Lecture Notes in Mechanical Engineering

D. A. Indeitsev
A. M. Krivtsov *Editors*

Advanced Problem in Mechanics III

Proceedings of the XLIX International Summer School-Conference “Advanced Problems in Mechanics”, 2021, St. Petersburg, Russia

Springer

The Algorithm of the Path Length Optimization on the Polyhedron Surface

Alia Gumirova[1], Ilia Marchevsky[1,2](✉), and Yurii Safronov[1]

[1] Bauman Moscow State Technical University, 2-nd Baumanskaya st., 5, 105005 Moscow, Russia

[2] Ivannikov Institute for System Programming of the RAS, Al. Solzhenitsyn st., 25, 109004 Moscow, Russia

iliamarchevsky@mail.ru

Abstract. Finding a short path between two given points lying on the polyhedron is an actual problem arising in various applications in science and technology; polyhedron faces without any restrictions can be considered triangular. An algorithm for the shortest pathfinding is suggested that uses Dijkstra's algorithm for the first (coarse) approximation. This path is split into parts consist of two segments. The positions of the middle nodes are optimized and they are placed on the corresponding edges of the polyhedron such as to provide a minimal length of each part. Then a similar procedure is performed once again, but the path is split into parts with the shift by one section. Thus, the positions are optimized for the nodes that have been fixed in the previous step. Such an algorithm is rather fast and it allows for a finding of an optimized path without "steep kinks" on a polyhedral surface, which is not optimal in the mathematical sense but is usually suitable for solving a number of applied problems. Further steps of optimization are based on the layouts considering and allow to achieve the exact solution of the problem.

Keywords: The shortest path · Polyhedron surface · Marquee · Layout · The Dijkstra's algorithm

1 Introduction

The ability to draw "straight" lines on a polyhedron surface, that provides the shortest path between two given points lying on the polyhedron, is required for solving various problems of science and technology. This problem arises in the planning of robots motion, navigation, *etc.* In particular, such a problem is actual in computational hydrodynamics, namely, in vortex methods of 3D flows simulation, when it is necessary to simulate the interaction of vortex structures being simulated by vortex loops with a streamlined body surface [1]. In contrast to classical problem of geodesics search on smooth enough surfaces, the

D. A. Indeitsev and A. M. Krivtsov (Eds.): APM 2021, LNME, pp. 55–68, 2024.
https://doi.org/10.1007/978-3-031-37246-9_5

considered problem is connected to geodesic reconstruction on piecewise-planar surfaces, where each panel (without loss of generality let us consider all panels to be triangular) is planar and there is an edge between the panels [2].

Nowadays, several algorithms are known, which allow for the mentioned problem solving and make it possible to find the shortest paths between two points on a polyhedral surface.

For example, in 1987, Mitchell et al. [3] suggested applying the E. Dijkstra's algorithm to construct a data structure, which usage allows one to find the shortest path between arbitrary two points lying on the triangulated surface. In 1999, Kapoor [4] used wave propagation technique to derive the shortest path algorithm. The algorithm developed by Chan and Han [5] is based on the surface layouts analysis.

Some other algorithms are also known for finding the shortest path on a polyhedral surface, developed mainly for specific problems. However, despite the simplicity of the problem statement, there is no "universal" algorithm, which would be suitable for a wide range of problems; existing algorithms are, as a rule, time- and memory-consuming.

In the present paper, an algorithm for the shortest pathfinding is suggested that uses Dijkstra's algorithm for the first (coarse) approximation and then local surface layouts are considered for the path optimization.

2 The Dijkstra's Algorithm

Let us first remind the basic information about the classical Dijkstra's algorithm.

2.1 Brief History of the Dijkstra's Algorithm

This algorithm seems to be the most well-known one for the shortest pathfinding on the graphs. It is named after the Dutch scientist Edsger Dijkstra, who have worked at the Mathematical Center in Amsterdam in 1956 as a programmer. He developed the algorithm to demonstrate the capabilities of a new computer called ARMAC. Later, the algorithm was used for efficient navigation in 64 cities in the Netherlands [5]. The final version of the algorithm was published in 1959 allowing finding the shortest paths in a directed (in general) weighted graph from one specified vertex to all other vertices, assuming that all the edges have non-negative weights.

