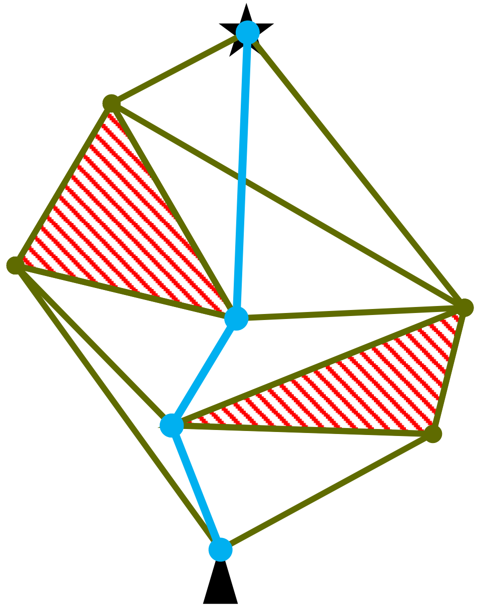
Aerial Robotics
Path Planning II

Prof. Arthur Richards

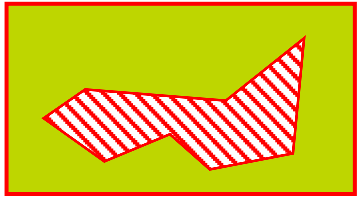
Visibility Graph

- Start with obstacles
- Evaluate all pairs of vertices
- Remove lines that go through obstacles
- Connect the start and goal
- Graph search

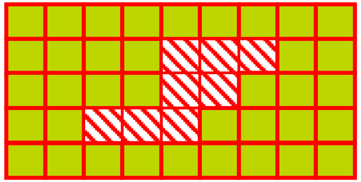


Representations

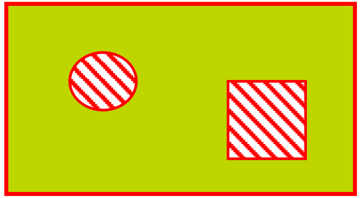
- All examples use line segments
  - In general terms, a “mesh”



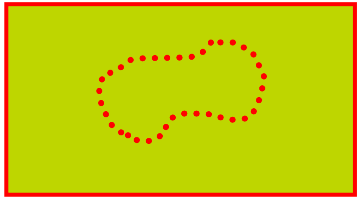
- Instead, occupancy grids?



- Convex primitives?

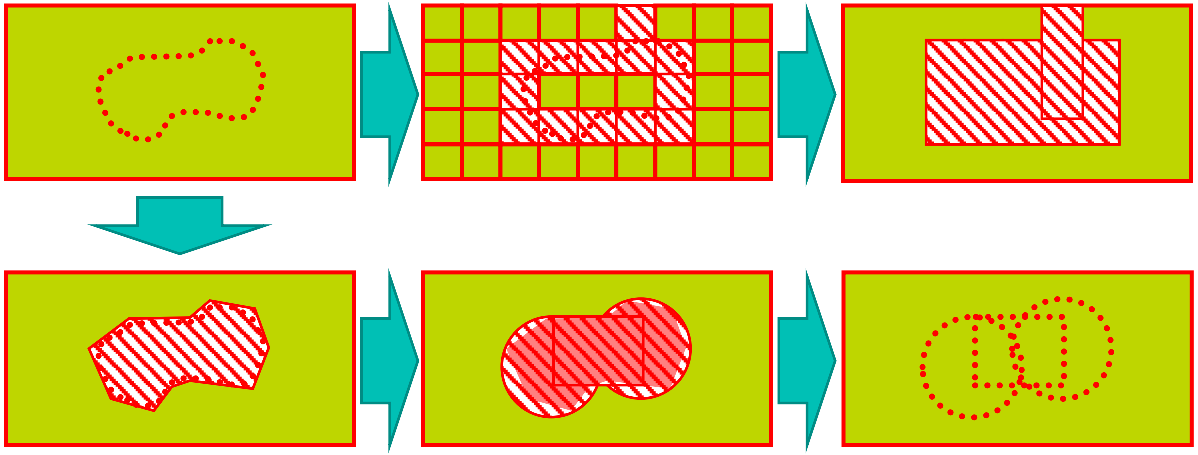


- Unstructured point cloud?



