General Techniques for Interpolation, Reconstruction, and Morphing of Polyhedral Surfaces

Amir Vaxman
General Techniques for Interpolation, Reconstruction, and Morphing of Polyhedral Surfaces

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Amir Vaxman

Submitted to the Senate of the Technion — Israel Institute of Technology
Adar 5771 Haifa February 2011
The research thesis was done under the supervision of Prof. Gill Barequet in the Computer Science Department.

The work described in this dissertation is based on the following publications:


The generous financial support of the Technion, and of the Jacobs-Qualcomm scholarship, are gratefully acknowledged.
Contents

Abstract 1

1 Overview 3
1.1 Background 3.3
1.1.1 Reconstruction from cross-sections 3
1.2 Discrete Shape Correspondence 6
1.3 Preliminaries 7
1.3.1 Two-Dimensional Straight skeletons 7
1.3.2 Discrete Multiscale Heat Kernels 9
1.4 Structure of the Dissertation 11

2 The Three-Dimensional Straight Skeleton 13
2.1 Definition 13
2.2 Ambiguity of the Definition 15
2.3 Properties of the Straight Skeleton 15
2.4 Constructing the Straight Skeleton 19
2.4.1 Data Structures 21
2.4.2 The Algorithm 22
2.4.3 Collecting Events 22
2.4.4 Memoryless Propagation 26
2.5 Complexity 26
2.5.1 A Combinatorial Lower Bound for General Polyhedra 26
2.5.2 Time Complexity 31
2.6 Degeneracies 31
2.7 Experimental Results 31
3 Reconstruction of multi-label domains from partial planar cross-sections 35
  3.1 Overview of our Method ........................................... 35
  3.2 An Arrangement of Partial Cross-Sections .......................... 38
  3.3 Using Straight Skeletons of Nonconvex Cells ....................... 39
  3.4 Projecting Images onto Skeletal Faces ............................ 40
    3.4.1 Projection from Degenerate Edges ............................ 43
  3.5 Mesh Reconstruction .............................................. 44
    3.5.1 Image Overlay .............................................. 44
    3.5.2 Mesh Creation .............................................. 44
    3.5.3 Correctness .............................................. 45
  3.6 Optional Improvements ........................................... 46
    3.6.1 Mesh Refinement ........................................... 46
    3.6.2 User-Guided Reconstruction ................................ 46
  3.7 Experimental Results ........................................... 46
  3.8 Conclusion ..................................................... 52

4 A Multi-Resolution Approach to Heat Kernels on Discrete Surfaces 53
  4.1 Previous Work .................................................. 54
    4.1.1 Multi-Resolution ........................................... 54
    4.1.2 Matrix Exponential Approximations ......................... 54
  4.2 Overview of the Algorithm .................................... 56
  4.3 Multi-Resolution Heat Kernels ................................ 57
  4.4 Sparse Heat Kernels ............................................ 63
    4.4.1 Sparse Scale and Square ................................ 64
    4.4.2 Binomial Approximation ................................. 66
  4.5 Results and Applications .................................... 72
  4.6 Conclusions and Discussion .................................. 77

5 Conclusions and Future Work 81
  5.1 Summary ....................................................... 81
  5.2 The Straight Skeleton .......................................... 81
  5.3 Multi-labeled Reconstruction .................................. 82
  5.4 Heat Kernels ................................................... 82
## List of Figures

1.1 The reconstruction of a hip from slices. .......................... 4

1.2 An example of a polygon ABCDE and its straight skeleton (The propagation is in dashed lines). The edges EA and CD vanish in edge events, and B splits the edge DE .......................... 8

1.3 Heat diffusion (heat kernel values) from a point on the hand of Buddha (left), and the heat kernel signature/diagonal (right). ..... 9

1.4 (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. .......................... 11

2.1 An example of the straight skeleton of a polyheron. The edges of the polyhedron are in solid lines, the skeletal edges are in dashed line, the skeletal vertices are marked by crosses, and examples of a skeletal face and 3-cell are featured in grey. ........................................... 14

2.2 A vertex of degree 5, and two possible topologies during the propagation, depicting the ambiguity of the straight skeleton. .......................... 16

2.3 The five types of events. The solid lines are the original edges, and the dashed lines are their locations after the propagation. The dotted arrows show the progression of these edges, up to the time of the event. .... 18

2.4 Changing the initial topology of a vertex of degree greater than 3. The skeleton is shown in dashed lines. .......................... 20

2.5 Treating a saddle vertex. The vertex 0 is the saddle, and vertices 1-7, marked as “up” or “down” vertices are shown in (a). In (b), the section of the down-edges is shown, and the portal vertices are marked by circles. These portals are connected to the respective up vertices (since there are three up streaks, each containing only one vertex). .......................... 21
2.6 The situation before and after edge events: in (a,b), edge \( e \) vanishes and edge \( e' \) is created, and in (c,d) face 4 vanishes. 24

2.7 Topology changes in hole, edge and, edge-split events. The dashed lines denote the topology before the event (in (d), dotted lines show the second split event done by the same edge. The endpoints of the propagating split edge here are marked by squares. The location of the immediate vertex event is marked by a circle), and the solid lines denote the topology after the change. 25

2.8 A worst-case-complexity skeleton in the convex case. 27

2.9 Illustrating the complexity of a three-dimensional straight skeleton. 28

2.10 Sample objects. 32

3.1 An overview of the algorithm in 2D. The arrangement of partial segments is constructed. The straight skeleton of a single cell is computed (dashed). Notice that singular edges evolve as degenerate rectangles. A skeletal cell of type ‘a’ has obtuse walls, and therefore, the purple image stretches onto these walls (see Section 3.4). A cell of type ‘b’ does not have walls, and so the image of the base edge simply projects onto it. A cell of type ‘c’ is created from a degenerate vertex (which matches degenerate edges in 3D), and the image adjacent to this vertex projects onto the roof of the cell. 36

3.2 Reconstruction from partial multi-colored slices. Two slices (with green and blue contours) are full, while the slice with the red contour lies only on a halfplane. The straight skeleton of the arrangement of partial planes is computed, the contours are projected, and, finally, the meshes are created accordingly. 37

3.3 The propagation of a planar patch (solid) with an image adjacent to the degenerate edges. The patch propagates into a triangular prism, and the image edges, which are adjacent to the degenerate edges, propagate into images faces. A similar example is depicted in the 2D setting, with a degenerate segment that propagates into a rectangle, and the images adjacent at the vertices propagating into image edges. 38
3.4 A typical cell with obtuse walls is shown in the left image. The base (dark gray) has one triangular image (red), which is projected onto the roof (solid black). All the walls form obtuse angles with the base. (One neighboring cell, which forms an obtuse angle with the base, is shown in light gray.) In the right image a 2D depiction of a skeletal cell is shown. The base is solid, the walls are dashed, and the roof is dashed-dotted.  

3.5 A 2D drawing of wall stretching. A simple orthogonal projection creates gaps in the surface although the images are continuous. Stretching the images on the obtuse walls fixes this effect.

3.6 Exemplifying the stretching of images onto walls.

3.7 The propagation of an image from a floating patch inside a box. Notice how the red material propagates from the degenerate edges, which propagate into faces, and stretches to envelope the entire patch.