2.2 The Dijkstra's Algorithm Description

The node X of the graph at which the path starts, is called the *starting* node. The length of the path from node X to the node Y, that fully lies on the surface, is called hereinafter *the distance to node* Y. At the first step of the Dijkstra's algorithm, some initial distance values are assigned for all the nodes, and then they are improved, step by step. Algorithm stops after final number of steps; it includes the following operations.

1. All the nodes are marked as unvisited; a one-dimensional array of distances from *starting* node to them is created, hereinafter called *main array*, together with one-dimensional array, called *Previous*, which length is also equal to the number of nodes.
2. For each node the initial distance is assigned and stored in the *main* array: it is equal to zero for the *starting* node and equal to infinity for all other nodes. The *starting* node having index s is taken as the *current* one, and the value (-1) is assigned to the s-th element in the *Previous* array.
3. For the *current* node, all its unvisited neighbors are considered and the distances are calculated to them through the *current* node (by weight adding of the corresponding edge of the graph). The newly calculated distances are compared to the current value of the distance, previously stored in the *main* array. The smallest of two these values is stored to the *main* array with the corresponding modification in the *Previous* array, if it is necessary. For example, if the current node A has in the *main* array *distance* value equal to D_A, and the edge connecting it to the neighboring node B has a weight W_{AB}, then the *distance* to B through A will be $\tilde{D}_B = D_A + W_{AB}$. If earlier node B had a distance value in the *main* array larger than $\tilde{D}_B$, it is replaced with $\tilde{D}_B$, and in the *Previous* array the index of the node A, namely i_A is stored at the position i_B, that corresponds to the node B. Otherwise (if $D_B \le \tilde{D}_B$), no changes should be done.
4. After considering all unvisited neighbors B of the *current* node A, the node A is marked as *visited*, *i.e.*, it will be never considered again. When all the nodes are marked as *visited*, the Dijkstra's algorithm finishes. If at some step all unvisited nodes have in the main array distance values equal to infinity, it means that these nodes cannot be reached from the *starting* node (*i.e.*, the considered graph is disconnected); in this case the algorithm terminates.
5. Otherwise, the unvisited node should be selected with the smallest value of the *distance* in the *main* array. Now it is considered to be a new *current* node and the algorithm is continued from the step 3.
6. After finishing the algorithm, the *Previous* array contains the information that makes it possible to reconstruct the shortest path from the *starting* node to all other nodes. Let us denote the values in this array as $\{P_i\}_{i=1}^{N}$, than in order to reconstruct the shortest path from the node s to the node t, we should first take the value, that corresponds to the last node t, namely P_t. Then we take a node, that corresponds to the node P_t, *etc.* When the index (-1) occurs, it means that *starting* node is achieved and the path is found. More precisely, to find the path from *starting* to target node, the obtained path should be reversed: $s \to \ldots \to P_{P_t} \to P_t \to t$.

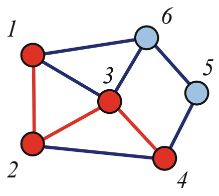

Fig. 1. The example of shortest path according to the Dijkstra's algorithm on a weighted graph

For example, if a graph consists of 6 nodes, $s = 1$ and $t = 4$, *i.e.*, nodes *1* and *4* are *starting* and target, respectively, the possible *Previous* array is the following:

i	1	2	3	4	5	6
P_i	-1	1	2	3	6	3

Thus, the shortest path is (Fig. 1).

$$1 \to 2 \to 3 \to 4$$

2.3 The Dijkstra's Algorithm as the Zeroth Step of Pathfinding

To solve the problem of the shortest pathfinding on a polyhedron surface, it is convenient to consider at the zeroth step a graph which nodes correspond to the vertices of the surface polygons, while edges correspond to their sides; the weights are equal to the length of edges. As a result, the undirected graph G is considered, and the previously described Dijkstra's algorithm is applied to it. Thus, the path from starting to target node is found, however it is not optimal in the general case, because it goes through vertices and edges of the polyhedral surface. The examples of found paths are shown in Fig. 2 for different model surfaces: a weight model and a fish model.