Representations

- Can switch between representations
  - Typically need to trade between efficiency and exactness

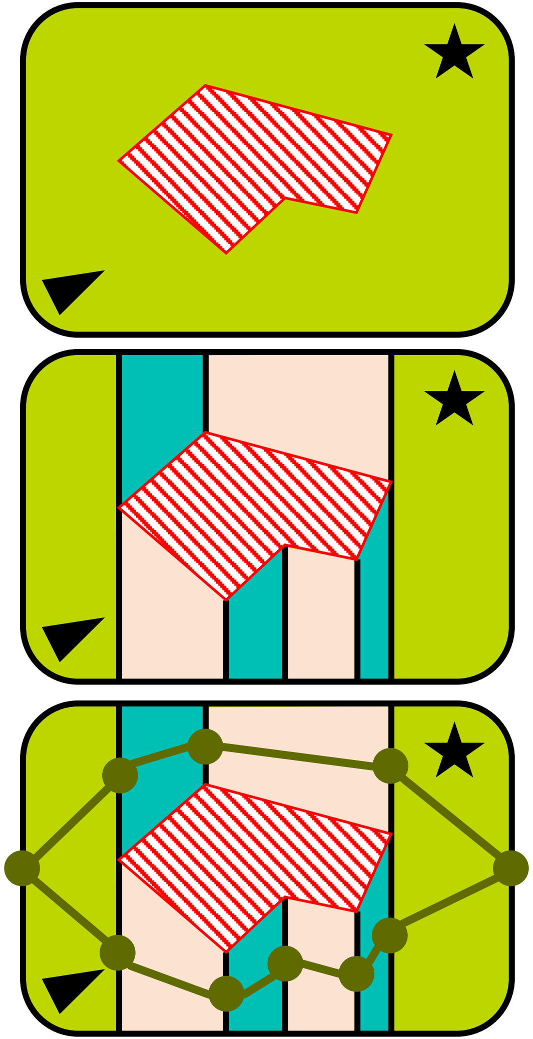


bristol.ac.uk

Cell Decomposition

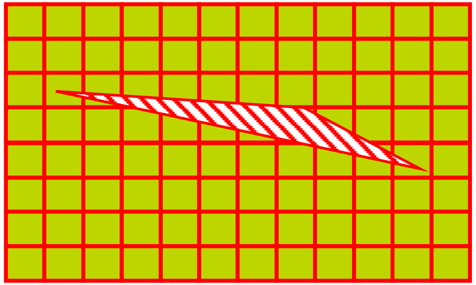
- Alternative idea: divide the free space into convex regions
  - Any two points in a convex region must be mutually visible

- Roadmap found by joining neighbouring regions
  - Cartoon (right) shows a sweep line method: sweep a vertical line across and drop a boundary wherever you cross a corner



Regular Mesh Decomposition

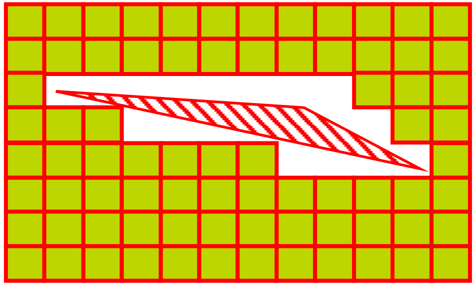
- Simple one: divide into uniform grid



Regular Mesh Decomposition

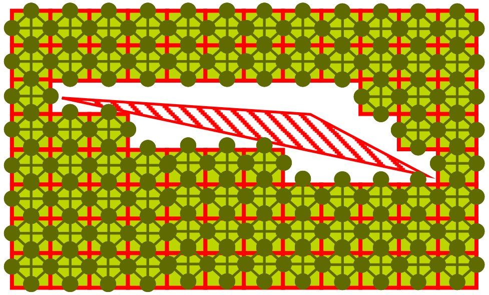
- Simple one: divide into uniform grid

- Keep only obstacle-free cells



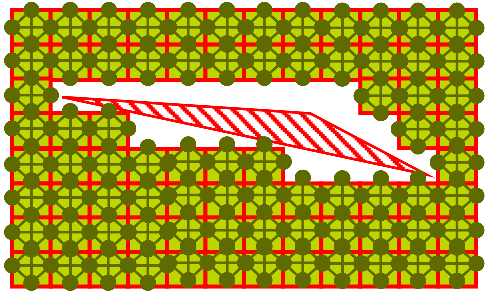
Regular Mesh Decomposition

- Simple one: divide into uniform grid
- Keep only obstacle-free cells
- Use midpoints as entry/exit nodes and join up neighbours



