A Multi-Resolution Approach to Heat Kernels on Discrete Surfaces

Amir Vaxman
Technion

Mirela Ben-Chen
Stanford University

Craig Gotsman
Technion

Abstract

Studying the behavior of the heat diffusion process on a manifold is emerging as an important tool for analyzing the geometry of the manifold. Unfortunately, the high complexity of the computation of the heat kernel – the key to the diffusion process - limits this type of analysis to 3D models of modest resolution. We show how to use the unique properties of the heat kernel of a discrete two dimensional manifold to overcome these limitations. Combining a multi-resolution approach with a novel approximation method for the heat kernel at short times results in an efficient and robust algorithm for computing the heat kernels of detailed models. We show experimentally that our method can achieve good approximations in a fraction of the time required by traditional algorithms. Finally, we demonstrate how these heat kernels can be used to improve a diffusion-based feature extraction algorithm.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

Keywords: heat diffusion, heat kernel, multi-resolution, matrix exponential

1 Introduction

Shape analysis and understanding is an important prerequisite to many shape manipulation algorithms. Whether it is shape deformation, repair, or compression, the better we understand the essence of the shape, the better the algorithm will perform. Many attempts have been made to extract salient shape information from the discrete representation of a surface, and here we focus on a recent approach – *heat diffusion* - which has great potential due to its multi-scale character. The heat diffusion process describes the evolution of a function on the surface over time. It is governed by a quantity called the *heat kernel* (HK): $k_t(x,y)$, which is uniquely defined for any two points x, y on the surface, and a time parameter t. The heat kernel has a geometric interpretation as an exploration process: starting at the point x, and exploring the surface by unit steps in random tangent directions, the probability that y has been reached at the time t is exactly $k_t(x,y)$. Since it aggregates information about all the possible ways to walk between two points on the surface, the heat kernel captures much of the structure of the surface.

In the context of shape analysis, the heat kernel of a surface has some remarkable properties, explored by Sun *et al.* [2009]. First, in a sense, the heat kernel uniquely defines a shape, as two shapes will have the same heat kernel if and only if they are isometric. Second, all the information in the heat kernel is encoded in its diagonal $k_t(x,x)$ - the probability of returning to x at the time t. The rest of the heat kernel is redundant in the sense that two heat kernels are identical if and only if their diagonals coincide. For this reason, Sun *et al.* [2009] call the diagonal the *heat kernel signature* (HKS). Finally, the heat kernel is multi-scale in the time parameter t, with larger times aggregating information about larger neighborhoods of the point x. These properties make the heat kernel (or its diagonal) a very effective tool in comparing different shapes at different scales, and in identifying prominent features of a shape.

A different way to express the information encoded in the heat kernel is through *diffusion distances* [Lafon 2004]. The diffusion distance between x and y contains information about all possible ways to walk from x to y and back, within the time t. As such, these distances are robust to small topological changes and have been recently applied to create an isometric-invariant hierarchical segmentation of a shape [deGoes *et al*. 2008].

Unfortunately, there is no free lunch, and such high quality and detailed information about a shape comes with a hefty computational price tag. On a surface M, the heat kernel k_t may be expressed in terms of the eigenfunctions of the *Laplace-Beltrami* operator of M. On a discrete surface, this operator is the $n \times n$ Laplacian matrix L, where n is the number of vertices of the mesh. Thus the most straightforward way to compute the heat kernel k_t - also a $n \times n$ matrix for any t - is by a spectral decomposition of L. For large t, a small number of eigenvectors having the smallest eigenvalues are sufficient for a good approximation of k_t, as the influence of an eigenvector φ with eigenvalue λ on k_t decreases exponentially like $\exp(-\lambda t)$. However, as t becomes smaller, prohibitively many eigenvectors are required to compute k_t, and this method is no longer practical. This limitation imposes constraints on applications based on the heat kernel; for example, the size of the features that can be extracted using the heat kernel signature depends on the minimal t for which k_t can be computed.

The heat kernel can also be expressed as the exponential of the Laplacian operator: $k_t = \exp(-tL)$, and computed accordingly using the power series of the exponential. However, such a computation is numerically error-prone, as the stability of existing methods for computing the matrix exponential strongly depend on the norm of the matrix, hence can only be applied if t is very small. Thus we conclude that existing methods can compute the heat kernel of a large mesh only for either very large or very small values of t.

As mentioned previously, one of the more useful properties of the heat kernel is its multi-scale nature. At each time scale it aggregates information from a local environment of a point whose diameter is proportional to the time scale. As the information is aggregated, it is smoothed out, so the detailed geometry of the surface plays a role only at small time scales. In fact, as observed

by Mémoli [2009], as t increases, the heat kernel provides a coarser view of the geometry of the surface. Hence, *the heat kernel at large time t can be approximated well by a sparse matrix related to a lower resolution version of the original surface*. Fig. 1 demonstrates this, by showing the heat kernel of a vertex of the gargoyle model, at three time scales at three different resolutions. Note that on the "diagonal" of the figure, where the time scale t matches the resolution of the mesh, it is indeed sparse. In addition, there is almost no noticeable difference between the heat kernel at the same time scales on the different resolutions, as long as the resolution is detailed enough for the given time scale (i.e. it is "above the diagonal" of the figure).

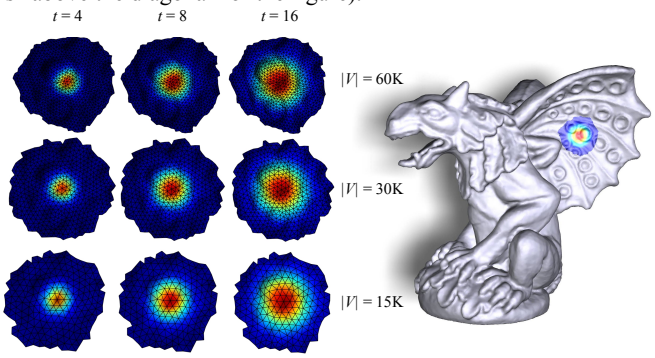


Figure 1: *(left) The heat kernel of a single vertex at three time scales and three resolutions. On the "diagonal" the heat kernel is sparse. (right) The relevant region of the gargoyle.*

From a linear algebraic point of view, the multi-scale property means that the numerical rank of k_t for large t is much smaller than n. This observation was also exploited by Coifman and Maggioni [2006] for generating a multi-scale basis based on diffusion maps, called *diffusion wavelets*.

1.1 Contribution

We describe an algorithm, taking advantage of the multi-scale property, to efficiently compute k_t for any t. We compute the low-rank operator on a *down-sampled* version of the surface, whose resolution is compatible with the time t. With this approach, the computation is efficient, in the sense that it uses only the bare minimum of information needed, by restricting it to the correct resolution level. As an added benefit, the locality of the heat kernel causes the operator to be sparse at the lower resolution, further contributing to the efficiency of the method. To efficiently compute the heat kernel on each resolution level, we describe a novel approximation to the matrix exponential, specifically tailored to our scenario – a sparse operator at a small time $t << 1$ - which converges faster and is more stable than existing approaches.

Although we do not provide a formal proof, we demonstrate empirically how our method allows computing the heat kernel of very detailed models, for all scales, ranging from the mesh edge-length to its full diameter.

1.2 Previous Work

Multi-resolution

The multi-resolution approach has emerged as a robust way of coping with the ever-growing size of digital datasets. The main idea is to process a low resolution version of the data set (which will probably consume much less computing resources than processing the original dataset), and then refine the result to fit a higher resolution version, assuming that the difference due to the additional detail can be computed relatively easily. This can be applied repeatedly over multiple resolution levels, using each level as a good starting point for the next level. A simple uni-directional sweep from lowest resolution to highest resolution is sometimes enough, but in some cases sweeps in both directions are required. This is the principle behind the modern multi-grid approach [Wesseling 2004]. When the data has a regular structure (i.e. a regular square grid), the structural relationship between levels is quite straightforward and is the basis for the standard multi-grid approach. For unstructured datasets (such as triangle meshes), *algebraic multi-grid* was developed. In many geometry processing algorithms, a uni-directional sweep is sufficient, sometimes called a *hierarchical* method. One of the first multi-resolution schemes for unstructured meshes, applied to surface modeling, was presented by Kobbelt *et al.* [1998], based on a progressive mesh structure [Hoppe 1996]. More recent work adapts algebraic multigrid approaches to surface meshes [Aksoylu *et al*. 2005], for parameterization [Sheffer *et al*. 2005], and deformation [Shi *et al*. 2006].

