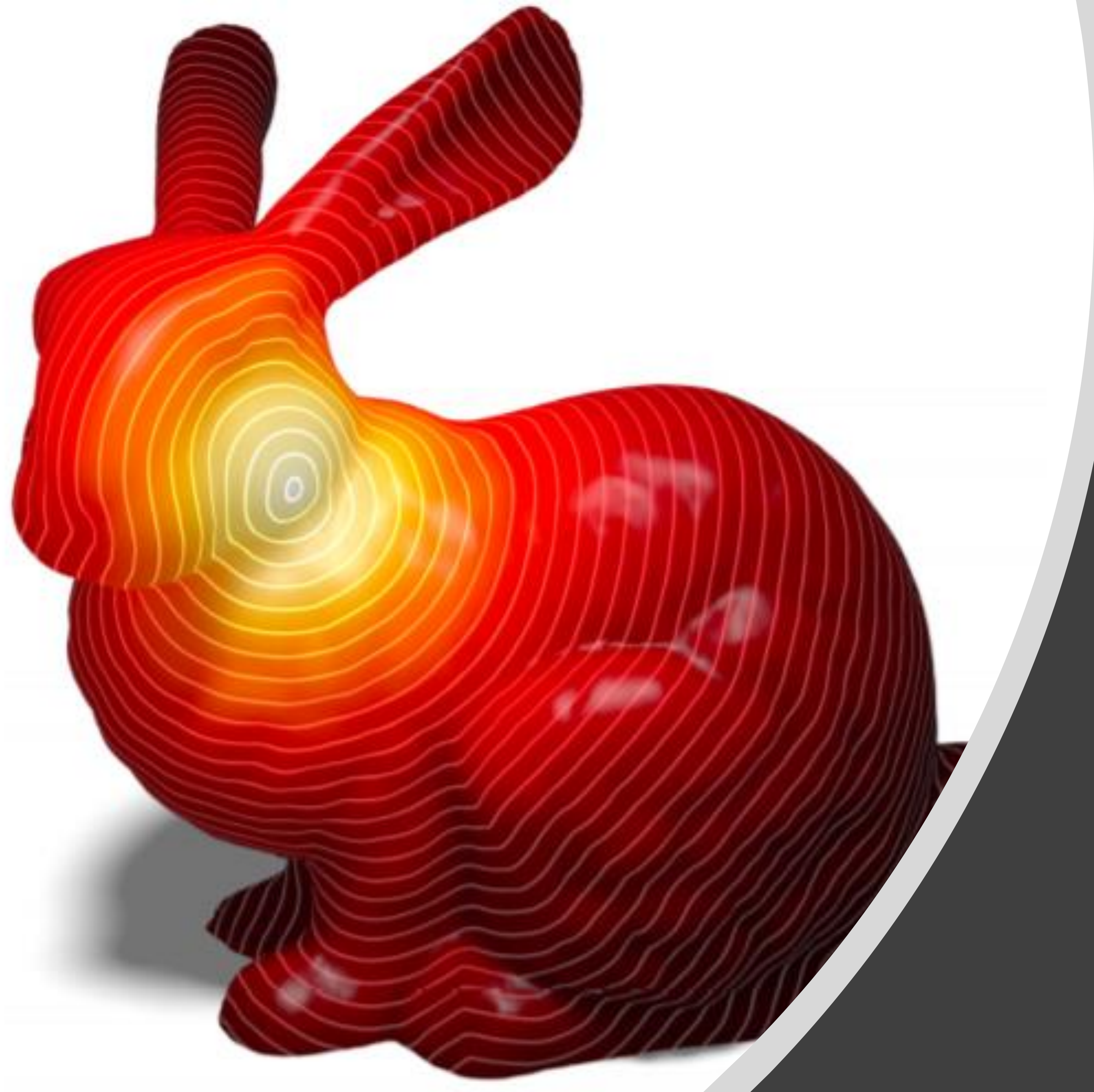


Nick Sale – 12/05/21 – Manifold Learning Working Group

Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow

Keenan Crane, Clarisse Weischedel, Max Wardetzky

Problem

Given a $\underbrace{\text{representation}}$ of a manifold M, $x, y \in M$,

Simplicial / Polygonal Mesh, Point cloud

approximate the geodesic distance $\underbrace{d_M(x,y)}$. or

find $\Phi_x(y) = d_M(x,y)$.

$$\min_{\substack{\gamma \in C^1([0,1], M) \\ \gamma(0), \gamma(1) = x, y}} L(\gamma)$$

