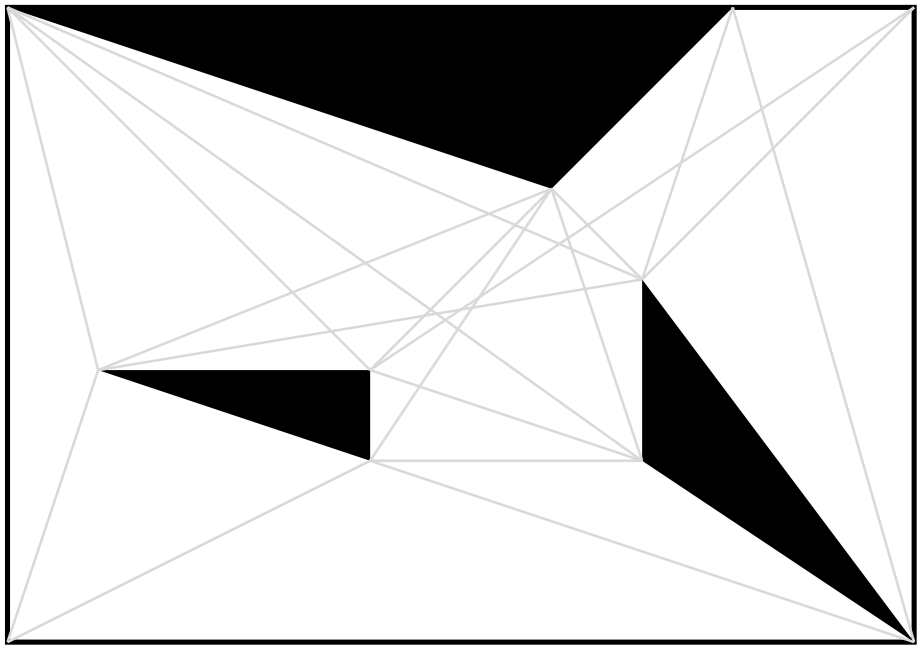
Outline

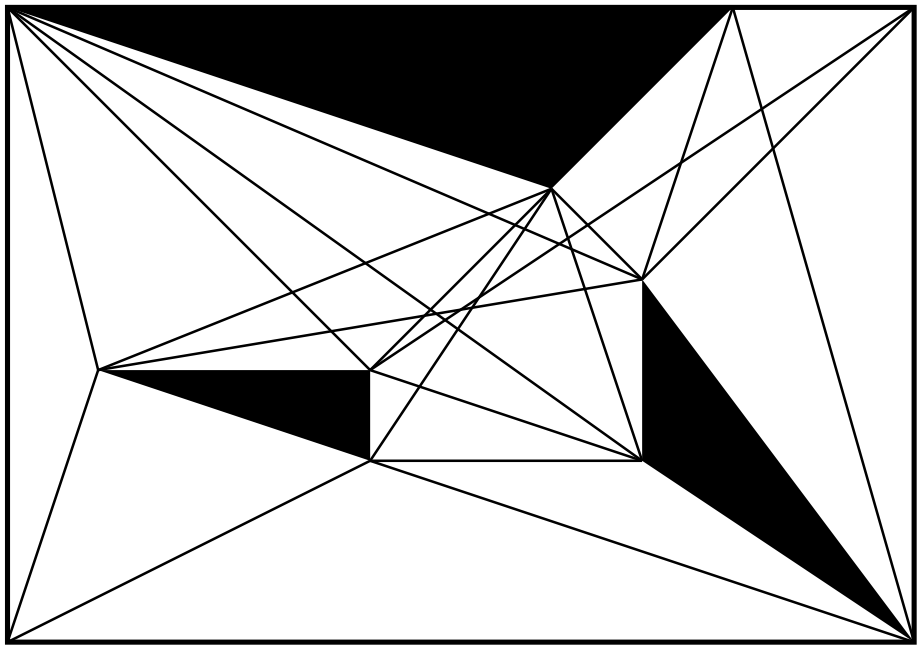
How to compute Obstacle Distance

- Existing works rely on *visibility graph* (VG)
  - any pair of visible points has an edge
- Run shortest path algorithm on *VG* (e.g. *Dijkstra*)
- Building global VG can be expensive - $O(V^2)$ (V: the number of vertex)



