Euclidean Shortest Path with Obstacles

Veera Venkata Surya Subrahmanyam Pendyala
Indiana State University
Terre Haute, Indiana
spendyala@cs.indstate.edu

December 14, 2011

Abstract

This paper is to find the Shortest Path within the Euclidean space with obstacles. Finding a path within Euclidean space where an object can fit within obstacles, and can traverse to destination from source. Implementation of this project is as follows:

1. Make a grid with the width r, where r is the radius of an object.
2. Find the valid vertices in the grid, where it makes a connected graph G from source to destination.
3. Now taking the gird as $G(\mathcal{V}, \mathcal{E})$, where G is graph, $\mathcal{V}$ is vertex and $\mathcal{E}$ is edge.
4. Once the connected graph G is formed, next step is to find the shortest path within G.

1 Introduction

There are algorithms to find the shortest path in polynomial time, using algorithms like $Breadth-first\ search$ or $Depth-first\ search$ or $\mathcal{A}^*$ *algorithm* and by using other greedy or dynamic algorithms. Usually the process of finding the shortest path on $2-Dimensional$ space is the polynomial time algorithm.
In this paper we discuss about the Euclidean Shortest Path, this is a problem within computational geometry. The shortest path in Euclidean space is found by a path within the space and it should be the shortest path.

1.1 An Overview of the Process

The goal is to find the shortest path in a given Euclidean space. To find a path in Euclidean space with obstacles, we need a graph. Dividing the Euclidean space into a grid, where the width and height are equal to the radius r of an object, as shown in the Figure 1 below.

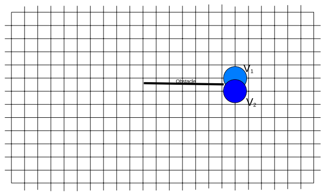

Figure 1: Example of an ESP problem.

2 What exactly is the problem?

Real-time problems occur while we need to find shortest path within the Euclidean space. Once the space is provided and obstacles are detected, to reach the goal we need a process within the shortest possible time. Here are examples of real-time applications:

- Shortest path between two cites when there are constrains.
- The shortest path for robot within a maze.

This paper proposed a method to find the shortest path within the Euclidean space. Below are the steps as follows:

- Formation of grid in Euclidean space.
- Find valid vertices within the grid.
- Build the graph using the valid vertices.
- Find the shortest path within the graph, from source to destination.

2.1 Formation of Grid.

Formation of a grid within the Euclidean space depends on the radius of the object. Considering, the object to be a sphere and equal size in all directions. Design the grid with arbitrary values of width as shown in the Figure 2.

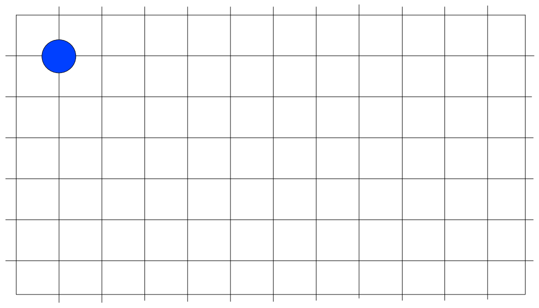

Figure 2: Arbitrary values of grid example 1.

In the Figure 3, the width of the grid is larger than the radius of an object. So, the obstacle is not within it's radius r of an object, this process of wrong validation of vertices creates wrong edges in a graph G. Here is another good example, Figure 4, when the width of the grid is reduced and it is still greater than the radius r of the object. It is clearly shown that the edge of the vertex is not detected and the object cannot move from V_1 to V_2. Thus, it cannot form an edge between these two vertices.

To overcome this error in validating the vertices we need to reduce the width of the grid less than or equal to the radius r of the object.

