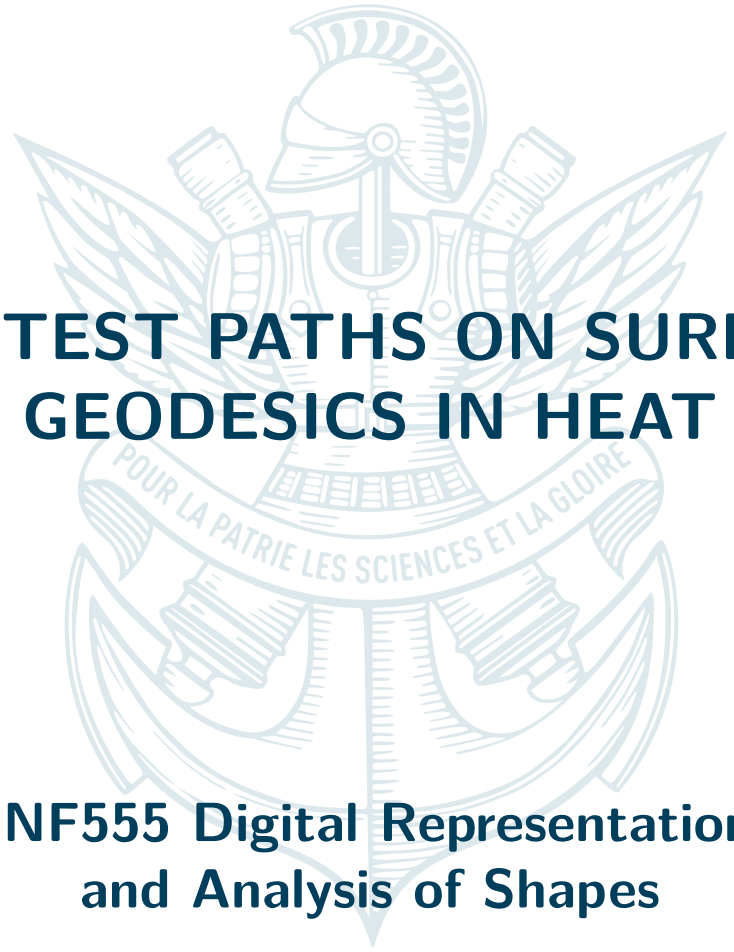

SHORTEST PATHS ON SURFACES GEODESICS IN HEAT

INF555 Digital Representation and Analysis of Shapes

26 novembre 2015

Ruoqi He & Chia-Man Hung

1 INTRODUCTION

In this project we present practical methods for computing approximate shortest paths (geodesics) on a triangle mesh. We implemented a naive method based on Dijkstra's algorithm and a method of Heat Flow based on *Geodesics in Heat : A New Approach to Computing Distance Based on Heat Flow*, written by Crane [2013]. We created a walking man on surface to represent the shortest path taken. The implementation is done in C# on the Unity platform.

2 ALGORITHMS

In this section we explain the main algorithms used in this project. The first part is a quick description of the method introduced in the paper. The second part is an algorithm created on our own.

2.1 Heat flow

This algorithm determines the geodesic distance to a specified subset of a given domain.
Notations : heat flow u, vector field X, distance function ϕ.

2.1.1 • First Step

Integrate the heat flow $\dot{u} = \Delta u$.
u of a point on a considered surface is a value between 0 and 1. Source have 1 as its value.

2.1.2 • Second Step

Evaluate the vector field $X = -\nabla u / |\nabla u|$.
We are only interested in the direction of ∇u and not in its value. X points to the opposite direction of the source.

2.1.3 • Third Step

Solve the Poisson equation $\Delta\phi = \nabla \cdot X$.
If a distance function ϕ exists, $\nabla\phi$ should give us a unit vector on every point, pointing to the opposite directin of the source. We approximate such a distance function ϕ by minimizing $\int |\nabla\phi - X|^2$, which is equivalent to solving the Poisson equation $\Delta\phi = \nabla \cdot X$.

2.2 Navigation

We also implemented a navigation system, represented by a walking man. He is positioned on the surface of the mesh and moves toward a given source point.

A walking man's position is defined by barycentric coordinates of a triangle of the mesh. A walk function takes a distance as its argument and first applys on the triangle on which he is standing and then recursively calls itself at the next triangle he comes across, until walking the given distance or reaching the source.

3 IMPLEMENTATION

3.1 Environment

We chose Unity to implement the heat method in order to better visualize the result. All codes are written in C#, and the scene is built with Unity Editor. For the main problem, we use the library ALGLIB to do sparse matrix operations and to solve linear equations. In addition, we use the C5 Generic Collection Library for the priority queue implementation.