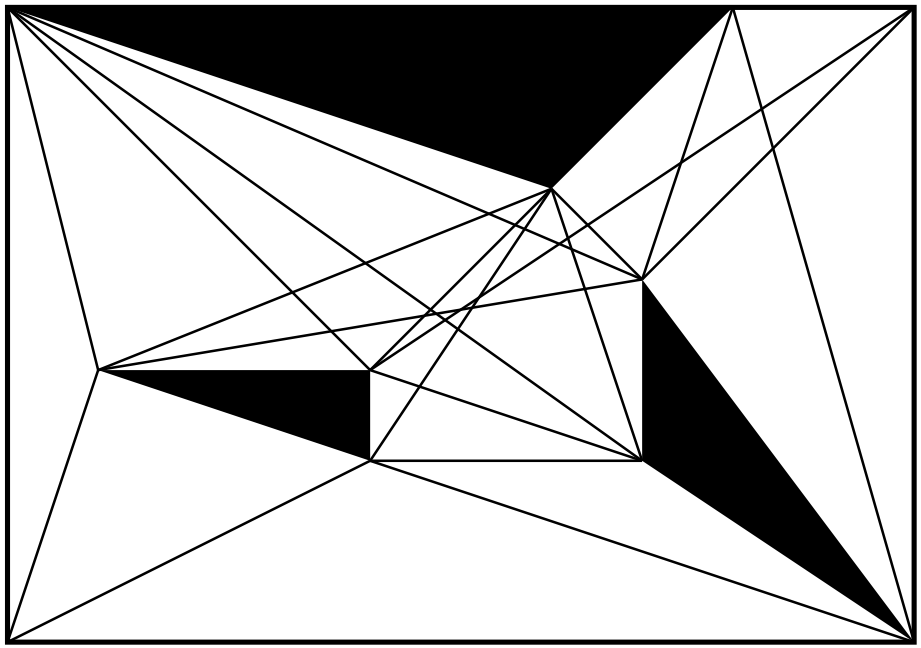
How to compute Obstacle Distance

- Existing works rely on *visibility graph* (VG)
  - any pair of visible points has an edge
- Run shortest path algorithm on *VG* (e.g. *Dijkstra*)
- Building global VG can be expensive - $O(V^2)$ (V: the number of vertex)



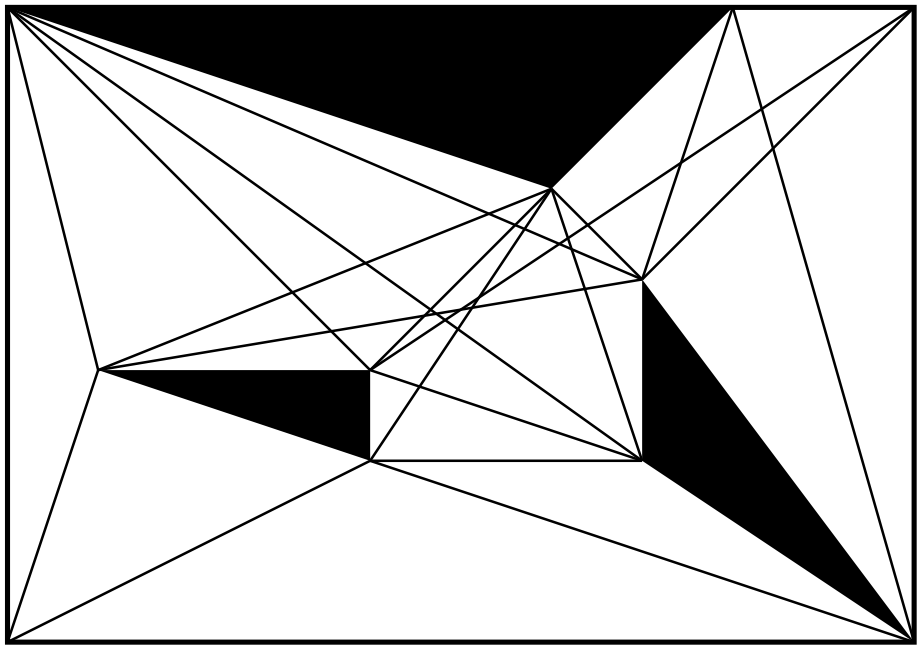
How to compute Obstacle Distance

- Existing works rely on *visibility graph* (VG)
  - any pair of visible points has an edge
- Run shortest path algorithm on *VG* (e.g. *Dijkstra*)
- Building global VG can be expensive - $O(V^2)$ (V: the number of vertex)