A completely different approach to multi-resolution in the context of heat diffusion is taken by the concept of *Diffusion Wavelets* [Coifman and Maggioni 2006]. There, the diffusion operator is simply assumed to have lower rank for lower resolution levels, and as such its rank is reduced in a brute-force manner, using a Gram-Schmidt procedure. As we will see, our method, in contrast, directly generates the "low rank" operator by defining it on a coarser mesh, avoiding expensive orthogonalization.

Matrix Exponential Approximations

As mentioned above, the HK is related to the matrix exponential of the Laplacian, thus efficient computation of exp(-tL) may be useful in approximating k_t. The problem of computing the exponential of a matrix is ubiquitous in scientific computing, thus a large body of research exists on this topic. We discuss only the most relevant approaches here, and refer the reader to Moler and Van Loan [2003] for a thorough survey of existing algorithms.

The matrix exponential is defined as the power series:

$$\exp(X) \triangleq S^T(X) \triangleq \sum_{k=0}^{\infty} \frac{1}{k!} X^k \quad (1)$$

A naïve approach to compute an approximation to exp(X) is to truncate the series (1) after the first M terms. This is known to be one of the *worst* methods for computing the exponential [Moler and Van Loan 2003], as it involves computations using very large numbers, hence is unstable and numerically error-prone.

A better approximation method is to use eigen-decomposition of X: If $XU=UD$, then

$$\exp(X) = U \exp(D) U^{-1} \quad (2)$$

Since D is diagonal, $(\exp(D))_{ij}$ is simply $\exp(D_{ij})$. However, since this method involves eigen-decomposition of X, it is not practical for large matrices.

The power series S^T is just the standard Taylor expansion of a function f as a polynomial. Alternatively, one can approximate f using a *rational function*, such as the *Pade approximation*:

$$f(x) \approx \sum_{k=0}^{N} a_k x^k \Big/ \sum_{k=0}^{M} b_k x^k$$

The coefficients are computed by requiring that this approximation agrees with the Taylor expansion, up to the highest coeffi-

cient. It is well known [Arioli *et al*. 1996] that for the same number of coefficients, the Pade approximation is always superior to the Taylor approximation. However, it may still suffer from large round-off errors, especially if the norm of X is large. A standard method to overcome this is the "scaling and squaring" method, which is based on the following property of matrix exponentials:

$$\exp(X) = \left(\exp(X/m)\right)^m$$

Rescaling X by dividing it by a constant reduces its norm, stabilizing the computation of the rational expansion. If m is a power of 2, the final result can be obtained by repeatedly squaring $\exp(X/m)$. This algorithm – combining scaling and squaring with the Pade approximation (for $\exp(X/m)$) - is one of the best available, and is implemented in MATLAB's *expm* function [Moler and Van Loan 2003]. However, it is still not suitable for computing the exponential of the Laplacian matrix of a large mesh, as the Pade approximation requires the inversion of a very large matrix. Our algorithm also uses the scaling and squaring approach. However, thanks to the multi-resolution component, the resulting heat kernel is sparse, thus the squaring is very efficient. In addition we avoid the expensive and potentially unstable matrix inverse operation, by using a polynomial instead of a rational expansion.

For some applications, such as solving partial differential equations, the full matrix exponential is not required, rather just the product of the matrix exponential with a vector:

$$\psi(t) = \exp(-tX)\psi(0)$$

This would, in fact, be the case if we would like to compute the heat kernel of just a single point x with respect to all other points (a single *column* of the heat kernel matrix). For such problems, specialized methods exist [Hochbruck and Lubich 1997] exploiting the fact that X is sparse, and do not compute the full exponential. These methods, however, will not be able to efficiently compute the *diagonal* of $\exp(-tX)$, as this will boil down to computing the entire matrix.

1.3 Algorithm Overview

Before diving into the details, we give a brief overview of our algorithm for computing the heat kernel. Given a triangle mesh M with n vertices, and a positive number t, we wish to compute select entries of the $n \times n$ heat kernel matrix K_t – these can be entries on the diagonal, a few columns, or any other small set of entries.

First, we generate a set of m meshes M_i, with $M_1 = M$ such that $n_i = |M_i|$, $n_{i+1} = n_i/2$, and define a multi-resolution structure, which allows us to map any point $x \in M_i$ to the original mesh M. Now, given t, we find the coarsest resolution i, such that the rank of the heat kernel k_t is smaller than the size of mesh M_i. Then we compute the heat kernel of the mesh M_i, using the matrix exponential of the Laplacian operator L_i, at the same time scale t. The computation is done using "scaling and squaring" combined with a new power sum series for the matrix exponential, which is tailored for fast convergence at $t << 1$. As the heat kernel is local at this resolution level, the resulting matrix $\exp(-tL_i)$ will be sparse. Finally, using the multi-resolution structure, we map back to the highest resolution level only the required entries of the heat kernel.

2 Multi-Resolution Heat Kernels

Let $M=(V,F)$ be a discrete surface given as a triangle mesh, where V are its vertices and F its faces, and let L be its Laplacian matrix. There are many definitions for L, and any of them can be used, as long as L has a full set of (linearly independent) eigenvectors with real eigenvalues. Let $\{\lambda_i\}$ be the set of eigenvalues of L, and $\{\varphi_i\}$ their corresponding eigenvectors. Then the *heat kernel* of M at time $t > 0$, is defined as the matrix:

$$K_t^M(u,v) = \sum_{i=1}^{n} \exp(-\lambda_i t)\varphi_i(u)\varphi_i(v) \quad (3)$$

where $n = |V|$ and $u,v \in V$. If, in addition, L is symmetric, then the eigenvectors are orthogonal, implying $K_t^M = \exp(-tL)$.

We now proceed to define our multi-resolution structure. We later use it to perform a coarse solve on a low resolution version of the mesh, followed by a projection to the higher resolution version.

***Definition* 1:**
Given a mesh M, and constants d, $C \in Z$, a *multi-resolution structure* $MR_{d,C}(M)$, is a set of meshes $\{M_1, M_2, \ldots, M_m\}$, and a set of mappings $\{f_2, f_3, \ldots, f_m\}$ such that $M_i = (V_i, F_i)$, $|V_i| = n_i$, satisfying:

1. $M_1 = M$
2. $n_{i+1} = n_i/d$
3. $M_{i+1} = \arg\min_N d_H(N, M_i)$, where N is a 2-manifold triangle mesh with n_i/d vertices, and d_H is the Hausdorff distance.
4. $n_m < C$
5. f_i: $V_i \rightarrow M_{i+1}$ maps each vertex of V_i to the point closest to it (not necessarily a vertex) on M_{i+1}.

Thus, we can define a group of meshes, and mappings between them, such that each resolution level has a fraction of the vertices of the finer level. The constant C is determined by the available resources, in the sense that C is the maximal number of vertices, for which it is feasible to compute the full spectral decomposition of the Laplacian operator using dense matrices. The number of meshes in the structure, m, is determined by C and d. Note that when computing (3) we use an approximation, as computing the true minimum is a difficult problem.