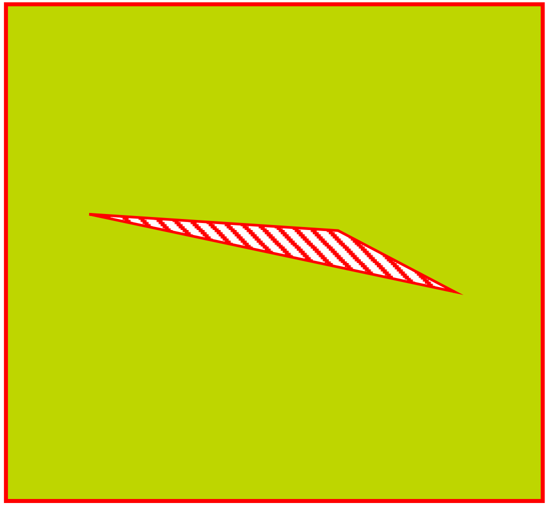
Regular Mesh Decomposition

- Simple one: divide into uniform grid
- Keep only obstacle-free cells
- Use midpoints as entry/exit nodes and join up neighbours
- Graph search for path



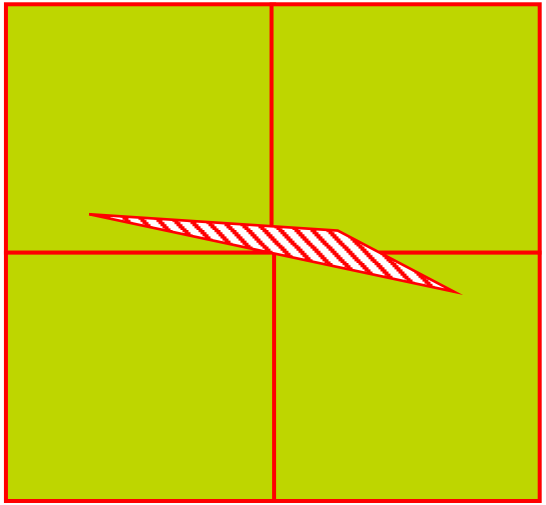
Quadtree

- Another cell decomposition method
  - Likes a square world – will see why



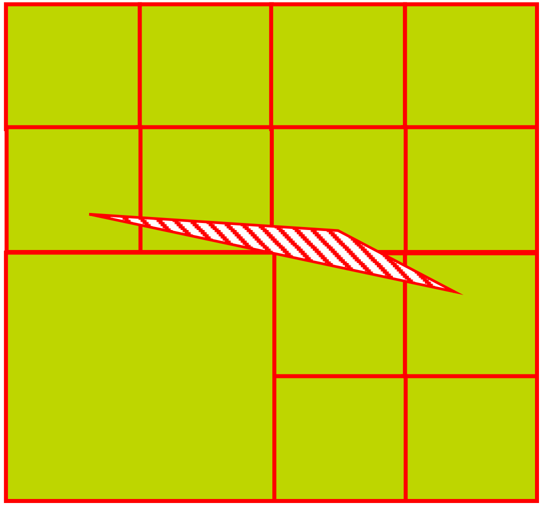
Quadtree

- Another cell decomposition method
  - – Likes a square world – will see why
- Cut world in four (*hence ‘quad’)*



