Journal of Theoretical and Applied Information Technology
30th September 2017. Vol.95. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

THE SHORTEST PATH FROM SHORTEST DISTANCE ON A POLYGON MESH

JINHYUNG CHOI, BO ZHANG, KYOUNGSU OH

Soongsil University, 369 Sangdo-ro, Dongjak-gu Seoul, Department of Media, South Korea

E-mail: jinham2049@gmail.com, zhangbo0037@qq.com, oks@ssu.ac.kr

ABSTRACT

We can use a Dijkstra algorithm to calculate shortest paths on a polygon mesh. However, if the number of vertices of a polygon mesh is large, the efficiency of conventional path searching method that using Dijkstra algorithm is low. In order to solve this problem, this paper proposes a method to search shortest distance and path by using the A* algorithm. According to an experiment with the bunny model with 2,503 vertices, the efficiency of searching the shortest path is improved approximately 80% than conventional methods.

Keywords: *A* algorithm, Dijkstra algorithm, Polygon Mesh, Shortest Distance, Shortest Path.*

1. INTRODUCTION

With the development of computers these days, most of case are processing calculations quickly. However, if the processing of the most basic mesh in 3D is slow, perhaps it will improve other areas without solving the most important problems. So we tried to figure out how to handle the best possible speed when process with the most basic polygon mesh in 3D. Recently, the field of computer graphics modeling has attracted many people. For example, in paper [1], presented a new approach that how to using a signal processing ideas, marked some specific vertices, to efficiently smooth and editing surfaces of polygon mesh. This research significantly improves the existing fairness-norm optimization approaches of editing 3D mesh. In paper [2], author Highly summarized about searching moving path on 3D space by using A* and Dijkstra algorithm[3], this novel work shows that A* algorithm is more efficient and better than Dijkstra in path searching problem, and indicated that A* algorithm is the best solution in solving the shortest pathfinding problem.

We find the shortest and shortest path of two points in a triangular mesh in 3D space. When looking at Daikstra, we found that many unnecessary calculations are performed on inadequate parts, such as open spaces in triangulated meshes. Therefore, we propose an optimized algorithm using A * algorithm.

This paper presents a method for finding the shortest path in a polygon mesh. This is useful for areas such as polygon mesh surface texture mapping, editing, and more recently developed computer graphics techniques [4, 5, 6] and methods for shortest path search in artificial intelligence. Shortest path on a polygon mesh will be useful for fields such as computer graphics or artificial intelligence.

For example, when we divide a polygon mesh from two user selected position, the shortest path between those points is a good option for division path. In addition, it will be useful for 3D game AI functions such as character automatic walking, obstacle avoidance, auto driving, and automatic maze escape and so on.

Melvær'et al suggested a shortest path algorithm on polygon mesh [7]. They used Dijkstra algorithm to find the shortest path from one source point to all vertices. However, because they used using Dijkstra algorithm, there performance have limitation.

We present a shortest path finding algorithm based on A* algorithm. Instead of finding a path from one source position to all destination points, we find a path between one source position and one destination point.

This method is divided into two steps as a whole. The first step is the forward process, which allows to calculate the shortest distance. Then, the second step is backward process, which allows to calculate the shortest path. This method have more advantage than the previous method when the number of vertices of the polygon mesh is high.

In the second section of this paper we will look at the outline of research. The third section will explain how to find the shortest distance and path.

Journal of Theoretical and Applied Information Technology
30th September 2017. Vol.95. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

And in the fourth paragraph we will compare and show our experimental results with Dijkstra algorithm. The last five paragraphs present our conclusion and discussion.

2. RESEARCH OUTLINE

We want to find the shortest distance and path between two vertices on a polygon mesh. Polygon mesh is the most basic representation to represent objects in computer graphics. In 3D games, characters and various 3D objects are represented by various polygon meshes [8]. A polygon mesh consists of a number of vertices. An edge connect two vertices and a face is composed of three or more edges. Figs. 1 and Figs. 2 are polygon mesh Bunny with 2,503 and 35,947 vertices respectively. It is the same rabbit shape. With more number of vertices, we can represent original shape more accurately but operations or calculations on polygon mesh become more expensive.

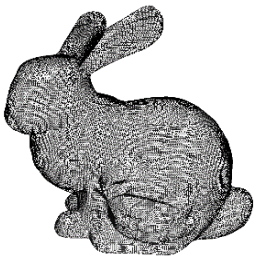

Figure 1: Rabbit model with 35947 Vertices.

