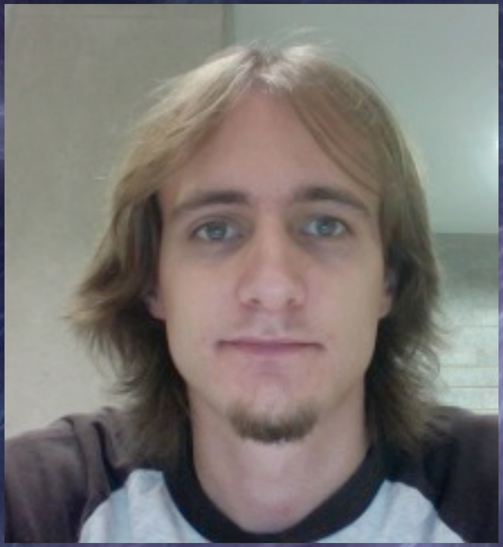
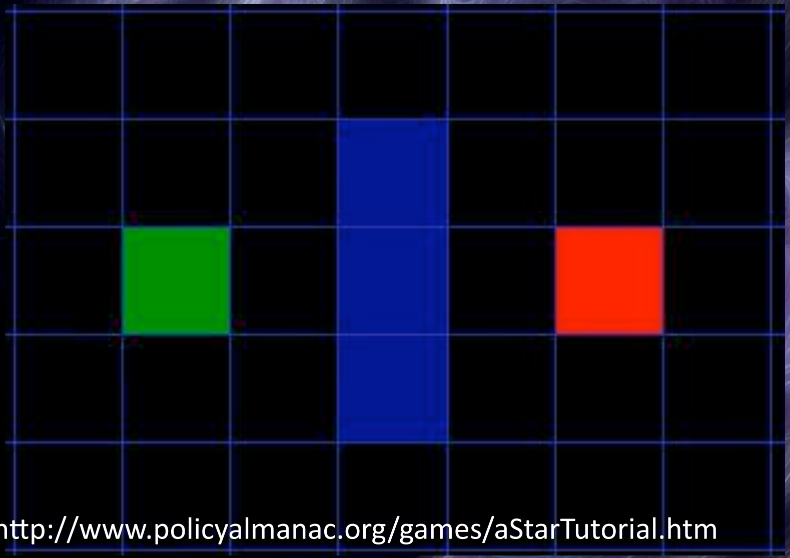
World Representation and Path Planning Using the A* Algorithm

By Alex Ufkes



Objective:

Find the shortest (or least weighted) path from point A to point B

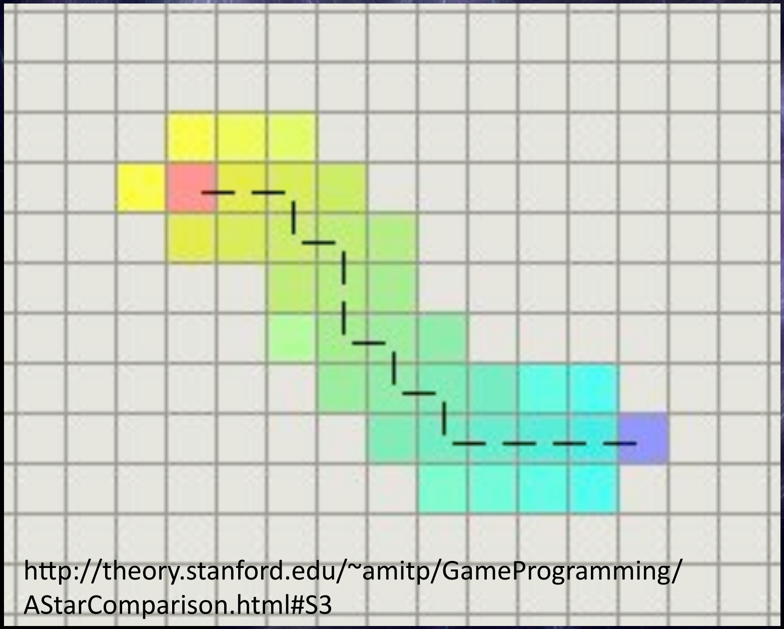

http://www.policyalmanac.org/games/aStarTutorial.htm

In this simple example, we want to go from the start point (green) to the end point (red) while avoiding the obstacles (blue).

We will use the A* (A star) algorithm to do this.

What is A* and how does it work?

- A* is a widely used pathfinding algorithm that finds a shortest path while using a *heuristic* to guide itself.


http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#S3

Terminology:

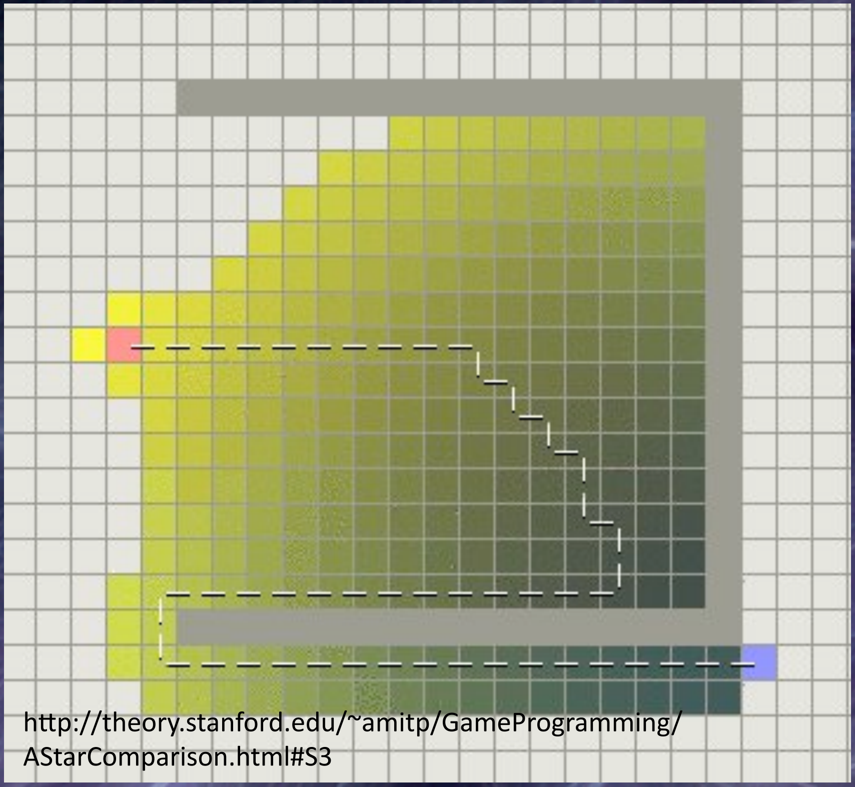
- G(n) is the distance (or cost) of the path from the starting point to any vertex n.