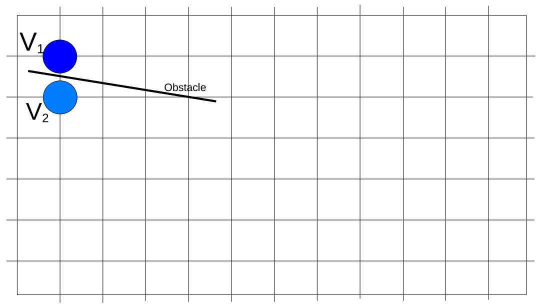

Figure 3: Arbitrary values of grid example 2.

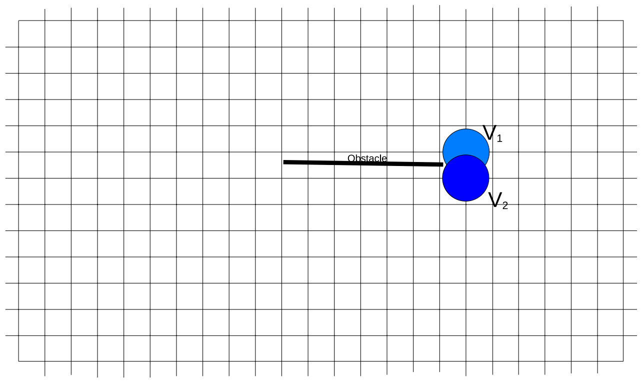

Figure 4: Arbitrary values of grid example 3.

2.2 Finding valid vertices in the grid.

In our example we have a grid, let's consider cross-sections of the grid to be our vertices. To find a valid vertex, the object needs to be fit at the vertex without any obstacle within radius r. The result as shown in Figure 5 and procedure to find the valid vertices are as follows.

1. Set $True$ as default values to all the vertices in the grid.
2. Get the range of x *coordinates* and y *coordinates* of every obstacle.
3. Find the valid points within the range of both x and y coordinates for every obstacle.

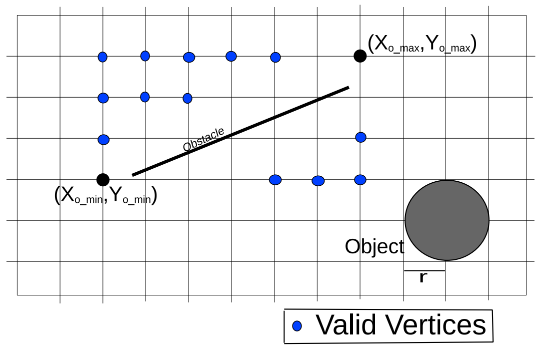

Figure 5: Valid vertices as output result shown in blue.

2.2.1 Algorithm to find the valid vertices in the grid.

```
*Finding valid vertices procedure*
begin
  $valid\_vertices$ :={ };
  for $i := X_{min}$ to $X_{max}$ step $grid\_width$ do
    for $j := Y_{min}$ to $Y_{max}$ step $grid\_width$ do
      $valid\_vertices[(i, j)] := True$;
    od
  od
  for $obstacle := 1$ to $n$ step $next\ obstacle\_list$ do
    $GET\ X_{o\_min}, X_{o\_max}, Y_{o\_min}\ Y_{o\_max}$;          # min and max of the obstacle co-ordinates;
    for $x := X_{o\_min}$ to $X_{o\_max}$ step $grid\_width$ do
      for $y := Y_{o\_min}$ to $Y_{o\_max}$ step $grid\_width$ do
        $dist := from\ point\ (x, y)\ to\ obstacle$;
        do if $dist < radius\ of\ object$ then exit fi;
          $valid\_vertices[(i, j)] := False$;
        od
      od
    od
  od
end
```

2.3 Building the graph.

Once we got the valid vertices of a grid, we need to build a graph G. This can be done by joining all the valid vertices. The valid vertices as set $\mathcal{V}$ and by joining all the vertices in $\mathcal{V}$, $\mathcal{E}$ is obtained. We get the graph $G(\mathcal{V}, \mathcal{E})$.
Firstly, as shown in Figure 6 with all the vertices in the grid are said to be

