Geodesics Distance on Triangulated Meshes

Andrew Owens
University of Calgary
Calgary, Alberta, Canada
arowens@ucalgary.ca

ABSTRACT

This is a project report for Computational Techniques in Graphic & Visualization (CPSC 601.13 W2014). My intentions were to implement a system allowing for geodesic distances to be efficiently solved on triangular meshes. Such an approach is detailed in *Geodesics in Heat* by Crane *et al.* [3]. I have implemented the salient aspects of this paper, and what follows is an overview of the method, my implementation considerations, and my results such as they are. This work was fruitful and motivating for future work.

1. INTRODUCTION

Certain mathematical principles and techniques are predicated on the notion of 'distance' being well defined. Distance may be defined differently in many cases, based on what is required of the metric and for which space it is to be defined. Of particular interest is the definition of geodesic distance within some domain. The geodesic distance is defined as the length of the shortest path(s) between two positions in arbitrary curved space. In regards to computer graphics, if this notion of geodesic distance can be found for certain discretized domains, such as 3D triangulated meshes, these principles and techniques can be exploited.

The classical approach to finding the distance function ϕ in previous methods is to solve the *eikonal equation*

$$|\nabla\phi| = 1$$

given the boundary conditions $\phi|_\gamma = 0$ for some subset of positions γ in the given domain. Many iterative methods have been developed to solve such problems which are non-linear and hyperbolic in nature. Popular methods for regular grids [7] and triangulated surfaces [8] exist, as do for point clouds [6] and polygon soup [1]. There has been substantial work done in defining geodesic distance in many domains. However, all fail to generalize to other domains outside of their own, as well tend to amortize poorly for consecutive distance queries for different γ. Along another approach, are a series of methods stemming from the idea of propagating geodesic distance and path information out from source points on a mesh like a *wavefront*, utilizing priority queues. Notably, the method proposed by Surazhsky *et al.* [9] demonstrated that their approach tended to subquadratic performance in time. However, these approaches fail in the same regards as the eikonal solvers above, as well tend to require a great deal of memory for the developed data structure. The wavefront propagation requires a large number of geometric calculations which are numerically sensitive, thus causing errors and gaps to appear in the wavefront [4]. As such the approach is not tractable in certain applications, requiring complex implementations. In particular, both I and another master's student in Dr. Przemyslaw Prusinkiewicz's Lab separately attempted this implementation, both finding that a great deal of special cases and implementation complexity was required to produce geodesics for certain meshes. In light of the complexities and perils detailed above, a more tractable and generic approach would be greatly beneficial to future work in my research.

2. OBJECTIVES

This project consists of a partial reimplementation of the approximate geodesic distance algorithm detailed in *Geodesics in Heat* by Crane *et al.* [3]. This method is both tractable in implementation and generic in application. In principle their method can be applied to any domain with a discrete gradient (∇), divergence ($\nabla\cdot$), and Laplace operator (Δ), opening up many possibilities of application. Once these operators are defined for a discrete domain, a simple, well known set of linear, hyperbolic problems are solved to produce a geodesic distance function. As they are well known linear problems, current linear solvers methods may be applied. As well, given the construction of problems, the set of linear problem may be prefactored (the bulk of computation) and backward solved (essentially linear) for different source points, γ. Thus the method has spectacular amortized costs for repeated unique distance queries. Although the algorithm generalizes to many domains, I have, as a proof of concept, implemented the method for 3D triangulated meshes alone, as they are of particular use in my research, and will lead to further work.

3. METHOD

The method of calculating geodesic distances on triangulated meshes is described by Crane *et al.* [3]. It is a numerical approximation scheme employing two linear solves of discrete differential systems. In principle their method can be applied to any domain with a discrete gradient (∇),

.

divergence ($\nabla\cdot$), and Laplace operator (Δ). If we let the Δ be the negative semidefinite Laplace-Beltrami operator, then the method follows the steps:

Steps

1. Integrate heat flow $\dot{u} = \Delta u$ for some fixed time t
2. Evaluate vector field $X = \frac{-\nabla u}{\|u\|}$
3. Solve the Poisson Equation $\Delta\phi = \nabla \cdot X$

The function ϕ of step 3 approximates the geodesic distance as t goes to zero. The distance computed is unique up to an additive constant, that must be shifted to have the minimum of the distance function be 0. Intuitively, this method works by integrating heat diffusion out from source points on the mesh for a given time interval, which will exponentially decay radially from these sources, but will do so monotonically. This means that no matter how faint the heat diffusion may be in the domain after this interval of time, if the heat value is greater than 0, the gradient will point towards the source at some magnitude. Thus as our distance function must satisfy the eikonal equation, of having a gradient of unit magnitude everywhere, and zero distance at the source points, we may simply negate and normalize the gradient of the heat diffusion $X = \frac{-\nabla u}{\|u\|}$, and the closest scalar potential ϕ to satisfy this vector field is the approximating distance function. The closest scalar potential is found by $min_\phi \|\nabla\phi - X\|^2$. This is equivalent to solving a Poisson Equation (Euler-Lagrange) $\Delta\phi = \nabla \cdot X$. This procedure is depicted below.