- H(n) is the heuristic estimated cost from any vertex n to the goal.

- F(n) is the value that A* will use to select the next step in the path. To find F(n), we simply add G(n) and H(n).

Calculating G(n) and H(n)

- G(n) is calculated by adding the cost of moving to the next square to the cost of getting to the current square.
- H(n) is basically a guess at how far away we are from the goal, and can be calculated in any number of ways.
- The method used will directly affect the speed and accuracy of the algorithm, so it is important to take care when determining H(n).

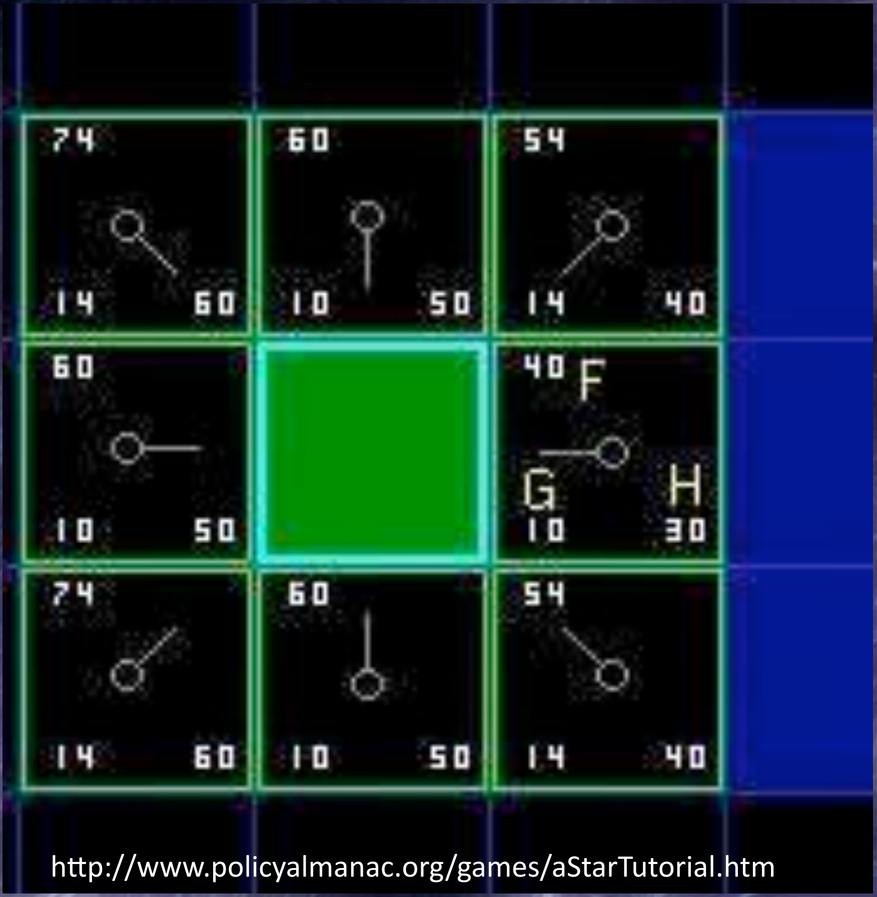

http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#S3

Back to our example…

- Here we see our starting point again, with G, H, and F values filled in for each adjacent square.

- In this case, each grid square is 10 units across. This means that G is 10 for directly adjacent squares, and 14 for diagonally adjacent squares. (sqrt(2) ~= 14/10)

- We use the Manhattan method to find H, which is the number of vertical and horizontal moves required to reach the goal, ignoring obstacles, and multiplying by 10.

