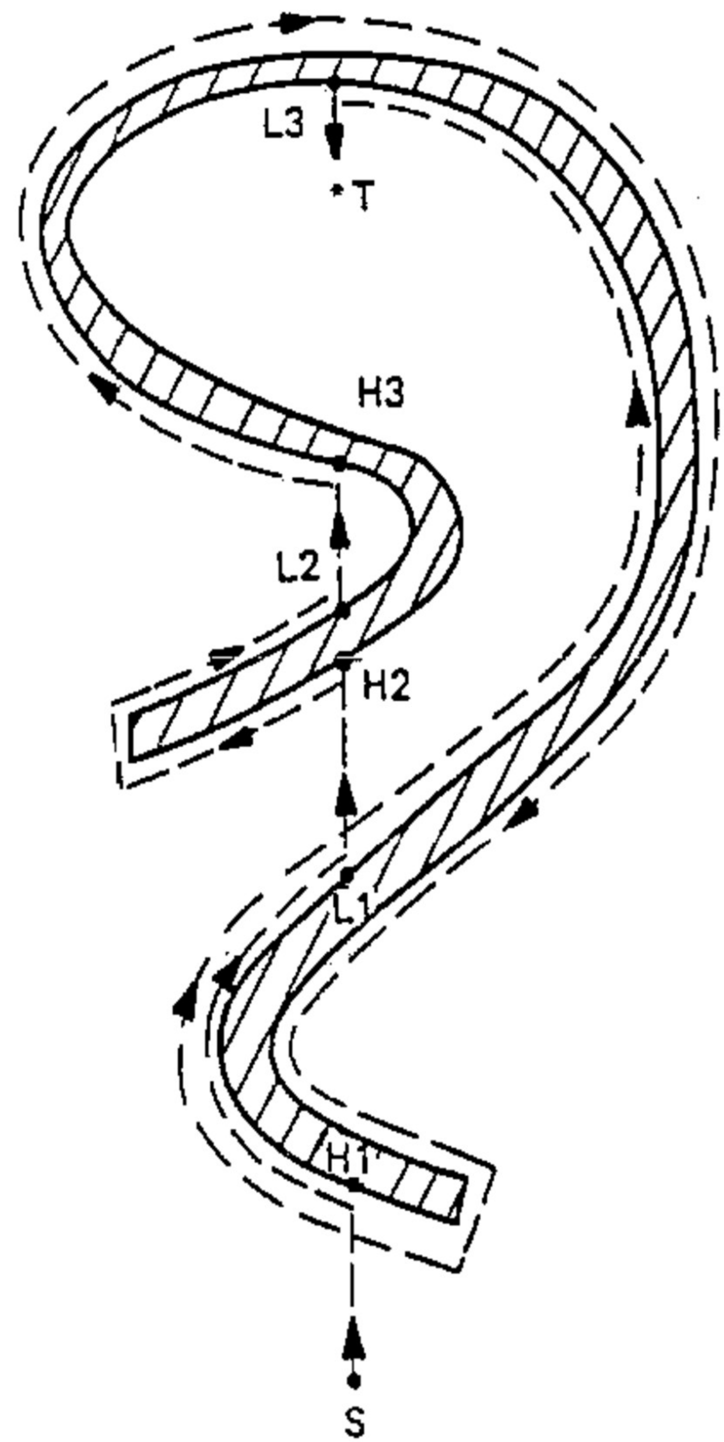
Bug Alg: 2nd version

- follow obstacle always on the left (or right) side
- leave obstacle if the direct connection between start and goal is crossed