3.8 Inserting a patch to aid reconstruction. The two blue stripes are expected to connect, but they do not project onto the same overlay. An aiding blue patch is put in the middle to ensure the continuity of the blue domain.

3.9 Reconstruction of a “3-knuckle.”

3.10 Reconstruction with partial sections.

3.11 Reconstruction of a cat from nonparallel sparse slices with noisy regions. Gray regions indicate the missing information.

3.12 Reconstruction of a hand.

3.13 A multi-labeled reconstruction with noisy regions.

3.14 A simultaneous reconstruction of several internal organs.

4.1 (left) The spectrum of different resolutions levels of the gargoyle model, and (right) exponential for different values of $t$.

4.2 (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.

4.3 The HKS and its prolonged version, for $t = 8, 64$. Note how noisy the prolonged version is at $t = 8$.

4.4 An approximation of the spectrum of a few meshes, using a linear extrapolation of the first 50 eigenvalues.
4.5 The geometric interpretation of sparse scale and square. The sum of $k_t(u,\cdot)$ and $k_t(v,\cdot)$ for times $t = 64, 128$ and $512$. Vertices whose sum is $\varepsilon$ at time $t$ do not influence the computation of $k_{2t}(u,v)$.

4.6 Comparison of the diagonal of $e^{-tL}$, computed using the scale and square approach (left), with the exact result (middle). Log$_{10}$ of the error per row (right).

4.7 Log of the norm of the residual $R_N(t, L) = e^{-tL} - S_N(t, L)$ of the approximation of $e^{-tL}$ with $N$ elements, using the Binomial series $S^B$ and the Taylor series $S^T$. $t \in \{0.25\text{(blue)}, 1\text{(red)}, 4\text{(green)}\}$.

4.8 Comparison of FastHKS and EigHKS for the Gargoyle and Armadillo, for $t = 6$, using a single resolution level.

4.9 Color coding of FastHKS on different meshes. Comparison with EigHKS for one mesh. The timings and errors are given in Table 4.1.

4.10 Improving the meshing quality improves the convergence of the Binomial series (and the Taylor series).

4.11 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 $t$.

4.12 Separating features from background by clustering the HKS with $k$-means.
Abstract

The topics of shape reconstruction from samples, and of morphing between objects are two of the major research branches of geometric processing. The two major problems in reconstruction which are commonly researched are reconstruction of two and three-dimensional objects from point sets, and reconstruction from cross-sections. The major issue of morphing between objects is usually the determination of correspondence between shapes, and of intrinsic symmetry.

In this work we develop three techniques to assist in the solutions of these major problems. We first develop a unique three-dimensional structure to find correspondences between arbitrarily-aligned planes, and on top of that we develop a geometric algorithm for reconstruction between partial slices. We continue by developing an algorithm to efficiently compute the heat kernel, which has been recently given much attention, as a quality descriptor in the study of the intrinsic properties of shapes.
Chapter 1

Overview

1.1 Background

1.1.1 Reconstruction from cross-sections

Reconstructing a three-dimensional object from a collection of its cross-sections has been a widely-investigated problem in the literature in the past thirty years. The problem is usually defined in the following manner: Given a set of contours $C$, embedded in a set of planes $P$, reconstruct object $O$ s.t. $O \cap P = C$. See Figure 1.1 for an example.

This reconstruction problem arises primarily in the fields of medical imaging, digitization of objects, and geographical information systems. Data obtained by medical imaging apparata, range sensors, or as elevation contours are interpolated in order to represent, reconstruct, and visualize human organs, CAD objects, or topographic terrains. It is assumed that a preprocessing step has already extracted from the raw data (usually a sequence of pixel images) the closed two-dimensional contours that delimit the material regions in each section. The goal is to compute a (usually triangulated) surface that tiles between these contours and forms a solid volume whose cross-sections at the given heights match the input slices.

Most work on the subject deals with the interpolation between parallel sections, which comprise contours defining the geometry and topology of the intersections of the object with a series of (usually equally-spaced) parallel planes. In the majority of papers these contours are all of the same “color,” that is, they describe the same type of material.

Various algorithms for two-dimensional based triangulated surface reconstruc-
tion have been suggested in the literature (See, e.g., [13, 36, 37, 40, 49, 51, 53, 72, 78, 81].) Many early algorithms fail in complex instances (such as multiple branching), leave gaps between the contours, and/or generate unacceptable solutions (e.g., self-intersecting surfaces.) Some algorithms [27, 29, 34, 62, 70, 84] reduce the more involved cases to the simple case where each slice contains only one contour. There have been several attempts [7, 9, 10, 12, 15, 18, 22, 39, 65] to handle the interpolation problem without limiting the number of contours in the slices, their geometries, or their containment hierarchies.

A practical assumption in almost all previous work has been that the input consists of parallel slices, and, in addition, that adjacent layers are independent. Thus, only a single pair of successive parallel slices are considered and interpolated at each instance, and the reconstructed object is the concatenation of the interpolating models computed for all the layers. The only studies, that we are aware of, which handle input with nonparallel cross-sections, are [11, 17, 32, 56, 68].

Another class of solutions attempts to reconstruct the object by defining an implicit three-dimensional function, from which the object is extracted as a level set, or define a partial differential equation, for which the sought surface is a solution. (see, e.g., [28, 30, 74]). These methods usually result in smooth surfaces, but are generally, up to the time our work was conceived, are limited to parallel slices, or at least a set of slices with some partial ordering defined on it.

Another practical assumption, which appears in almost all previous works, is
that all the contours are “unicolored,” that is, they describe the same type of material. Thus, there are no constraints on which contours should or should not be interpolated with other contours. This is a natural assumption, for example, when one reconstructs a single organ (e.g., bone) from medical data. However, input exists in which the cross-sections simultaneously describe several types of tissues (e.g., muscles, fat, bone, blood, etc.), so that each contour is given a label (“color”) that characterizes its type of material. For such input, reconstruction of each label regardless of the other labels might result in inconsistencies and intersections between the reconstructed outputs. The only studies, that we are aware of, which address multi-colored cross-sections, are [48, 56]. These papers also suggested the term “curve network” to denote the partition of a planar region into colored subregions.

Thus, to the best of our knowledge, only Liu et al. [56] suggested an algorithm that interpolates between multi-colored curve networks in nonparallel cross-sections. Liu et al. considered the arrangement of the planes supporting the sections, and handled each cell of this arrangement separately. In every cell of the arrangement, they projected the portions of the contours that lie on the boundary of the cell onto the medial axis of the cell. By erecting “walls” connecting the original contour portions and their projections on the skeletons, they reconstructed portions of the colored surfaces (boundaries of the sought volumes) within each cell. Finally, they corrected jagged surfaces by general mesh-smoothing techniques that are independent of the reconstruction process.

Finally, all previous work assumed that all slices are complete, in the sense that no data are missing within the sections. This assumption is not practical, for example, for data originating from ultrasound scanning. Such areas (within the input sections) where data are not available, can either belong or not belong to the reconstructed object, or, in the multi-label case, can belong or not belong to any tissue type, at all. We are not aware of any previous reconstruction algorithm that is able to handle such partial-slice situations.