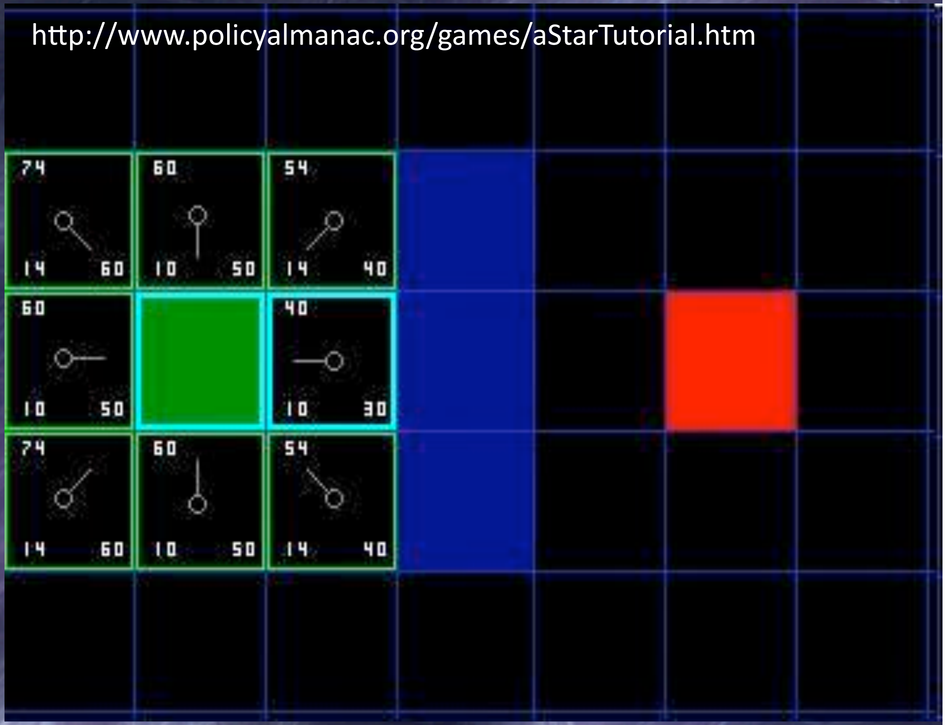


http://www.policyalmanac.org/games/aStarTutorial.htm

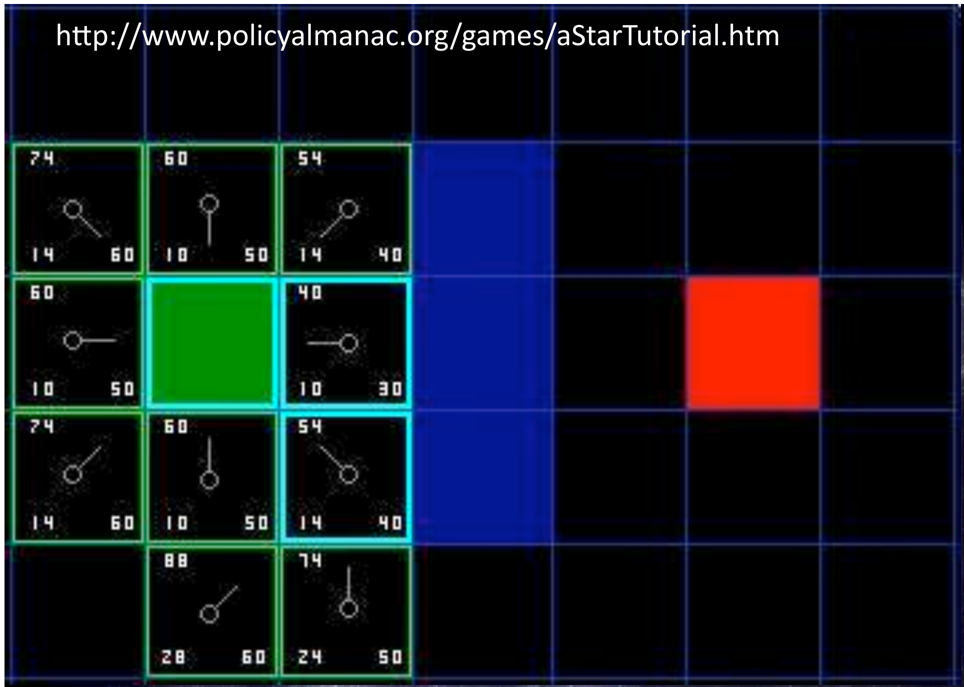
- The algorithm begins by creating an open and a closed list. The open list contains all the squares that can be reached from the current square, and the closed list contains all the squares that we don't need to bother looking at, such as the parent of the current square.

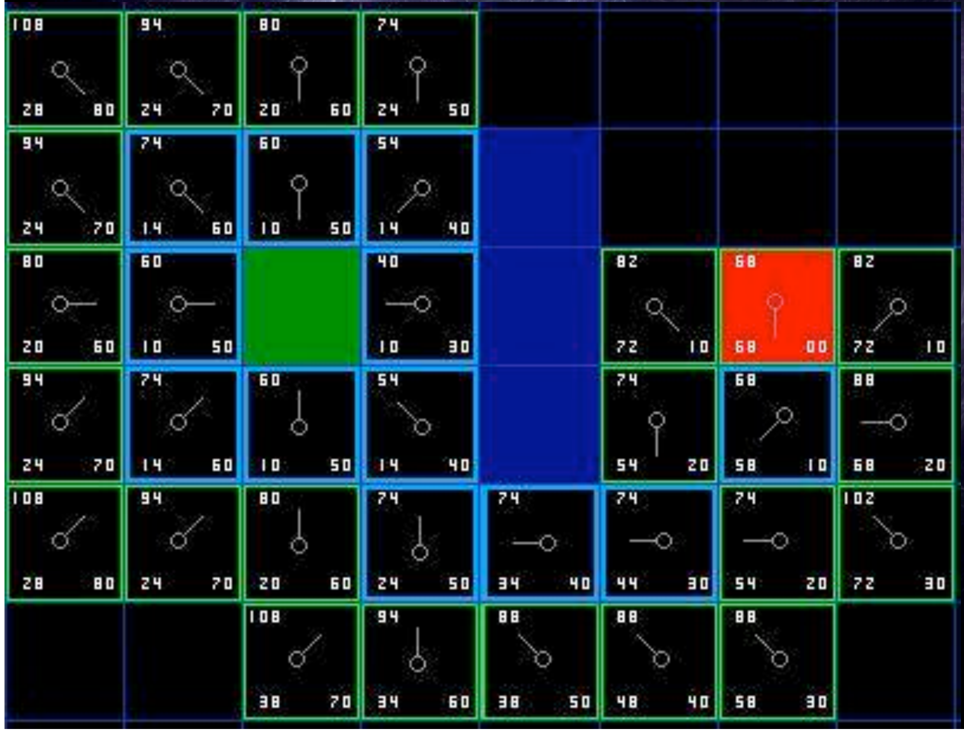
- We now pick from the open list the square with the lowest F score. This becomes our new current square, and the previous square is added to the closed list. The new square in this case is directly to the right.

- Now that we have a new current square, we need to check the squares adjacent to it. If the adjacent square is impassable or on the closed list, we ignore it. If the adjacent square is not on the open list, we add it. Finally, if the adjacent square is already on the open list, we check if this new path is shorter, and, if it is, we change that square's parent to the current square.



http://www.policyalmanac.org/games/aStarTutorial.htm

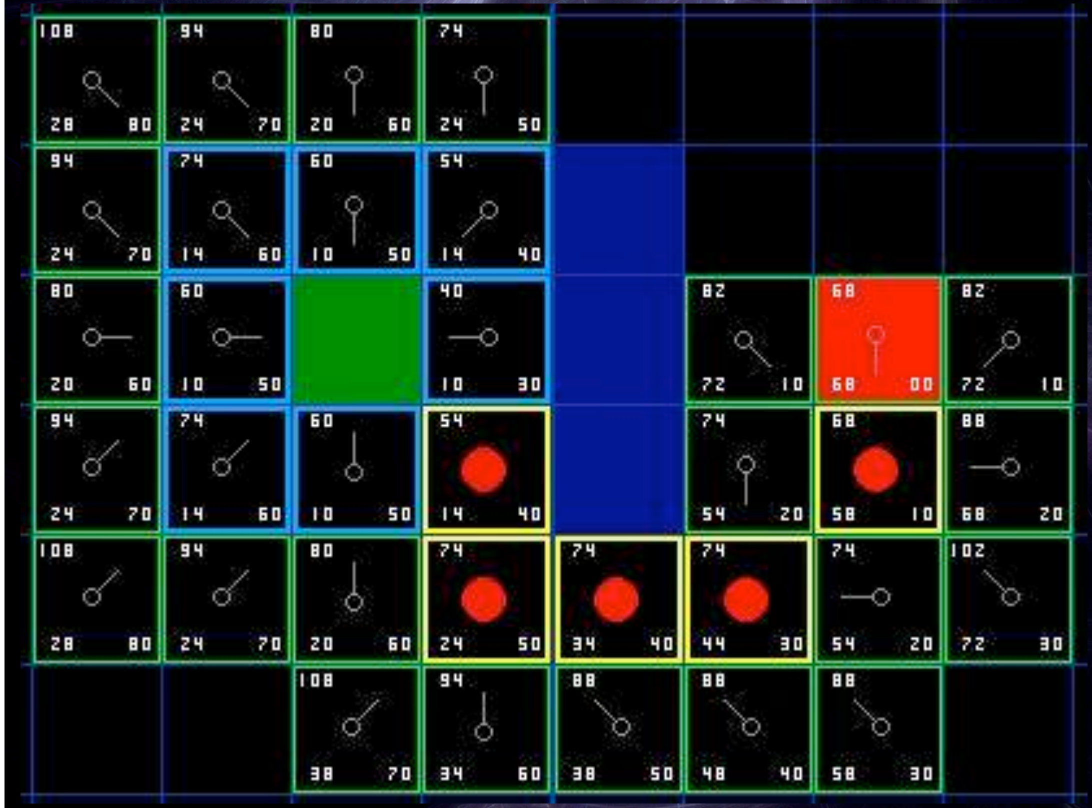




- The next iteration chooses the square below. Notice how the arrows point to that square's parent, and how they change as smaller paths are found.