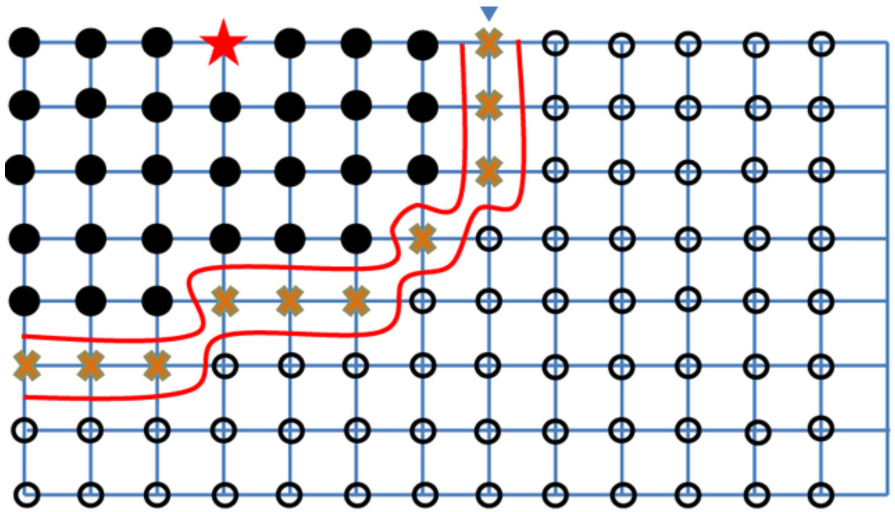
Traditional Approach

Approximate a solution to the $\underbrace{\text{eikonal equation}}_{|\nabla \phi_x| = 1}$

with boundary conditions $\phi_x(x) = 0$.

E.g. via fast marching

$O(N \log N)$, must recompute from scratch for each $x, y \in \mathcal{M}$.