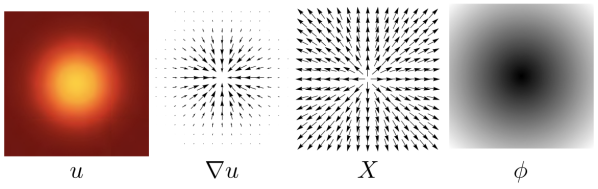


Figure 1: Overview of heat method. Left to right: Heat is diffused creating scalar field u *(left)*. Temperature gradient ∇u *(center left)* is normalized and negated to produce $X = \frac{-\nabla u}{\|u\|}$ *(center right)*. The distance function ϕ is the solution to the Poisson Equation $\Delta\phi = \nabla \cdot X$ *(right)*.

3.1 Time Discretization

A backward Euler step of a fixed time t is used to integrate the *heat diffusion*. This is realized by solving the linear system

$$(id - t\Delta)u_t = u_0 \tag{1}$$

over the domain. This is producing a solution u_t of heat diffusion such that, if undone on the mesh, would produce sole points of heat at our sources. This approach will ensure our solution respects the maximum energy of the input. As such we may only use this over the domain that is integrable, which is any part of the domain that is not a heat source (e.g having distance of 0). This equates to considering the following elliptical boundary value problem for heat source points γ of the domain M

$$(id - t\Delta)u_t = 0 \qquad M\backslash\gamma \tag{2}$$

$$u_t = 1 \qquad \gamma \tag{3}$$

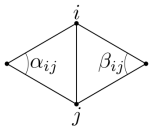
The time step t prescribed by the paper is different between the 2 versions of the paper. In one of the papers $t = h^2$ where h is the mean spacing between adjacent nodes, while the other is $t = A_m/|F|$ where A_m is the surface area of the mesh and $|F|$ is the number of faces. Both of these are to address, as they say, scale and refinement invariance in the method.

3.2 Spatial Discretization

The discrete differential operators needed for this method on simplical meshes (such as our triangulated mesh) are defined in the paper as follows. Let $u \in \mathbb{R}^{|V|}$ be a piecewise linear function on a triangulated mesh, such as our heat diffusion defined at the vertices. The standard discretization of the (negative-semidefinite) Laplace-Beltrami operator was chosen by paper. The Laplacian at a vertex i is given by

$$(Lu)_i = \frac{1}{2A_i} \sum_j \left(cot\alpha_{ij} + cot\beta_{ij}\right)(u_j - u_i) \tag{4}$$

where A_i is one third the area of all the triangles incident with vertex i (see inset). The sum is over all of neighboring



vertices j to that of vertex i. The angles α_{ij}, β_{ij} are the angles opposite the edge of i and j [5]. The Laplacian is then expressible as a matrix $L = A^{-1}L_c$ where $A \in \mathbb{R}^{|V|\times|V|}$ is a diagonal matrix of the vertex areas, and $L_c \in \mathbb{R}^{|V|\times|V|}$ is the *cotan operator* which encoded the remaining sum.

$$L = A^{-1}L_c \tag{5}$$

$$= \begin{bmatrix} \frac{1}{A_1} & 0 & \dots & 0 \\ 0 & \frac{1}{A_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{A_{|V|}} \end{bmatrix} L_c \tag{6}$$

L_c is a highly sparse symmetric negative-semidefinite system of equations dependent on the mesh connectivity. The heat flow can be solved by integrating the diffusion for a set of *heat sources* $u_0 \in \mathbb{R}^{|V|}$

$$(I - tL)\,u = u_0 \tag{7}$$

$$\Rightarrow \left(I - tA^{-1}L_c\right)u = u_0 \tag{8}$$

$$\tag{9}$$

where u_0 is 1 at vertices that are heat sources and 0 for other vertices. This system is equivalent to solving the symmetric positive-semidefinite system

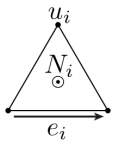
$$(A - tL_c)\,u = Au_0 \tag{10}$$

Once heat diffusion u is solved for, its gradient ∇u can be computed for each face of the mesh. This is computed as

follows

$$\nabla u_f = \frac{1}{2A_f} \sum_i u_i \left(N \times e_i\right) \tag{11}$$

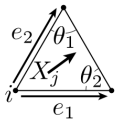
Where A_f is the area of the face, N is the unit face normal, e_i is the edge vector (oriented counterclockwise) opposite the i^{th} vertex u_i of the face f, and the sum is over all vertices of f.