- The algorithm ends when the goal square has been added to the closed list.

Finding the Path



http://www.policyalmanac.org/games/aStarTutorial.htm

- The smallest path is obtained by walking back through the list of parents starting at the goal square.

- Go from the goal square to its parent, then from that square to its parent, etc. Repeat this until we are back at the starting square.

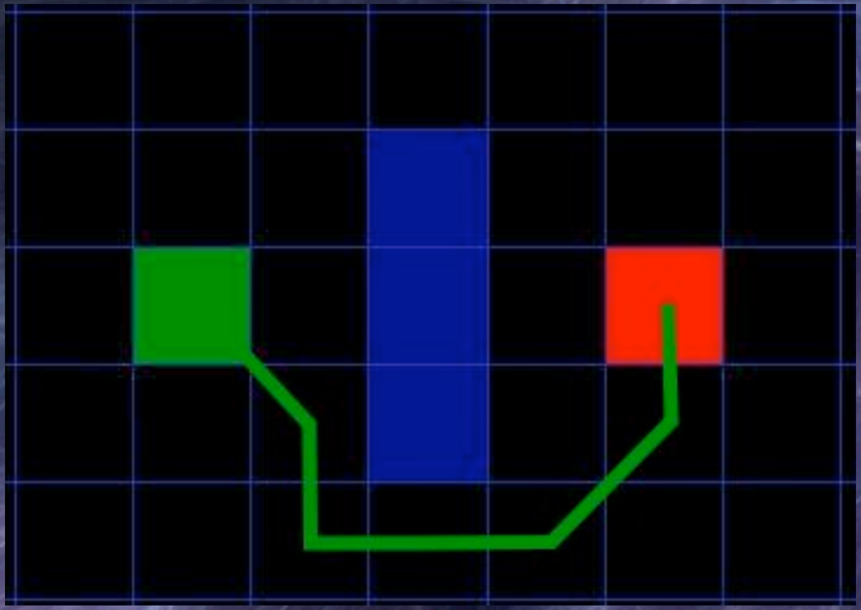
- With the right heuristic we will have found the shortest path.

Triangles?

- One way to improve on this result is to change the way we divide our search area.

- A square grid is the simplest way of dividing a space, but it does not always generate a smooth path, and requires a large number of nodes to be effective.

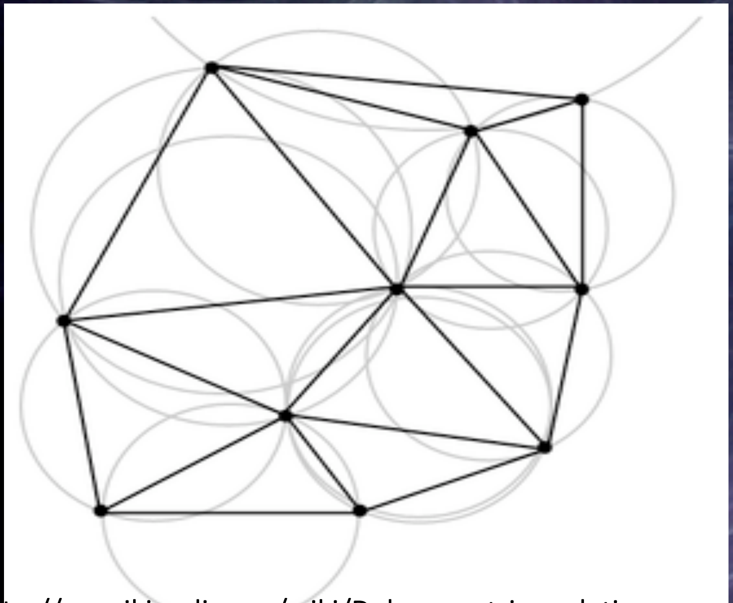
- Another option is to divide the search area into triangles. This provides an advantage because triangles can easily be constructed around any polygon, whereas fitting polygons into a square grid tends to be clumsy and unnatural.



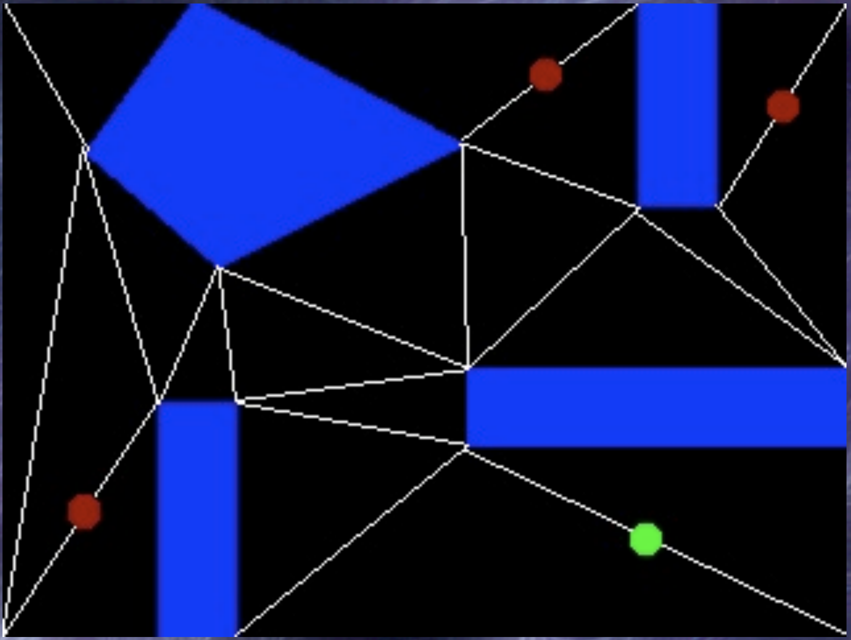
Delaunay Triangulation

- The Delaunay triangulation of a set of points is a triangulation such that no point lies within the circumcircle of any of the triangles.