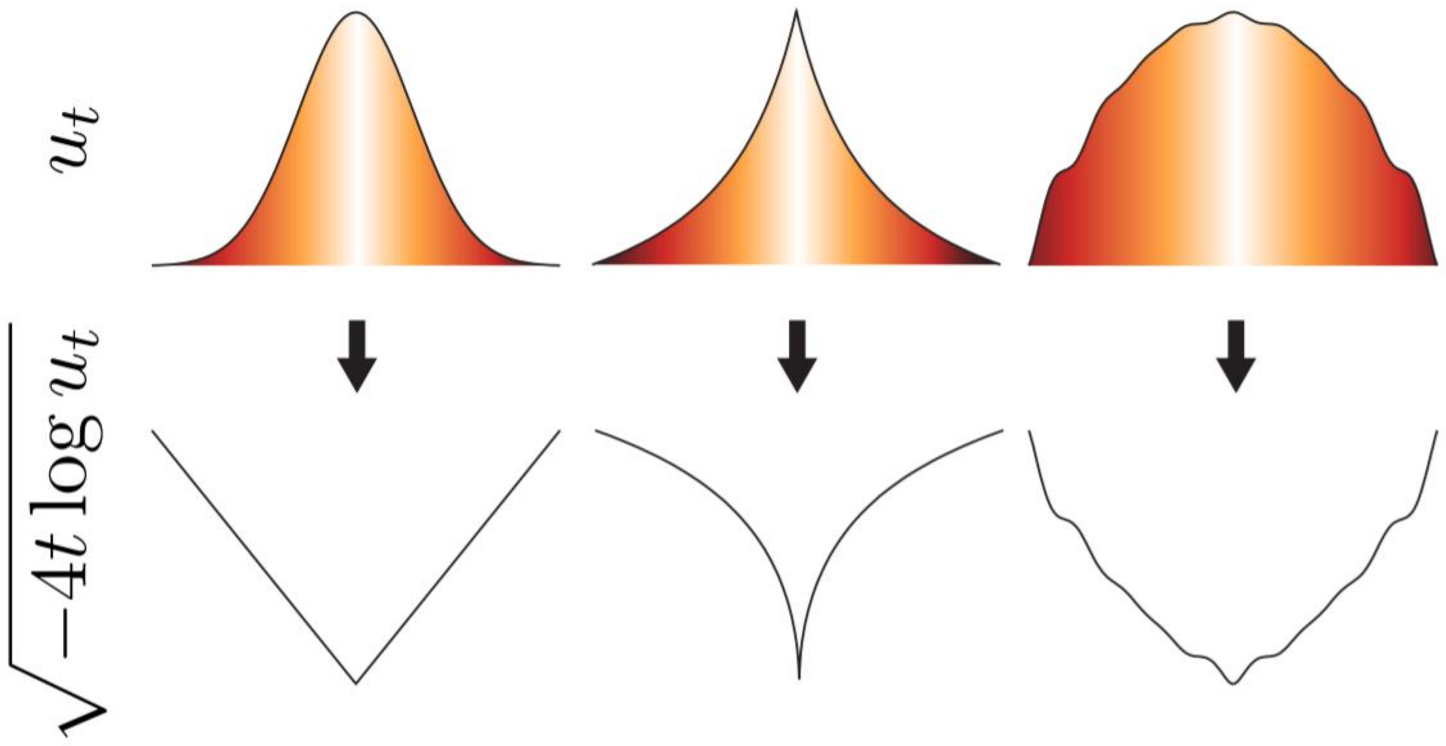
Idea

Simulate heat flow for a robust, recomputable distance estimation, appealing to $\underbrace{\text{Varadhan's formula}}$

$$d_M(x,y) = \lim_{t \downarrow 0} \sqrt{-4t \log k_{t,x}(y)} \ ,$$

$k_{t,x}$ solves $\dot{u} = \Delta u$
$= \nabla \cdot \nabla u$
$u_0 = \delta_x$

Is that it? No: There's a problem



This isn't robust.

Idea Continued

Don't use the magnitude of $k_{t,x}$, just its gradient $\underbrace{\text{direction}}$ (After all, we know that $|\nabla \phi_x| = 1$.)

$$X = \frac{-\nabla k_{t,x}}{|\nabla k_{t,x}|}$$

Then find a ϕ_x whose gradient looks like X.

$$\phi_x = \min_{\phi} \int_M |\nabla \phi - X|^2 \quad \overset{}{\underset{E\text{-}L}{\Longleftrightarrow}} \quad \text{Solve } \nabla \cdot \nabla \phi = \nabla \cdot X$$

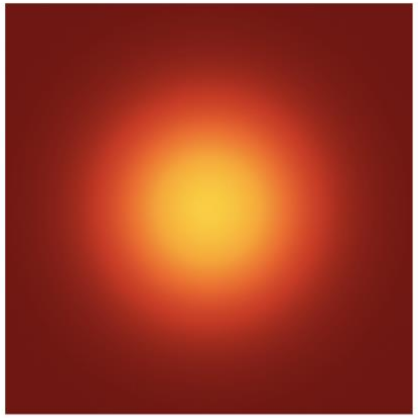
(Poisson eqn.)

Algorithm

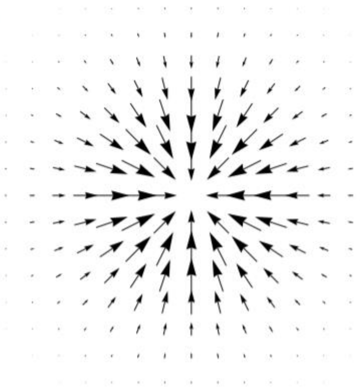
I. Integrate heat flow $\dot{u} = \Delta u$ for fixed time t

II. Evaluate the vector field $X = \dfrac{-\nabla u}{|\nabla u|}$

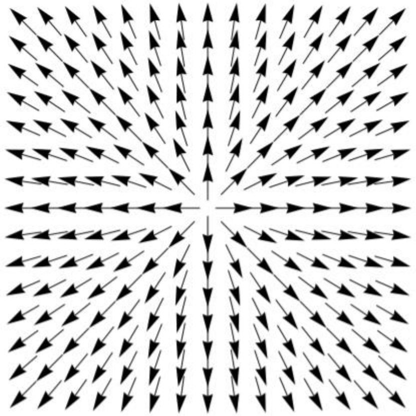
III. Solve the Poisson eqn. $\Delta \phi = \nabla \cdot X$