The algorithm we describe in Chapter 3 attempts to solve the slice-interpolation problem in its full generality. It handles parallel, nonparallel, or any combination of cross-sections. It supports the multi-label case, with no limit on the number of different labels. It can also handle the situation of partial sections, where data may be missing in portions of the sections. Naturally, within each section, the algorithm has no restrictions whatsoever on either the geometries or topologies of the labeled domains; furthermore, it does not make any assumptions about similarities
between partitions of different sections. The algorithm reconstructs the original three-dimensional partition by interpolating simultaneously all the cross-sections, so that planar domains are connected only to other domains with the same label, no two reconstructed spatial domains intersect, and no unnecessary gaps remain between the reconstructed labeled domains. The algorithm is guaranteed to interpolate a valid spatial partition for any possible input, and is intuitive in the sense that it tends in practice to minimize the surface area of the reconstruction. This is because it uses an offset distance function to locally decide which contour features to bind.

Our geometric algorithm relies on finding the correspondence between partial planar objects, representing the slice planes. The three-dimensional medial axis is not adequate for this purpose anymore, because it is piecewise-linear only for convex polyhedra, and fails to be so in arrangements of partial planes. We therefore define and investigate the three-dimensional generalization of the straight skeleton in Chapter 2, that will guide us through the reconstruction algorithm.

1.2 Discrete Shape Correspondence

Morphing between two three-dimensional objects is defined as the gradual change from one to the other. Much like the reconstruction problems, this problem is ill-defined, and usually requires the solution of two subproblems: finding the correspondence between the objects, i.e., which portion of the first object morphs unto the second object, and then the design of the gradual motion, which usually amounts to defining a computational interpretation of smooth and valid transitions. Finding the correspondence is a key in defining a good morph; a human usually perceives a natural morph as one that takes portions of variable similarity from one object to the next one, and which also regards the symmetries of the objects. This problem, usually referred to as shape matching, has received much attention in the last decade (for an exhaustive survey, see [54]). Recent works attempted to create an intrinsic shape matching, in which the objects are investigated within the group of isometric transformations. These works usually apply discrete intrinsic metrics in order to compare and extract similar features between objects, and within one object. Kim *et al.* [57] used conformal factors, Kazhdan and Funkhauser [52] applied a spherical harmonics decomposition, and the works [6, 64] investigated the use of Diffusion Distances, which rely on the computation of the Heat Kernel of an object. Diffusion distances and heat kernels have proven to be very useful
tools in multiscale shape investigations. However, such information comes with the price of a cumbersome computation. In Chapter 4 we investigate a method to compute this heat kernel efficiently, and show its benefits for such feature extraction applications.

1.3 Preliminaries

1.3.1 Two-Dimensional Straight skeletons

Definition and computation

The straight skeleton, introduced in [1], can be viewed as an approximation of the Medial Axis of a polygon. The skeleton is the result of offsetting all the edges inwards at equal speed, tracing the movement of vertices along the angular bisectors of adjacent edges, until the polygon vanishes. Two types of events occur in the course of this process (see Figure 1.2):

1. **Edge event**: An edge shrinks to a point, and two bisectors meet to create a node of the skeleton.

2. **Split event**: An edge splits when the bisector of an opposite vertex (denoted as a Reflex Vertex) hits it. A skeleton node is created here as well. The polygon then splits into two polygons that continue to shrink to zero. If the reflex vertex and the edge are from different polygonal chains originally (such as a polygon and its hole), this might lead to uniting of two offsetting fronts.

The union of the traces of all bisectors in the process is the straight-skeleton of the polygon $S(P)$, and is a unique structure that serves as a partition of the polygon. The area that an edge $e$ sweeps along the process is called the face of $e$.

Various algorithms exist in the literature to compute the straight skeleton of a polygon, all of which are based on the simulation of the shrinking process, such as the one given in [1]. Unfortunately, since local changes in the polygon have a significant global effect on its straight skeleton, the algorithms to compute it are limited in utilizing traditional methods, such as divide-and-conquer or incremental methods. In this work, we employ the algorithm of [38] which runs in $O(n(r + \log n))$ (where $n$ is the number of vertices in the polygon and $r$ is the number
Figure 1.2: An example of a polygon ABCDE and its straight skeleton (The propagation is in dashed lines). The edges EA and CD vanish in edge events, and B splits the edge DE of its reflex vertices), because of its intuitive approach. The fastest (running-time-wise) algorithm known in the literature is due to Eppstein and Erickson [33], which running time is \( O(n^{1+\varepsilon} + n^{8/11+\varepsilon} \cdot r^{9/11+\varepsilon}) \) for any \( \varepsilon > 0 \).

The straight skeleton \( S(P) \) has the following properties, investigated in [1]:

- The straight skeleton of a simple polygon is a tree. For an \( n \)-gon \( P \), there are \( n \) faces, \( n - 2 \) vertices and \( 2n - 3 \) edges (not including the original vertices and edges of \( P \)) in \( S(P) \). In case a polygon has \( h \) holes, \( h \) edges can be removed from the skeleton to produce a tree. When a polygon is not in general position (e.g., when more than one edge event occurs at the same point, like in a regular polygon), some skeletal edges may be degenerate (having a zero length).

- Exactly three skeletal edges (considering degenerates, which create skeletal nodes with zero-length edges between them) coincide at each skeletal inner node.

- Every face of \( S(P) \) is monotone with respect to its defining edge of \( P \).
1.3.2 Discrete Multiscale Heat Kernels

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 is formally defined as the amount of heat present at point $y$ in time $t$, when a delta heat source (i.e., of infinite magnitude and unit energy) is put in point $x$ at time $t = 0$. 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. See Figure 1.3 for an example.

In the context of shape analysis, the heat kernel of a surface has some remarkable properties, explored in [66]. 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, the diagonal is denoted as the heat kernel signature (HKS) in [66] (see Figure 1.3). 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 [55]. 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 [31].

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 a large value of $t$, a small number of eigenvectors having the smallest eigenvalues are sufficient for a good approximation of $k_t$, as the influence of an eigenvector $\phi$ with a corresponding eigenvalue $\lambda$ on $k_t$ decreases exponentially like $e^{-\lambda t}$. 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, $k_t = e^{-tL}$, 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 [59], 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. Figure 1.4 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).

1.4 Structure of the Dissertation

In Chapter 2 we define the three-dimensional straight skeleton and examine its properties. In Chapter 3 we use this skeleton to create an algorithm for reconstruction from cross-sections, and in Chapter 4 we propose an efficient algorithm to compute the heat kernel according to its properties.
Chapter 2

The Three-Dimensional Straight Skeleton

2.1 Definition

We define the three-dimensional straight skeleton, and address its unique issues.

**Definition 2.1.1** Given a set of polyhedra $P$, their straight skeleton $S(P)$ is the trace of the intersections between the faces defining $P$, as they propagate inward at equal rate in the direction of their respective normal vectors.

In a propagation of the boundary of a polyhedron, the faces move inwards until the polyhedron vanishes. In this process, the topology of the polyhedron changes at certain points in time to which we refer as events. A polyhedron may split into several connected components or vanish in one piece. The three-dimensional straight-skeleton comprises of 3-cells, planar faces, and edges (see Figure 2.1), traced by the faces, edges, and vertices of the polyhedron, respectively. The vertices of the skeleton are the locations of events.