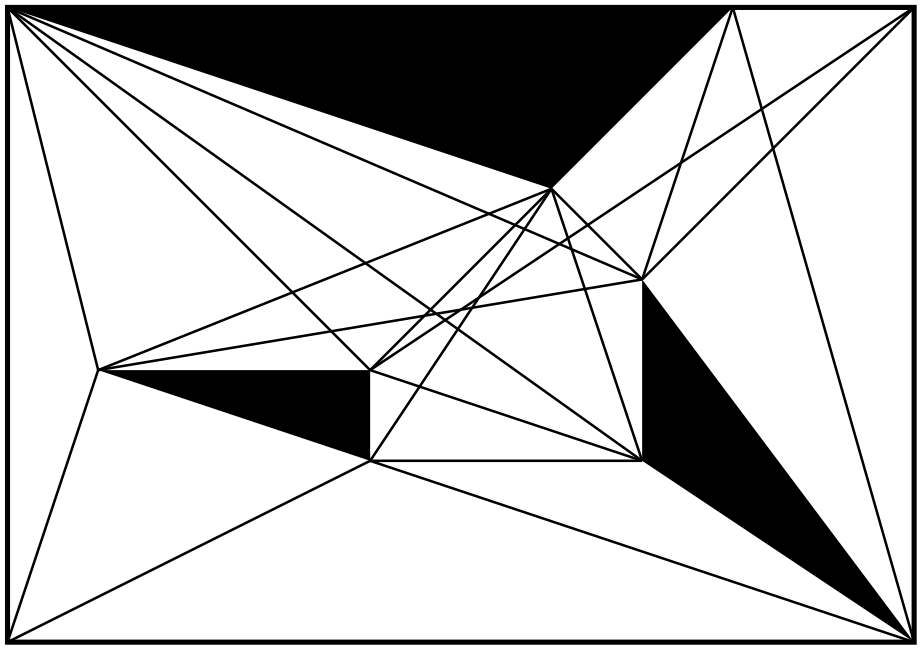
How to compute Obstacle Distance

- Existing works rely on *visibility graph* (VG)
  - any pair of visible points has an edge
- Run shortest path algorithm on *VG* (e.g. *Dijkstra*)
- Building global VG can be expensive - $O(V^2)$ (V: the number of vertex)



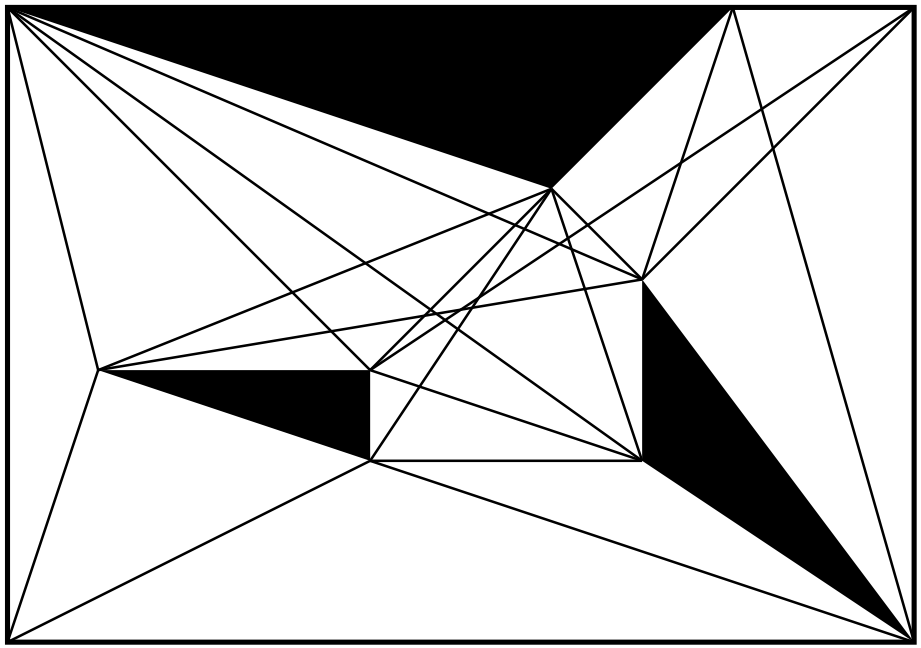
How to compute Obstacle Distance

- Global VG: expansive
- Only consider query related area?
- *Zhang, EDBT 2004: Local Visibility Graph (LVG)*