The advantage of this algorithm is connected with extremely low computational cost of its implementation. However, since a graph of vertices/edges is considered, the resulting path contains "steep kinks", that is usually undesirable for applications. For example, in the shortest pathfinding in the framework

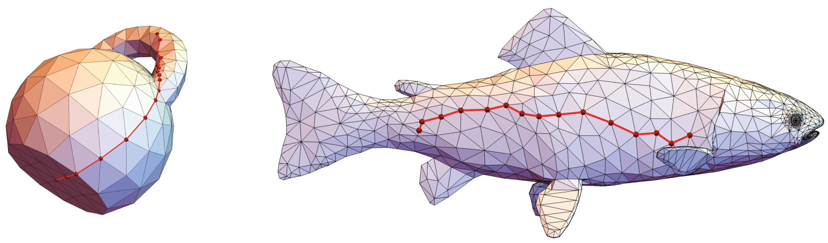

Fig. 2. The shortest path according to the Dijkstra's algorithm on different surfaces

of vortex methods in CFD, it is not a critical issue when non-optimal path is constructed, while the mentioned "kinks" are highly undesirable: such paths are considered as trajectories of vortex filaments on the streamlined body surface, and to avoid physically implausible high velocities, their shapes should be as smooth as possible. It is clear, taking into account the polyhedral shape of the initial surface, that the path cannot be smooth in the mathematical sense, so its "smoothness" should be understood in sense of the path length minimality.

We note also, that if the initial surface has edges or "corner lines", the closest path between the points lying on different sides from the edge, of course, has a "steep kink", however, such behavior is now desirable and physically plausible since it corresponds to singularity of the velocity field at such "corner points".

So the further path optimization can be performed either "exactly", *i.e.*, up to achieving the shortest path, or partially to obtaining a smooth enough path in the above-mentioned sense.

We consider only the case when the initial polyhedral surface is closed, so all the edges between vertices are *internal*, *i.e.*, there are two faces adjacent to it.

3 Path Optimization

Let us describe in details the developed algorithm, that has low computational cost and allows for constructing path, that smooths Dijkstra's path. The resulting path is called *optimized*.

3.1 Algorithm Description

It is suggested to perform the optimization according to the sequential layout algorithm, which is described below.

Let us introduce some definitions. We call as a *nodes of a marquee* an arbitrary vertex of the triangulated surface together with all neighboring vertices, *i.e.*, vertices connected with initial one by an edge, Fig. 3. Initial vertex of the marquee forms triangles with all pairs of adjacent bottom vertices (we call *adjacent* those of bottom vertices which are neighboring, *i.e.*, connected by an edge). Thus, the marquee's surface consists of triangles having common vertex, which we call as a *marquee's top*. Note that each node of the graph (or, the same, the vertex of the triangulated surface) is uniquely associated with a marquee, in which it is a top.

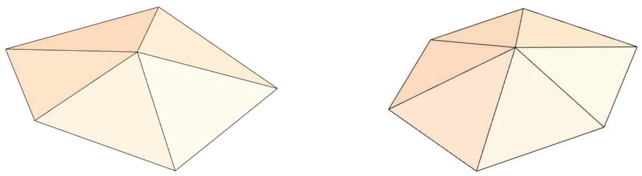

Fig. 3. A marquees with 6 and 7 vertices

Therefore, in the path obtained as a result of Dijkstra's algorithm, the corresponding marquee can be found for each vertex. Such representation seems to be convenient because at the first step of the optimization procedure these marquees are to be analyzed separately. Note also, that each of these marquees always contains three vertices of the Dijkstra's path, one of which is the marquee's top. These facts lead to a suggestion of the marquees' layouts consideration, which allows performing the first ("local") step of the path optimization.