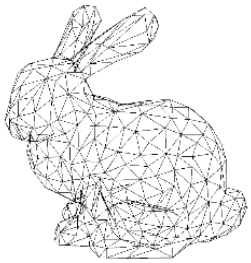

Figure 2: Rabbit model with 2503 Vertices

For example, in the case of Figs. 3, there are two vertices on the surface of a polygon mesh Bunny. We want to find shortest path on surface of polygon mesh between two points. This paper discusses how to find the shortest distance and shortest path between two vertices S and D on a polygon mesh by using the A * algorithm.

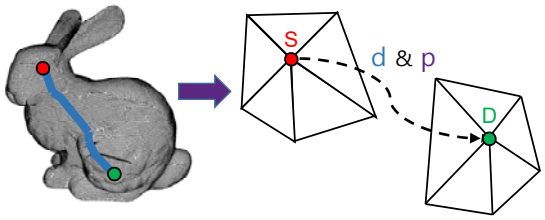

Figure 3: Two points on polygon mesh

The A * algorithm is a general graph search algorithm that finds the shortest path consisting of nodes and edges [9]. In our problem, the graph is defined by each vertex as a node and the edge on the mesh (the line connecting the two vertices) as the edge. This expresses the shortest path length (price) through a node n in the A * algorithm.

$$F(n) = G(n) + H(n) \quad (2.1)$$

G is the length of the calculated shortest path from the starting vertex to the n, H is the estimate of the shortest distance from the n to the target vertex, and F is the sum of G and H. A* algorithm starts from source node. Candidate set is set of nodes to be selected next. At first, souce node is added to candidate set. In each iteration, a node in open set with minimum F value is selected and move to visited set. Then neighbor of selected node is added to candidate set. The iteration continue until destination node is selected from candidate set.

Figs. 4 shows a simplified representation of the A * algorithm compared with the Dijkstra algorithm. The Dijkstra algorithm proceeds all the way from the starting vertex when searching for the shortest distance, but A * is a more efficient algorithm because it looks toward the target vertex like the black arrows in Figs. 4. The overview of our algorithm is as shown in Fig. 5.

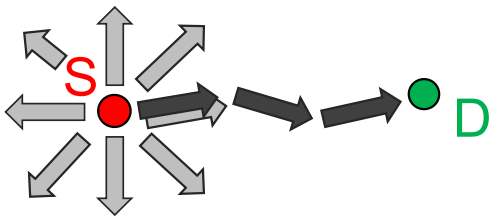

Figure 4: Comparison of A and Dijkstra Algorithms*

Journal of Theoretical and Applied Information Technology
30th September 2017. Vol.95. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