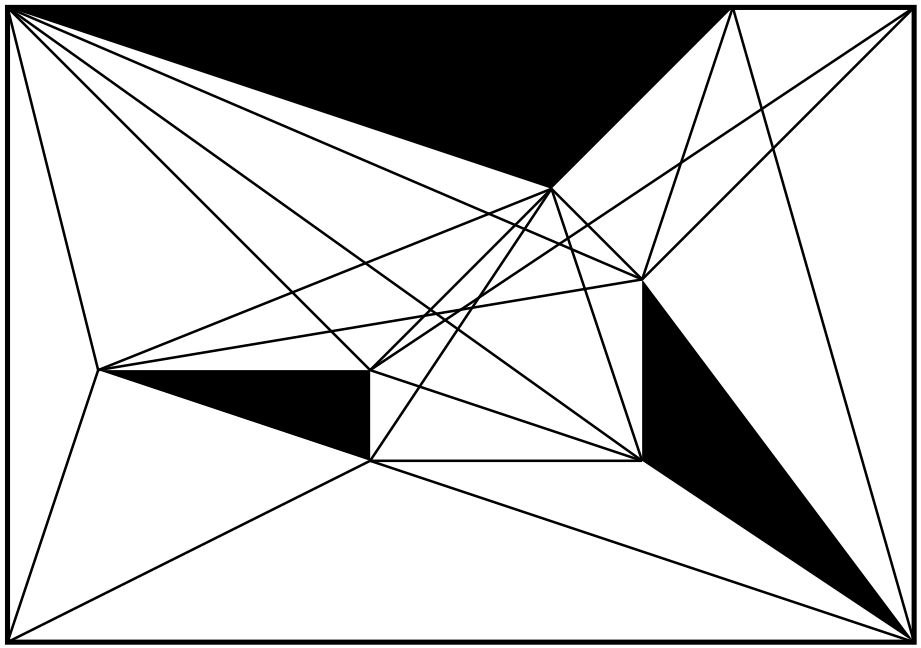
How to compute Obstacle Distance

- Global VG: expansive
- Only consider query related area?
- *Zhang, EDBT 2004: Local Visibility Graph (LVG)*



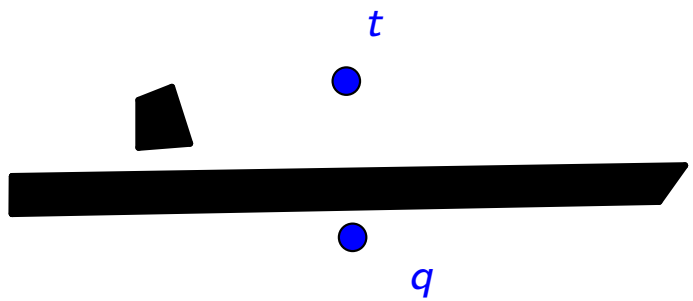
How to compute Obstacle Distance

- Global VG: expansive
- Only consider query related area?
- *Zhang, EDBT 2004*: *Local Visibility Graph (LVG)*



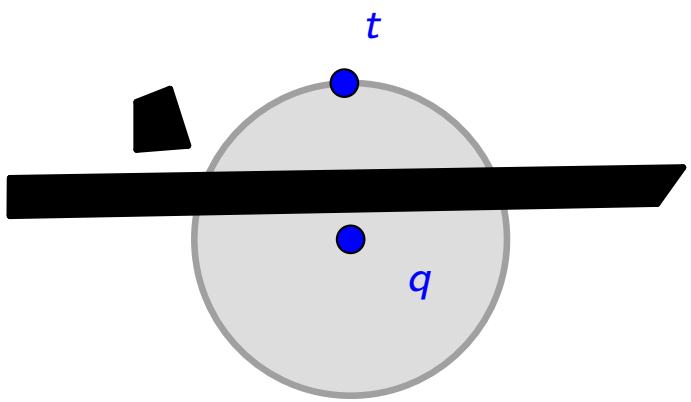
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
  - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
  - build VG incrementally
  - update shortest path
- Terminate when $r > d_o(q, t)$