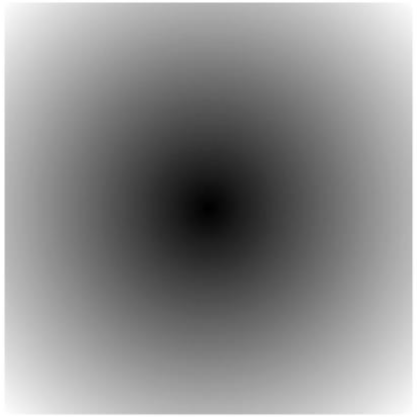
u



∇u



X



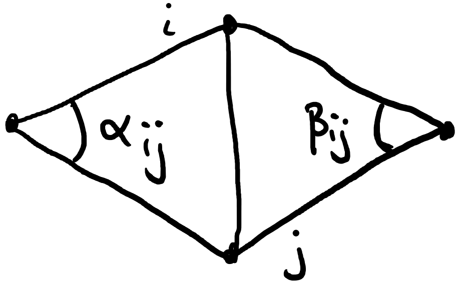
ϕ

Computation on Discrete Domain

- $u \in \mathbb{R}^{|V|}$ ← # of vertices
- Δu becomes $\mathcal{L} u$ ← appropriate choice of Laplacian matrix
- Heat equation solved in 1 backwards-Euler step ← $(I - t\mathcal{L})u = \delta_x$
- Poisson eqn. becomes $\mathcal{L}\phi = b$ ← Appropriately computed divergence of X.

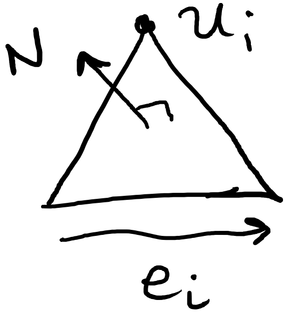
e.g. Simplicial Meshes

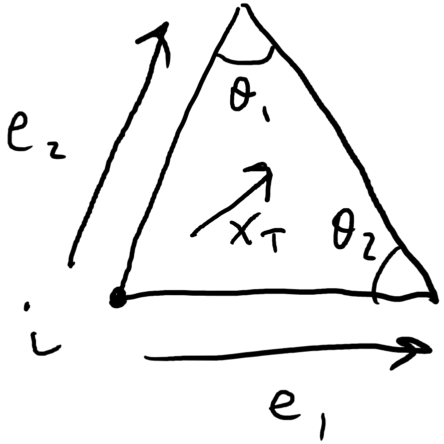
$\frac{1}{3} \sum_{T \ni i} \text{Area}(T)$



- $$(\Delta u)_i = \frac{1}{2A_i} \sum_j (\cot \alpha_{ij} + \cot \beta_{ij})(u_j - u_i)$$

- $$(\nabla u)_T = \frac{1}{2A(T)} \sum_{i \in T} u_i (N \times e_i)$$





- $$(\nabla \cdot X)_i = \frac{1}{2} \sum_{T \ni i} \cot \theta_1 (e_1 \cdot X_T) + \cot \theta_2 (e_2 \cdot X_T)$$

Time Step

In the discrete setting it's not necessarily the case that using a smaller t gives better results.

LEMMA 1. *Let $G = (V, E)$ be the graph induced by nonzeros in any real symmetric matrix A, and consider the linear system*

$$(I - tA)u_t = \delta$$

where I is the identity, δ is a Kronecker delta at a source vertex $u \in V$, and $t > 0$ is a real parameter. Then generically

$$\boxed{\phi = \lim_{t \to 0} \frac{\log u_t}{\log t}}$$

where $\phi \in \mathbb{N}_0^{|V|}$ is the **graph distance** (i.e., *number of edges*) *between each vertex $v \in V$ and the source vertex u.*

i.e. with small t we end up with u a function of the graph distance.

Time Step Continued

$h = \operatorname{mean}_{i,j} \{ \|v_i - v_j\|_2 \}$

Instead use $t = mh^2$ for some choice of m.

$h^2 \Delta$ is invariant wrt scaling/refinement

In practise, $m = 1$ works well — e.g. recovers ℓ_2 distance on regular grid.

Increasing beyond this $(m > 1)$ yields a smoothed distance function.

Performance

Computation comes down to solving 2 linear systems both amenable to $\underbrace{\text{Sparse Cholesky factorisation}}$.

not actually, but pretty much $O(n)$ for these sorts of systems.

This easily allows reuse of computations.

Table I. Comparison with fast marching and exact polyhedral distance. Best speed/accuracy in **bold**; speedup in orange.