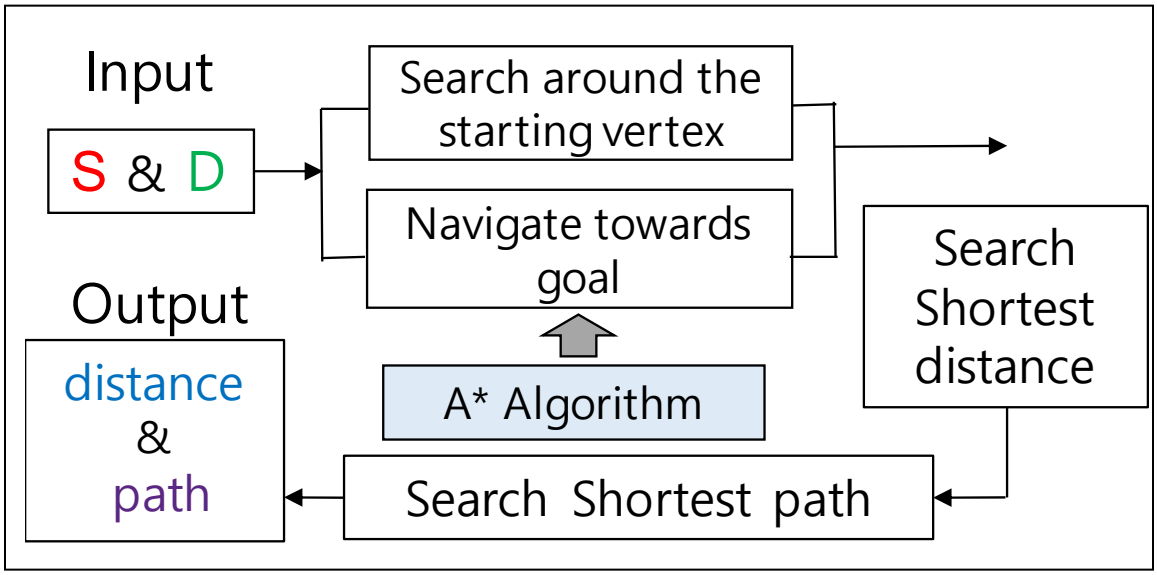

Figure 5: Research composition diagram

3. SHORTEST DISTANCE AND PATH

Let us assume that there are two vertices S (starting vertex) and D (target vertex) on a polygon mesh M. If the i-th point on the polygon mesh is N_i, its coordinates and other information can be defined as follows.

$$N_i(x_i, y_i, z_i, g_i, h_i) \in M, (i = 1,2,\dots n \in N^*) \quad (3.1)$$

x_i, y_i, z_i are the three-dimensional coordinate values of the vertex N_i, g_i is the length of the shortest path between N_i and the starting vertex S on polygon mesh , h_i is the straight line distance between, N_i and the target vertex D . Then information of S and D are as follows. h_S is the three-dimensional straight line distance between S and D.

Starting vertex: $S(x_S, y_S, z_S, 0, h_S)$

Target vertex: $D(x_D, y_D, z_D, g_D, 0)$

3.1 Start search from the starting vertex

We start search from starting vertex S. Get neighbor vertices around the start vertex S, calculates each coordinates and information on them, and stores them in the candidate set Q. In this paper, F_i is calculated as follows.

$$F_i = g_i + h_i \quad (3.1.1)$$

h_i can be easily obtained by the distance formula between two points in 3D space. The vertex with the smallest F value is selected.

$$h_i = \sqrt{(x_i - x_D)^2 + (y_i - y_D)^2 + (z_i - z_D)^2} \quad (3.1.2)$$

For example, in Figs. 6 there are five vertices $N_1 \sim N_5$ around the start vertex S. F_1 is smallest among F values ($F_1 < F_2, F_3, F_4, F_5$), the vertex N_1 is selected as next vertex. Herein, g_1 (The shortest distance on polygon mesh) which is the information of the vertex N_1, is the straight line distance between the starting vertex S and itself. $g_2 \sim g_5$ are calculated in the same way.

$$g_1 = \|S - N_1\| \quad (3.1.3)$$

After performing the neighbor search, the visited set P and the candidate set Q are as follows. S who has already visited does not visit again.