The “rooftop” analogy of two-dimensional straight skeletons applies here as well; a “height” field can be associated with the vertices, being the time of the creating event. Thus, every vertex has four coordinate values, and the straight skeleton becomes a four-dimensional polyhedron embedded in three dimensions.
Figure 2.1: An example of the straight skeleton of a polyheron. The edges of the polyheron are in solid lines, the skeletal edges are in dashed line, the skeletal vertices are marked by crosses, and examples of a skeletal face and 3-cell are featured in grey.
2.2 Ambiguity of the Definition

A convex polyhedron is uniquely defined by the supporting planes of its faces. However, a nonconvex polyhedron is defined by both the supporting planes of its faces, and a given topology, which is not necessarily unique. Thus, a polyhedron can propagate from a given intermediate (or initial) state into one or more topological configurations, all of which are valid, as shown in Figure 2.2. This is the problem of the weighted rooftop: having a base polygon and slopes of walls, determine the topology of the rooftop of the polygon, which does not have a unique solution in all cases. However, in our definition of the skeleton, we define a consistent method for establishing topological changes in events and for initial topology, based on the two-dimensional weighted straight skeleton.

2.3 Properties of the Straight Skeleton

We assume general position for now, and then discuss degeneracies in Section 2.6.

**Lemma 2.3.1** A 3-cell is monotone with respect to its defining face.

**Proof.** Let $N$ be a ray orthogonal to a defining face $f$, based on $f$ and extending into the 3-cell it sweeps. Assume for contradiction that there exist points $a$ and $b$ on $N$ at which $N$ leaves the 3-cell and returns to it, respectively. Between these points, the ray $N$ is crossed by propagating faces which are not perpendicular to $N$, and therefore will reach $b$ before $f$ does, which contradicts the existence of the point $b$. ■

**Lemma 2.3.2** All propagating vertices have degree 3. An event, a topological change, occurs only at the intersection of exactly four planes.

**Proof.** Propagating vertices are defined as the intersection of propagating planes. Such a vertex is uniquely defined by exactly three planes, which also defines the vertex’s three adjacent propagating edges.\(^1\)

The topology of the polyhedron remains unchanged during the propagation between events. We define all possible events (see Figure 2.3):

\(^1\)We discuss the propagation of polyhedra that initially have vertices of degree greater than 3 in Section 2.4.
Figure 2.2: A vertex of degree 5, and two possible topologies during the propagation, depicting the ambiguity of the straight skeleton.
1. **Edge Event.** An edge vanishes, as its two endpoints meet. This is the meeting point of the four planes around the edge.

2. **Hole Event.** A reflex vertex (adjacent to three reflex edges, also called “spike”) runs into a face. The three planes adjacent to this vertex meet the supporting plane of the face.

3. **Split Event.** A ridge vertex (adjacent to one or two reflex edges) runs into an opposite edge. The faces adjacent to the ridge meet the face adjacent to the twin of the split edge.

4. **Edge-Split event.** Two reflex edges cross each other. Every edge is adjacent to two planes.

5. **Vertex event.** Two ridges sharing a common reflex edge meet. This is a special case of the edge event, as it is the meeting of the endpoints of the reflex edge, but it has different effects, and so it is considered a different event. Vertex events occur when a reflex edge runs twice into a face, and the two endpoints of this edge meet.

A convex polyhedron induces only edge events during propagation, and reduces to a single tetrahedron before the last event, which is the simultaneous edge events of all four last edges. A general (nonconvex) polyhedron may split into several connected components, which will be reduced into tetrahedra and similarly vanish. All these events are meeting points of four planes, and other types of events are not accounted for, as they do not occur in general position (two reflex vertices running into each other, etc.), which are meeting points of more than four planes at a location.

**Corollary 2.3.3** Every skeletal edge is adjacent to exactly three 3-cells and three skeletal faces. Every skeletal vertex is adjacent to exactly four skeletal edges, four 3-cells, and six skeletal faces.

**Proof.** Every skeletal edge is the trace of a propagating vertex, and, therefore, is adjacent to exactly three propagating planes, which create 3-cells, and to three propagating edges, tracing skeletal faces. Every skeletal vertex is the result of an event, and, therefore, is adjacent to four 3-cells. Between four adjacent 3-cells, there are $\binom{4}{2} = 6$ skeletal faces, and four skeletal edges (one for each pair of skeletal faces).
Figure 2.3: The five types of events. The solid lines are the original edges, and the dashed lines are their locations after the propagation. The dotted arrows show the progression of these edges, up to the time of the event.
2.4 Constructing the Straight Skeleton

Our algorithm is an event-based simulation of the propagation. We do not construct the intermediate polyhedra, but only the skeleton itself.

Initial Topology

Vertices of a polyhedron can initially have a degree greater than three. Therefore, upon initiating the propagation, each such vertex splits into several vertices of degree three (see Figure 2.4), a topology which we seek to discover. This weighted-rooftop problem can have several valid solutions. Our approach is based on sectioning the faces surrounding the initial vertex with one plane or more, and establishing the weighted straight skeleton of the intersection of these faces with the section plane, with the weights determined by the dihedral angle of these faces with the cutting plane, after an infinitesimally-small propagation. This approach always yields a unique valid solution. We establish this method for all types of vertices:

- **Convex vertices and spikes.** In a convex vertex, all of the edges are on the negative side of its osculating plane. Edges adjacent to convex vertices do not have to be convex. A spike is the opposite of a convex vertex, as all edges are on the positive side of the osculating planes. The topologies of both types can be determined by sectioning and propagating and finding the weighted straight skeleton.

- **Saddle Vertices.** In saddle vertices, some of the edges lie on the negative side of the osculating plane, denoted “up” edges, and some on the positive side, denoted “down” edges. We use two sectioning planes. First, we section the edges in the negative side of the plane. Then, we construct their straight skeleton. The section is not a closed polygon, as the intersections of the faces that lead from a “down” edge to an “up” edge, and vice versa, are infinite rays. Every pair of such infinite rays creates a wedge (see Figure 2.5(b)). When computing the weighted straight skeleton of this planar shape, one skeletal vertex will be adjacent to an infinite ray for each such wedge. We call these vertices “portals.”

Next, we take each “up”-streak (i.e., a set of “up” edges that are consecutive

---

2 Any consistent choice of a cutting plane which intersects all faces and parallel to none would suffice.
(a) The original polyhedron. Vertex $v$ has degree 5.

(b) The cross-section and its weighted straight skeleton. The polyhedron. Notice that $v$ vertex $v$ becomes three new spawned three skeletal edges. vertices: $v_1$, $v_2$, and $v_3$.

(c) The straight skeleton of the polyhedron. Notice that $v$ vertex $v$ has spawned three skeletal edges.

(d) The propagated polyhedron. Vertices $v_1$, $v_2$, $v_3$ trace their skeletal edges.

Figure 2.4: Changing the initial topology of a vertex of degree greater than 3. The skeleton is shown in dashed lines.
in a counterclockwise order around the saddle vertex) and section it by a second plane above the osculating plane. We get a segment-chain, beginning and ending in infinite rays (see Figure 2.5). We calculate the straight skeleton of the segment-chain, resulting in a single vertex of degree 2. Every “up”-streak corresponds to a “down”-wedge, and we connect the last “up” vertex with its corresponding “portal” vertex by an edge. Thus, we get a new fully-connected topology for saddle vertices.