Using our multi-resolution structure, we can infer functions on the vertices of a fine level from the functions defined on the vertices of a coarser level, as follows. Let g^{i+1} be a function on the vertices, at the resolution level i+1, g^{i+1}: $V_{i+1} \rightarrow$R. The *prolongation of* g^{i+1} to the level i is:

$$g^i(v) = \sum_{l=1}^{3} w_l(p,u^l) g^{i+1}(u^l), \quad p = f_i(v) \in (u^1,u^2,u^3) = t_p \quad (4)$$

where w_l are the barycentric coordinates of p in t_p with respect to u^l. We can write (4) in matrix notation, using a *prolongation matrix* P_{i+1}^i:

$$\boldsymbol{g}^i = P_{i+1}^i \boldsymbol{g}^{i+1} \quad (5)$$

By recursively applying (5), we can infer the values at the vertices of the finest level from the values on the vertices on any resolution level h:

$$\boldsymbol{g}^M = \boldsymbol{g}^1 = P_2^1 P_3^2 \ldots P_h^{h-1} \boldsymbol{g}^h = P_h^1 \boldsymbol{g}^h$$

Define $P_i^{i+1} = (P_{i+1}^i)^T$. Then, similarly, given a matrix of values A^{i+1} representing a bivariate function A^{i+1}: $V_{i+1} \times V_{i+1} \rightarrow R$, we can prolong it to the i-th level using:

$$A^i = P_{i+1}^i A^{i+1} P_i^{i+1} \quad (6)$$

which is the natural extension of (4) to functions of two vertices. Note that if A^{i+1} was symmetric, then A^i will be symmetric as well.

Finally, we define the *multi-resolution (MR) heat kernel* on M:

***Definition* 2:**
Given $t > 0$ and $\varepsilon > 0$, let $r_\varepsilon(t)$ be the numerical rank of K_t^M, i.e. the number of eigenvalues λ_i of the Laplacian matrix, such that $\exp(-t\lambda_i) > \varepsilon$. In addition, let h be the coarsest resolution level, such that $cn_h > r_\varepsilon(t)$, for some constant $0 < c \le 1$. Then the *multi-resolution heat kernel* on M is:

$$\hat{K}_t^M = P_h^1 K_t^h P_1^h \tag{7}$$

In the special case that h=1, we obtain $\hat{K}_t^M = K_t^h$.

Intuitively, given a time t, we compute the multi-resolution heat kernel by finding the coarsest resolution level in which the fraction of *active* eigenvectors – those for which $\exp(-t\lambda_i) > \varepsilon$ – is smaller than the constant c. Then we prolong its values through the multi-resolution levels, until we reach the finest level. This idea is based on the relationship between the first eigenvalues/eigenvectors of L^M and the first eigenvalues/prolonged eigenvectors of L^h, respectively.

To make this more precise, consider the case $c = 1$. Given that the numerical rank of K_t^M is $r_\varepsilon(t) = r$, we may approximate (3) by:

$$K_t^M(u,v) \approx \sum_{i=1}^{r} \exp(-\lambda_i t)\varphi_i(u)\varphi_i(v) \tag{8}$$

Now we look for a low-resolution mesh, M_h, upon which we can approximate the first r eigenvectors and eigenvalues of M. Obviously, if $n_h < r$, we would not have enough eigenvectors to use. Hence, we choose the coarsest resolution level h, such that $n_h > r$. Let $\tilde{U}$ be the matrix whose columns are the first r eigenvectors of L, and $\tilde{D}$ a diagonal matrix of the first r eigenvalues, and similar for L^h. Re-writing (8) in matrix notation, we get:

$$K_t^M \approx \tilde{U}\exp(-t\tilde{D})\tilde{U}^T$$

Using the same formulation for K_t^h, we have:

$$\hat{K}_t^M = P_h^1 K_t^h P_1^h \approx P_h^1 \tilde{U}^h \exp(-t\tilde{D}_h)\tilde{U}^T P_1^h$$

If the exponential of the eigenvalues on the coarse and fine meshes were the same, and similarly for the eigenvectors and their prolonged version, then $\hat{K}_t^M \approx K_t^M$. Unfortunately, this is not the case, as the eigenvectors are unique only up to sign. Furthermore, if two eigenvalues are close, their matching eigenvectors might switch. However, Sun *et al* [2009] showed that, in the continuous case at least, the heat kernel is resilient to these changes. This leads us to the conjecture that even though the eigenvectors and their prolonged versions may not be exactly the same, this effect is cancelled out in the heat kernel, and the multi-resolution heat kernel is a good approximation of the true heat kernel.

Although we do not provide a formal proof, we show empirically that the MR heat kernel at time t provides a good approximation of the true heat kernel on the finest level, provided that it is computed on a resolution level compatible with t.

Given a mesh M, its multi-resolution structure can be generated by repeatedly simplifying it using any reasonable mesh simplification method, with varying target number of vertices, as required by Property 2 of Definition 1. We used the "quadric edge collapse" method [Garland and Heckbert 1997] as implemented in MeshLab [Cignoni *et al*. 2008]. Although we cannot reach the theoretical minimum as defined in Property 3 of Definition 1, this mesh simplification software does a good job at generating meshes which are very close to the input mesh.

To test our conjecture about the behavior of the spectral decomposition of the simplified mesh, with respect to the spectral decomposition of the original mesh, we simplified the gargoyle mesh, from an initial 60K vertices, to three resolution levels, having 30K, 15K and 7.5K vertices respectively. We then computed the full spectral decomposition of all four meshes, using the Graphite software [Graphite 2009], which is based on the algorithm of Vallet and Lévy [2008].

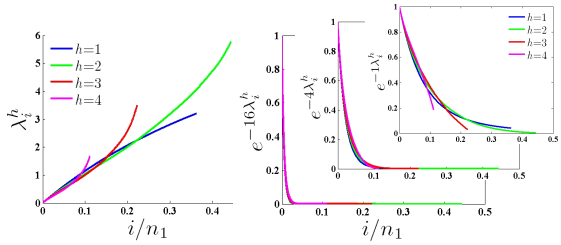


Figure 2: (*left*) *The spectrum of different resolutions levels of the gargoyle model, and* (*right*) *exponential for different values of t.*

Fig. 2 shows λ_i^h and $\exp(-t\lambda_i^h)$ as a function of i/n_1, for the four resolutions of the gargoyle mesh at a few values of t. It is evident from the figure that the exponentials of the spectrums are very similar, for the appropriate values of t.

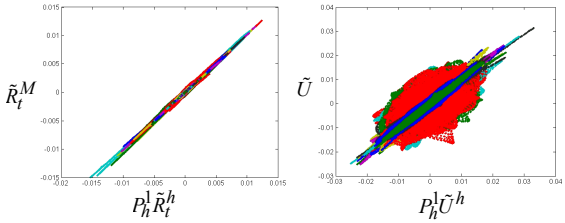


Figure 3: *(left) Diffusion map on the fine mesh vs. its prolonged version. Different colors represent diffusion maps of different vertices. (right) Eigenvectors on the fine mesh vs. their prolonged versions. Different colors represent different eigenvectors.*

As was mentioned before, the eigenvectors and their prolonged versions will not be the same, in general. However, significant discrepancies are visible only in higher eigenvectors. To see that, consider the expression $\tilde{R}_t^M = \tilde{U}\exp(-0.5t\tilde{D})$ - the first r axes of the *diffusion map* [Lafon 2004] of M, and respectively $\tilde{R}_t^h = \tilde{U}_h \exp(-0.5t\tilde{D}_h)$. R_t is the square root of the heat kernel matrix: $K_t = R_t R_t^T$. Fig. 3(b) shows the effect of t on our approximation. For visualization purposes, we plotted $\tilde{U}$ vs. $P_h^1\tilde{U}^h$, and $\tilde{R}_t^M$ vs. $P_h^1\tilde{R}_t^h$, where a good match will be indicated by a straight line, and errors in the prolongation will be "fat" areas off the line. As is evident from the figure, some of the eigenvectors are very different from their prolonged version, however the $\exp(-t)$ expression attenuates that, and makes $\tilde{R}_t^M$ very similar to $P_h^1\tilde{R}_t^h$, up to a sign per column. This shows, as expected, that for large values of t the heat kernel can be prolonged from a coarser level, whereas this cannot be done accurately for too small values of t. Fig. 4 shows this phenomenon, by comparing the true (exact) and prolonged heat kernel signatures (the diagonal of the heat kernel), for small and large t.

One issue remains: since we do not actually know the full spectrum of M, how, given t, can we know what the appropriate resolution level is? To solve this, we use a simple technique of extrapolating the beginning of the spectrum linearly, given just the first

50 eigenvectors corresponding to the finest resolution. As can be seen in Fig. 5, this is a reasonable approximation.