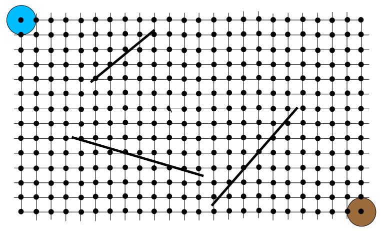

Figure 6: Graph with default values of valid vertices.

valid and after finding the valid vertices within the grid we get Figure 7. Connecting the valid vertices gives us the graph $G(\mathcal{V}, \mathcal{E})$

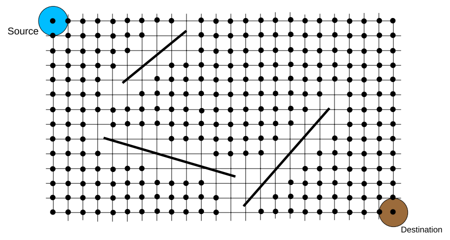

Figure 7: Graph with exact valid vertices.

as shown in Figure 8 is the result after joining the valid vertices in the grid.

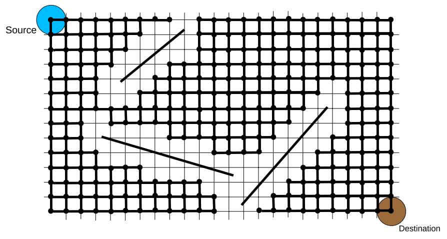

Figure 8: Graph $G(\mathcal{V}, \mathcal{E})$ in a grid without touching the obstacles.

2.4 To find the path within the graph $G(\mathcal{V}, \mathcal{E})$

Using any of the shortest path algorithms we can find the *path* within the graph $G(\mathcal{V}, \mathcal{E})$. After finding the shortest path here is the result in Figure 9.

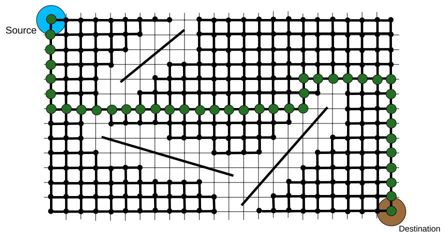

Figure 9: Path in the graph shown as green high-lighted vertices.

A Appendix: Euclidean Shortest Path with Obstacles using Python GTK.

An application developed using PyGTK. To demonstarte, the process of how the application works. The step by step process is discussed below. Here is the link to the code.
`http://cs.indstate.edu/~spendyala/pdf/esp.py`

B Appendix: Output of the program.

Here are the screen-shots of the program. To show the process of the software designed, when we start our program the first screen is as Figure 10 is shown.
After that, we need to check for the option"Create Obstacles". Once the option is selected, we can create obstacles in the drawing area. For every two clicks, an obstacle is created as shown in Figure 11.

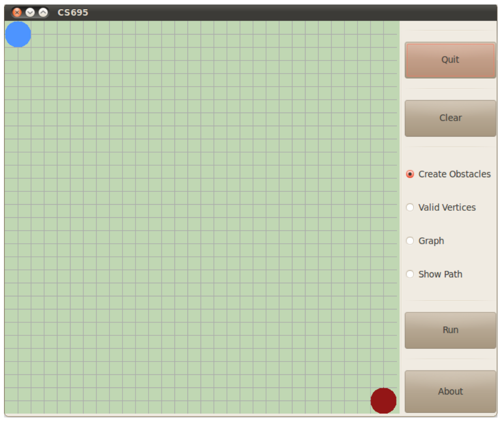

Figure 10: Screen-shot while starting the application.

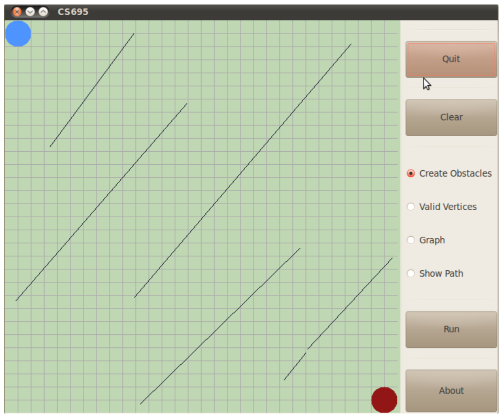

Figure 11: Screen-shot after creating obstacles.