$$P = \{S\}, \; Q = \{N_1, N_2, N_3, N_4, N_5\} \quad (3.1.4)$$

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

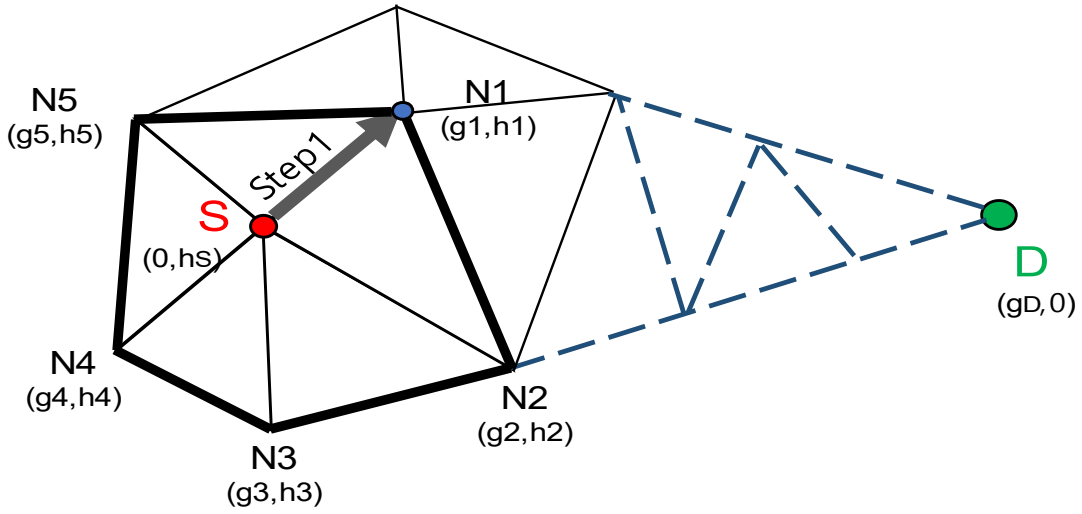

Figure 6: Navigate around the starting point

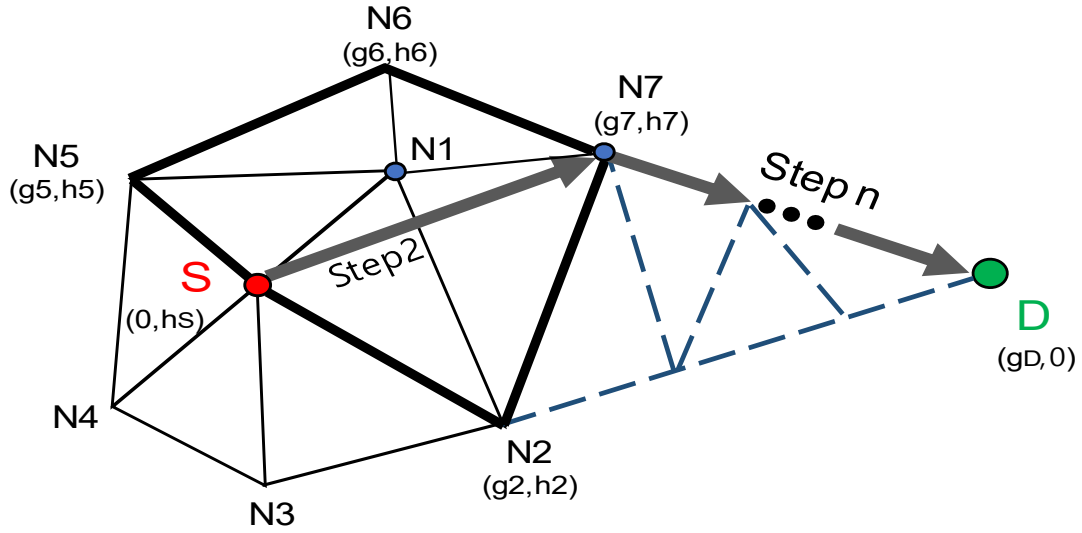

*Figure 7: Step of A * algorithm search*

3.2 Navigate towards goal

If the selected vertex N_i is calculated after the search of the start vertex S is completed, the vertex N_i becomes the new center point. Again, A * algorithm is applied to search for the final target vertex.

In Figs. 7, there are five vertices around N_i: vertices S, N_2, N_3, N_6, N_7. The F_6 value of the vertex N_6 and the F_7 value of N_7 should be newly calculated. If F_7 is the smallest among the F values ($F_7 < F_6$, F_2, F_3, F_5), the vertex N_7 is selected as next node. This is repeated until we arrive at the final target point D.

In Figs. 8, It shows the process of step 3 in the same way as the above. As a result, we go ahead and search for N_6.

3.3 Calculate shortest distance

Let's think about calculating vertex g_7. We want to calculate shorted distance from S to vertex N_7 on polygon mesh. We started from S and currently selected vertex is N_1. There are two triangles $\Delta N_1N_7N_2$ and $\Delta N_1N_7N_6$. The shortest path is from S to N_7 through one of these triangles

Journal of Theoretical and Applied Information Technology
30th September 2017. Vol.95. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