What this is describing is how much the heat is changing orthogonal to, or independent of, the flow between the other heat samples of f. This sum will give a direction of greatest heat increase within the face. Applying this to every face of our mesh produces a *vector field* X over the faces of the mesh.

The integrated divergence of a vertex i is defined as

$$\nabla \cdot X = \frac{1}{2} \sum_j cot\theta_1 \left(e_1 \cdot X_j\right) + cot\theta_2 \left(e_2 \cdot X_j\right) \tag{12}$$



where the sum is over all faces j that are incident with i, X_j is the gradient of face j, e_1 and e_2 are the edge vectors directed outward from i on face j, and θ_1, θ_2 are the opposing angles. Let per vertex integrated divergence values of X be stored as a vector $d \in \mathbb{R}^{|V|}$. Then the distance function ϕ can be computed by solving the symmetric Poisson problem

$$L_c \phi = d \tag{13}$$

4. IMPLEMENTATION

This project was implemented using C++(0x), using GLUT windowing system with OpenGL 4.0, and GLSL for GPU programmable pipeline, for visualization. The implementation suite is 3500+ lines of code. This project and the results to follow were developed and run on a laptop with an i7-2630QM CPU at 2.00GHz, 8GB of DRAM, and an GeForce GTX 460M with 1.5GB VDRAM, running Ubuntu (12.04 32bit). The program suite that I implemented for this project follows that of the implementation described in *Geodesics in heat* [3] which employs the use of a fast sparse Cholesky factorization library. The two linear systems using the Laplacian and the Backward Euler integration are both symmetric, and negative and positive semi-definite, respectively. Thus they may be prefactored using Cholesky factorization, if certain hard constraints are applied. CHOLMOD[1] is a ANCI C library for sparse Cholesky factorization and update/downdate that is distributed with SparseSuite[2]. It is a highly completive option for these types of applications as it out preforms many other libraries in a large number of matrix tests. The installation and setup of CHOLMOD is notoriously difficult for Windows OS based setups, however for Linux based systems the setup is relatively straight forward. In my *Makefile* I am only required to link to the CHOLMOD library compiled for my system. To utilize CHOLMOD a *relatively* complicated system of handles to the library must be managed and made available to each matrix that will be factored or used in a solve. Luckily, a different, but related implementation from Crane et al. [2] is available[3] that utilizes CHOLMOD, and gave guidance as to how to implement my mesh and matrix classes for this project. As well, their implementation tipped me off to a glaring issue in the use of Cholesky factorization of our linear systems. Cholesky requires the matrix in question to be positive definite, whereas we have negative and positive semi-definite matrices in our Laplacian and Euler integration, respectively. Thus our systems are under constrained. This issue is glossed over in both versions of the paper, with no suggestion of how to address it. Crane et al. [2] remove a row and column from their Laplace matrix to ensure that it is positive definite. When I employed the same tactic, I get rather good results everywhere, except at the vertex that is ignored, as no distance or heat diffuse information is made available there. Another approach was to replace the last row of the Laplacian with an all zeros except a constant contstraint value (i.e 10) at the bottom most diagonal value. This produced a positive definite matrix and produces slight perturbing of the distance field at the vertex. The third promising, yet currently unsuccessful, artificial constraint is to enforce a known invariant of conservation of energy, in that the total heat in the system at the beginning of integration must equal that found diffused throughout the system. Thus we may add a constraint to the Laplacian, and Euler integration thereafter, that the total heat from the right hand side (source vertices) must be conserved over the time step. We make the entire bottom row of the Laplacian literal 1s, so that regardless of the right hand side source points, we may precondition our bottom most constant term to sum to the incoming heat. This method is still requiring work, but hopefully will lead to an elegant solution. As it turns out, it was easier to define a positive-semidefinite Laplacian L_c^+ and distribute the negative time t into the integration

$$\left(A + tL_c^+\right) u = Au_0 \tag{14}$$

As this allowed me to pre-factor L_c^+ as a positive-definite matrix for the Poisson solve.

$$L_c^+ \phi = -d \tag{15}$$

To produce the evenly spaced distance contours, I use a fragment shader that calculates distance per-pixel per triangle face. Each triangle has an established distance at its three vertices, and distance at any given pixel area of the triangle is an interpolated value of these distances. Thus the fragment shader uses a modulated sine function of this distance to create the banding effect.