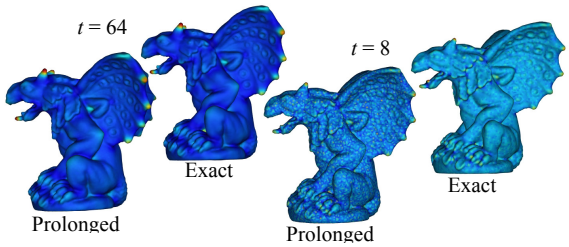


Figure 4: *The HKS and its prolonged version, for t=8, 64. Note how noisy the prolonged version is at t=8.*

Thus, given a range of timescales $[t_{min}, t_{max}]$ for which we would like to compute the heat kernel, we can partition it into segments: $\{[t_{min}=t_1, t_2), [t_2,t_3),\ldots,[t_k,t_{k+1} = t_{max}]\}$, such that the heat kernel of M for the i-th time segment $[t_i,t_{i+1})$ can be approximated by computing the heat kernel of M_i at times $[t_i,t_{i+1})$, and prolonging it to the finest level. As the heat kernel on M_i at times $[t_i,t_{i+1})$ describes details of M_i which do not exist at the next coarser level, it is also local on M_i and thus sparse. This phenomenon is shown in Fig. 1. On the coarsest level, we can compute the full heat kernel matrix, since by the construction of the multi-resolution structure, it is so small that sparsity is no longer a concern.

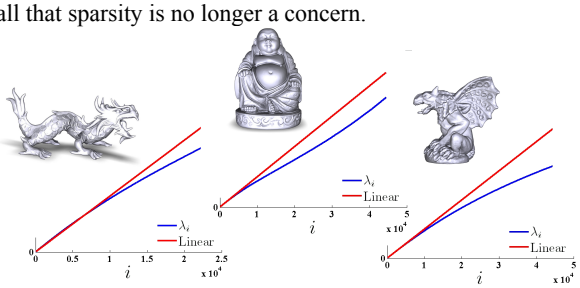


Figure 5: *An approximation of the spectrum of a few meshes, using a linear extrapolation of the first 50 eigenvalues*

Note that the heat kernel on the finest resolution level is still a large dense matrix, so we will usually not compute all of it, rather just a few of its columns, or its diagonal. This can be easily done, since both the prolongation matrix P_h^1 and the heat kernel on the lower resolution are sparse matrices. Furthermore, it is worth mentioning that other algorithms that compute the action of the heat kernel on a vector v [Hochbruck and Lubich 1997] cannot efficiently compute the diagonal of the heat kernel – the HKS, without resorting to the computation of the entire matrix, whereas in our case computing the HKS is a simple matter, using:

$$\left(HKS_t^M\right)_{ii} = \boldsymbol{p}_i K_t^h \boldsymbol{p}_i^T$$

where $\boldsymbol{p}_i$ is the i-th row of the prolongation matrix P_h^1.

To summarize, we have shown how the problem of computing select elements from the heat kernel matrix of M can be reduced to computing *sparse* heat kernels on *low-resolution* versions of M at small times t. In the next section, we confront the complementary problem: how to compute these sparse heat kernels accurately and efficiently.

3 Sparse Heat Kernels

Given a mesh M, whose Laplacian matrix is L, and a *small* time $t > 0$, our goal is to compute the heat kernel matrix K_t^M. Applying the standard computation of the heat kernel in terms of eigenvectors of L, obviously the smaller t is, the larger the computational effort, as more eigenvectors are required for an accurate result. This "top-down" approach is somewhat counter-intuitive, since the smaller t is, the more "local" k_t is, in the sense that the region of the mesh where $k_t(v,\cdot)$ is non-negligible, is limited to small regions around v. If we are only interested in values of the heat kernel which exceed some threshold ε, then we would expect it to be *easier* to compute k_t for smaller t, rather than harder.

Thus, we depart from the definition of the heat kernel through the Laplacian eigenvalues, and adopt the alternative definition using the matrix exponential. From now on, we will limit our discussion to Laplacians of the type: $L = A^{-1}W$, where A is a positive diagonal matrix, and W is a symmetric matrix having the structure of the mesh adjacency matrix. Many of the existing Laplacian discretizations fall in this category [Pinkall and Polthier 1993; Meyer *et al.* 2002; Reuter *et al.* 2006; Belkin *et al.* 2008; Xu 2004], and it guarantees that the Laplacian has a full set of eigenvectors with real eigenvalues. Given these definitions, the heat kernel matrix of M at time t is given by Sun *et. al* [2009]:

$$K_t^M = \exp(-tL)A^{-1}$$

Note, that if L is symmetric, then A is the identity matrix, and the heat kernel is identical to the usual exponential of the Laplacian.

It would appear at first glance that any of the (many) existing algorithms for computing the exponential of a matrix can be used in our case. However, as was explained in previous sections, this is not the case, as we are dealing with large sparse matrices, for which it is infeasible to compute the full heat kernel without additional assumptions. In the sequel, we adapt an existing algorithm to our specialized scenario of small t, where we know that many of the entries of K_t^M are close to 0.

3.1 Sparse Scale and Square

The fact that computing the heat kernel for small t should be easier than for large t, leads us to the following "bottom-up" approach. First, assume that t is small, and define the sparse HK:

Definition 3:
Let K_t^M be the heat kernel matrix of mesh M at time t. The *ε-sparsified heat kernel matrix* of M is:

$$\left(SK_{t,\varepsilon}^M\right)_{ij} = \begin{cases} \left(K_t^M\right)_{ij} & \left(K_t^M\right)_{ij} > \varepsilon \\ 0 & otherwise \end{cases}$$

Now, assume we can efficiently compute K_t^M for a very small $t=t_0$. Then we can compute the heat kernel matrix for $t=2t_0$, using: $K_{2t}^M = \left(SK_{t,\varepsilon}^M A\right)^2 A^{-1}$. Note that although, in general, sparse matrix multiplication might result in many non-zero entries, the number of significant entries in K_{2t}^M is determined by the influence regions at times $2t$, and thus for a small enough t, the number of non-zeros will remain small. This leads to Algorithm 1 for computing the sparse heat kernel of M at time $2^s t_0$, given the heat kernel at time $t_0 << 1$.

***Algorithm* 1**

Input: $K_{t_0}^M$, ε, s
Output: $SK_{t,\varepsilon}^M$, $t = 2^s t_0$
$K_t^M = K_{t_0}^M$; $t = t_0$
For $i = 1$ to s do:
  $K_{2t}^M = (SK_{t,\varepsilon}^M A)^2 A^{-1}$
  $t = 2t$
end

From a geometric point of view, clumping heat kernel values together also makes sense. Consider, for example, the sum of the heat kernels of two vertices u and v, which are far from each other with respect to the time t. See Fig. 6 for an example. If some third vertex w satisfies $k_t(u,w) + k_t(v,w) < \varepsilon$, then w is too far away from both v and u to influence the computation of $k_{2t}(u,v)$, and the sparsification will introduce only negligible errors. In fact, the clumping strategy is consistent with the locality property of the HK. In a sense, clumping achieves the same "localization" effect, without the need to actually re-compute the heat kernel for each vertex separately, on a different sub-mesh. Of course, from a purely algebraic point of view, our method is very similar to the classic "scale and square" approach, with the added sparsification twist.

Figure 6: *The geometric interpretation of sparse scale and square. The sum of $k_t(u,.)$ and $k_t(v,.)$ for times t=64, 128 and 512. Vertices whose sum is $< \varepsilon$ at time t do not influence the computation of $k_{2t}(u,v)$.*

Fig. 7 shows empirical evidence of the accuracy of our sparse "scale and square" approach. We computed exp(-tL) for t=4, once exactly, and once by our sparse scale and square approach, with t_0=0.25, and $\varepsilon=10^{-7}$. The figure shows the color coding of the diagonal of the exponential matrix, using the exact solution, computed using the full eigen-decomposition, vs. the sparse scale and square solution. We also show the color coding of the RMSE per row on a logarithmic scale. For all rows, the error was the order of 10^{-9}. It seems that error is slightly larger in concave regions, which is reasonable since their heat kernels tend to spread out further than in convex regions.