3.2 The Sequential Layout Algorithm

The proposed algorithm can be described as the following.

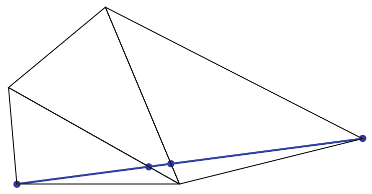

Fig. 4. The layout of a "half" of the marquee and a new path on it

1. The Dijkstra's path is divided into threes (containing three vertices each), so that the last vertex of the first three coincides with the first vertex of the second three, *etc.*
2. Each of these threes determines the marquee, where the first and last vertices correspond to the beginning and ending of the path on the marquee, and the middle vertex is the top of the marquee.
3. For each vertex of the marquee, that is a point in 3D space, some point on the 2D plane is assigned according to the following rule: point (0, 0)—the origin on the plane—is assigned to the top of the marquee. The image of a vertex, that corresponds to the beginning of the path, is placed on Ox axis at the point (ρ, 0), where ρ is the distance in 3D space between these points. Then the angle measures should be found for all the triangles of the marquee, that are adjacent to the top vertex. The rays corresponding to these angles (and, thus, to the marquee's edges) should be drawn, until the edge containing the third point of the path is achieved. This procedure of *2D layouts* constructing should be performed twice, for both directions of the marquee's traversal. If the sum of the angles in some traverse directions exceeds π, this case is out of interest, because it is impossible to draw a path over the corresponding marquee's faces, that is shorter than the initial path, containing two edges. If both angle sums are less than π, both cases are considered, and the marquee is split into two "halves". Otherwise, if the both angle sums are greater than π, the optimal path coincides with Dijkstra's path on this marquee.

4. 2D layouts are considered for one or two "halves" of the marquee. The shortest path between the first and last points corresponds to the straight line between their images on the layout, Fig. 4. If the straight line doesn't intersect some edge on the layout (and intersects its prolongation), the shortest path is constructed as a polyline which internal vertices coincide with edges, that are images of bottom edges of the marquee.
   If two "halves" are considered, that path should be chosen, which length is smaller. Since the transformation from the "half" of the initial marquee to its layout preserves length, the inverse mapping of the found straight line determines the shortest path on the initial marquee between the first and third points. For consistency, let us determine the positions of the points, at which this path intersects the "inner" edges of a marquee, in some invariant manner, that is suitable both for 3D geometry and its 2D layout representation: each such point is characterized by a tuple $\{x, y, \alpha\}$, where x and y are the indices of the endings of the marquee's edge, that is intersected by new path (to be more specific, let us assume that x corresponds to the top of the marquee, y—to the vertex on its bottom); α is a ratio, in which the $x-y$ line is split by new path. 3D coordinates of the obtained intermediate points can be easily found by using the proposed representation. The path, containing these intermediate points instead of the top of the marquee, is certainly shorter (Fig. 5).
5. The next marquee is considered and steps 3 and 4 are repeated.
6. As a result, the first step of path length optimization is finished, and a new path is shorter in comparison to the initial one.

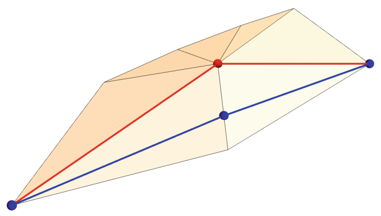
Fig. 5. New path with one intermediate point

Note that in the obtained path between the intermediate points lying on the edges, there are points that coincide with the vertices: these points were the first and third points in the threes, considered earlier on the first step of optimization. Now, at the second step, we consider, vice versa, these points to be tops of the marquees and perform optimization similarly to the above-described algorithm. The only difference is that in each such marquee the first and third points now do not coincide with bottom vertices, lying at some (known) intermediate points on bottom edges. However, it is not essential for the optimization procedure; for simplicity two axillary edges on the marquee can be drawn: connecting the top with two considered points on bottom edges.