[1] http://www.cise.ufl.edu/research/sparse/cholmod/

[2] https://www.cise.ufl.edu/research/sparse/SuiteSparse/

[3] http://www.cs.columbia.edu/ keenan/Projects/

5. RESULTS

With my implementation, I was able to reproduce results comparable to those seen in the paper on a number of models. Most notably was the speed at which the distance function was able to be solved for different sets of source points. Below are some of the results I was able to produce using my implementation. Note, all results seen here use a time step $t = h^2$ where h is the mean length of edges in the mesh, as used in one version of the paper. The alternative method did not seem to produce stable or adequate results, with many of the distance functions unable to be solved. The distance function is visualized as evenly spaced contours of the distance field, and the near (red) to far (blue) color interpolation. Table 3. is a chart which recounts the running times on the laptop described above. The computation time is the same for any number of heat source points, as they are right hand constant terms for any given set of sources. There is a consistent range in the variability of the computation times, thus I give a range within which the distance function has yet to fall outside of. Thus it is easy to see the amortized benefits of this approach for multiple and possibly disparate heat source vertices. In future, geodesic paths will be extracted in one way or another from this method. Geodesic paths can be produced as *streamlines* of the vector fields that come from the heat method. We have two options, the gradient of heat diffusion X, or the gradient of the distance function itself. As the distance function ϕ is the solution to an optimization problem with regards to X, the vector fields gleaned from will potentially differ. Figure 5. demonstrates my investigation into the matter. The black arrows are of the diffuse gradient X, the blue arrows are gradient vectors of the distance function, $\nabla\phi$. The blue arrows are scaled proportional to the angle difference between the black and blue arrows (gradients), with max scaling once they differ by more than 90°. What this shows is that the diffusion attempts to create a divergence free vector field, causing smoothing vectors along the cut loci (areas that are equidistant to source points in opposing directions) of the distance field. The Poisson Equation allows for this and produced much more reasonable distance and gradient measurements at this cut loci (blue arrows on either side generally face away from one another).

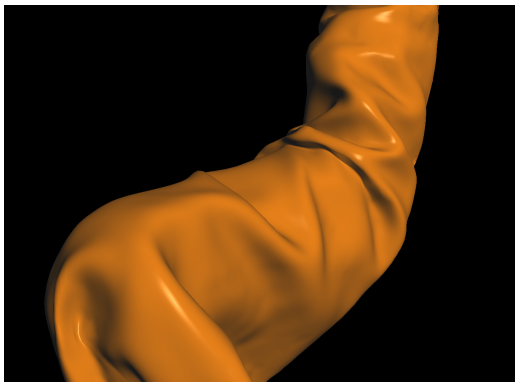


Figure 2: *Sleeve:* basic model view.

6. FUTURE WORK

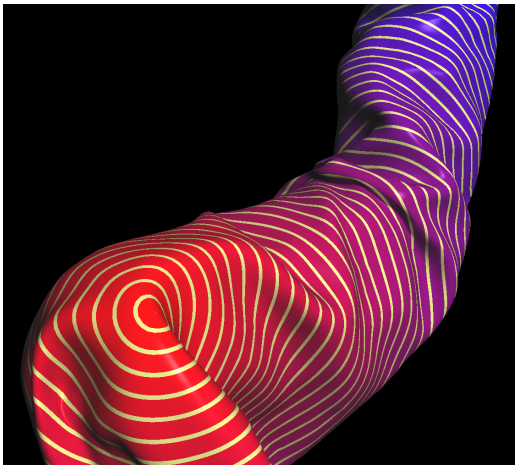


Figure 3: *Sleeve:* distance function visualized to one source point.

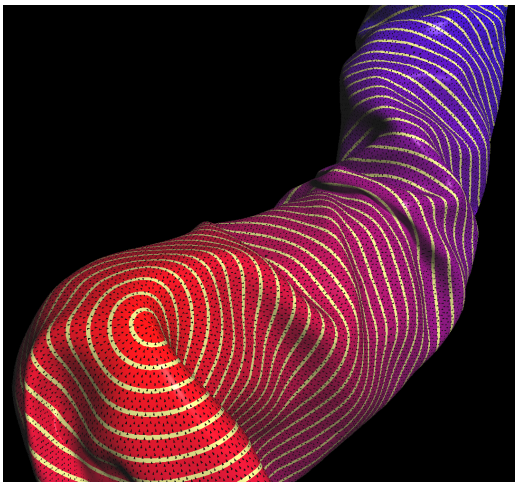


Figure 4: *Sleeve:* diffusion gradient.