3.2 Mesh representation

We obtain ordinary triangle meshes via various ways. Those meshes are represented by a vertex array (array of Vector3) and a triangle array (sets of 3 indexes stored in an int array). We first wrote a method to convert them to half-edge representation, defined in Geometry.cs. The conversion can be done in time of $O(nd^2)$ where n is the number of vertices and d is the maximum degree of vertices. There are however 2 important things to consider :

1. The half-edge representation is not well defined when using meshes with boundaries. In order to incorporate with other methods built on this representation, we decided to add a new face to cap each boundary. Those faces are marked as "boundary faces", and all vertices around them are marked as "boundary vertices". This way we can easily implement the boundary conditions in the heat method.
2. Many 3D models obtained from the internet have UV mappings, and thus have UV seams. This means that at the same position there can be 2 separate points having different UV coordinates. So the geometry we built may have seam-like boundaries blocking the way. To cope with this, we implemented a method to weld all overlapping vertices with a complexity of $O(nlogn)$, based on kdTree range searching.

3.3 Matrix precalculation

For each mesh loaded, we calculate in the first place its discrete unweighted laplacien matrix $-Lc$, and the matrices $A - tLc$ adapted to 2 different boundary conditions (if there are boundaries).

— Dirichlet condition :
We set all elements in the rows/columns of the boundary vertices to 0, except the diagonal elements. This way the heat value will always be 0 at the boundary.

— Neumann condition :
The original laplacien matrix described in the paper satisfies Neumann condition.

We also build a Vector3 array of size (3 * triangle count) keeping all the values of cot(angle) * opposite edge vector, in order to accelerate the calculation of divergence.

For the first time, we skipped the Cholesky decomposition step since the overall performance without it is still reasonable. However, because of the numerical problems that we will explain afterwards, we finally implemented the Cholesky decomposition. It is applied on all precalculated matrices.

3.4 Main calculation

For single source problem, we follow these steps :

1. Calculate the heat flow u by solving the heat equation $(A - tLc)u = \Delta(\text{source})$.
2. Calculate X, the normalized gradient of the heat flow, on every triangle.
3. Calculate $DivX$ on every vertex using the value of X on its surrounding triangles.
4. Calculate the distance field ϕ by solving the Poisson equation $Lc\phi = DivX$
5. We then calculate the gradient of the distance field $\nabla\phi$ on every triangle which can be used to calculate the shortest paths.

3.5 Navigation

4 RESULTS

4.1 Main results

success !

4.2 Boundary conditions

lots of graphs here!

4.3 Time step t

numerical error when t = 1 (choice of t), tweak parameter

4.4 Calculation Time Cost

5 ADDITIONAL WORK

5.1 How we obtain meshes ?

Below are the four options we used :

1. OFF format (high genus, bague, triceratops, horse) -> mesh -> halfedge mesh
2. FBX format (dragon) -> processing (weld uv seams) -> halfedge mesh
3. 3Dmax handmade mesh (maze) -> halfedge mesh
4. Code generated mesh (sphere) -> halfedge mesh

weld uv seams by Kd tree
Fix vertices (triceratops)

5.2 Optimization

Our solver ALGLIB solves parse symetric positve-definite matrix system a lot faster than only symetric matrix system and Cholesky decomposition works only with symetric positve-definite matrix. We noticed that $-Lc$ is a symetric positve-semidefinite matrix. Therefore, $A - tLc$ (for $t > 0$) in the first step is a symetric positve-definite matrix and $\epsilon I - Lc$ (for $\epsilon > 0$) in the third step is also a symetric positve-definite matrix. Below is a math proof.

Prove that $-Lc$ is a symetric positve-semidefinite matrix.

We note $U_{ij} = E_{ii} + E_{jj} - E_{ij} - E_{ji}$ where E_{ij} is an elementary matrix with only one nonnull value 1 on position (i, j). By definition, $-Lc$ is in form $-Lc = \sum_{i,j} u_{ij} U_{ij}$ with all $u_{ij} > 0$. Since

the eigenvalues of U_{ij} is 0 and 2, it is a symetric positve-semidefinite matrix. We conclude by saying that any positive combination of symetric positve-semidefinite matrices is also a symetric positve-semidefinite matrix. (Another proof is given in Lecture 9.)

5.3 Multisource

5.4 Navigation

5.5 Mapping

Albedo, Normal, Specular Smoothness, Emission

6 CONCLUSION & EXTENSIONS

The heat method is a simple and efficient way that allows us to calculate the shortest paths to a point source or a specific domain of sources. In this project we succeeded to implement this method on surfaces represented by triangle meshes. A system of navigation is added to visualize the path taken. Some extra work is done to improve the visualization, such as texture mappings

add weight

applications : video games, robotics