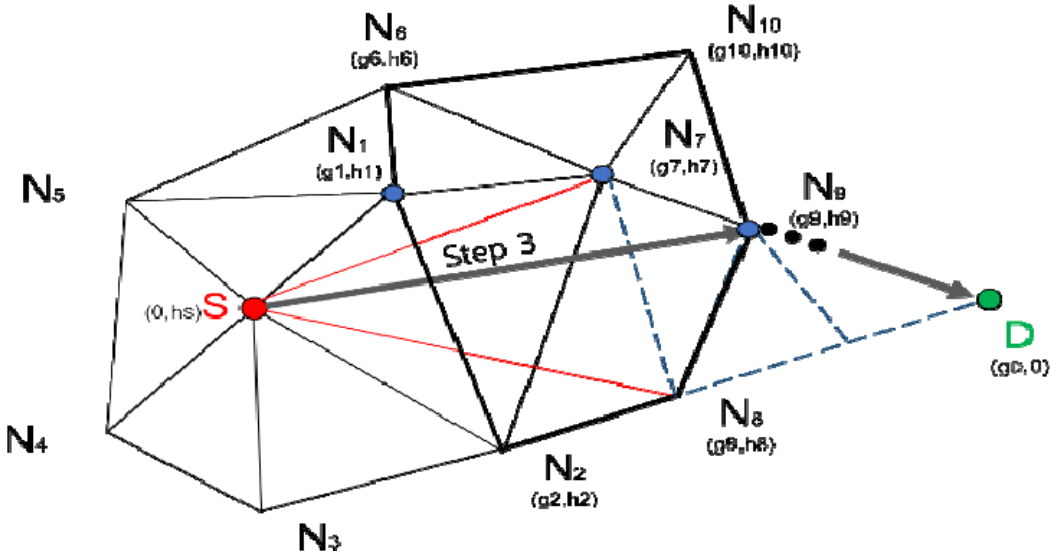

*Figure 8: Step 3 of A * algorithm search*

Let's think about the shortest path through $\Delta N_1N_7N_2$. The two triangles ΔSN_1N_2 and $\Delta N_1N_2N_7$ on the polygon mesh have a common edge N_1N_2, and the two triangles are fixed on the mesh and their shapes do not change. Therefore, if we rotate any one of two triangles around edge N_1N_2 they can be in a same plane. Then $SN_1N_7N_2$ is a rectangle. Using this rectangle, the distance $g(g_7)$ of the shortest path between the two vertices S and N_7 can be calculated.

This situation is expressed like in Figs.9. g can be obtained by using the cosine law. Here, letting e_{ij} be the straight line distance between the vertices N_i and N_j. The calculation formula thereof is as follows.

$$e_{ij} = \|N_i - N_j\|, (i,j = 1,2,\dots n \in N^*) \quad (3.3.1)$$

The shortest distance g is as follows.

$$g = \sqrt{g_1^2 + e_{17}^2 - 2g_1e_{17}\cos(\alpha + \beta)} \quad (3.3.2)$$

The α is the angle of the edges SN_1 and N_1N_2, and β is the angle of the edges N_1N_2 and N_1N_7. The calculation is like this.

$$\alpha = \arccos\frac{g_1^2+e_{12}^2-g_2^2}{2g_1e_{12}} \quad (3.3.3)$$

$$\beta = \arccos\frac{e_{12}^2+e_{17}^2-e_{27}^2}{2e_{12}e_{17}} \quad (3.3.4)$$

Applying equations (3.3.3) and (3.3.4) to (3.3.2), we can compute the value of g.

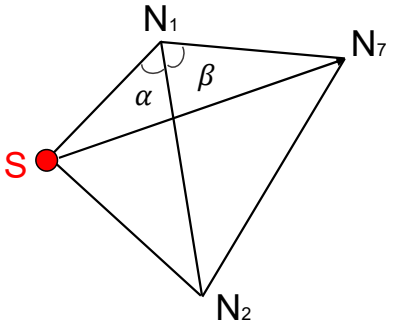

Figure 9: Calculating rectangle diagonal (1)

After calculating the shortest distance g, the visited set P and the candidate set Q are the same as follows.

$$P = \{S, N_1\}, \; Q = \{N_2, N_3, N_4, N_5, N_6, N_7\} \quad (3.3.5)$$

Journal of Theoretical and Applied Information Technology
30th September 2017. Vol.95. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.4 Calculate shortest distance

Now, we have obtained the shortest distance value through the Forward process, but the shortest path is not yet accurate, so a backward process is needed. The backward process is the process of approaching from the target vertex D to the start vertex S by using the information that calculated in the forwarding process. Through this step, we can know the exact shortest path.

First, finding the key point (the last vertex entered the set P) closest to the target vertex D through the set P. Assuming it is N_i, so there is a unique edge DN_i between N_i and the target vertex D. The two triangles sharing this edge are ΔDN_iN_L and ΔDN_iN_R. In this paper, we use the half-edge data structure, so we know which direction that the shortest path actually comes from. If the shortest path actually comes from ΔDN_iN_j as shown in Figs. 10, we need to calculate the exact position of the intersection Y of the diagonal lines on the rectangle SN_iDN_j. We can see that the actual shortest path comes from Y to D. The location of Y can be calculated by computing the intersection of the rectangle SN_iDN_j like the method of section 3.3 in this paper.