Now, the second step of optimization is finished, the examples are shown in Figs. 6–8. Note, that these two steps are "local", *i.e.*, the pairs of adjacent legs are considered separately from the other pairs.

The resulting path, in contrast to the Dijkstra's path, passes through the faces of the polygonal surface instead of the edges end vertices. As a rule, it does not contain many "steep kinks" in comparison to the Dijkstra's path (*i.e.*, it is much smoother in the above-mentioned sense), and in some applications such optimization can be quite enough for practical purposes. However, below we describe the following steps of optimization, which allows for obtaining the shortest path in the mathematical sense. Of course, further steps are not "local", *i.e.*, they touch long parts of the path simultaneously.

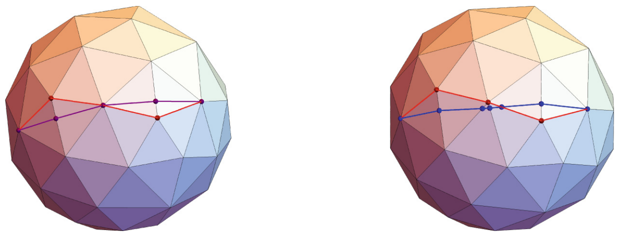

Fig. 6. The path ot triangulated spherical surface after the first (left) and the second (right) step of local optimization in comparison to the Dijkstra's path

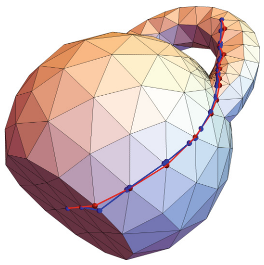

Fig. 7. The example for the weight model: blue path corresponds to the second step; red path is initial Dijkstra's one

4 The Shortest Path Determination

The shortest path approximation obtained after the above described two "local" steps of optimization is not the shortest, because, according to the technique of its constructing, it certainly goes through the triangular faces, that are adjacent

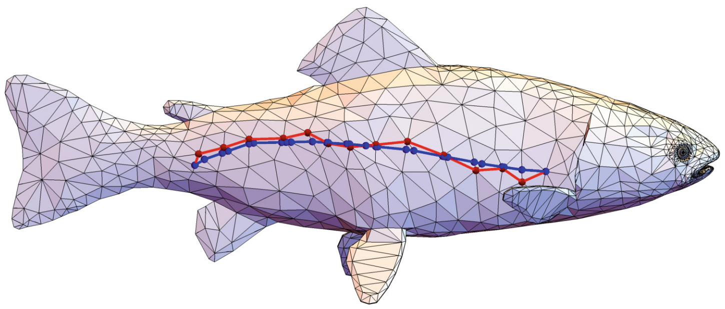

Fig. 8. The example for the fish model: blue path corresponds to the second step; red path is initial Dijkstra's one

to the edges of the Dijkstra's path, while for the mathematically optimal path this statement in the general case is not true. In order to overcome this issue and to construct the optimal path, we again use the idea of the layouts consideration, but now not "local" ones for a small number of adjacent faces, but "global" for all the paths between starting and target points on the body surface (Fig. 9).

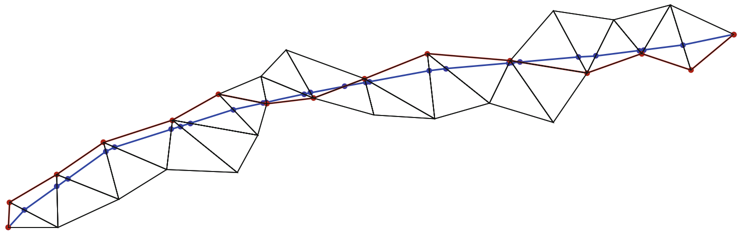

Fig. 9. Path layout for the fish model; blue line—the path after two steps of optimization; red line—the initial Dijkstra's path

4.1 The Path Layout

Recall that each vertex of the path lies on some edge and is described by a tuple $\{x, y, \alpha\}$ that contains two vertex indices and a ratio of the edge between them.