With the success of producing a geodesic distance field as prescribed by Crane *et al.* [3], there are many avenues of future work to explore. Most notably is to produce geodesic paths over the mesh between points of interest. As mentioned above, I will use the gradient of the distance field as the bases for creating streamlines. Exploring other means of solving the linear systems have been discussed among the members of Dr. Przemyslaw Prusinkiewicz's lab, such as applying psuedo-inverse solutions, preconditioning and setting constraints, and adaptive/dynamic updating of the factorization for continuously deforming meshes. Potentially, this method will be applied to other domains (i.e volumetric simplices), and I will then re-implement the method to be a stand alone program with no external library dependencies.

7. CONCLUSION

This project has produced a reliable and scalable means of measuring distance on 2D triangulated meshes, with promising extensions of application and further development. As

a proof of concept, this project has succeeded, and will be built upon to suit various other domains of inquiry. I have learned a great deal from this project and look forward to working further on this and related projects which will use it.

8. REFERENCES

[1] M. Campen and L. Kobbelt. Walking on broken mesh: Defect-tolerant geodesic distances and parameterizations. In *Computer Graphics Forum*, volume 30, pages 623–632. Wiley Online Library, 2011.

[2] K. Crane, U. Pinkall, and P. Schröder. Spin transformations of discrete surfaces. *ACM Trans. Graph.*, 30, 2011.

[3] K. Crane, C. Weischedel, and M. Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG)*, 32(5):152, 2013.

[4] Y.-J. Liu, Q.-Y. Zhou, and S.-M. Hu. Handling degenerate cases in exact geodesic computation on triangle meshes. *The Visual Computer*, 23(9-11):661–668, 2007.

[5] R. H. MacNeal. *The solution of partial differential equations by means of electrical networks*. PhD thesis, California Institute of Technology, 1949.

[6] F. Mémoli and G. Sapiro. Distance functions and geodesics on submanifolds of \r^d and point clouds. *SIAM Journal on Applied Mathematics*, 65(4):1227–1260, 2005.

[7] J. A. Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3. Cambridge university press, 1999.

[8] J. A. Sethian and A. Vladimirsky. Fast methods for the eikonal and related hamilton–jacobi equations on unstructured meshes. *Proceedings of the National Academy of Sciences*, 97(11):5699–5703, 2000.

[9] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 553–560. ACM, 2005.

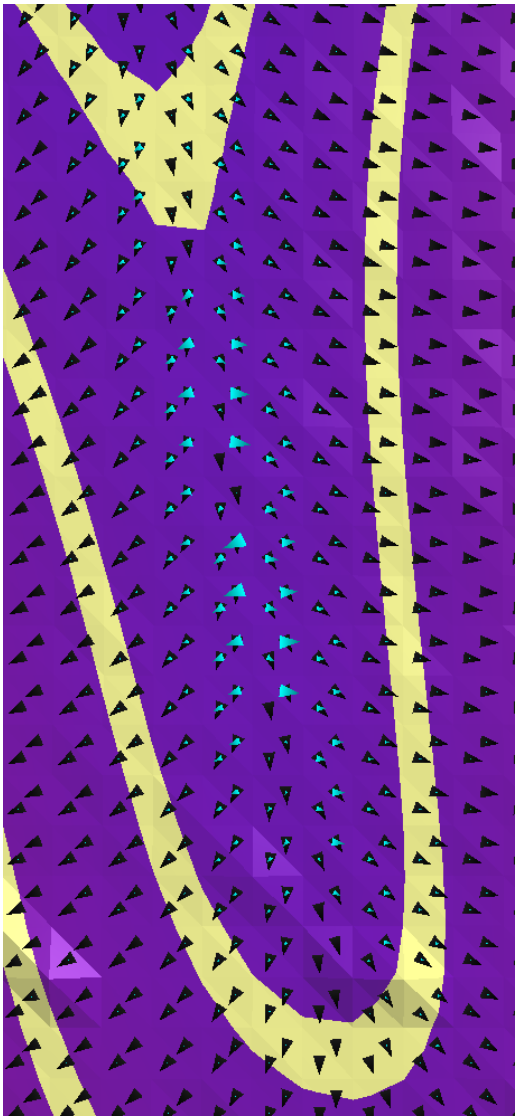


Figure 5: *Gargoyle:* X (black), accentuated $\nabla\phi$ (blue). This is to show how the Poisson solve, can remove the enforced smoothing of the diffusion.