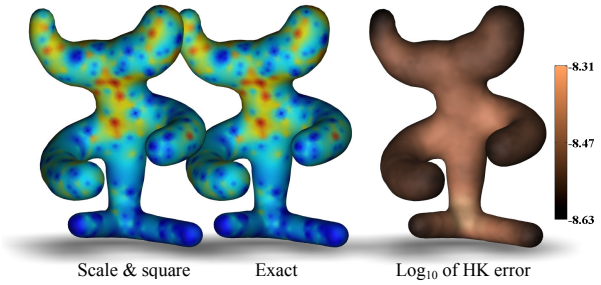


Figure 7: *Comparison of the diagonal of exp(-tL), computed using the scale and square approach (left), with the exact result (middle). Log_{10} of the error per row (right).*

To complete our solution, we proceed by describing a method for computing the heat kernel matrix for $t_0 << 1$.

3.2 Binomial Approximation

The most basic building block of our algorithm is the computation of K_t for $t << 1$, by computing exp(-tL). As discussed previously, using an approximation by a rational function is problematic, as it will require inverting a very large matrix. Hence, we opt for a polynomial approximation instead.

An obvious solution would be to use the power series of the matrix exponential, given by S^T. However, taking advantage of the assumption that t is small, we can generate an approximation which converges much faster than S^T, and is more stable.

***Definition 4*:**
Let $L = A^{-1}W$ be the Laplacian matrix of the mesh M, with A positive diagonal and W symmetric, and $t > 0$. The *binomial representation of the matrix exponential* is the series:

$$(11) \qquad S_N^B(t,L) = \sum_{m=0}^{N} Q_m(L)\left(\exp(-t)-1\right)^m,\ Q_m(L) = \binom{L}{m} = \frac{1}{m!}\prod_{k=0}^{m-1}(L-kI)$$

***Proposition* 1:**
Let S_N^B be defined as in (11), and $t > 0$. Then:

$$S^B(t,L) \triangleq \lim_{N\to\infty} S_N^B(t,L) = \exp(-tL)$$

To see why this is correct, consider a new variable $s=e^{-t}$, and the function $f(s) = s^x = e^{-tx}$. The Taylor expansion of f around $s = 1$ is given by:

$$s^x = 1^x + x1^{x-1}(s-1) + \frac{1}{2!}x(x-1)1^{x-2}(s-1)^2 + \ldots$$
$$= \sum_{m=0}^{\infty}\binom{x}{m}(s-1)^m \quad , \quad \binom{x}{m} = \frac{x(x-1)\ldots(x-(m-1))}{m!}$$

which is just the binomial series $(1+y)^x$ for $y = s - 1$. This series is well-defined for any complex x, and is guaranteed to converge if $|s-1| < 1$. In our case, since $t > 0$, this implies $0 < s < 1$, and convergence results.

This series can be easily extended to a diagonal matrix, by applying it independently to each element in the diagonal. Finally, it remains valid for any matrix of the type $L = A^{-1}W$, via the eigen-decomposition of L. The full derivation is given in Appendix A.

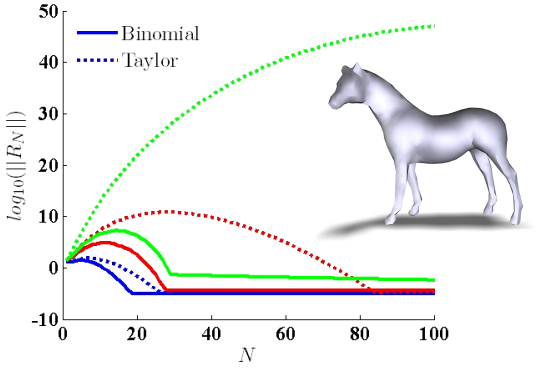


Figure 8: *Log of the norm of the residual $R_N(t,L) = \exp(-tL) - S_N(t,L)$ of the approximation of exp(-tL) with N elements, using the Binomial series S^B and the Taylor series S^T. $t \in \{0.25$ (blue), 1 (red), 4 (green)$\}$. $||L||_2 = 30.5$.*

To see the advantage of using the binomial series S^B instead of the standard one given by S^T, consider the following. In the scalar case, S^T is the Taylor expansion of exp(-tx) around $tx = 0$, and as such converges slowly as tx is farther from 0. Instead, we define a new variable s = exp(-t), and use the Taylor expansion of s^x around $s = 1$. Since t is close to 0, s is *very* close to 1, thus S^B converges faster than S^T. This is evident in Fig. 8, which shows the residual vs. the number of terms in the series S^B, and the standard Taylor approximation S^T, for various t, for the approximation of exp(-tL) on the horse mesh. Since the mesh is small, it was possible to compute the "ground truth" needed for computing the residual using Matlab's *expm* function.

As with any other matrix power sum, our method also faces numerical problems when $\alpha = \|tL\|_2$ is large, albeit is less sensitive than the regular Taylor sum, due to the fast decay of $s = \exp(-t)$. For large α, the m-th element in the series, B_m, will be so large that round-off errors are the same magnitude as the final result, rendering it unusable. To avoid this, and to limit the highest power of the Laplacian which needs to be computed, we fix the number of iterations to N, and choose t_1 such that $\|B_N\|_2 \le \varepsilon$, using the following bound:

***Proposition* 2:**
Let $B_m(t,L)$ be the m-th element of the binomial series S^B. Then:

$$\left\|B_m(t,L)\right\|_2 \le \left|\binom{\lambda_{\max}}{m}\left(e^{-t}-1\right)^m\right|\sqrt{\frac{\max(a_i)}{\min(a_i)}}$$

where λ_{max} is the maximal eigenvalue of L, and $a_i = A_{ii}$. The proof is given in Appendix B.

As the actual t we wish to compute K_t^M for is larger than t_1, we apply our sparse scale and square approach on top of the series expansion, as follows. Given t for which we want to compute K_t, we find an integer s such that $t/2^s = t_0 < t_1$, compute KS_{t_0} using S^B, and scale back the result using Algorithm 1. This procedure is summarized in Algorithm 2.

***Algorithm* 2**

Input: t, ε , N
Output: $K_{t,\varepsilon}^M$, s
Find t_1 such that $\|B_N\|_2 \le \varepsilon$
Find s such that $t/2^s = t_0 \le t_1$
Set $K_{t_0} = S_N^B(t_0, L)$
Apply Algorithm 1.

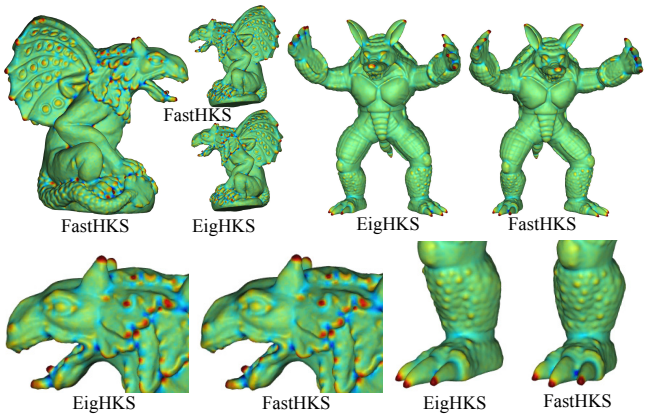


Figure 9: *Comparison of FastHKS and EigHKS for the Gargoyle and Armadillo, for t=6, using a single resolution level.*

Fig. 9 shows an example of the result of our computation of the HKS at small times. To make the times meaningful, we have scaled the mesh such that the total area is equal to the number of vertices, thus t=1 results in an average influence region of about one 1-ring. We used $\varepsilon = 10^{-6}$, and 15 iterations for the series approximation. The figure shows our HKS – denoted *FastHKS* - compared to the true HKS, computed from the spectral decomposition of the mesh, using the algorithm of Vallet and Lévy [2008] – denoted *EigHKS* - on the "gargoyle" and "armadillo" meshes, for t = 6. Both HKS are color-coded over the mesh, and it is evident that they are visually indistinguishable. Table 1 provides the quantitative comparison for these and other meshes. Note that about 12K eigenvectors were needed in order to compute *EigHKS* for t = 6 on the gargoyle mesh, which has 122K vertices. Our computation, on the other hand, took only a few minutes.

