

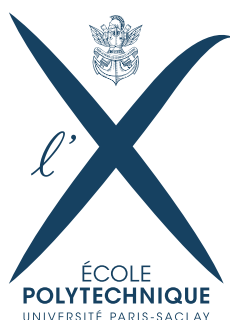


SHORTEST PATHS ON SURFACES GEODESICS IN HEAT

**INF555 Digital Representation
and Analysis of Shapes**

28 novembre 2015

Ruoqi HE & Chia-Man HUNG



1

INTRODUCTION

In this project we present a practical method 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

HEAT FLOW ALGORITHM

In this section we explain the Heat Flow algorithm used in this project. It determines the geodesic distance to a specified subset of a given domain. We describe it on a triangle mesh.

Notations : heat field u , vector field X , distance function ϕ .

2.1 FIRST STEP

Integrate the heat field $\dot{u} = \Delta u$.

The heat u of a point on a considered surface is a value between 0 and 1. Source has 1 as its value.

In other words, we solve $(id - t\Delta)u_t = u_0$ where u_t is the heat field after a time step t and u_0 is the initial heat field.

On a triangle mesh and in matrix form, we have $(A - tLc)u = u_0$ where A is a diagonal matrix containing the vertex areas, $A^{-1}Lc$ is the discrete Laplacian matrix of a triangle mesh, and u_0 is a vector which has value 1 on sources and 0 otherwise. Remark that here u_0 should be multiplied at left by A . However, in the case of one point source, this does not affect the second step.

2.2 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.

On a triangle mesh, by using the formula given in the article, we calculate the vector field on every triangle.

2.3 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 direction 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 $\Delta\phi = \nabla \cdot X$.

On a triangle mesh and in matrix form, we solve $Lc\phi = b$ where b is the vector of divergences of the vector field X .

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 $O(n \log n)$, based on kdTree range searching.

3.3 MATRIX PRECALCULATION

For each mesh loaded, we calculate in the first place its discrete unweighted laplacian 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 laplacian matrix described in the paper satisfies Neumann condition.

We also build a Vector3 array of size $(3 * \text{triangle count})$ keeping all the values of $\cot(\text{angle})$ * 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 field u by solving the heat equation $(A - tLc)u = u_0$.
2. Calculate X , the normalized gradient of the heat field, 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.

At first, without Cholesky decomposition, we used LinCG (Linear Conjugate Gradient) solver to solve linear equations. This solver solves symmetric positive definite problems, but it works fine even with our second semi-definite problem (Poisson equation). We just need to shift the result such that the distance value at the source equals to zero.

However, from the result we obtained, we observed that LinCG solver is sensible to numerical errors. The solution of the heat equation contains values ranging from 1 to 10^{-20} or smaller. The longer the distance to the source, the smaller the heat value is. LinCG returns values of 0 when smaller than $\sim 10^{-12}$, so the heat gradient of the farther area cannot be computed. We can only improve the solution by increasing the time step t , which increases the heat value but creates a smoothed distance field.

Therefore, we chose to implement the Cholesky decomposition of matrices. It consists of decomposing a symmetric positive matrix M into LL^T , where L is a lower triangular matrix. This means the linear equation $LL^Tx = y$ becomes two basic triangular systems, that can

be solved by simple substitution. This improves greatly the calculation time when switching sources (The decomposition is only calculated once for every mesh). And since we calculate the exact solution this way, we managed to have much smaller numerical errors, so as to eliminate the problems above. We overcame 2 difficulties regarding the Cholesky decomposition :

- While M is a sparse matrix, L isn't necessarily sparse, especially when the entries of M are mostly far away from the diagonal. This comes to one of our model which has randomly ordered vertices – it costs more than 4G RAM space to store the matrix L . After we reorder the vertices of the mesh along a certain axis, L becomes much sparser and takes much less time to compute.
- We need to add a small regularization term to the diagonal entries of the laplacian matrix to get strict positive-definiteness (needed for the Cholesky decomposition of ALGLIB).