In order to obtain the path layout, the following steps should be performed.

1. Firstly, a list of triangles should be prepared, which are intersected by the current path. In order to do it, consecutive vertices of the path are considered; each of such pair stores indices of two vertices between which the edge is placed. Since the surface mesh topology is known, it is easy to determine the triangles, which contains the specified edges and, thus, the triangular

face, through which the path goes (*i.e.*, that contains both the edges). The sequence of all such triangles defines the region that wraparounds the entire path (more precisely, the current approximation to the optimal path) goes.

2. Similarly to the above-described algorithm of the marquee layout construction, now the layout for all the wraparounding triangles should be built. In order to do it, we consider the first triangle from the list and put it onto the plane Oxy, coinciding the starting vertex for simplicity with the origin (0, 0). The second vertex (chosen arbitrarily from two possible variants) is placed to the point (ρ, 0), where ρ is the length of the corresponding edge (the distance between the first and second vertices). The image of the third vertex of the initial triangle is found in such a way, that provides the geometric equivalence between the triangular face and its image on the layout.
3. Each next triangle from the above-found sequence has a common edge with the previous one, so to draw its image on the layout we only need to find the image of its remaining vertex. Again, it is found outside the previously drawn triangle and provides the equivalence of the considered triangular face and its image. Note, that the number of approaches can be used to do such operations; that one should be chosen that provides the smallest execution time. As the result of this step, we obtain the layout of the domain, that includes all the triangles wraparounding the current path (Fig. 10, *a*, *b*, *c*).
4. According to the ratios, in which the path points divide the corresponding edges, it is easy to mark them on the layout.

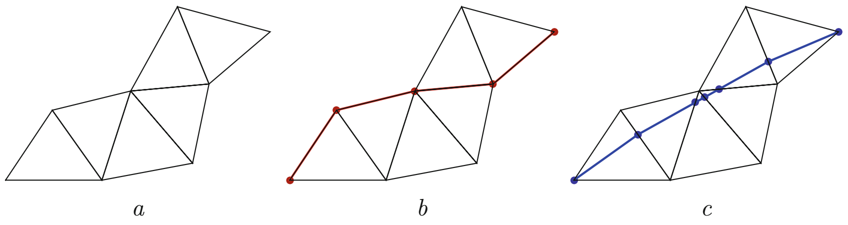

Fig. 10. The layout example: *a*—images of all the triangles wraparounding the current path; *b*—the Dijkstra's path; *c*—the path after two steps of "local" optimizations

4.2 The Path Straightening Algorithm

Path considering on a plane is much more convenient than in 3D space. Obviously, the shortest path between two points on the plane is a straight line between them. Until the current approximation of the shortest path is one straight line on the layout, we sequentially straighten it according to the following algorithm.

1. We try to draw straight lines from the starting point to sequential points and to check if such secant line intersects the boundary of the domain (*i.e.*, the

outer boundary of the union of all the triangles on the layout). When for some point the secant line leaves the domain, we mark the previous point (for which it still lies inside the domain) as *kink point.* Then we should consider the last kink point as the beginning of the next secant line and search similarly the next kink point, *etc.*, until the target point is reached.
Note, that after each substep is performed (*i.e.*, after the straight line between kink points or starting/target points is obtained), we should replace the corresponding part of the current path with new points, which correspond to the intersection points of the found straight line with the edges. In fact, the only α ratios should be updated in the corresponding tuples, while the "topology" of the approximation of the shortest path remains the same.