Once the obstacles are created, change the option to“Valid Vertics” and find the valid vertices within the grid, and hit “Run” to continue. Then the output is shown in Figure 12.
To build a graph using valid vertices we need to select the option “Graph” and hit “Run” to continue. Result shown below in Figure 13

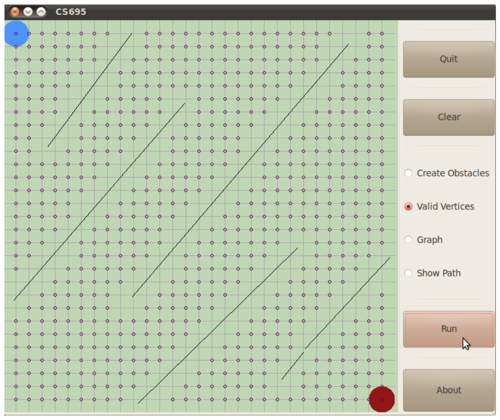

Figure 12: Screen-shot after generating valid vertices.

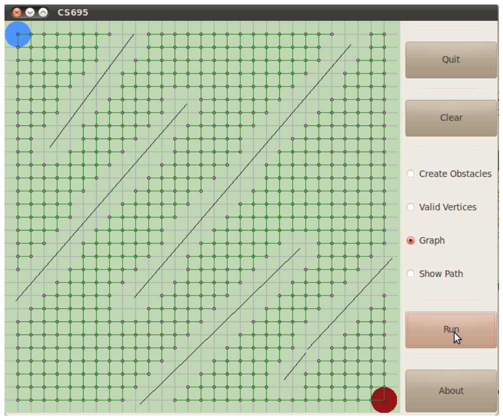

Figure 13: Screen-shot of graph of the Euclidean space with obstacles.

Change the option to “Path” and hit “Run” to get the path shown in Figure 14
If there is a disconnected graph from source to destination, it generates a dialogue box saying “Sorry, Path not found !”.

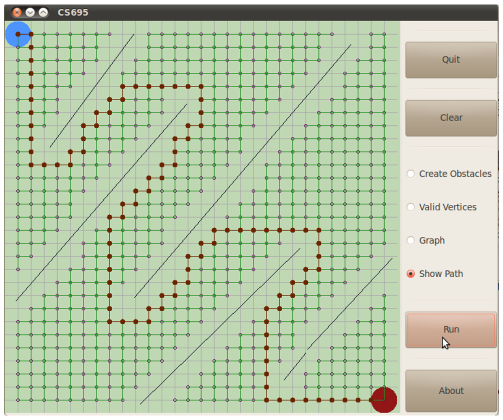

Figure 14: Screen-shot of graph of the Euclidean space with obstacles.

References

[1] Franz, Aurenhammer and Rolf, Klein
http://www.pi6.fernuni-hagen.de/publ/tr198.pdf
Voronoi Diagrams.

[2] *http://en.wikipedia.org/wiki/Euclidean_shortest_path*

[3] *http://en.wikipedia.org/wiki/Breadth-first_search* *Breadth-first Search*

[4] Li,Fajie and Klette, Reinhard *Euclidean Shortest Paths in a Simple Polygon* Computer Science Department, The University of Auckland, Auckland, New Zealand

[5] Kapoor, Sanjiv and Sachindra N. Maheshwari and Joseph S. B. Mitchell
http://www.ams.sunysb.edu/~jsbm/papers/sp-holes.pdf
An Efficient Algorithm for Euclidean Shortest Paths Among Polygonal Obstacles in the Plane

[6] *http://en.wikipedia.org/wiki/A⋆_search_algorithm* $\mathcal{A}^*$ Algorithm