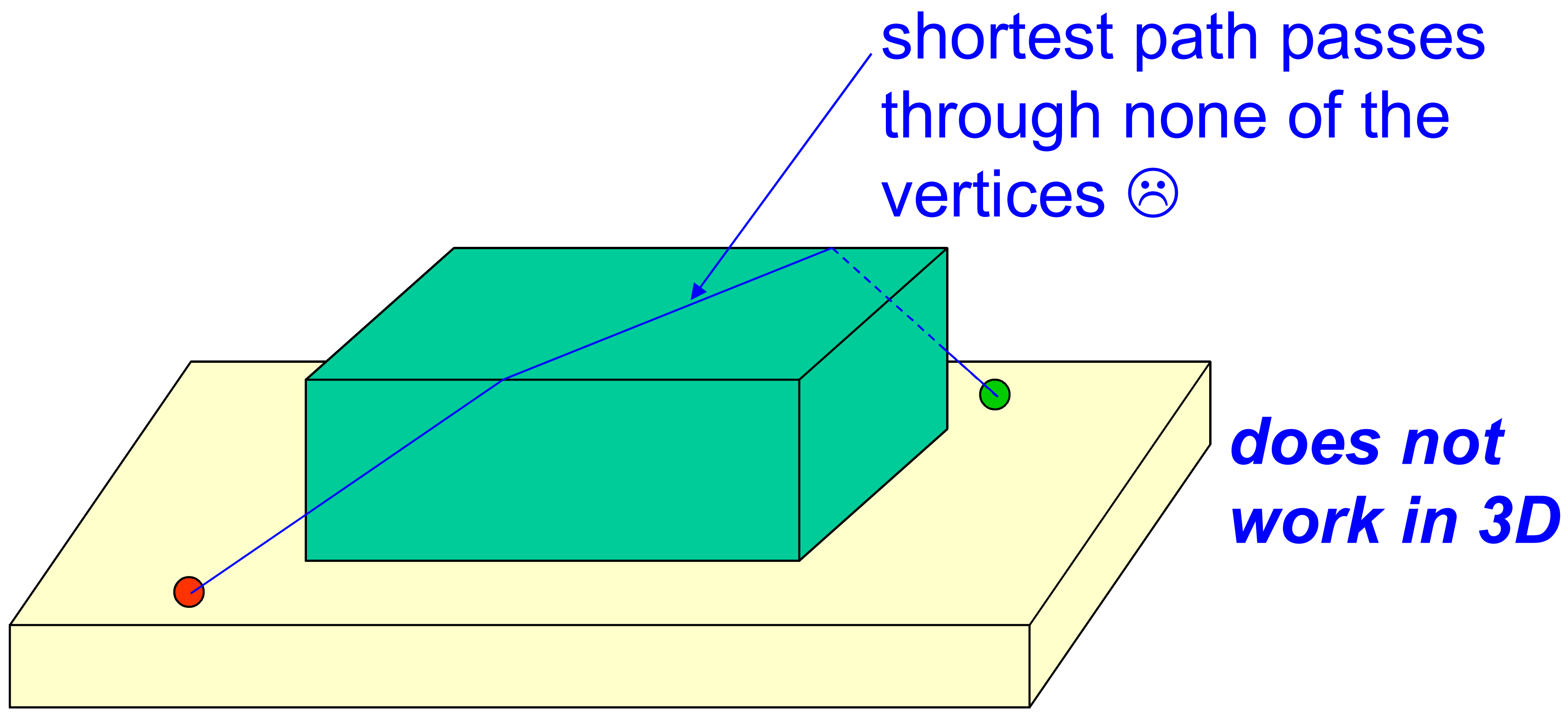
Bug Alg: 2nd version

can also
lead to
long paths



Path Planning beyond 2D

Example: Visibility Graph in 3D



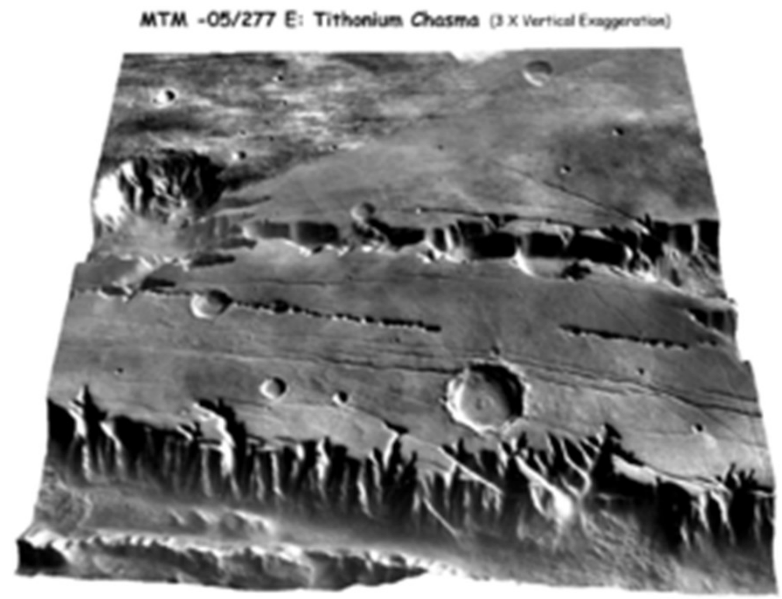
shortest collision-free path
in a polyhedral space is NP-hard

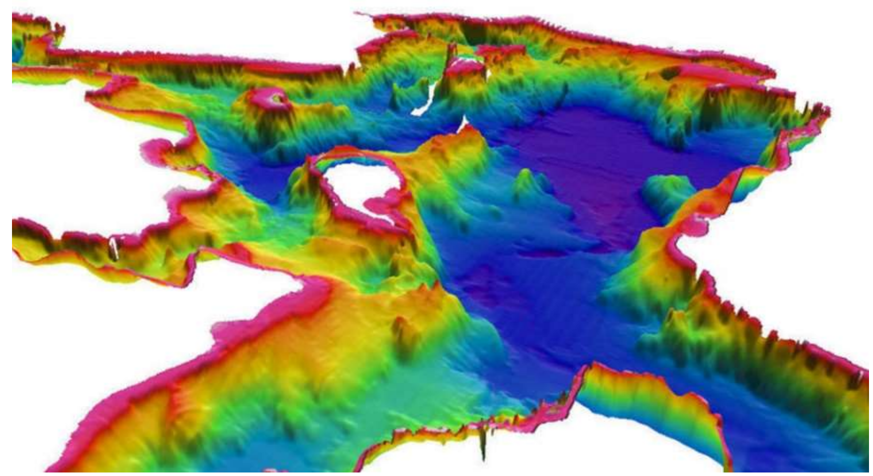
Path Planning in 2.5D Maps

2.5D Elevation

elevation map

- (typically regular) grid
  - cell value = elevation
- aka
  - Digital Elevation Model (DEM),
  - Digital Terrain Model (DTM),
  - Digital Surface Model (DSM)
  - marine: bathymetry
- often falsely denoted as 3D
- but 2D manifolds in 3D space





2.5D Cost Maps

- grid map: edge values = cost to traverse that edge, e.g.,
  - local gradients from elevation (slope) by absolute differences
  - terrain classification (road, grass, sand, gravel, etc.)
- path-planning as before (e.g., A*)
  - but now on weighted graph
  - edge $e=(v_1,v_2)$ & weight $w(e)$