If the mesh was small enough, we could get away with a single resolution level – computing for small times using the *sparse heat kernel* approximation, and large times using a few eigenvectors. However, even for medium sized meshes this approach will break down, as "medium" times are introduced - too large to be sparse and efficiently computed using the sparse method, yet too small for the extraction of enough eigenvectors to be feasible. This is where our multi-resolution approach kicks in, by allowing us to treat medium times as if they were small and sparse. The following section summarizes our complete algorithm, and shows some results of our computation of select entries of the heat kernel matrix of detailed models.

4 Results and Applications

First, let us explain how the full algorithm comes together, and provide additional implementation details.

Let M=(V,F) be a mesh, and L its Laplacian matrix, such that L=$A^{-1}W$. We wish to compute select entries of K_t^M, for a given set of timescales $\{t_i\}$. The actual time scales depend on the application in question, however, to have comparable times between models, we normalized all models, such that their surface area equals to their vertex count. This does not mean that the same time t will have the same meaning for different meshes, but rather that a time will indicate the size of the influence area to some degree. For example, in this setup, t = 1 matches roughly the average influence area of a 1-ring.

Our algorithm may be summarized as follows:

1. Compute the multi-resolution structure $MR_{d,C}(M)$
2. Choose the coarsest resolution level h, such that $cn_h > r_\varepsilon(t)$
3. Compute the sparse heat kernel on the resolution level h using Algorithm 2.
4. Project the sparse heat kernel to the finest resolution level using (7).

Several clarifications are in order. First, let us examine the influence of the constants d and C on the algorithm. Together, these parameters determine m – the number of meshes in the multi-resolution structure. The smaller m is, the more resources we need to allocate for computing the heat kernel, with the extreme being m=1, in which case there is only a single mesh in the structure. We used d=2 for all our experiments, with C varying from 6K to 20K vertices. As is seen in Table 1, larger values of C (such as were used for the Armadillo and Gargoyle models) result in a larger computation time for large times, as the heat kernel will not be sparse enough on the coarsest resolution level. The actual computation of the multi-resolution structure was done using the MeshLab software [Cignoni *et al*. 2008], decimating the original mesh using the "Quadric edge-collapse decimation" method [Garland and Heckbert 1997] by prescribing the number of target vertices. Next, the appropriate resolution level for a given time t was chosen, so that $cn_h > r_\varepsilon(t)$, for c = 0.2. Using a larger value for c would yield errors after the prolongation, as higher eigenvalues on the coarse level no longer match their counterpart on the next

resolution level. Using a smaller value for c will increase the computational burden, as we will compute K_t^M on a level on which it is not sparse. We have found $c = 0.2$ to be a good compromise, and used it in all our examples. For ε, we use $\varepsilon = 10^{-4}$ for the numerical rank computations, and $\varepsilon = 10^{-6}$ for the sparsification process.

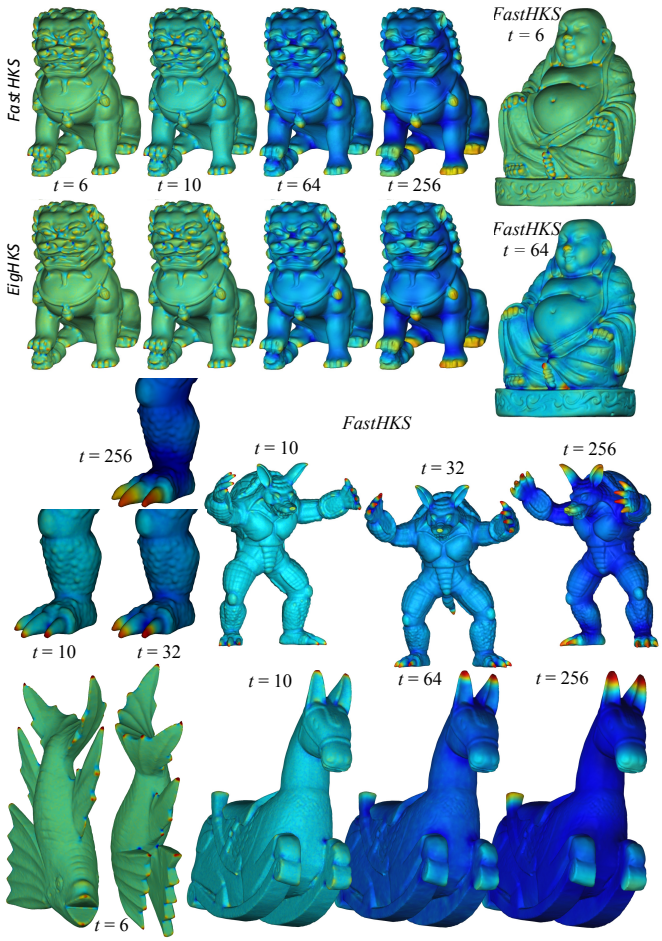


Figure 10: *Color coding of FastHKS on different meshes. Comparison with EigHKS for one mesh. The timings and errors are given in Table 1.*

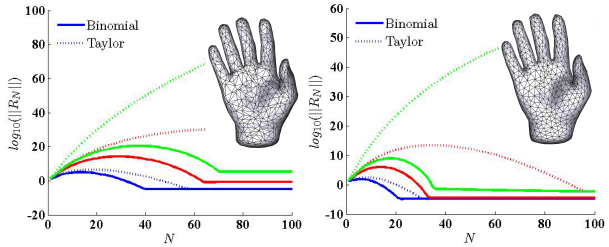
Our algorithm was implemented in MATLAB, using C code for the sparsification process. If the amount of available memory is limited, Algorithm 1 can be performed out-of-core, such that only part of the matrix is resident in memory. We have used this approach, and partitioned the matrix into 5×5 blocks. We compared our results to those obtained with the EigHKS algorithm, for a variety of meshes and time scales, summarized in Table 1, and visualized in Fig. 10. The eigen-decomposition of EigHKS was computed out-of-core, as the matrices involved are too large to keep in memory. Thus, for EigHKS, we report two time measurements – that of the eigen-decomposition, and that of computing the HKS given that. In addition, for each mesh and time scale, we report how many eigenvalues are required for an "exact" computation. This number is computed by finding the first index i, for which $\exp(-t\lambda_i) < \varepsilon$, where $\varepsilon = 10^{-3}$. The additional computational cost of computing the first 50 eigenvalues was less than 40 seconds for all models, and constructing the multi-resolution scheme took less than 35 seconds for all models.

In almost all cases our FastHKS is must faster than EigHKS. However, as t grows larger the advantage over EigHKS is less obvious, and for extremely large t (1024 for the Buddha for example, when only about 60 eigenvectors are needed), EigHKS would be faster. However, such a scale is very large, with the influence regions of vertices encompassing close to half the mesh.

Model	t	$\|V\|$ (K)	$\|V\|$ coarse (K)	Scale Square (sec)	S^B (sec)	Total FastHKS Time (sec)	*RMSE* (x10^{-6})	#eigens required	Eigens (sec)	HKS given eigens (sec)	Total EigHKS Time (sec)
Buddha	4	112	112	87	23	**110**	19	17,971	31,528	252	**31,780**
	32		14.5	21	6	**44**	97	1,946	3,421	31	**3,452**
	256		6.2	44	<1	**91**	140	242	426	6	**432**
	512		3.5	20	1.3	**72**	98	109	202	3	**205**
	1024		3.5	31	1.4	**113**	1000	61	95	1	**96**
Chinese lion	3	153	153	205	51	**256**	5.7	34,477	62,486	406	**62,892**
	16		42	92	8	**120**	93	5,415	10,502	78	**10,580**
	128		10	31	5	**76**	98	661	1,470	10	**1,480**
Arma	4	112	112	95	19	**114**	21	17,901	32,405	287	**32,692**
	24		30	131	19	**179**	61	2,593	4,986	46	**5,032**
	200		18	204	7	**576**	75	314	603	7	**610**
Garg	3	122	122	129	32	**161**	11	28,144	51,172	319	**51,491**
	45		16	36	7	**72**	110	1,515	2,754	29	**2,783**
	100		16	81	7	**139**	71	682	1,262	15	**1,277**
Fish	4	100	100	85	14	**110**	16	16,068	29,723	235	**29,958**
	64		5	14	<1	**32**	150	861	1594	13	**1,607**
	512		5	75	<1	**115**	114	110	219	2	**221**
Horse	6	100	100	270	18	**288**	10	10,122	9,824	196	**10,020**
	45		12	34	2	**51**	78	1,239	1,176	24	**1,200**
	200		6	32	<1	**71**	75	278	286	6	**292**