### 2.4.1 Data Structures

We use an event queue which holds all possible events sorted by time, and a set of propagating polyhedra, initialized to the input polyhedron (or polyhedra), after the initialization of topology. The propagating polyhedra (the generalization of the SLAV structure [38] in two-dimensions) contains only topological information and locations of vertices at their creation; no intermediate offset polyhedra are contructed. When an event occurs, we connect the vertices that participate in the

---

[3] In practice, we extend the last two segments by a large-enough distance, and connect them, so this vertex is the last edge event of this faraway edge of a closed polygon. We do a similar process in the “down” step in order to avoid computing the straight skeleton of unbounded shapes.

[4] Technically, it is easy to construct such offset polyhedra by computing the intersections of adjacent planes, offset by the given time.
event with skeletal edges, with one end at the vertices in their former location, and the other at the location of the current event, thus connecting the locations of last and current events. The faces and 3-cells are formed in a post-processing step, although they can easily be formed on-line.

2.4.2 The Algorithm

The general algorithm comprises the following steps:

1. Collect all possible initial events.

2. While the event queue is not empty:
   (a) Retrieve the next event and check its validity (see Section 2.4.3).
   (b) If the event is not valid, return to Step 2.
   (c) Create a vertex at the location of the event and connect the vertices participating in the event.
   (d) Change the topology of the propagating polyhedron according to the actions taken in Step 2(c). Set the location of the event to the newly-created vertices.
   (e) Create new events for newly-created vertices, edges and faces and their neighbors, if needed.

We will next describe how to deal with the different types of events defined in Section 2.3. The algorithm is bound to terminate, as the number of all possible events is always bounded by the number of combinations of four propagating faces.

2.4.3 Collecting Events

The heart of the computation of events is finding the intersection of four planes. Let $P_i : a_ix + b_iy + c_iz + d_i = 0$, $0 \leq i \leq 4$, be the plane equations, and let $N_i = \sqrt{a_i^2 + b_i^2 + c_i^2}$. By solving the linear system

$$
\begin{pmatrix}
  a_1 & b_1 & c_1 & N_1 \\
  a_2 & b_2 & c_2 & N_2 \\
  a_3 & b_3 & c_3 & N_3 \\
  a_4 & b_4 & c_4 & N_4
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z \\
  t
\end{pmatrix}
= 
\begin{pmatrix}
  -d_1 \\
  -d_2 \\
  -d_3 \\
  -d_4
\end{pmatrix},
$$

(2.1)
we obtain the coordinates and time of the event. In general position, this system has a unique solution. However, the time can be less than zero (or less than the current time if the event is computed within Step 2 of the algorithm), in which case the event is not valid. In addition, the locations of the events must be within the area the participating edges sweep, i.e., bounded between the edge and the trisectors of its endpoints.

**Edge events.** There are three types of edge-event outcomes (see Figure 2.6):

- An edge vanishes as its four surrounding planes meet. In this event the two planes that were not adjacent become adjacent, sharing a newly-created edge. Therefore, an edge vanishes, and a new edge appears.

- A face vanishes when its last three edges vanish simultaneously. The three planes adjacent to the vanishing face now share a vertex. No new edges are created.

- A simplex vanishes when its four faces (and four edges) vanish simultaneously. This is the last event for each connected component in the propagation of a polyhedron.

**Hole events.** The reflex vertex establishes a new triangular hole within the face it hits (see Figure 2.7(a)). The face is then adjacent to the three faces defining the reflex vertex, sharing the edges of the hole with them. The three vertices of the hole are new ridges in the hit face.

**Split events.** The ridge vertex splits through the edge and creates a new ridge in the other face adjacent to the split edge (see Figure 2.7(b)). The edge splits into two parts.

**Edge-split events.** New ridges are created, in the same manner as in the split event, on all four faces adjacent to the edges (see Figure 2.7(c)).

**Vertex events.** Two ridges sharing a reflex edge unite. The face which both ridges share is split (see Figure 2.7(d)).

---

5 The rays traced by vertices during propagation.
6 A convex polyhedron propagates into a single connected component, as it does not split.
Figure 2.6: The situation before and after edge events: in (a,b), edge $e$ vanishes and edge $e'$ is created, and in (c,d) face 4 vanishes.
Figure 2.7: Topology changes in hole, edge and edge-split events. The dashed lines denote the topology before the event (in (d), dotted lines show the second split event done by the same edge. The endpoints of the propagating split edge here are marked by squares. The location of the immediate vertex event is marked by a circle), and the solid lines denote the topology after the change.
2.4.4 Memoryless Propagation

Given our definition of events and handling of initial vertices, the propagation of any polyhedron is memoryless; the result of a future propagation depends only on the current topological structure, and the transition is smooth. Thus, we can pretend to initialize the propagation in any given time, including the time of an event. This means that the aforementioned events are actually a hard-coded enumeration of all possible topological outcomes when initializing a vertex of four coinciding faces. This resolves any possible ambiguity of event outcomes in difficult situations.

2.5 Complexity

It is known that the two-dimensional straight skeleton of a polygon with \( n \) edges has exactly \( n - 2 \) skeletal vertices and \( 2n - 3 \) skeletal edges. Unfortunately, the number of skeletal vertices, edges, and faces of the three-dimensional straight skeleton may vary even for two polyhedra of the same complexity, but having a different topology. We establish some bounds on the maximum complexity of the straight skeleton.

2.5.1 A Combinatorial Lower Bound for General Polyhedra

We first address the convex case. As mentioned in the introduction, Held [44] showed that the complexity of the straight skeleton of an \( n \)-vertex convex polyhedron is \( \Omega(n^2) \) (see Figure 2.8). The following theorem shows a matching upper bound. Thus, we establish a tight bound of \( \Theta(n^2) \) on the complexity of the straight skeleton in the worst case.

**Theorem 2.5.1** Let \( n \) be the number of faces of a polyhedron. Then, the straight skeleton of the polyhedron contains exactly \( n \) 3-cells and \( O(n^2) \) faces, edges, and vertices.

**Proof.** The straight skeleton of the polyhedron contains \( n \) 3-cells because each face \( f \) of the polyhedron sweeps exactly one 3-cell of the skeleton, which is monotone with respect to \( f \). Since the polyhedron is convex, no face of it is split during the propagation of the polyhedron’s boundary. In addition, every 3-cell is convex. Therefore, every 3-cell of the polyhedron, during the entire offsetting process, shares at most one skeletal face with any other 3-cell (otherwise faces of the polyhedron would be split, or 3-cells would not be convex). Thus, there are \( O(n^2) \)
Figure 2.8: A worst-case-complexity skeleton in the convex case.
skeletal faces, where each 3-cell is bounded by $O(n)$ of them. Hence, every 3-cell is bounded by $O(n)$ skeletal edges and vertices, for a total of $O(n^2)$.

We now show that the three-dimensional straight skeleton of an $n$-vertex general simple polyhedron can have asymptotic combinatorial complexity strictly greater than the complexity of the three-dimensional straight skeleton of an orthogonal polyhedron.

**Theorem 2.5.2** The complexity of a three-dimensional skeleton for a simple polyhedron is $\Omega(n^2 \alpha^2(n))$ in the worst case, where $\alpha(n)$ is the inverse of the Ackermann function.