- This is a very effective way of dividing the world. It maximizes the smallest angles and thus generates as few skinny triangles as possible.



tp://en.wikipedia.org/wiki/Delaunay_triangulation

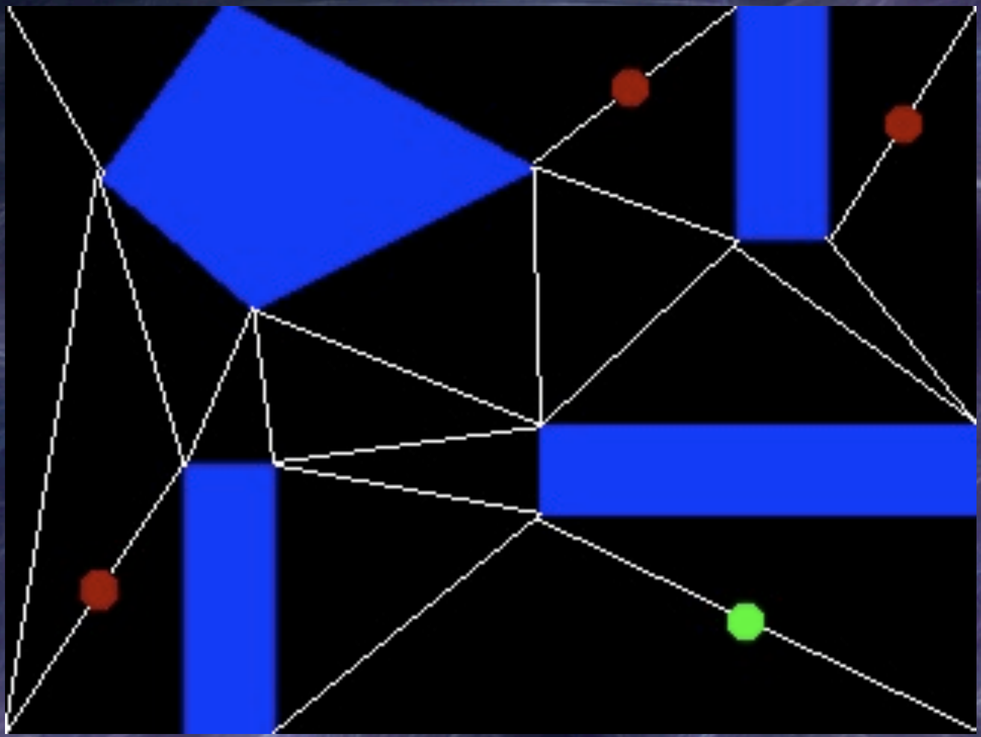


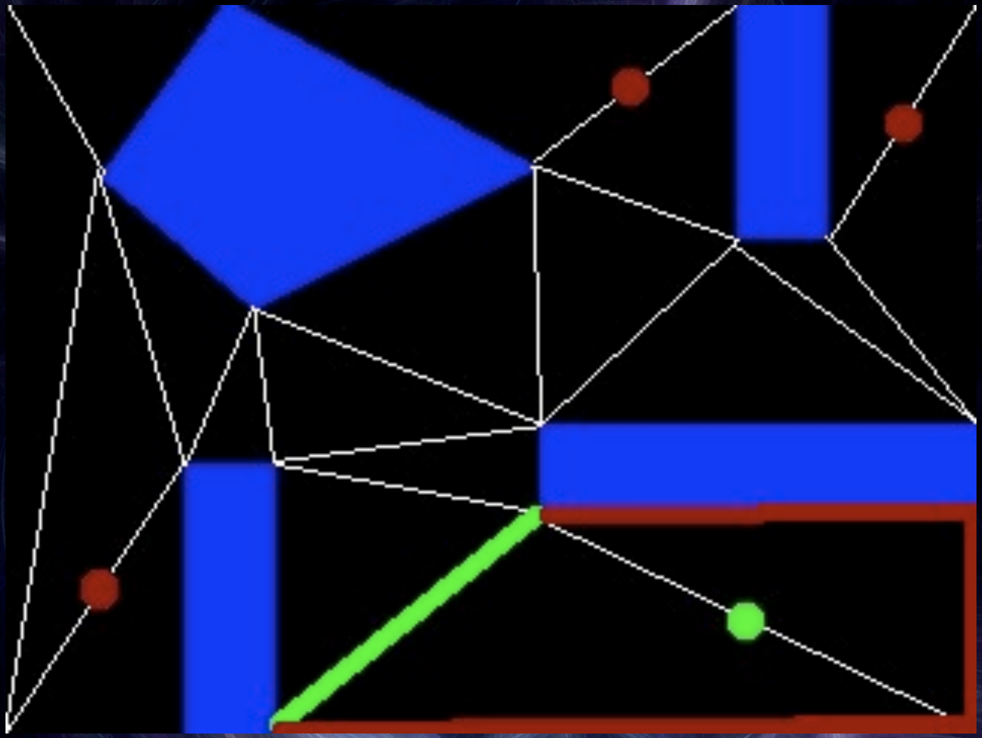
Triangulated Example

- The simplest way to traverse a triangulated world is to use the midpoints of the triangle edges as nodes.

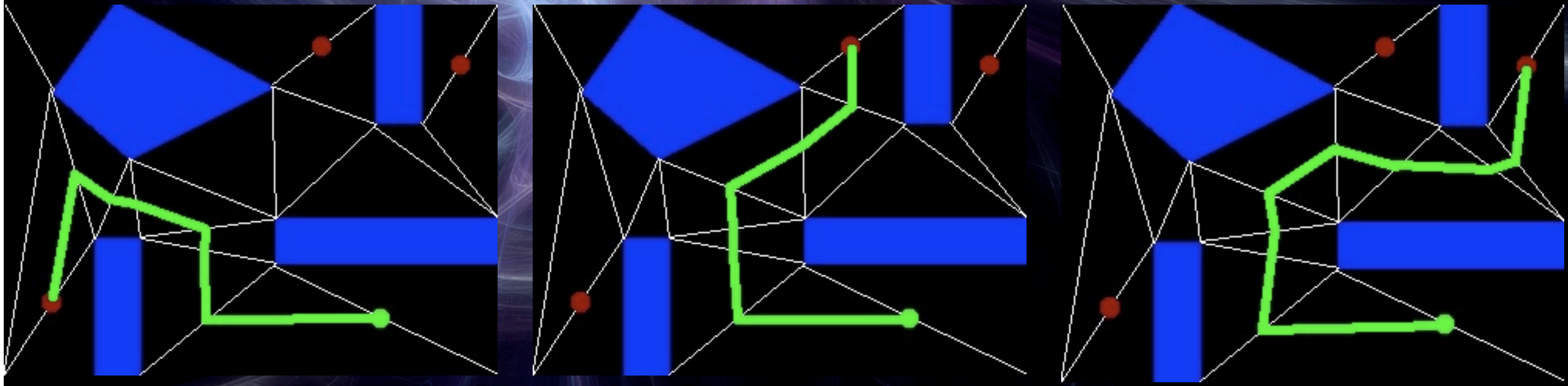
- G is calculated as an estimate of the distance traveled so far, and H is the Euclidean distance from the current node to any point on the goal triangle's entry edge.