Table 1: *Comparison between FastHKS and EigHKS. Columns: model, timescale, number of vertices (in thousands), number of vertices in coarsest resolution, time for scale and square, time for computing S^B, total FastHKS time, RMS error from EigHKS, # eigens required, time for extracting HKS given the out-of-core (OOC) eigen-decomposition, time for OOC eigen-decomposition. Experiments were performed on a Windows Server 2003 64 bit single core machine, with 2.33GHz CPU, and 4GB memory.*

Our method has some limitations. First, we do not provide theoretical guarantees on the quality of our approximation, yet we show empirically that the results are quite accurate. For example, our algorithm is heavily based on the fact that the heat kernel can be accurately prolonged from a coarse mesh to a higher resolution level. However, there is no existing theoretical result bounding the difference between the two, in terms of the meshes' Hausdorff distance. Although in practice the prolongation seems to work quite well, a theoretical bound would have been very satisfying.



(*a*) (*b*)

Figure 11: *Improving the meshing quality improves the convergence of the Binomial series (and the Taylor series).*

Furthermore, as with other Laplacian-based methods, the quality of our result might deteriorate if the surface is very badly meshed. For example, the bound in Proposition 2 depends on the maximal eigenvalue of the Laplacian operator, which in turn depends on the meshing quality. Thus, if we repeat the experiment of Fig. 8 on a badly meshed model, more iterations would be required for convergence (although still less than required by the Taylor series). Fig. 11(a) demonstrates this on a badly meshed model, whose Laplacian norm is $||L||_2 = 74.5$. The number of iterations

until convergence (both for the Binomial series and the Taylor series) has doubled relative to Fig. 8. However, when the mesh is smoothed, reducing the norm to $||L||_2 = 35.3$, the number of iterations until convergence decreases, as evident in Fig. 11(b).

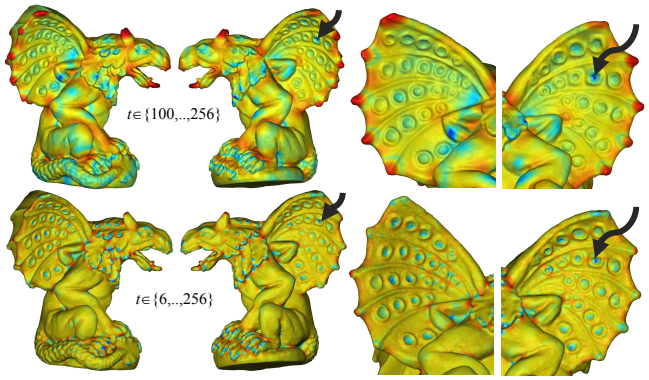


Figure 12: *The HKS-based distance function from the marked vertex, using different ranges of t. The other round features are close in HKS distance to the marked vertex only when using small enough values of t.*

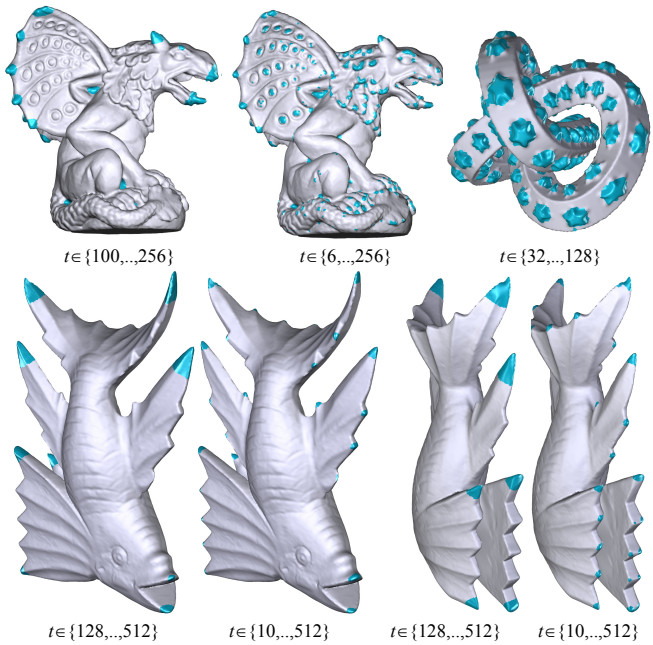


Figure 13: *Separating features from background by clustering the HKS with k-means*

To demonstrate the benefit gained by using our methods for applications based on computing the heat kernel, we experimented with a diffusion-based feature extraction algorithm, following Sun *et al* [2009]. Let $T = \{t_1, \ldots, t_k\}$ be the set of interesting time scales, and let HKS_T be the corresponding set of heat kernel signatures. Two points u and v on the surface are considered similar if $d_{HKS}(u,v) = ||HKS_T(u)-HKS_T(v)||_2$ is small. Given a point on the gargoyle, we show the *HKS* distance from it to all other points, for two sets of times T_1 and T_2. T_1 is sampled logarithmically in the range [100,…,256], whereas T_2 also includes smaller times, and is sampled from [6,…,256]. K_{T_1} can be computed using the eigen-decomposition, but K_{T_2} cannot. Fig. 12 shows the marked point, and the HKS distance from it using T_1 and T_2. As evident in the figure, the small round features which are similar to the surrounding region of the marked point are missed when using only large timescales, whereas they are clearly visible as "blue" – meaning close – points, when using the full time scale.

To further emphasize this, we used simple *k*-means clustering to cluster the *HKS* values of the vertices, again for time scales T_1 and T_2. We used $k=2$, as we want to classify the points as "features" vs. "background". Fig. 13 shows the clustering result for the Gargoyle, knot and fish models – blue vertices were classified as one group, and gray vertices as another. On the Gargoyle and the fish models, when using T_2, even such a straightforward clustering approach could detect fine features, whereas they were completely missed using T_1. The star shapes on the knot model exhibited similar behavior.

6 Conclusions and Discussion

Diffusion maps, and specifically the heat kernel and diffusion distances, have many applications in geometry processing, yet are notoriously difficult to compute. We have proposed a method, taking advantage of the multi-scale property of the heat kernel, for its efficient computation. In addition, we showed how for very small t, a power series based on the Binomial expansion is more appropriate for approximating the matrix exponential than the standard Taylor series. Combining these properties led to a fast algorithm for computing the heat kernel of large meshes, for all values of t.

We showed how to apply our algorithm to improve a diffusion-based feature extraction method, but we believe its applicability lies far beyond that. For example, diffusion distances are an important tool for the analysis of graphs (e.g. in communication and social networks), and it is not unlikely that if a reasonable scheme of "simplifying" a graph is used, the diffusion distances on a very large graph could be computed in a similar multi-resolution manner. Furthermore, perhaps similar ideas could be used for computing the diffusion map itself (and not only the heat kernel) in an efficient manner.

Acknowledgments

Thanks to Irad Yavneh for helpful numerical discussions. This work was partially supported by NSF grants 0808515 and 0914833, and by a joint Stanford-KAUST collaborative grant.

References

AKSOYLU, B., KHODAKOVSKY, A., AND SCHRÖDER, P. 2005. Multilevel solvers for unstructured surface meshes. *SIAM Journal of Scientific Computing* 26, 4.

ARIOLI, M., CODENOTTI, B. AND FASSINO, C. 1996. The Padé method for computing the matrix exponential. *Linear Algebra and its Applications* 240.

BELKIN, M., SUN, J., AND WANG, Y. 2008. Discrete Laplace operator on meshed surfaces . In *Proceedings of SOCG 2008.*

CIGNONI, P., CORSINI, M., AND RANZUGLIA, G. 2008. Meshlab: An open-source 3D mesh processing system. *ERCIM News* 73.

COIFMAN, R. AND MAGGIONI, M. 2006. Diffusion wavelets. *Applied and Computational Harmonic Analysis,* 21, 1.