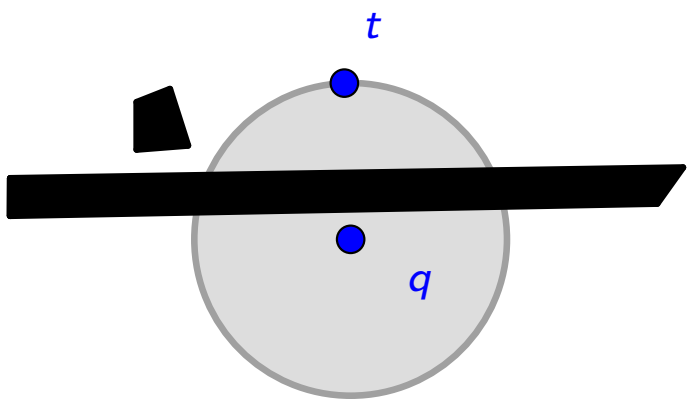
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
  - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
  - build VG incrementally
  - update shortest path
- Terminate when $r > d_o(q, t)$



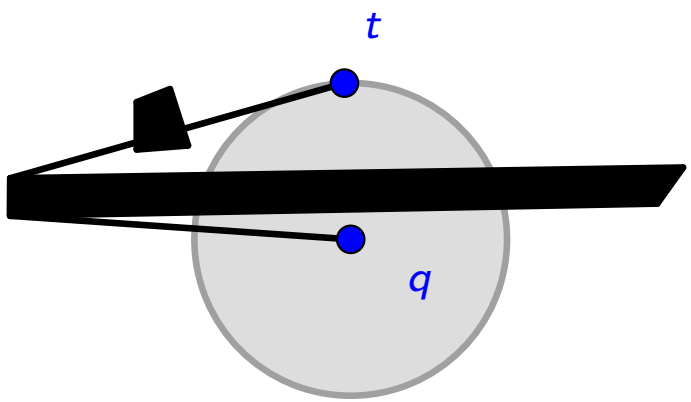
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
  - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
  - build VG incrementally
  - update shortest path
- Terminate when $r > d_o(q, t)$



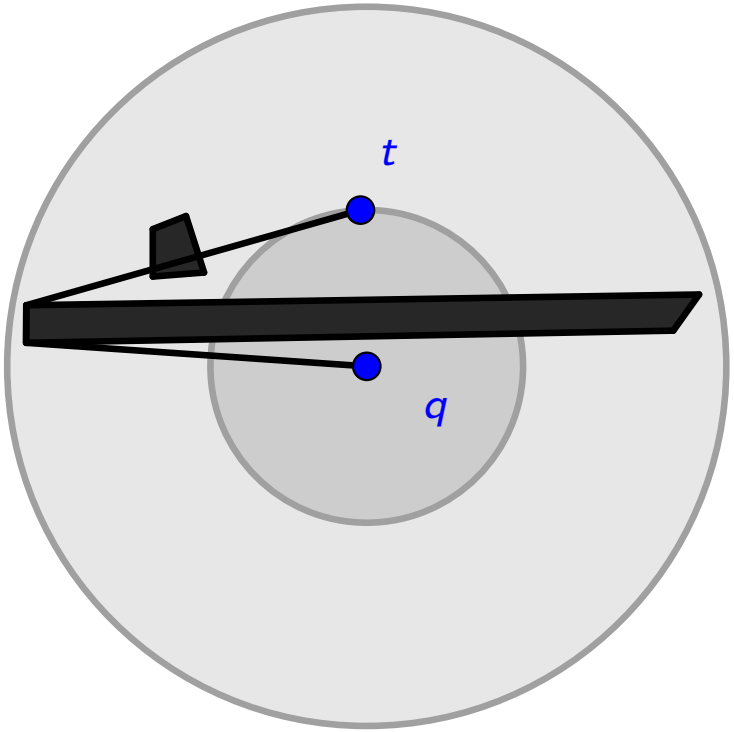
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
  - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
  - build VG incrementally
  - update shortest path
- Terminate when $r > d_o(q, t)$