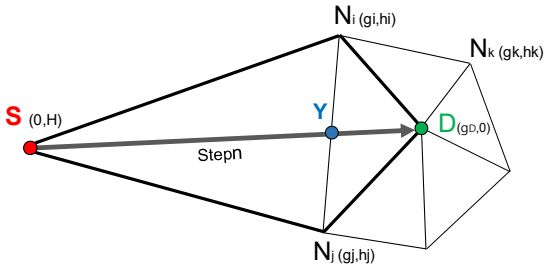

Figure 10: Calculating rectangle diagonal intersection (1)

Then, in Figs. 11, the intersection Y is at the edge N_iN_j, the two triangles that sharing it are ΔDN_iN_j and $\Delta N_iN_jN_Q$. Since ΔDN_iN_j is a triangle that has already been used, it is not considered. Therefore, the shortest path should come from the other two edges N_iN_Q and N_jN_Q except for edge N_iN_j in $\Delta N_iN_jN_Q$. Compute Y_{n-1}(The front vertex of Y on the shortest path) of the two possibilities, then select the one with the shorter distance. Repeat these steps until the start vertex S is reached. Finally, we can get the final shortest path by connecting all the intersections.

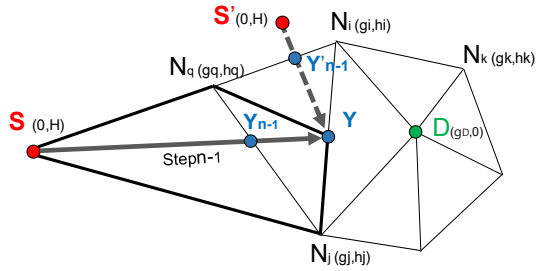

Figure 11: Calculating rectangle diagonal intersection (2)

In Code 1 shows the Pseudocode code of the A* algorithm.

Code 1: A Algorithm Pseudocode*

```
function A*(start, goal)

    open_list = set containing start
    closed_list = empty set
    start.g = 0
    start.f = start.g + heuristic(start, goal)
    while open_list is not empty
      current = open_list element with lowest f cost
      if current = goal
        return construct_path(goal) // path found
      remove current from open_list
      add current to closed_list

      for each neighbor in neighbors(current)
        if neighbor not in closed_list
          neighbor.f = neighbor.g
          + distance(neighbor, goal)
          if neighbor is not in open_list
            add neighbor to open_list
          else
            openneighbor = neighbor in open_list
            if neighbor.g < openneighbor.g
              openneighbor.g = neighbor.g
              openneighbor.parent
               = neighbor.parent
return false // no path exists
```

```
function neighbors(node)

    neighbors = set of valid neighbors to node
    for each neighbor in neighbors
        if neighbor is diagonal
          neighbor.g = node.g + diagonal_cost
        else
          neighbor.g = node.g + normal_cost
        neighbor.parent = node
    return neighbors
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
function construct_path(node)

    path = set containing node
    while node.parent exists
        node = node.parent
        add node to path
return path
```

4. EXPERIMENT

This experiment is performed by using the Bunny model with a total of 2,503 vertices to compare the results of Dijkstra and A* algorithm, and set starting vertex to the first vertex of the Bunny model. The target vertex is calculated from the second vertex to the last vertex of the mesh, and the experimental result is as follows. Therefore, this experiment is to find the shortest path and the shortest path from the mesh to all the vertices. As a result, we have found the distance of the vertices, the number of calculations.

In Figs. 12, the X-axis of the graph is the shortest path's value between the start vertex and the target vertex, and it is sorted according to the size. The Y-axis means that dividing A* and Dijkstra algorithm's Push Count values, and multiplying 100 to them. The results demonstrate that the A* algorithm improves the searching efficiency about 80% on average, compared with the results of Dijkstra algorithm.

Next, In Figs. 13, the X-axis value of this graph is the same with Figs. 12, and the Y axis means a push count value when the shortest distance test is completed. As a result, it can be seen once again that the number of calculations of searching the shortest distance and path of 2,503 vertices on polygon mesh is improved by about 80% compared with the A* and Dijkstra algorithm.