DEGOES, F., GOLDENSTEIN, S., AND VELHO, L. 2008. A hierarchical segmentation of articulated bodies. *Computer Graphics Forum* 27, 5.

GARLAND, M. AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proc. SIGGRAPH 1997.*

GRAPHITE. 2009. In http://alice.loria.fr/software/graphite.

HOCHBRUCK, M., AND LUBICH, C. 1997. On Krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 34, 5.

HOPPE, H. 1996. Progressive meshes. In *Proc. SIGGRAPH 1996.*

SUN, J., OVSJANIKOV, M., AND GUIBAS, L. 2009. A concise and provably informative multi-scale signature based on heat diffusion. *Computer Graphics Forum* 28, 5.

KOBBELT L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P.1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. SIGGRAPH 1998.*

LAFON, S. 2004. Diffusion maps and geometric harmonics. *PhD Thesis, Yale University,* 2004.

MÉMOLI, F. 2009. Spectral Gromov-Wasserstein distances for shape matching. In *Proc. of Workshop on Non-Rigid Shape Analysis and Deformable Image Alignment*.

MEYER, M., DESBRUN, M., SCHRÖDER, P. AND BARR, A. H. 2002. Discrete differential geometry operators for triangulated 2-manifolds. In *Proc. VisMath'02*.

MOLER, C. AND LOAN, C. V. 2003. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review,* 45, 1.

PINKALL U. AND POLTHIER K. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics* 2, 1.

REUTER, M., WOLTER, F.-E., AND PEINECKE, N. 2006. Laplace-Beltrami spectra as 'Shape-DNA' of surfaces and solids. *Computer-Aided Design* 38, 4.

SHEFFER, A, LÉVY, B., MOGILNITSKY, M. AND BOGOMJAKOV, A. 2005. ABF++: fast and robust angle based flattening. *ACM Trans. Graph.* 24, 2.

SHI, L., YU, Y., BELL, N., AND FENG, W.-W. 2006. A fast multigrid algorithm for mesh deformation. In *Proc. SIGGRAPH 2006.*

VALLET, B. AND LÉVY, B. 2008. Spectral geometry processing with manifold harmonics. *Computer Graphics Forum* 27, 2.

WESSELING P. 2004. *An Introduction to Multigrid Methods*. R. T. Edwards, Inc.

XU G. 2004. Discrete Laplace-Beltrami operators and their convergence. *Computer-Aided Geometric Design* 21, 8.

Appendix A

***Proposition* 1:**
Let $L = A^{-1}W$ be the Laplacian matrix of the mesh M, with A positive diagonal and W symmetric, and $t > 0$. Define

$$\text{(A1)} \qquad S_N^B(t,L) \triangleq \sum_{m=0}^{N} Q_m(L)\left(e^{-t}-1\right)^m,\ Q_m(L) = \binom{L}{m} = \frac{1}{m!}\prod_{k=0}^{m-1}(L-kI)$$

then:

$$S^B(t,L) \triangleq \lim_{N\to\infty} S_N^B(t,L) = e^{-tL}$$

Proof:
It is well known, that if L fulfills the conditions of the proposition, then it has a full set of eigenvectors with real eigenvalues, which are the solution to the generalized eigenvalue problem $Wv=A\lambda v$. Let V be the matrix whose columns are the right eigenvectors of L, and D the diagonal matrix of eigenvalues. Then, $L=VDV^{-1}$. Furthermore, it is easy to check that $L - kI = V(D - kI)V^{-1}$, hence:

$$\text{(A2)} \qquad \frac{1}{m!}\prod_{k=0}^{m-1}(L-kI) = V\left(\frac{1}{m!}\prod_{k=0}^{m-1}(D-kI)\right)V^{-1} = VE_mV^{-1}$$

where E_m is a diagonal matrix. Plugging (A2) into (A1), we have:

$$\text{(A3)} \quad S_N^B(t,L) \triangleq \sum_{m=0}^{N}\left(e^{-t}-1\right)^m VE_mV^{-1} = V\left(\sum_{m=0}^{N}\left(e^{-t}-1\right)^m E_m\right)V^{-1} = VF_NV^{-1}$$

where, F_N is a diagonal matrix. Considering a single element in F_N:

$$(F_N)_{ii} = \sum_{m=0}^{N}\frac{1}{m!}\prod_{k=0}^{m-1}(\lambda_i - k)\left(e^{-t}-1\right)^m = \sum_{m=0}^{N}\binom{\lambda_i}{m}\left(e^{-t}-1\right)^m$$

where λ_i is the i-th eigenvalue of L. The right hand expression is the partial sum of the Binomial series for $(1+(e^{-t}-1))^{\lambda_i} = e^{-t\lambda_i}$ which is known to converge for any λ_i, when $|e^{-t} - 1| < 1$. Since $t > 0$, we have $0 < e^{-t} < 1$, hence the convergence condition holds, and we get:

$$\lim_{N\to\infty}(F_N)_{ii} = \lim_{N\to\infty}\sum_{m=0}^{N}\binom{\lambda_i}{m}\left(e^{-t}-1\right)^m = e^{-t\lambda_i}$$

Hence, the entries on the diagonal of F_N converge absolutely, and thus F_N itself converges: $\lim_{N\to\infty} F_N = e^{-tD}$. Plugging this into (A3), we conclude:

$$\lim_{N\to\infty} S_N^B(t,L) = \lim_{N\to\infty} VF_NV^{-1} = Ve^{-tD}V^{-1} = e^{-tL}$$

which proves Proposition 1.

Appendix B

***Proposition* 2:**
Let $B_m(t,L)$ be the m-th element of the binomial series S^B. Then:

$$\|B_m(t,L)\|_2 \le \left|\binom{\lambda_{\max}}{m}\left(e^{-t}-1\right)^m\right|\sqrt{\frac{\max(a_i)}{\min(a_i)}}$$

where λ_{max} is the largest singular value of L, and $a_i = A_{ii}$.

Proof:
Let $L_1 = A^{1/2}LA^{-1/2}$. By definition we have that:

$$\text{(B1)} \qquad B_m(t,L) = \frac{1}{m!}\prod_{k=0}^{m-1}(L-kI)\left(e^{-t}-1\right)^m$$

Plugging L_1 into (B1) instead of L we get:

$$\frac{1}{m!}\prod_{k=0}^{m-1}(L_1-kI) = \frac{1}{m!}\prod_{k=0}^{m-1}\left(A^{1/2}(L-kI)A^{-1/2}\right) = A^{1/2}\left(\frac{1}{m!}\prod_{k=0}^{m-1}(L-kI)\right)A^{-1/2}$$

Thus, we have:

$$\text{(B2)} \qquad B_m(t,L_1) = A^{1/2}B_m(t,L)A^{-1/2}$$

Now we proceed to bound $B_m(t,L_1)$. Since L_1 is symmetric, we have:

$$\text{(B3)} \quad \left\|\frac{1}{m!}\prod_{k=0}^{m-1}(L_1-kI)\right\|_2 \le \frac{1}{m!}\prod_{k=0}^{m-1}\|(L_1-kI)\|_2 = \frac{1}{m!}\prod_{k=0}^{m-1}(\lambda_{\max}-k) = \binom{\lambda_{\max}}{m}$$

where $\lambda_{\max}$ is the largest eigenvalue of L_1 (and also of L), and we assumed that $\lambda_{\max} > m$. Going back to $B_m(t,L)$ using (B2):

$$\text{(B4)} \qquad \|B_m(t,L)\| = \left\|A^{-1/2}B_m(t,L_1)A^{1/2}\right\| \le \left\|A^{-1/2}\right\|\left\|B_m(t,L_1)\right\|\left\|A^{1/2}\right\|$$

where all the norms are L_2. Finally, since A is a diagonal matrix, $\|A^{1/2}\|$, and $\|A^{-1/2}\|$ are given in terms of the diagonal values a_i of A, and combining (B3) and (B4), we finally obtain:

$$\|B_m(t,L)\|_2 \le \left|\binom{\lambda_{\max}}{m}\left(e^{-t}-1\right)^m\right|\sqrt{\frac{\max(a_i)}{\min(a_i)}}$$

which proves the proposition.