SHORTEST PATHS ON SURFACES GEODESICS IN HEAT

INF555 Digital Representation and Analysis of Shapes

28 novembre 2015

Ruoqi HE & Chia-Man HUNG

l'X
ÉCOLE
POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

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

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 $O(nlogn)$, 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 * 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 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 added a small regularization term to the diagonal entries of the laplacian matrix to get strict positive-definiteness (needed for 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

5
ADDITIONAL WORK

5.1 How we obtain meshes ?

Below are the four options we used :

1. OFF format meshes from TD (high genus, bague, triceratops, horse) -> Unity mesh -> halfedge mesh

2. Internet resources (dragon, skull) -> Unity mesh -> processing (weld uv seams) -> halfedge mesh
3. 3Dmax handmade mesh (maze, sphere 2 and 3, hemisphere) -> Unity mesh -> halfedge mesh
4. Code generated mesh (sphere 1) -> halfedge mesh

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

We also tried calculating the geodesics on surfaces when multiple vertices are marked as sources. However, by simply changing the initial heat vector to u_0 (sources) we generate incorrect heat field and distance field – the solved heat field u_t cannot guarantee equal values at each source vertex, and thus not every source vertex has a distance of zero. Typically, the vertex surrounded by more source vertices has a higher temperature and a negative distance.

[tuuuuuuuuuuuuuuuuuuuuuuuuu]

We solved this by imposing the constraints $u_t = 1$ at every source vertex in the heat equation, and also the constraint $\phi = 0$ at every source vertex in the Poisson equation. (The first constraint cannot imply the second since the distance field we get is only the closest potential of the heat gradient field.)

[tuuuuuuuuuuuuuuuuuuuuuuuuu]

One downside, however, is that the matrices need to be modified to contain these extra constraints related to the source. For exemple, to have $u_t = 1$ at the source we put 1 at the diagonal entries of the sources, and 0 everywhere else in the same lines and columns. We then compensate the terms we deleted by adding an additional vector at the right side of the equation. Because of this, the Cholesky decomposition need to be carried out each time we change the source. This slows down the calculation a lot.

5.4 Navigation

Now that we have the distance field, we would like to calculate the path from a given point to the source point. We achieved this by first calculating the gradient of the distance field (step 5 of the main calculation) which is considered uniform on every triangle, then trace a trajectory by recursively following the gradient in every triangle and entering the next one.

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. A walk function takes a distance as its argument and first applies 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.

5.5 Mapping

Albedo, Normal, Specular Smoothness, Emission

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 tetrehedron meshes.