**Proof.** We begin by showing that the cross-section of a set of growing wavefronts can have the same complexity as the upper envelope of a set of line segments in the plane. The main idea to produce such a cross-section is to set up a sequence of $n$ triangular prisms sticking up out of a side of the polyhedron. (One such prism is shown in Figure 2.9(a).) The prisms have long skinny rectangular bases aligned with the $xy$ plane, where the long edge is parallel to the $x$ axis. Their $xz$ cross-sections are very sharp isosceles triangles. The prisms are positioned on the $xy$ plane such that in a side view, looking at the positive $y$ direction, the set of their top edges looks like a set of intersecting line segments.

The goal is to show that, although the prisms do not really intersect in space, they produce a wavefront whose combinatorial complexity at any height $z$ is at least that of the upper envelope of the imaginary two-dimensional set of segments (top edges of the prisms) which is highlighted in Figure 2.9(b).
To this aim, we start from a set of prisms, for which the complexity of the upper envelope of the segments, obtained by computing the $xz$ projection of the top edges of prism, attains the maximum possible value. By choosing an appropriate $y$ span for the base of each prism, we make sure that all the propagating wavefronts (induced by the prisms) have the same (arbitrarily chosen) $z$ component. This is done by choosing the $y$ spans of the bases so that the $yz$ projection of the dihedral angle between the two slanted faces of a prism (marked by $\theta$ in Figure 2.9(a)) has the same value for all prisms. Thus, all the fronts induced by the prisms advance along the $z$ direction at the same speed. The imaginary upper envelope of the prisms consists of segments, each having its own midpoint (shown in Figure 2.9(b)). Our goal is to guarantee that regardless of the topological changes that the top edges of the prisms undergo while being offset, the respective midpoints of all the initial segments never vanish. Moreover, we also want to guarantee that if we extend any such midpoint along the $y$ axis, this line extension is completely ruled by the respective prism, that is, the entire line is swept first by the offset of that prism and cannot be “taken over” by another prism.

We already assured that the midpoint of every original segment of the initial envelop is not taken over during the offsetting process. This follows from the fact that all prisms propagate upward at the same speed. We now need to assure that the entire $y$ extension of any midpoint is not taken over, that is, it is always swept first by the wavefront of the same prism. The only situation in which this can be violated is when a lower prism is closer to the extension of a midpoint (along the $z$ direction) than the original prism (along the $y$ direction). This situation is illustrated in Figure 2.9(b) by the respective distances $d_1$ and $d_2$. To avoid this undesired situation we stretch the entire construction along the $z$ axis, so that the vertical distance between every such midpoint and the segment below it (in the $xz$ projection of the set of segments) is more than the width ($y$ span) of the entire construction. This is obviously possible, and it does not hamper the property that all prisms propagates upwards at the same speed.

It remains to guarantee that the midpoint of every segment of the imaginary upper envelop remains “alive” during the evolution of this envelop. That is, we need to avoid any situation in which a midpoint vanishes from the upper envelop. This can in principle happen because the prisms are not only offset upward but they also grow in both directions along the $x$ axis. These undesired situations can also be avoided by stretching the entire construction, once again, along the $x$ axis, so that the horizontal ($x$-wise) distance from each midpoint of a segment of the
original imaginary upper envelope to the endpoints of the respective segment is at least the width \((y\text{ span})\) of the entire construction. This is also obviously possible, and it does not ruin the previously-obtained properties since we leave the \(y\) span of the entire construction intact.

Thus, if we consider any fixed value of \(z\), we find in the respective \(xy\)-parallel plane \(\pi_z\) a set of \(y\)-oriented “slabs,” each of which is ruled (in the sense defined above) by a different prism. No matter what happens in between these slabs, this implies that the complexity of the intersection of the skeleton with \(\pi_z\) is at least the complexity of the original imaginary upper envelope of the line segments.

Wiernik and Sharir [80] show that the upper envelope of a set of line segments in the plane can have \(\Omega(n\alpha(n))\) complexity in the worst case. Thus, the complexity of the cross-section (at \(\pi_z\)) of the set of growing wavefronts in our construction can also be made to be \(\Omega(n\alpha(n))\).

Our lower bound for the three-dimensional skeleton follows, then, by having such a set of growing wavefronts attached to the “floor” of a simple polyhedron interact with an orthogonal set of similar growing wavefronts attached to the “ceiling” of a simple polyhedron. The interaction of the two orthogonal sets of “slabs” will produce a number of pieces of the straight skeleton that is quadratic in the number of each such set of slabs.

We conjecture, based on empirical tests, that the upper bound of the straight skeleton is \(O(n^3)\), and even that is probably far from being tight, as it is hard to conceive such an example. We feel that the real upper bound is superquadratic as the lower bound is, but have not attained a full proof yet. However, we have noticed that in a propagation without any hole events (initial surface containing neither spikes nor tunnels), the complexity of the skeleton is \(O(n^2)\), as two 3-cells share only one skeletal face. In addition, we have empirically noticed that the total number of skeletal faces shared by the same pair of neighboring 3-cells grows linearly, or slightly superlinearly in the number of spikes and tunnels in the input. If proven, this will lead to a tight quadratic, or slightly-superquadratic bound.

Another question that remains open is the complexity of the propagating structure in the course of the algorithm, at every given point in time. Here, again, we conjecture that the structure has a linear complexity, based on our experiments.
2.5.2 Time Complexity

The time complexity also remains an open question which is directly linked to the complexity of the propagating structure. Suppose that we have a total of $O(r)$ reflex edges and reflex vertices to begin with (whether split or spike vertices). Thus, collecting all the initial events requires the matching of every reflex component against every other component of the polyhedra, which takes $O(nr)$ time. At any event we have to match all newly-created reflex components to all other components. Thus, if the maximal complexity of the propagating structure is $O(p)$, and since $O(1)$ components are created at any event, the complexity of handling a single event is $O(p)$. Denote the complexity of the skeleton as $O(k)$, then the complexity of the entire algorithm is $O(pk + nr)$. Based on our conjecture that the combinatorical complexity of the skeleton is superquadratic, and that the complexity of the propagating structure is linear, we also conjecture that the time complexity of the algorithm is slightly supercubic.

2.6 Degeneracies

A degeneracy is usually created where the polyhedron has some symmetry. The most common degeneracy involves parallel features (mostly faces, like in a box) collapsing one to the other, or concurrent propagation of more than three planes (such as in a vertex which cross-section is a regular polygon). These can be easily handled with a slight perturbation and the usage of zero-length edges to get correct results. Uncommon (but existing) degeneracies are reflex events that involve more than four planes—such as events that involve two “spikes” or two ridges not sharing a common reflex edge (i.e., unlike our vertex event). All of these events can be treated as a composition of the five basic events (e.g., two meeting ridges can be treated as a split event following edge events), but they must be identified precisely beforehand, as regular perturbation can result, since in the two-dimensional case, in wrong results.

2.7 Experimental Results

We have implemented the algorithm for computing the straight skeleton of a general polyhedron in Visual C++ .NET2005, and experimented with the software on a 3GHz Athlon 64 processor PC with 1GB of RAM. We used the CGAL library to
Figure 2.10: Sample objects.