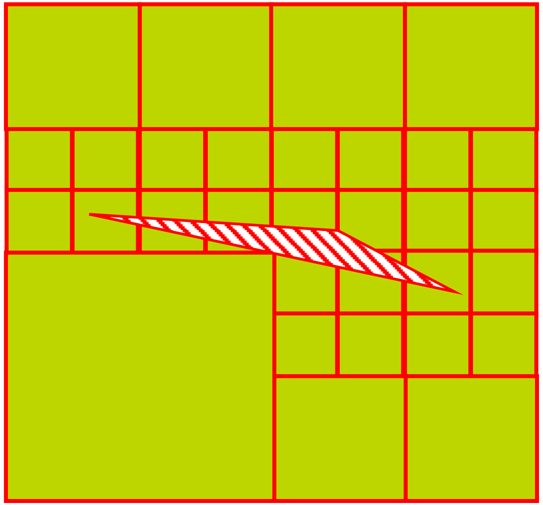
Quadtree

- Another cell decomposition method
  - Likes a square world – will see why
- Cut world in four (*hence ‘quad’)*
- For every child that isn’t empty, do quadtree decomposition
  - A *recursive* algorithm



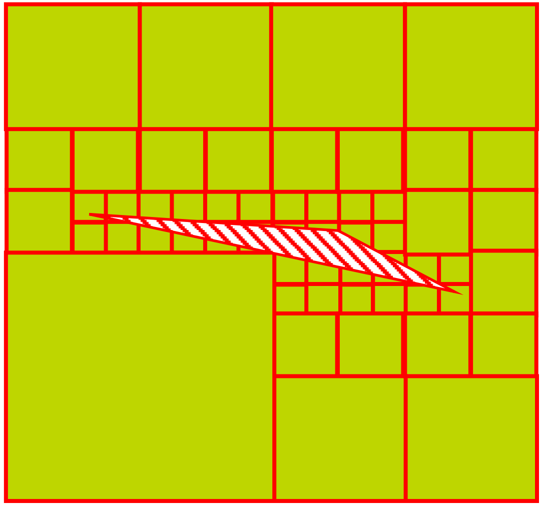
Quadtree

- Another cell decomposition method
  - Likes a square world – will see why
- Cut world in four (*hence 'quad'*)
- For every child that isn't empty, do quadtree decomposition
  - A *recursive* algorithm



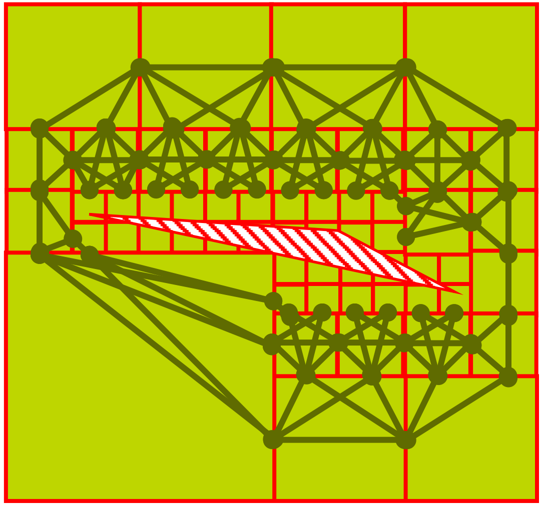
Quadtree

- Another cell decomposition method
  - Likes a square world – will see why
- Cut world in four (*hence 'quad'*)
- For every child cell that isn't empty, do quadtree decomposition
  - A *recursive* algorithm



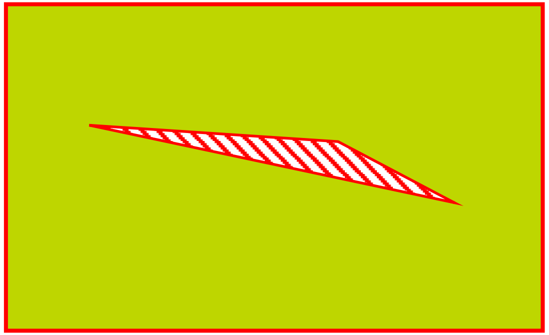
Quadtree

- Another cell decomposition method
  - Likes a square world – will see why
- Cut world in four (*hence 'quad'*)
- For every child cell that isn't empty, do quadtree decomposition
  - A *recursive* algorithm
  - Stop when you're bored…
- Join up obstacle-free neighbours via midpoints, as before