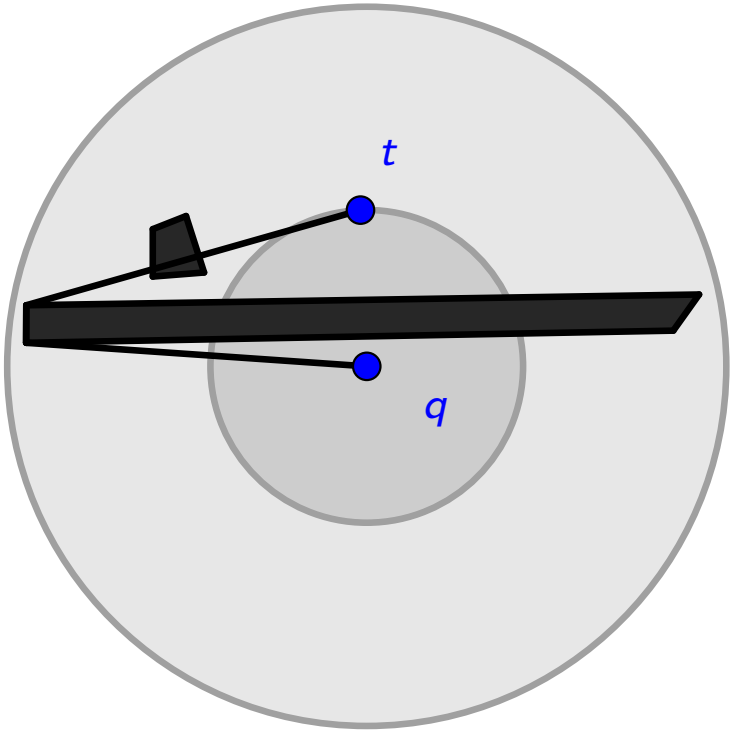
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
  - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
  - build VG incrementally
  - update shortest path
- Terminate when $r > d_o(q, t)$



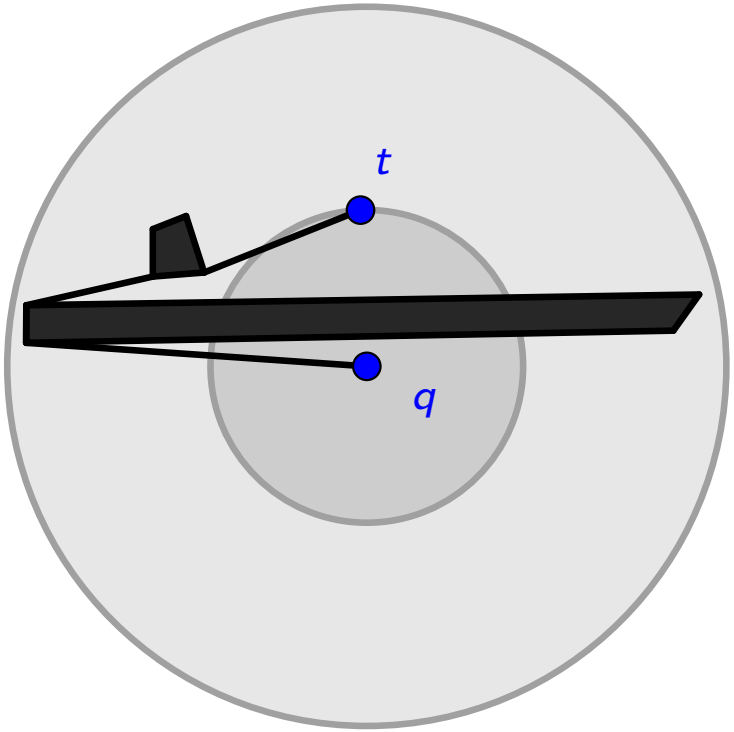
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
  - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
  - build VG incrementally
  - update shortest path
- Terminate when $r > d_o(q, t)$



Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
  - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
  - build VG incrementally
  - update shortest path
- Terminate when $r > d_o(q, t)$



Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
  - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
  - build VG incrementally
  - update shortest path
- Terminate when $r > d_o(q, t)$