5. CONCLUSION & DISCUSSION

In this paper, we propose a method to find the shortest distance and the path between with two vertices on a polygon mesh by using the A* algorithm. Experimental results show that the proposed method improves the path searching efficiency by about 80% compared with the conventional method by using the Dijkstra algorithm. Therefore, when calculating the shortest distance and the path of a polygon mesh with a large number of vertices, using this method can be calculated more efficiently than the conventional method.

Our method is faster than using Dijkstra's algorithm. But we have a trivial problem from the results of the experiment. It is sometimes the case that the value of the calculated shortest

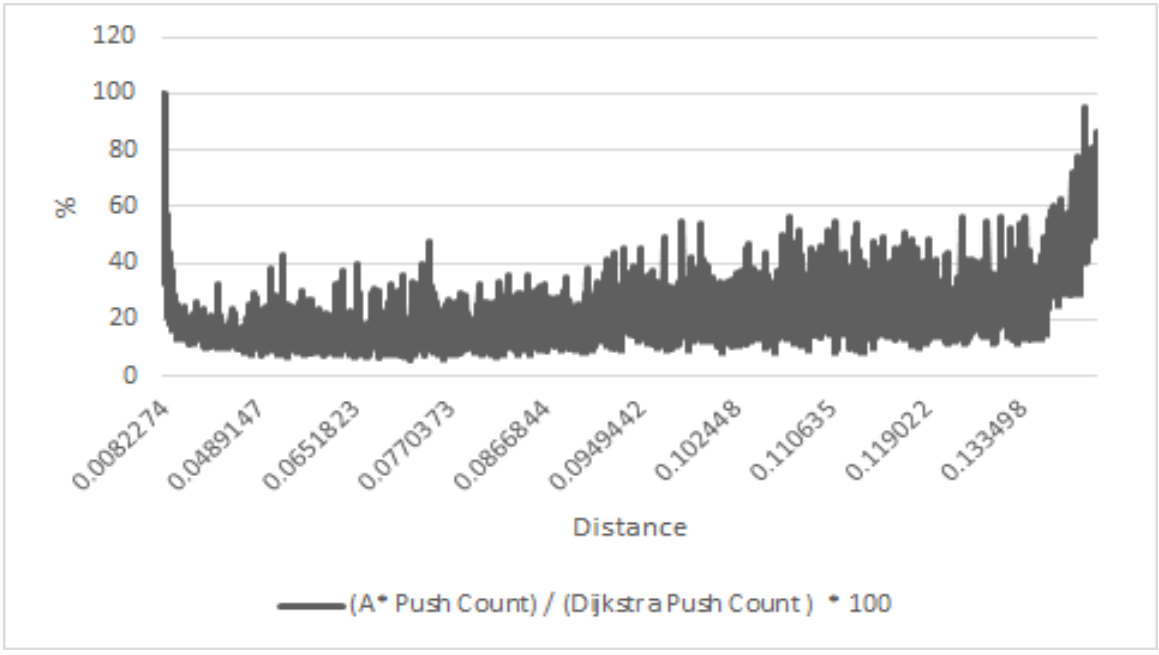

Figure 12: Comparison of A and Dijkstra algorithms*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