MODEL	TRIANGLES	HEAT METHOD				FAST MARCHING			EXACT
		PRECOMPUTE	SOLVE	MAX ERROR	MEAN ERROR	TIME	MAX ERROR	MEAN ERROR	TIME
BUNNY	28k	**0.21s**	**0.01s** (28x)	3.22%	**1.12%**	0.28s	**1.06%**	1.15%	0.95s
ISIS	93k	**0.73s**	**0.05s** (21x)	1.19%	**0.55%**	1.06s	**0.60%**	0.76%	5.61s
HORSE	96k	**0.74s**	**0.05s** (20x)	1.18%	**0.42%**	1.00s	**0.74%**	0.66%	6.42s
KITTEN	106k	**1.13s**	**0.06s** (22x)	0.78%	**0.43%**	1.29s	**0.47%**	0.55%	11.18s
BIMBA	149k	**1.79s**	**0.09s** (29x)	1.92%	0.73%	2.62s	**0.63%**	**0.69%**	13.55s
APHRODITE	205k	**2.66s**	**0.12s** (47x)	1.20%	**0.46%**	5.58s	**0.58%**	0.59%	25.74s
LION	353k	**5.25s**	**0.24s** (24x)	1.92%	0.84%	10.92s	**0.68%**	**0.67%**	22.33s
RAMSES	1.6M	**63.4s**	**1.45s** (68x)	0.49%	**0.24%**	98.11s	**0.29%**	0.35%	268.87s

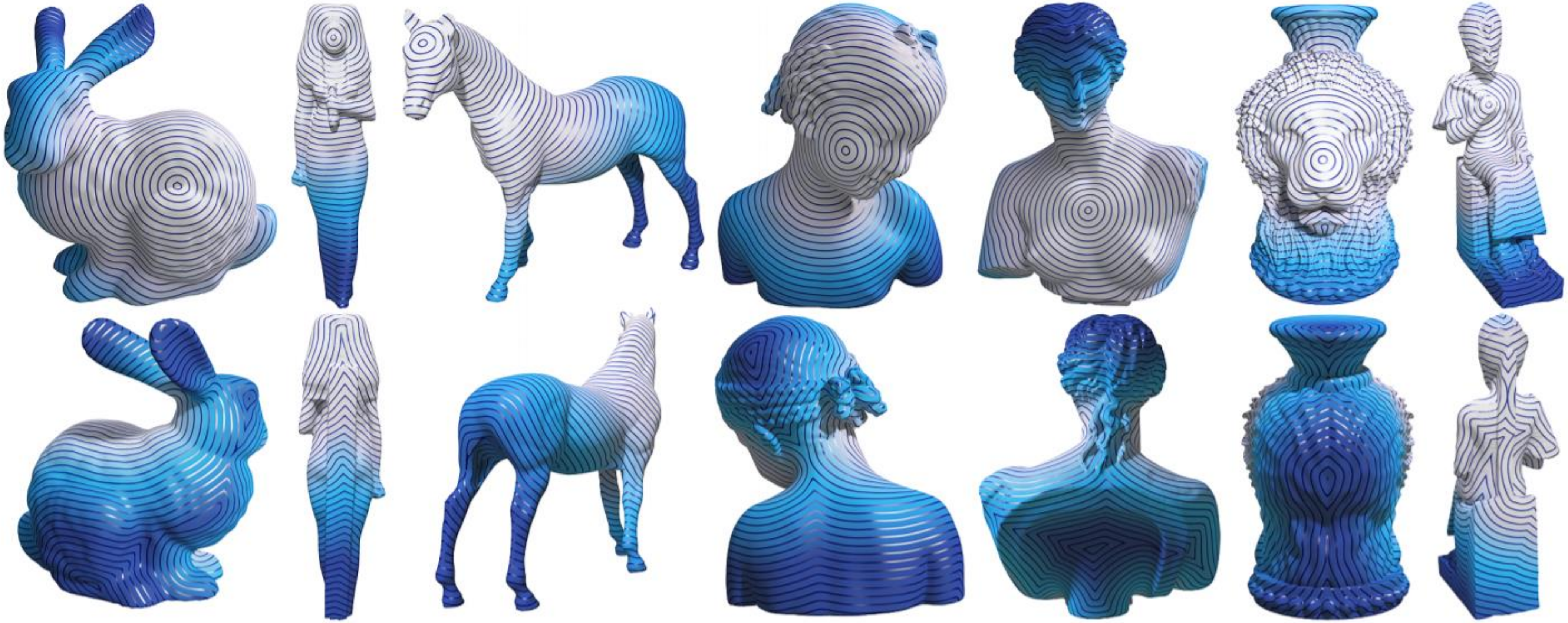


Fig. 13. Meshes used in Table I. Left to right: BUNNY, ISIS, HORSE, BIMBA, APHRODITE, LION, RAMSES[1].