# Fast and exact geodesic distance on triangular mesh

## Motivation

Given a 3D mesh, one could want to be able to compute distance on the mesh. One possible application for an exact computation of geodesic distance on 3D mesh would be the ability to draw Beziers and Splines curves on those meshes.

## Existing methods and limitations

We could easily approximate distance between two points by limiting ourselves to edges, using Djikstra algorithm. Using this method, the shortest path between two given points would mainly consist of a walk along the edge of the mesh. This approximation might hold for graph that are sufficiently tight and regular, but the error with this method is arbitrary high if the mesh or the points are not well chosen.

## Exact computation of geodesic distance

Even if Djisktra’s method is not adapted to the problem we are trying to tackle, we can use the same paradigm to compute the exact geodesic distance : propagating information step by step, keeping at any given point in time the optimal information, and stopping when there is no information left to propagate.

In this case, the information is a “window” on an edge h. This window describes a way of reaching a subset of h from a point S called the source or the pseudo-source. Furthermore, the window can be reached from S using only straight lines in the planar unfolding of the faces that lie between s and the edge.

On one window, the distance to point S on the mesh is completely known, and if we know the distance from S to the starting point, we can derive the distance field over the window through S.

In a Djikstra-like manner, we would now like to propagate this information to edges that are across a new adjacent face. We can do this by adding the new face to the planar unfolding, then drawing straight lines from S through the window. This will define a certain number of new windows on which the distance field through S is known.

However, the new windows might intersect with existing windows that were already here. We then need to merge the existing windows and the new window so as to keep only the minimum distance to the starting point. We can then carry on propagating the resulting new windows.

We will continue this propagation-merge loop until no more windows are left to propagate. Barring any unreachable points, the distance field should then be known over the entire mesh.

If the method is pretty simple in theory, there are a few problems that arise on edge cases. Propagating the windows is done according to a set of rules described in the article, which we will explain in the second part. Merging was more loosely described, and we derived our own algorithm that did a correct, albeit not really computationally efficient job.

## Our Implementation

### Main Objective

Even if the implementation of the algorithm described in the paper provided is a goal in itself, we wanted to go a little further by creating a small video-game based on geodesic distance.

Given a complicated and big 3D mesh, the game objective is to find the “source” using only information given by the distance field (i.e. You’re getting closer or farther from the source). This information is represented using colours on the mesh. We take advantage of the exact nature of the algorithm using subdivision to represent the information on a much more detailed level that what could be achieved with an approximate Djikstra algorithm on the initial mesh.

### Data Structure

We chose to use the halfedge representation of graphs. This allows us to achieve balance between the size of the object we’re manipulating and the ease with which we can traverse the graph.

The only extensive custom data structure we’re implementing I the window we described in the first paragraph. A window is an object that can locally describe the distance field, and possesses method to be able to propagate itself along neighbouring edges.

### Propagation

Given a window, we first need to be able to propagate it along the neighbouring edge. We identified 6 main cases and a few sub-cases leading to different results.

In order to distinguish between those cases, we will introduce a few notations. Given a window (B0, B1) on an edge h, and knowing the source S associated to the window, we can identify M0, M1, M2 and M3 as the intersection between 4 pairs on lines. We then define a point as *valid* if it lies on one on the triangle edge. We distinguish between our cases depending on which of the previously mentioned points are valid.

#### Case 0

#### Case 1

#### Case 2

#### Case 3

### Merging

As we propagated a window on a new edge,

### Backtracing and shortest-paths

### Subdivision and representation of results

## Results

### Efficiency

### Accuracy

### Problems we encountered

## Extensions

### Bezier Curves

### Find the Source!

### Procedural texturing