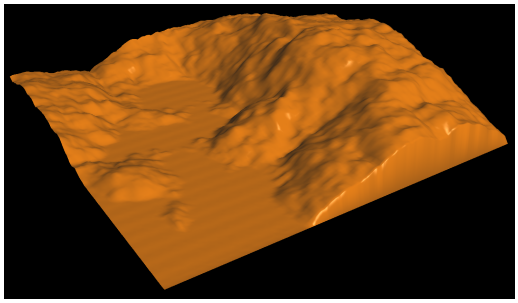


Figure 6: *Terrain:* **basic model view.**

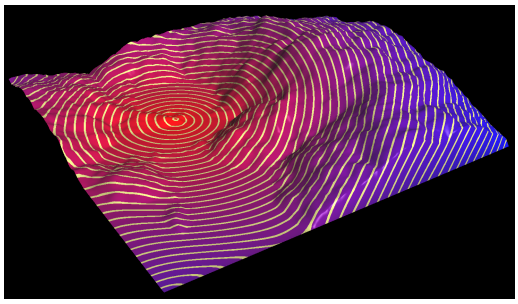


Figure 7: *Terrain:* **single source.**

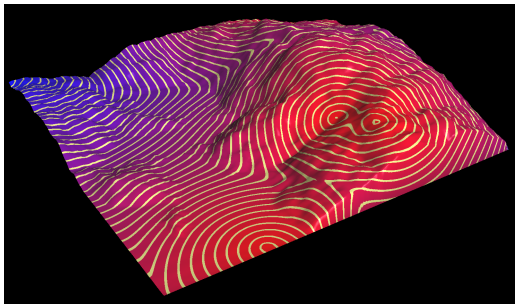


Figure 8: *Terrain:* **multiple sources.**

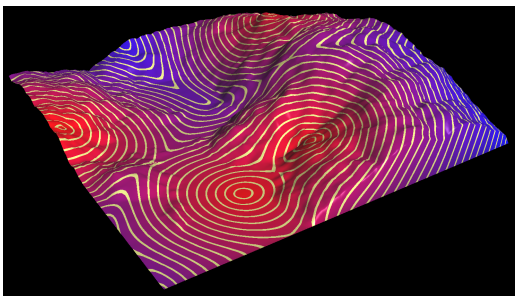


Figure 9: *Terrain:* **multiple sources.**

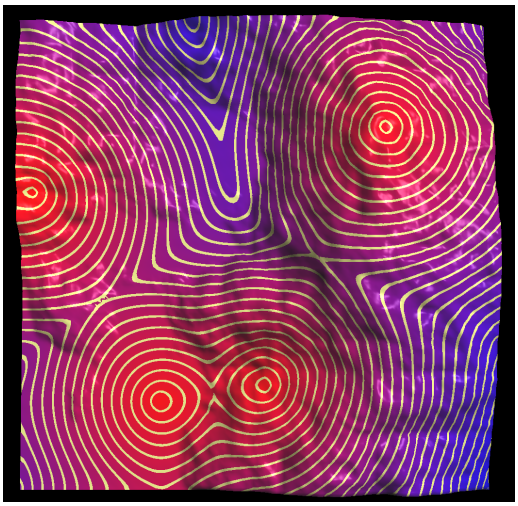


Figure 10: *Terrain:* **multiple sources.**

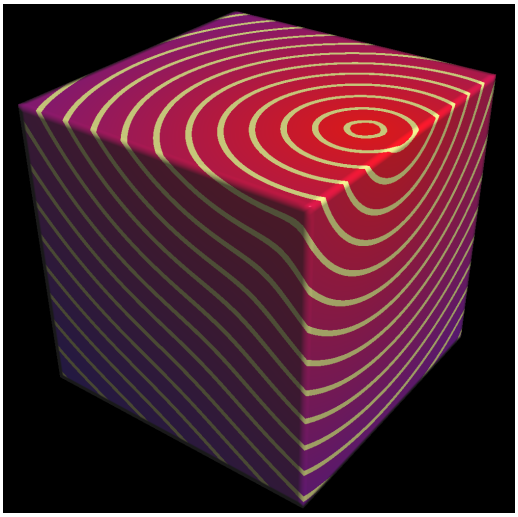


Figure 11: *Cube:* **demonstrates how distance propagates across flat and sharp features.**

Table 1: Program Performance

Model	number of faces	Pre Factorizations	Solve of Heat and Poisson equation
Bunny	+4k	30 ms	0-10 ms
Terrain	+33k	170 ms	10-20 ms
Sleeve	+37k	290 ms	20-30 ms
Cube	+30k	220 ms	10-30 ms
Gargoyle	+200k	2330 ms	120-130 ms