<table>
<thead>
<tr>
<th>Object</th>
<th>Vertices</th>
<th>Edges</th>
<th>Facets</th>
<th>Vertices</th>
<th>Edges</th>
<th>Faces</th>
<th>Cells</th>
<th>Time (Sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>12</td>
<td>20</td>
<td>10</td>
<td>8</td>
<td>24</td>
<td>25</td>
<td>10</td>
<td>0.312</td>
</tr>
<tr>
<td>(b)</td>
<td>20</td>
<td>30</td>
<td>12</td>
<td>25</td>
<td>60</td>
<td>46</td>
<td>12</td>
<td>0.719</td>
</tr>
<tr>
<td>(c)</td>
<td>28</td>
<td>42</td>
<td>16</td>
<td>45</td>
<td>104</td>
<td>74</td>
<td>16</td>
<td>0.567</td>
</tr>
<tr>
<td>(d)</td>
<td>20</td>
<td>30</td>
<td>12</td>
<td>16</td>
<td>42</td>
<td>37</td>
<td>12</td>
<td>0.188</td>
</tr>
<tr>
<td>(e)</td>
<td>20</td>
<td>18</td>
<td>9(+1)</td>
<td>15</td>
<td>45</td>
<td>56</td>
<td>9</td>
<td>0.250</td>
</tr>
<tr>
<td>(f)</td>
<td>12</td>
<td>18</td>
<td>10</td>
<td>21</td>
<td>48</td>
<td>37</td>
<td>10</td>
<td>0.484</td>
</tr>
<tr>
<td>(g)</td>
<td>16</td>
<td>24</td>
<td>11</td>
<td>6</td>
<td>21</td>
<td>25</td>
<td>11</td>
<td>0.177</td>
</tr>
<tr>
<td>(h)</td>
<td>16</td>
<td>24</td>
<td>11</td>
<td>12</td>
<td>36</td>
<td>33</td>
<td>11</td>
<td>0.146</td>
</tr>
<tr>
<td>(i)</td>
<td>16</td>
<td>24</td>
<td>10</td>
<td>32</td>
<td>32</td>
<td>29</td>
<td>10</td>
<td>0.172</td>
</tr>
</tbody>
</table>

Polycubes
perform basic geometric operations, such as plane intersection, with the embedded rational exact number type GMPQ. The source code consists of about 6,500 lines of code. Figure 2.10, given in the appendix, shows the straight skeletons of a few simple objects, and the performance of our implementation. (Note that object (e) contains one hole polygon in addition to the 9 facets.)
Chapter 3

Reconstruction of multi-label domains from partial planar cross-sections

In this chapter, we describe the first method we devised for reconstruction, based on the straight skeleton we defined in the previous chapter. The input to this problem is a set of arbitrarily-aligned cross sections, possibly with several domains, and with unknown regions. We utilize the three-dimensional straight skeleton of the cells of the arrangement of planes in order to match portions of contours from different slices.

3.1 Overview of our Method

The input to our algorithm is a set of planar cross-sections, partitioned into a set of closed labeled (“colored”) domains. Some planes might contain areas with an “unknown” color; these areas are treated as “holes” in the plane. The output is a colored triangulated mesh that represents the boundaries of the different domains in a partition of the space, such that this partition interpolates the known portions of the input planar cross-sections.

To distinguish between the arrangement of the planar cross-sections, and the labeled domains, we denote the latter as “images.”

Our algorithm proceeds as follows (see Figure 3.2, and a two-dimensional exemplification of the algorithm in Figure 3.1):
Figure 3.1: An overview of the algorithm in 2D. The arrangement of partial segments is constructed. The straight skeleton of a single cell is computed (dashed). Notice that singular edges evolve as degenerate rectangles. A skeletal cell of type ‘a’ has obtuse walls, and therefore, the purple image stretches onto these walls (see Section 3.4). A cell of type ‘b’ does not have walls, and so the image of the base edge simply projects onto it. A cell of type ‘c’ is created from a degenerate vertex (which matches degenerate edges in 3D), and the image adjacent to this vertex projects onto the roof of the cell.
Figure 3.2: Reconstruction from partial multi-colored slices. Two slices (with green and blue contours) are full, while the slice with the red contour lies only on a halfplane. The straight skeleton of the arrangement of partial planes is computed, the contours are projected, and, finally, the meshes are created accordingly.

1. Construct the spatial arrangement of the partial cross-sections (portions of planes).

2. Compute the three-dimensional straight skeletons of every cell of the arrangement of cross-sections; the skeletal cells are a refinement of the arrangement cells [8], such that every skeletal cell is defined by one arrangement cell (denoted the “base face”).

3. For each skeletal cell of each arrangement cell:
   
   (a) Project the images from every base face onto the other faces of that skeletal cell;
   
   (b) Each skeletal face is shared by two skeletal cells on both sides. For each such skeletal face, compute the overlay of the projected images on both sides. For each such overlay, compute a constrained Delaunay triangulation;
   
   (c) Reconstruct portions of the mesh from these triangulated overlays.

4. Stitch together all the pieces of the mesh in a consistent orientation.

5. Apply mesh fairing and refinement.

We now describe in detail the algorithm steps.
Figure 3.3: The propagation of a planar patch (solid) with an image adjacent to the degenerate edges. The patch propagates into a triangular prism, and the image edges, which are adjacent to the degenerate edges, propagate into images faces. A similar example is depicted in the 2D setting, with a degenerate segment that propagates into a rectangle, and the images adjacent at the vertices propagating into image edges.

3.2 An Arrangement of Partial Cross-Sections

Using the arrangement of arbitrary-oriented planar cross-sections is a recognized technique for reconstruction of an object from nonparallel cross-sections (see, e.g., [11, 56]). Computing the arrangement of full planes is a topic well-studied in the literature. The cells of such an arrangement are all convex. In our setting, sections may be partial, so that the intermediate problem with which we deal is the computation of the arrangement of portions of planes. The cells of such an arrangement are not necessarily convex, and may contain degenerate or even disconnected planar patches “floating” inside the cell. We use the general Nef polyhedra [42], provided by [23] to construct such an arrangement.

For practical purposes, in order to avoid infinite cells, we “wrap” the meaningful region of the arrangement (where there are data) with a large-enough bounding box. The images within the cross-sections are subdivided by the faces of the arrangement. The resulting arrangement contains (not necessarily convex) cells, on whose faces portions of the original images of different colors lie.

From this point up to Section 3.6.1 we discuss the reconstruction of the mesh (portions of the boundaries of the reconstructed colored objects) within a single arrangement cell. We ignore arrangement cells of two types: (a) Cells that are
empty of images; and (b) Cells that do not contain any image vertex or edge. A cell of the former type is free of any material, while a cell of the latter type is fully contained in a single domain. In both cases, such a cell does not contribute any portion of the reconstructed boundary mesh. In a post-processing step, we simply “glue” together all the interpolated pieces of mesh reconstructed within all the cells of the arrangement. Since the algorithm is interpolatory, that is, it produces colored meshes that are connected exactly to the images on the boundaries of the cells, and each image appears identically twice on twin faces, shared by two neighboring cells of the arrangement, the validity of the gluing step is guaranteed.

3.3 Using Straight Skeletons of Nonconvex Cells

It is well-known that the cells of the arrangement of full planes are always convex polyhedra. Recent works [11, 56] used the medial axis of a convex polyhedron in order to subdivide the cell, project on this axis the images lying on the boundary of the cell, and match portions of images accordingly. However, the cells of partial planes, supporting partial cross-sections, are not necessarily convex, and therefore, may have nonlinear medial axes, which complicates projective methods.
considerably. In order to avoid these complications, we use the three-dimensional straight-skeleton described in this chapter.