2. In case of no kink points found on the layout after step 1, the shortest path is found. If there are some kinks, they should be "smoothed". Firstly, for each kink point, we should try to find as long as possible straight line that connects some of its left neighbors with some of its right neighbors, and lies fully in the considered domain on the layout. For example, sequential left and right neighbors, step-by-step, can be checked until the line between them lies inside the domain. If the involvement of some left neighbor breaks the controlled condition, the line widening to the left should be stopped, and the last "good" point is marked as a new kink point, and then the second new kink point is found on the right, and wise versa for the case when some right neighbor first breaks the condition. The part of the path, that had contained the "old" kink point, is replaced by the found straight line: again, only the ratios should be updated in the corresponding tuples (the "old" kink point is removed since its point is not included now in the current path). The described procedure is performed several times until at least two kink points are close enough to a triangle vertex (Fig. 11).
3. The obtained after step 2 path "rests" against some vertex (or vertices), *i.e.*, the kink points are concentrated around this vertex (or several vertices). Having in mind the analogy with a fiber, being stretched in the considered domain between the starting and target points, it is clear, that it becomes piecewise-straight, while some triangle vertices become "obstacles" that the fiber goes round. Now we consider sequentially such vertices (around which kink points are found) as the tops of the corresponding marquees. For each such top point the marquee's bottom (that is understood now as the closed line) should be found, together with two points at which it intersects the current path. The above described local optimization algorithm now should be executed, similar to its second stage. As the result, we obtain the next approximation for the shortest path, which has a different topology: it goes through the other triangular faces in comparison to the previous one (the removed and introduced faces correspond to the faces of two "halves" of the mentioned marquee). So, the problem-causing vertex is now overcome.
Note, that after overcoming each of such vertices due to changes in topology the layout should be constructed once again (Fig. 12).

4. Step 1 should be performed once again for new path topology and new layout shape. The cycle continues until there are no kink points at step 2, which means, in turn, that the straight path on the layout is achieved and the shortest path on the initial surface is found (Fig. 13).

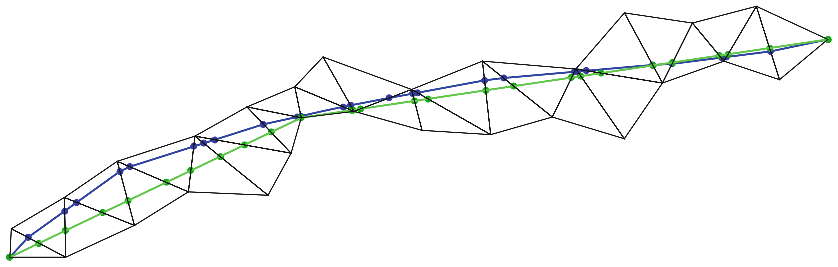

Fig. 11. Green line—piecewise-straight path; blue line—initial path

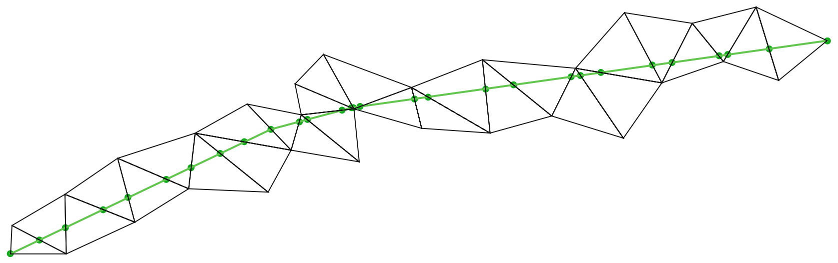

Fig. 12. Piecewise-straight path after the problem-causing vertex overcoming

For the obtained points on the layout, defined by the tuples, it is easy to find their prototypes on the edges of the initial triangulated surface in 3D space by splitting the corresponding edges in given ratios.

The results, obtained with this algorithm, are shown in Figs. 14 and 15.