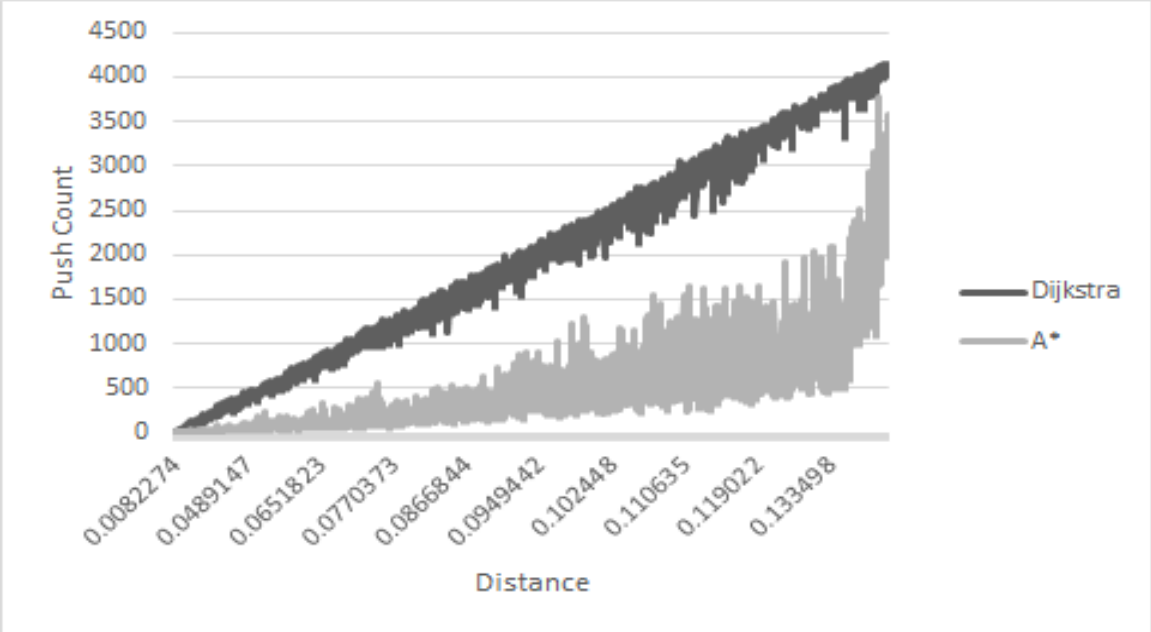

Figure 13: Comparison of A and Dijkstra algorithms' Push Count*

distance is not the same as that of the Dijkstra's method.

When using the A * algorithm, since we calculate the $H(n)$ values by measure the distance of two points, this cases are sometimes occurred. The assumptions about how to find the shortest distance of our algorithm are as follows. The $H(n)$ in Eq. (2.1) must always be equal to or less than the distance between two points on the actual 3d polygon mesh, but occasionally there are occasions when it is not. That's why this happens.

However, this case is very rare. In our experiment, this cases are two out of 2503 results. We have found that the distance error is less than about 0.1% of the distance of Dijkstra's method. Therefore, we purpose our algorithm better than dijkstra's method, because of faster than dijkstra's method although this method rarely occurred error.

Recently, artificial intelligence has been developed in various ways in the field of graphic arts. Among them, areas that can utilize various 3D space shortest path and distances such as automatic 3D modeling field, user input based mesh generation, and 3D texture mapping are being developed. We believe our algorithm can be used more efficiently in this area.

In terms of future work, firstly, we decide to study and compare with another good shortest pathfinding algorithm on 3D polygon mesh, then obtain the results of the advantages and disadvantages of each algorithm. Secondly, we plan to improve our Algorithm on its efficiency and accuracy by using technique of artificial intelligence, such as BP neural network or SVM. Thirdly, we are considering an artificial intelligence framework that can improve the algorithm of our algorithm and automatically generate the shortest distance in the polygon mesh model as preprocessing.

REFRENCES:

[1] Gabriel Taubin., "A Signal Processing Approach to Fair Surface Design", In Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, SIGGRAPH '95, ACM, 1995, pp.351-358

[2] Moh. Zikky., "Review of A* (A Star) Navigation Mesh Pathfinding as the Alternative of Artificial Intelligent for Ghosts Agent on the Pacman Game", EMITTER International Journal of Engineering Technology, Vol. 4, No. 1, 2016, pp. 141-149

[3] Addison-Wesley, "Dijkstra's algorithm", *Implementing discrete mathematics: Combinatorics and graph theory with mathematica*, Reading, MA (1990), pp. 225–227

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[4] Varady T., Martin R R., and Cox J., "Reverse engineering of geometric models—an introduction", *Computer-Aided Design*, 29(4), 1997, pp. 255-268

[5] Jinsuk Yang, Kyoungsu Oh, Hyung-Il Choi., "Mesh Editing with an Intuitive User Interface". *ICSCME'16*, April 12-13, 2016, pp. 47-53

[6] Schmidt R., Grimm C., Wyvill B., "Interactive Decal Compositing with Discrete Exponential Maps", *ACM Trans*, Graph, 25, 2006, pp. 605–613.

[7] Melvær E., Reimers M., "Geodesic Polar Coordinates on Polygonal Meshes", *Computer Graphics Forum*, Volume 31, number 8, 2012, pp. 2423–2435

[8] Colin Smith, "On Vertex-Vertex Systems and Their Use in Geometric and Biological Modeling", *Doctoral Dissertation*, 2006.

[9] Delling D., Sanders P., Schultes D., Wagner, D., "Engineering route planning algorithms", *Algorithmics of Large and Complex Networks*: Design, Analysis, and Simulation. Springer, 2009, pp. 117–139