Table 2: Program Usage

Command Line Usage	Comments
./Geodesic path/to/model.obj	loads obj model for distance function calculation
Makefile pre-made commands	
make bunny	load Stanford Bunny model
make cubeBig	loads cube model
make terrain2	loads terrain model
make gargoyle	loads gargoyle model

Table 3: Program Interface

Command	Comments
mouse move (mm)+left click	rotate model around X and Y axes
(mm)+left click+CTRL	zoom in/out
(mm)+left click+SHIFT	move model in XY-plane
p	animate model rotating
1	wireframe view
2	flat shading view
3	smooth shading view
4	view *heat source* vertices
5	view heat diffusion gradient X
6	view distance function gradient error (see Results)
7	view distance field contours
+/-	randomly add/remove *heat source* vertices
h	solve distance function for random new set of *heat source* vertices

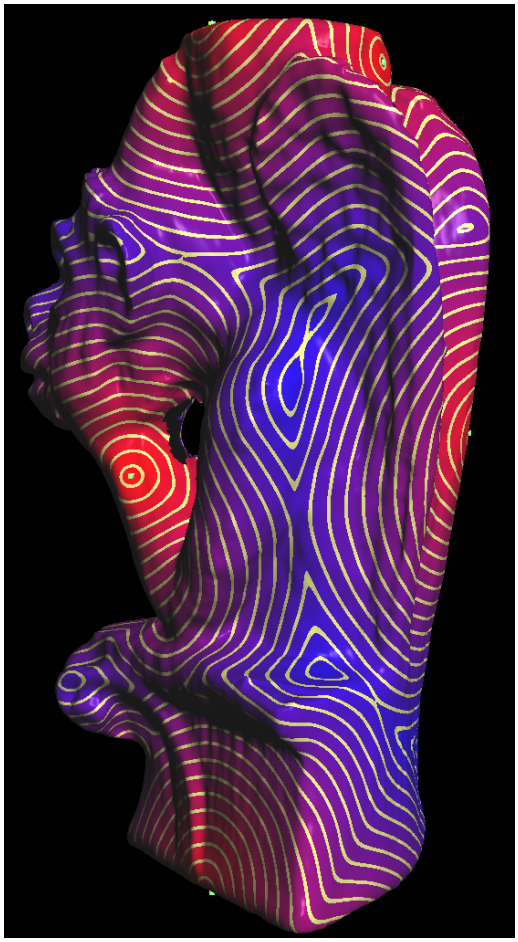


Figure 12: *Gargoyle:* **multiple sources, distance fields colliding.**

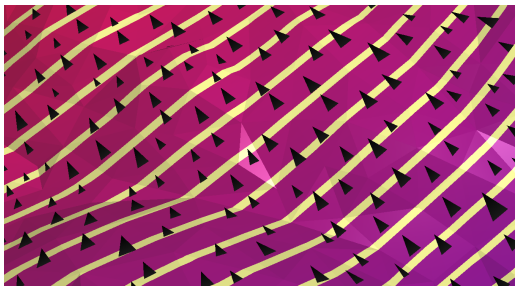


Figure 13: *Gargoyle:* **source.**

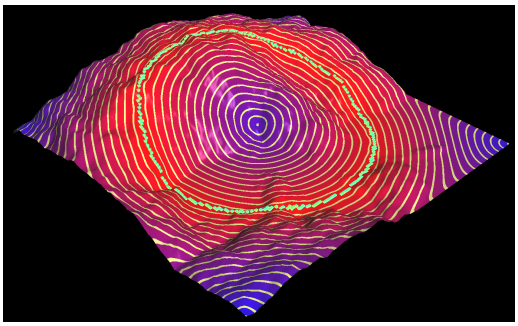


Figure 14: *Terrain:* **curve of sources.**

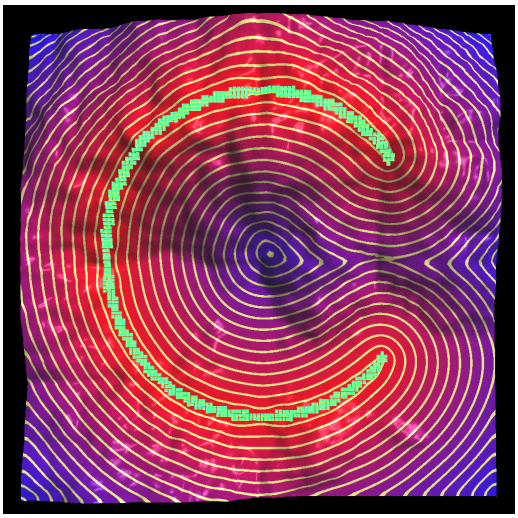


Figure 15: *Terrain:* **curve of sources.**

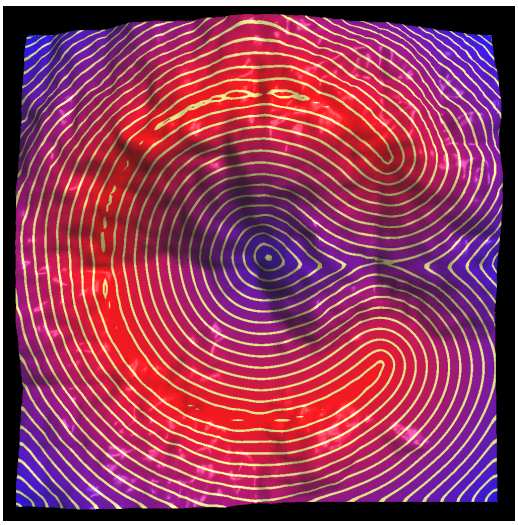


Figure 16: *Terrain:* **curve of sources.**

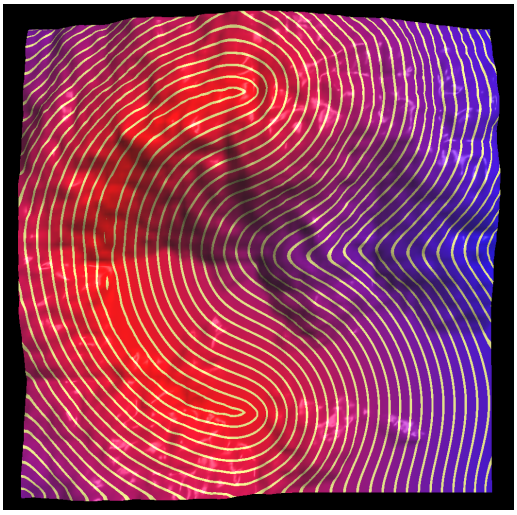


Figure 17: *Terrain:* **curve of sources.**

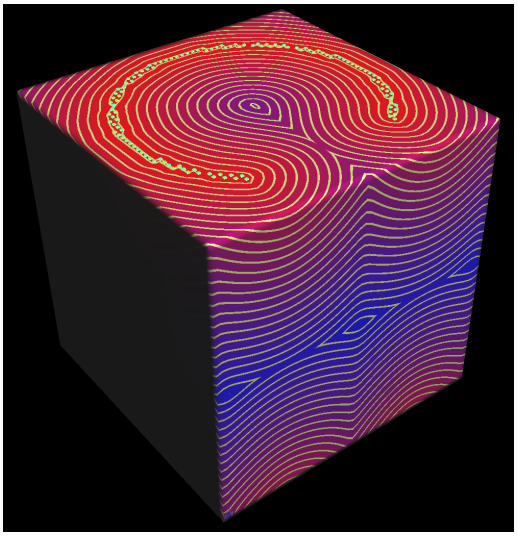


Figure 18: *Cube:* **curve of sources.**

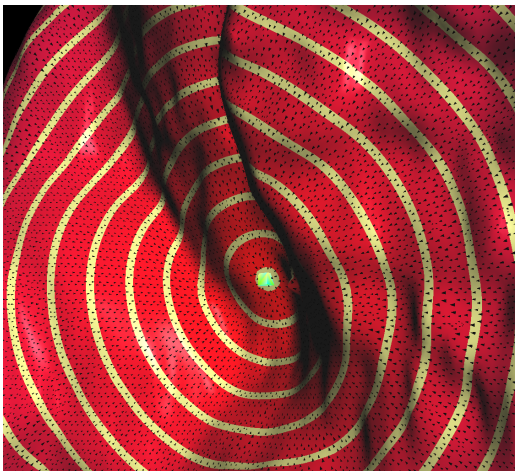


Figure 19: *Gargoyle:* **source.**

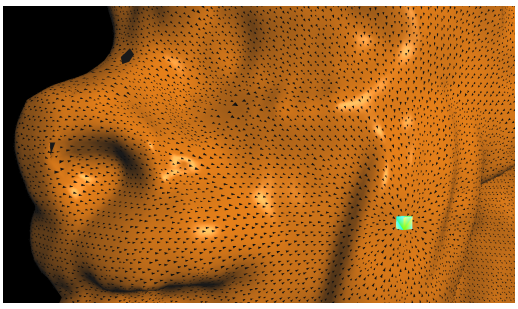


Figure 20: *Gargoyle:* **source.**