Constrained Delaunay Triangulation

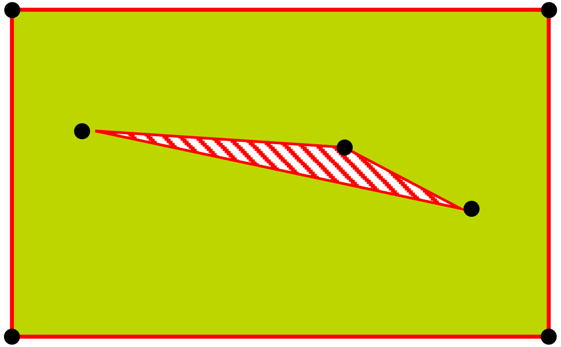
- Need an *outer boundary* of the workspace and some obstacles



Constrained Delaunay Triangulation

- Need an *outer boundary* of the workspace and some obstacles

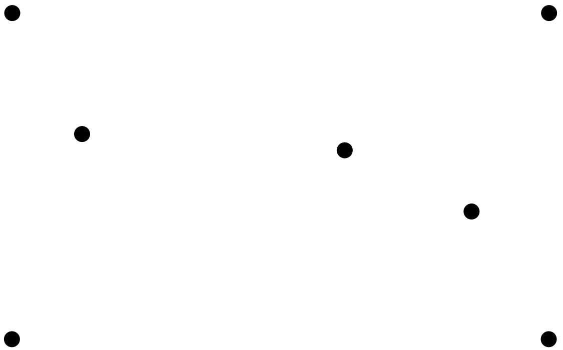
- Just take the vertices



Constrained Delaunay Triangulation

- Need an *outer boundary* of the workspace and some obstacles

- Just take the vertices

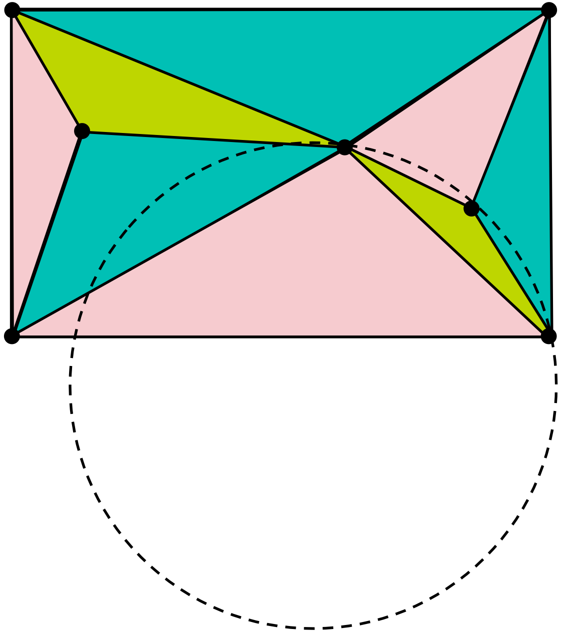


Constrained Delaunay Triangulation

- Need an *outer boundary* of the workspace and some obstacles

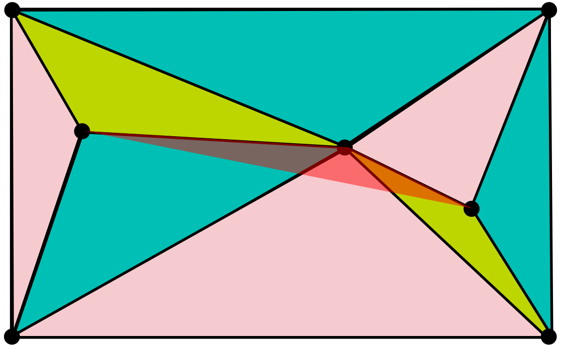
- Just take the vertices

- Delaunay triangulation
  - No points in circumcircle of any Δ
  - Off-the-shelf code