- The A* algorithm can then be carried out in the exact same way as before.

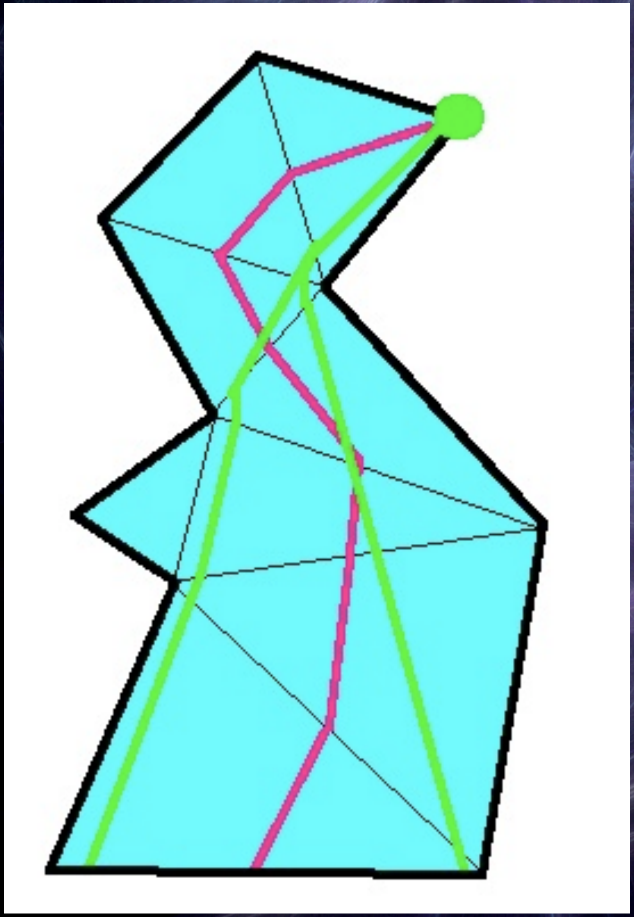




- Adjacent nodes are simply the edges that comprise the two triangles of which the current node is also an edge.
- The open and closed lists are maintained just as before, with closed edges being those previously visited and those that share an edge with an obstacle.



Triangulation-Based Pathfinding Cont.



- Using triangle edge midpoints as nodes produces a smoother path traversing fewer nodes than the grid-based method, but there is still room for improvement.

- Once the path along the midpoints is found, it creates a “channel” of triangles going from the start point to the goal. We now want to find the shortest path through this channel.

- The disadvantage here is that we don’t know the actual path we have taken until after the goal is found. Until then, we only have information about the channel itself.

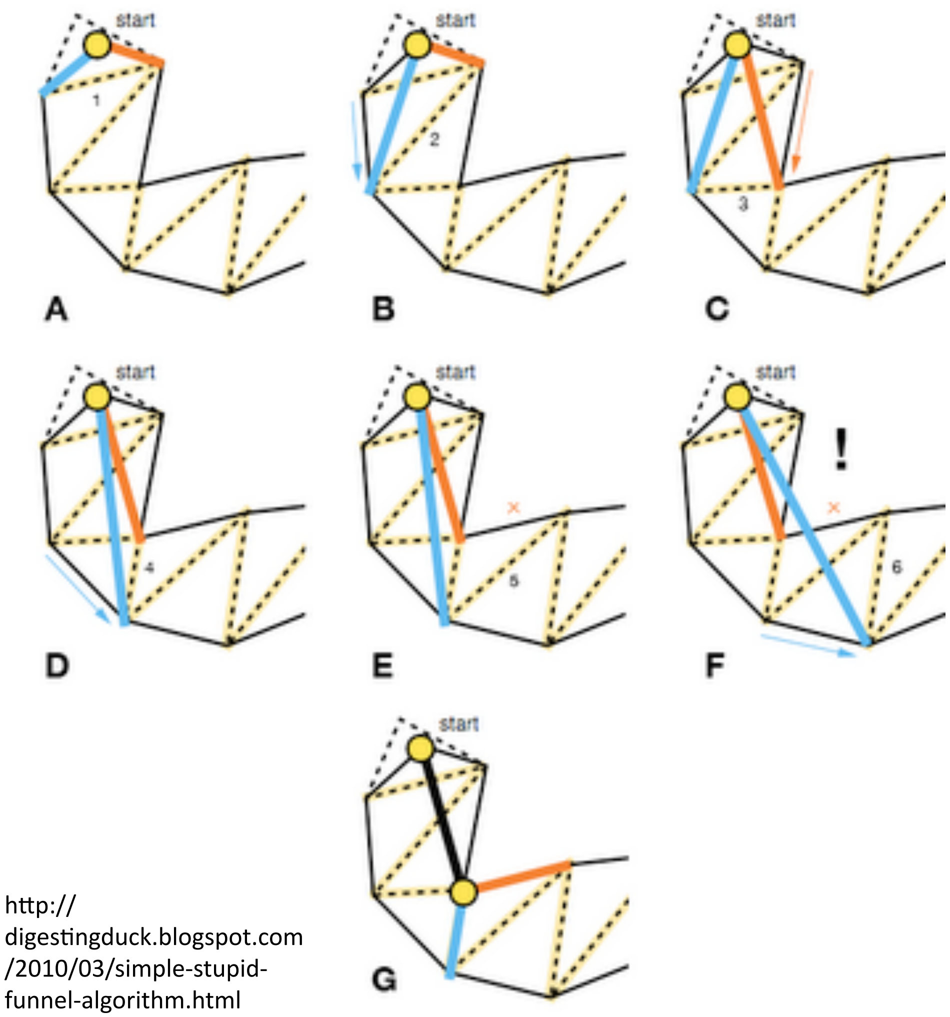
Funnel Algorithm

- Once we have our channel, we apply the funnel algorithm to find the shortest path through it.

- The algorithm essentially steps through the vertices of the channel that lie inside the initial funnel (A-D).