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

Model	Triangles	Precalculation	Main calculation
High Genus	3328	1.1	1.2
Bague	5304	0.25s	2.2
Cow	5804	2.1	2.2
Triceratops	5660	2.1	2.2
Sphere 1	9024	2.1	2.2
Sphere 2 & 3	6912	2.1	2.2
Hemisphere	3456	2.1	2.2
Skull	12200	2.1	2.2
Maze	5120	2.1	2.2
Horse	39996	2.1	2.2
Dragon	20388	2.1	2.2

5

ADDITIONAL WORK

5.1 HOW WE OBTAIN MESHES ?

Below are the four options we used :

1. OFF format mesh from TD (High Genus, Bague, Cow, Triceratops, Horse) -> Unity mesh -> halfedge mesh
We wrote a OFF parser to transform OFF format meshes into Unity meshes.
2. Internet resources (Skull, Dragon) -> Unity mesh -> processing (weld uv seams) -> halfedge mesh
3. 3Dmax handmade mesh (Maze, Sphere 2 & 3, Hemisphere) -> Unity mesh -> halfedge mesh
4. Code generated mesh (Sphere 1) -> halfedge mesh

5.2 OPTIMIZATION

Our solver ALGLIB solves sparse symmetric positive-definite matrix system a lot faster than only symmetric matrix system and Cholesky decomposition works only with symmetric positive-definite matrix. We noticed that $-Lc$ is a symmetric positive-semidefinite matrix. Therefore,

Prove that $-Lc$ is a symmetric positive-semidefinite matrix.

5.3 MULTISOURCE

```
[tunnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn]
```

[illegible]

5.4 NAVIGATION

We visualized this process by moving a walking man on the surface of the mesh. His position is defined by barycentric coordinates of the triangle he is standing on. The distance gradient on it is also converted to barycentric coordinates, but with a sum of 0. (In this way, we can add it to the position coordinates and still get a barycentric coordinate with a sum of 1.) A walk function takes a distance as its argument and first moves on the triangle on which he is

standing. By finding the first coordinate that reaches zero we find the edge the man will come across. It then recursively calls itself at the next triangle he enters, until walking the given distance or reaching the source.

There are however 2 extra cases to consider :

- The gradient is pointing to a boundary. In this case, when reaching a boundary, we walk alongside it towards the smaller-distance vertex.
- 2 neighboring triangles have opposing gradient. In this case, when reaching the middle edge, we choose to follow the edge instead.

5.5 MAPPING

In order to better visualize the distance field, we chose to map a striped texture onto the surface of the mesh. It is actually quite simple : We create some striped textures with color gradient that are only one pixel tall, assign it to the mesh, and for each vertex we set the U value of its UV coordinates to its distance to the source. And that's it ! When rendering the texture, each point on a triangle has its U coordinate interpolated from the 3 vertices of the triangle, and it is exactly the interpolated distance of this point – so it will be colored by the pixel of the striped texture representing this distance. To make it even better, in addition to the main texture, we also apply generated normal map, specular map and emission map to each model, making it especially realistic. We just need to set the tangent vector of each vertex to the average distance gradient on it to make the normal map work.

5.6 DIJKSTRA

We implemented a simple Dijkstra shortest path algorithm on the graph made by the vertices and edges of the mesh. This allows us to easily create a line of source vertices connecting 2 selected vertices. (In the demo, press shift and click on the surface of the mesh.)

6

CONCLUSION & EXTENSIONS

Path finding is an important topic and has applications in several domains, such as video games and robotics. The heat method is a simple way that allows us to compute approximate geodesics to a point source or a specific domain of source (multisource) efficiently. In this project we succeeded to implement this method on surfaces represented by triangle meshes. A system of navigation is added to visualize the shortest path taken. Some extra work is done to improve the visualization, such as texture mappings.

Regarding improvement we could make, in the case of multisource repeated calculation of Cholesky decomposition is required due to the constraints of multisources. It might be possible to operate on the Cholesky decomposition of Lc to cleverly avoid recalculating the decomposition each time we change the source.

One extension we can think of is adding different weight on different areas to represent the difficulty of passing through. Another is changing the form of the considered domain from surfaces represented by triangle meshes to bounded spaces represented by tetrahedron meshes.