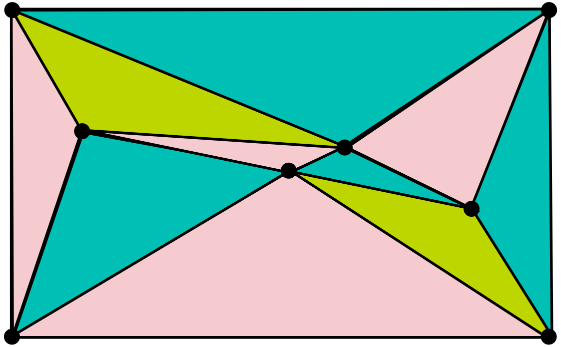
Constrained Delaunay Triangulation

- Need an *outer boundary* of the workspace and some obstacles
- Just take the vertices
- Delaunay triangulation
  - No points in circumcircle of any Δ
  - Off-the-shelf code
- Check obstacle edges included



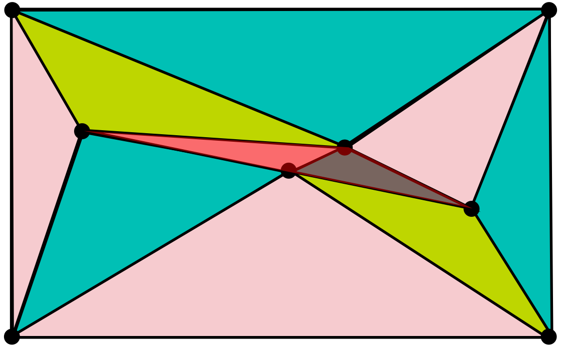
Constrained Delaunay Triangulation

- Need an *outer boundary* of the workspace and some obstacles
- Just take the vertices
- Delaunay triangulation
  - No points in circumcircle of any Δ
  - Off-the-shelf code
- Check obstacle edges included
- If not, insert midpoint of missing edge and repeat



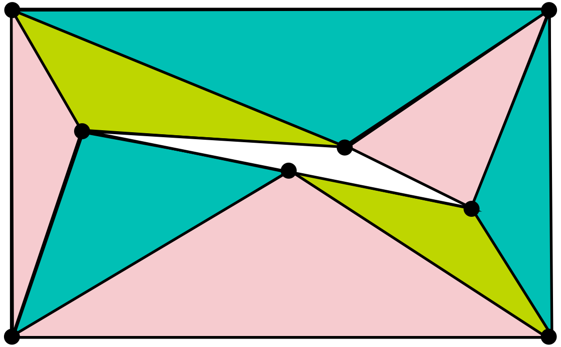
Constrained Delaunay Triangulation

- Need an *outer boundary* of the workspace and some obstacles
- Just take the vertices
- Delaunay triangulation
  - No points in circumcircle of any Δ
  - Off-the-shelf code
- Check obstacle edges included
- If not, insert midpoint of missing edge and repeat



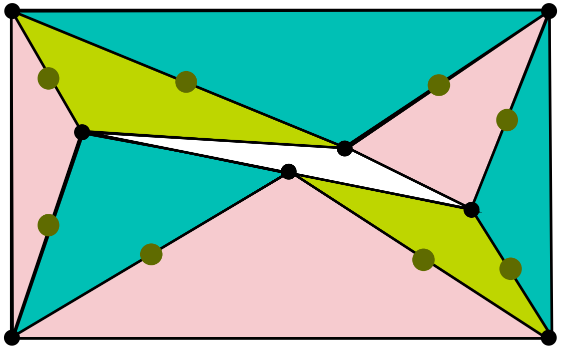
Constrained Delaunay Triangulation

- Roadmap made by joining up neighbouring triangles
  - Ignore triangles in obstacles



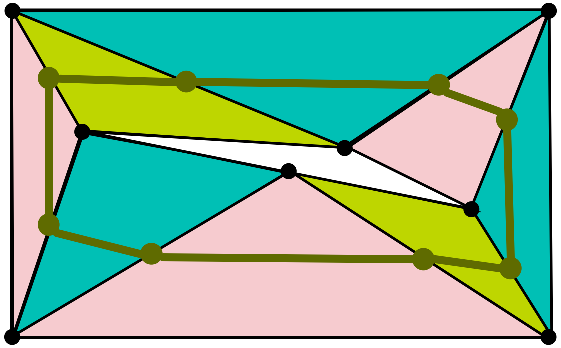
Constrained Delaunay Triangulation

- Roadmap made by joining up neighbouring triangles
  - Ignore triangles in obstacles
  - Take midpoint of every boundary between neighbours as entry/exit