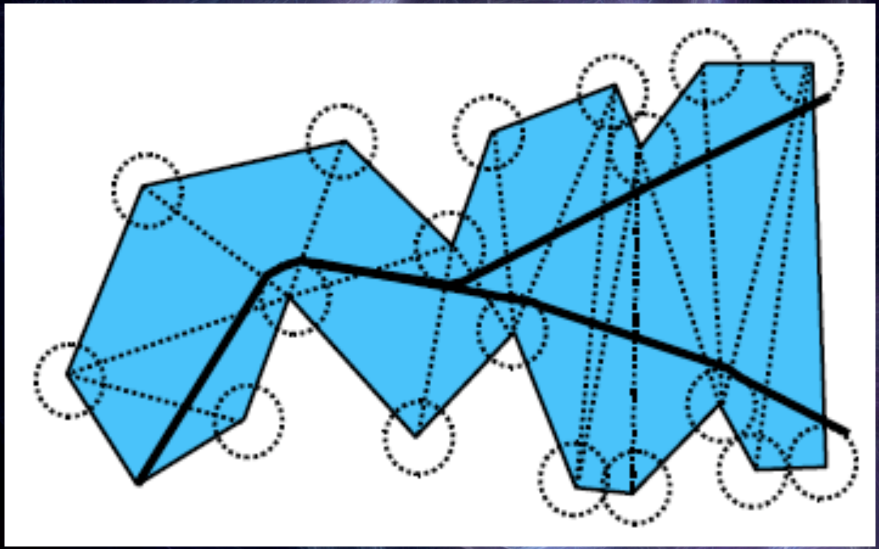
- Once one side can move no further, we continue moving the other side until it overlaps with the first (E-F).

- At this point, the endpoint of the edge that is being overlapped becomes the new apex, and the algorithm repeats (G).



http://digestingduck.blogspot.com/2010/03/simple-stupid-funnel-algorithm.html

Accounting for Size



- Mobile robots are not points, they have both width and depth.

- In order to avoid driving against a wall or corner, we can use a modified version of the funnel algorithm that leaves a specified amount of clearance.

- By drawing circles around every vertex and modifying the funnel algorithm to create paths that are tangent to these circles, we can ensure that a robot of non-zero size can safely travel this path.

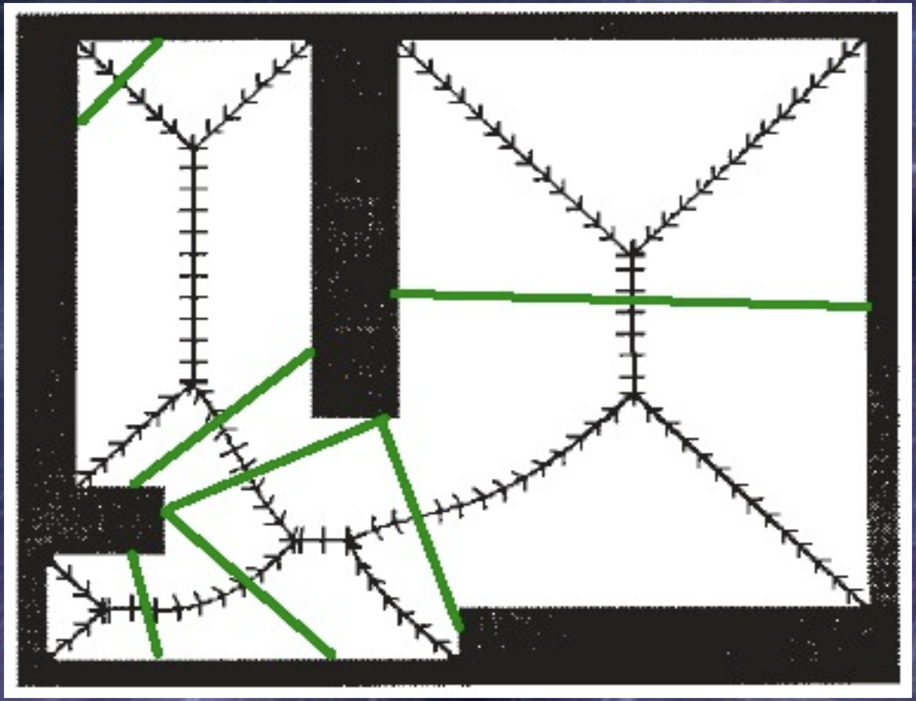
Voronoi Diagrams in Brief

- Generalized Voronoi diagrams and their corresponding graphs provide another tool that can be used in robot navigation and path planning.

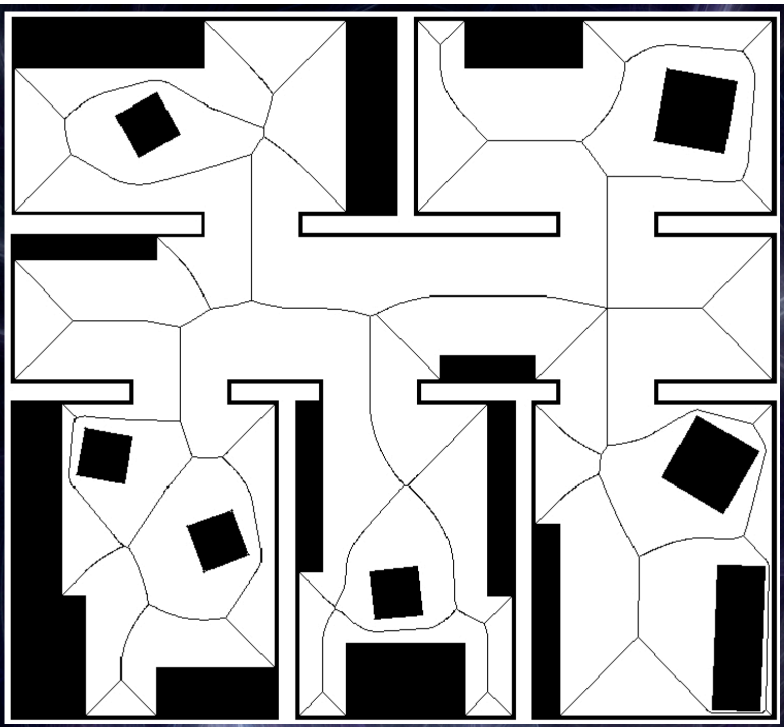
- The GVD is obtained by drawing all possible line segments that meet the following conditions:

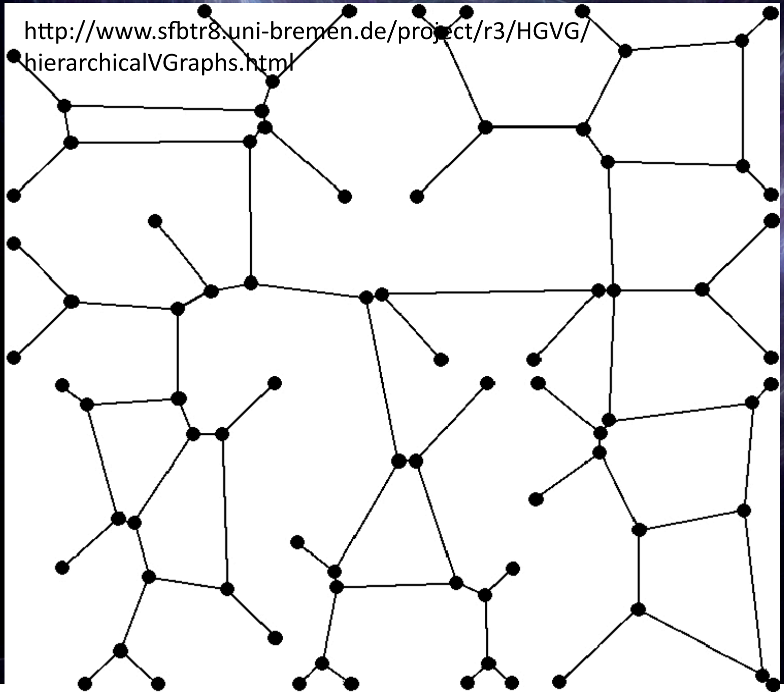
1. The line segment must be equidistant from two obstacle faces that are within line of sight of each other.

2. No part of the line segment may be closer to a third obstacle face than it is to the two from which it is equidistant.



- Green lines connect a few sample lines and their respective obstacle edges.





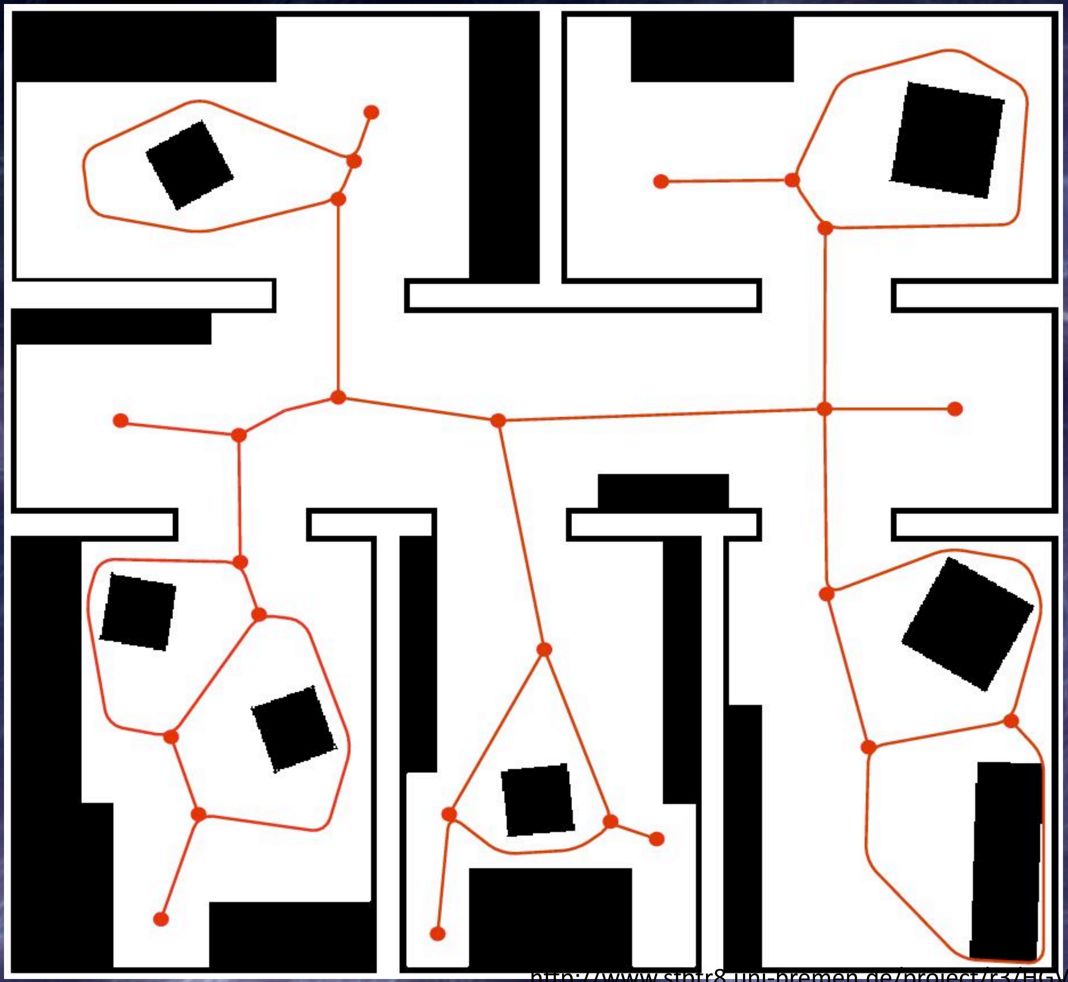
http://www.sfbtr8.uni-bremen.de/project/r3/HGVG/hierarchicalVGraphs.html

- A Generalized Voronoi Graph, or GVG, is created by placing a vertex at the endpoints and meeting points of the line segments in the GVD.
- Additional information such as curve description and length is added to the lines, and the distance from obstacles is added to the vertices.
- Given this information, we can assess the relevance of each vertex in the graph. For example, does the robot have clearance to reach this vertex, does the robot have any reason to travel to this vertex, etc.

Simplifying the GVG

- The value of a vertex can be determined in many ways, such as the degree of the vertex, the clearance on all sides of the vertex, etc.

- The end result is a simpler graph that ideally provides coverage to all areas of the map and provides enough clearance for the robot to travel freely within the world.



http://www.sfbtr8.uni-bremen.de/project/r3/HGVG/hierarchicalVGraphs.html

Further Reading

http://theory.stanford.edu/~amitp/GameProgramming/
http://www.policyalmanac.org/games/aStarTutorial.htm
http://en.wikipedia.org/wiki/Delaunay_triangulation
http://en.wikipedia.org/wiki/Voronoi_diagram
http://harablog.wordpress.com/category/pathfinding/
http://www.sfbtr8.uni-bremen.de/project/r3/HGVG/hierarchicalVGraphs.html