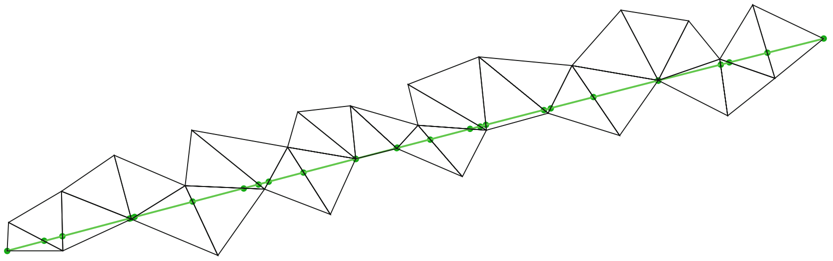

Fig. 13. The shortest path (straight line) on the layout

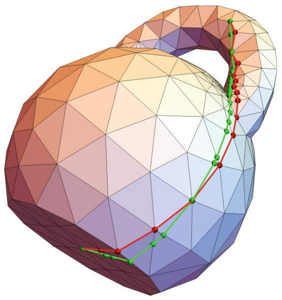

Fig. 14. The shortest path (green line) between two specified points on the triangulated surface of a weight model in comparison to the Dijkstra's path (red line)

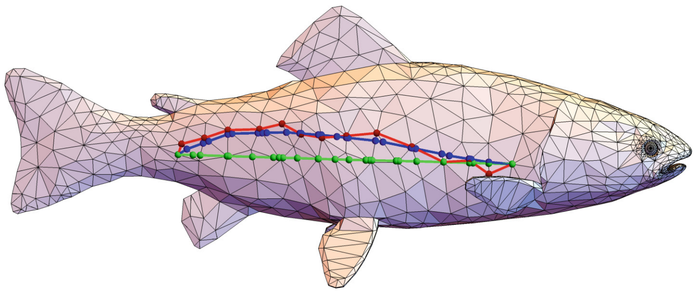

Fig. 15. The shortest path (green line) between two specified points on the triangulated surface of a fish model in comparison to the Dijkstra's path (red line) and the results of its local optimization (blue line)

5 Conclusion

The algorithm of a finding of the path without "steep kinks" on a polyhedral surface is suggested. It consists of several steps: the Dijkstra's algorithm and two stages of local optimization. The resulting path is not optimal but can be suitable for engineering applications. The further optimization algorithm is also developed; it is iterative and is based on path layouts considering of the polyhedron surface. The shortest path can be found after a finite number of iterations. The algorithm is implemented as a prototype in the computer algebra system *Wolfram Mathematica*.

Acknowledgements. The work of I. K. Marchevsky is supported by Russian Foundation for Basic Research (proj. 20-08-01076).

References

1. Dergachev, S.A., Marchevsky, I.K., Shcheglov, G.A.: Flow simulation around 3D bodies by using Lagrangian vortex loops method with boundary condition satisfaction with respect to tangential velocity components. Aerosp. Sci. Technol. **94**, art. 105374 (2019). https://doi.org/10.1016/j.ast.2019.105374
2. Marchevsky, I.K., Shcheglov, G.A., Dergachev, S.A.: On the algorithms for vertex element evolution modelling in 3D fully Lagrangian vortex loops method. In: Šimurda, D., Bodnár, T. (eds.) Topical Problems of Fluid Mechanics 2020, pp. 152–159. Prague (2020). https://doi.org/10.14311/TPFM.2020.020
3. Mitchell, J.S.B., Mount, D.M., Papadimitriou, C.H.: The discrete geodesic problem. SIAM J. Comput. **16**(4), 647–668 (1987). https://doi.org/10.1137/0216045
4. Kapoor, S.: Efficient computation of geodesic shortest paths. In: Proceedings of 31st Annual. ACM Symposium on Theory Computing, pp. 770–779. Atlanta, USA (1999). https://doi.org/10.1145/301250.301449
5. Chen, J., Han, Y.: Shortest paths on a polyhedron. In: Proceedings of 6th Annual. ACM Symposium on Computational Geometry, pp. 360–369. Berkeley, USA (1990). https://doi.org/10.1142/S0218195996000095