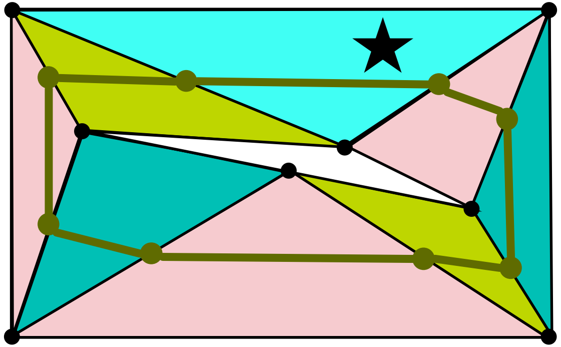
Constrained Delaunay Triangulation

- Roadmap made by joining up neighbouring triangles
  - Ignore triangles in obstacles
  - Take midpoint of every boundary between neighbours as entry/exit
  - Join up entry/exit points of each triangle



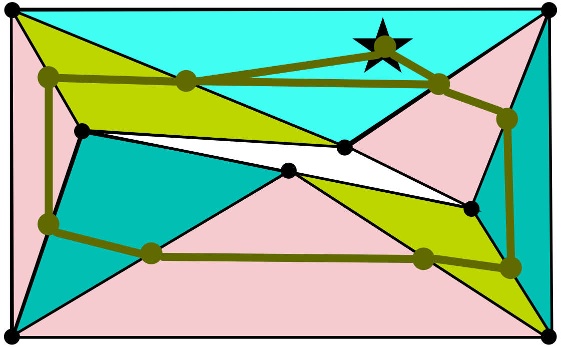
Constrained Delaunay Triangulation

- To *query* for a path:
  - Identify which triangle contains the goal



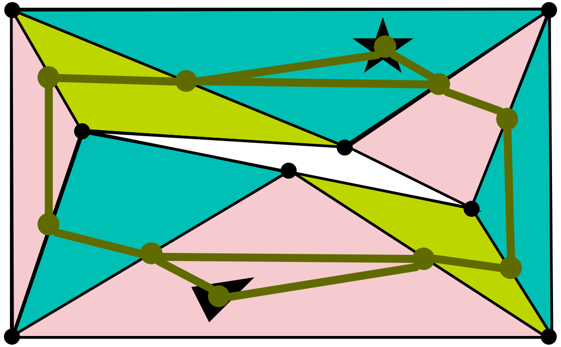
Constrained Delaunay Triangulation

- To *query* for a path:
  - Identify which triangle contains the goal
  - Connect goal to entry/exit points in that triangle



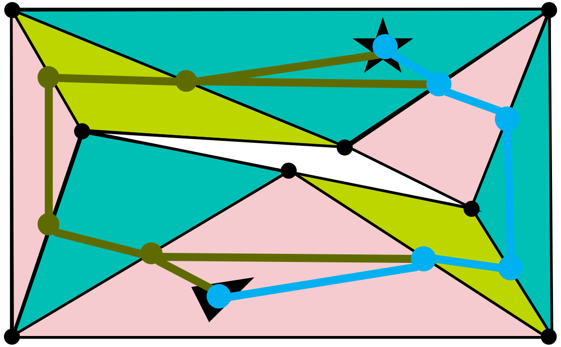
Constrained Delaunay Triangulation

- To *query* for a path:
  - Identify which triangle contains the goal
  - Connect goal to entry/exit points in that triangle
  - Same for start



Constrained Delaunay Triangulation

- To *query* for a path:
  - Identify which triangle contains the goal
  - Connect goal to entry/exit points in that triangle
  - Same for start
  - Graph search



Joining it all up

	Likes	Dislikes	Pros	Cons
Visibility graph	Primitives, meshes	Grids, point clouds	Optimal* paths	Heavy computation
Quadtree	Grids, point clouds	Primitives, meshes *but only mildly*	Flexible input	Longer paths, recursive computation
Delaunay	Primitives, meshes	Grids, point clouds	Efficient computation, paths away from obstacles	Longer paths

*Is anything ever really *optimal*? Only for the model it's applied to, which is already an approximation laden with assumptions. Perhaps better just to say "good".