Every skeletal cell, induced by a base face of the arrangement of planes, contains skeletal faces of two more types: skeletal faces that form an obtuse dihedral angle with the base, and are denoted as “walls,” and faces that form an acute angle with the base, which are called the “roof” of the skeletal cell (see Figure 3.4 for an illustration). The monotonicity property of skeletal cells determines that both the entire roof and entire wall are connected. That is, every wall (similarly, roof) face is connected to any other wall face by a path, which can only pass through wall faces. Note that neither all wall faces nor all roof faces must be adjacent to the base face. In a convex skeletal cell, which is the only case in an arrangement of full planes, all angles are acute, and therefore, such cells have no walls.

To support degenerate situations created by partial slices (in which portions of both sides of a plane belong to the same cell), we expand the definition to include the propagations of portions of planes and planar patches as follows. Every edge of the cell, which is adjacent to two faces of the cell lying on opposite sides of the same plane, propagates into a new face orthogonal to this plane. (This is a generalization of the definition of the straight skeleton of a straight-line graph [1], in which each segment is modeled as a degenerate rectangle.) Following this definition, we consider every floating planar patch as a degenerate prism, all of whose isolated edges (adjacent to only two faces of the arrangement) induce new faces as described above. Figure 3.3 illustrates these situations. Having computed the straight-skeleton for each arrangement cell, we obtain a partition of the arrangement cell into skeletal cells, such that every face (or a degenerate edge) of the arrangement is associated with such a skeletal cell, for which the arrangement face is the “base face.” Each skeletal face, other than the base, has a counterpart face belonging to a neighboring skeletal cell.

### 3.4 Projecting Images onto Skeletal Faces

#### Extending Images onto Skeletal Walls

We proceed with projecting the images, lying on a single base face $f$, onto the faces of the skeletal cell induced by $f$. Previous methods that applied a similar approach (e.g., [48]) used orthogonal projection onto the faces of the skeleton. However, this method works well only for an arrangement of full planes, in which
Figure 3.5: A 2D drawing of wall stretching. A simple orthogonal projection creates gaps in the surface although the images are continuous. Stretching the images on the obtuse walls fixes this effect.

the cells of the arrangement are convex, and hence the skeletal cells and faces are also convex (i.e., there are no walls). Consider a case in which the dihedral angle between two arrangement faces $f_1, f_2$ is reflex, resulting in an obtuse skeletal wall between them. In this case, a simple orthogonal projection of the images on both faces creates unnatural gaps in the resulting mesh, even if the images are continuous (see Figure 3.5).

The monotonicity of the three-dimensional skeletal cell guarantees that any ray originating at either the base or the walls, directed orthogonally to the base and toward the inside of the cell, intersects exactly one point of the roof. To solve the gap issue, we extend the image edges that lie on the faces of the base onto the walls, “stretching” them upward until they reach the boundary of the roof. The vertices of the edges stretch across piecewise-linear routes through the faces of the walls, chosen in the following manner: for each face of the wall, if any image vertex is at the bottom of a wall edge, it is stretched along this edge. Otherwise, the vertex is stretched along a steepest ascent line inside this wall face. This stretching method has two appealing properties:

- No two routes cross each other, as all steepest-ascent lines are parallel, because the wall faces are planar. Occasionally, two routes may unite, if they both reach a common wall vertex, which means not all images may be stretched up to the boundary of the roof.

- If every point on the boundary of the face base is on an image edge, then every point on the wall is on an image. In other words, if there are no image gaps on the boundary of the base face, there are no image gaps on the walls.
Figure 3.6: Exemplifying the stretching of images onto walls.
Figure 3.7: The propagation of an image from a floating patch inside a box. Notice how the red material propagates from the degenerate edges, which propagate into faces, and stretches to envelope the entire patch.

It is easy to see that the stretching of the edges propagates as a watertight front, which sweeps the entire wall. The stretching process is exemplified in Figure 3.6. The stretching process can be considered a morphing process from the boundary of the base to the boundary of the roof.

Having stretched the image, we can now project the image orthogonally onto the roof from both the base and the wall faces. When there are no walls in a skeletal cell, which is always the case where arrangement cells are convex, the boundary of the roof is exactly the base boundary, and so we simply project the image orthogonally from the base, thus conforming in this case to the projection method of [48].

### 3.4.1 Projection from Degenerate Edges

As defined in Section 3.3, degenerate edges of the arrangement propagate into faces orthogonal to the faces adjacent to this degenerate edge. Therefore, this edge is a degenerate base face of its skeletal cell. To properly produce the effect of skeletal propagation on the reconstruction, every image edge on a degenerate edge is also considered a degenerate image face on the propagating base. Thus, if every point on a degenerate edge is on an image edge, then every point on the propagating base is also on an image edge, and no new gaps are created. With the no-gap property of the wall-stretching step, and the property of degenerate propagation of images, we conclude that when every point on an arrangement face is on an image, then every point on the respective skeletal cell is on an image, original, stretched, or projected. For such an example, see Figure 3.7.
3.5 Mesh Reconstruction

3.5.1 Image Overlay

At this stage, every skeletal face contains portions of images, either projected or stretched onto it from the original images on the base faces of the arrangement of planes. In fact, every skeletal face contains two sets of such images, because every such face has a twin skeletal face that is shared by a neighboring skeletal cell. We proceed by computing the two-dimensional overlay of these two partitions of the skeletal face. In the overlay, every face is associated with one of the following: two identical colors, two different colors, a color and the null color (“air”), or with the null color on both sides. As in the reconstruction algorithm of [10], we ignore the uniform cases (same color or null color), and compute the constrained Delaunay triangulation of overlay faces originating from images of two different colors (including a color and the null color).

3.5.2 Mesh Creation

We generalize the “spatial lifting-up” method of [10]. Every overlay of the projected images (described in the previous section) is associated with three planes: the overlay plane (the one that supports the skeletal face), denoted as the “middle” plane, and the two planes supporting the bases of the respective skeletal cells (which are also the planes supporting the respective cross-sections), denoted as the “upper” and “lower” planes (arbitrarily selected). The reconstructed mesh connects the images in the upper and lower planes, through the middle plane, within the two respective skeletal cells. The reconstruction of the portion of the mesh induced by the skeletal face has two steps. The first step is the creation of triangles in the overlay, as discussed in the previous section. The second step is the creation of the mesh triangles associated with this overlay, in a similar manner to that of [56]: two back-to-back mesh triangles are created per triangle in the overlay plane (in the case when the overlay is between a color and the air, only one triangle is created), and mesh triangles are erected between portions of original, or stretched images and their projection in the overlay.

After computing the mesh portion for every such overlay in each skeletal face within every skeletal cell, and within every cell of the arrangement of planes, we obtain a full consistently-orientated portion of the mesh in these cells, in which the boundary of every domain (color) is an oriented 2-manifold, and is adjacent,
via back-to-back triangles, to other domains (or to the “air”). By gluing the reconstructed portions of mesh in all the arrangement cells, we obtain the fully reconstructed partition of space into colored domains.

### 3.5.3 Correctness

First, we claim that the boundary of every reconstructed spatial domain is a closed 2-manifold, and no two such boundaries intersect each other. This follows from the fact that every face of a section image, which is the result of intersecting a spatial domain with a plane, is projected onto both sides of the supporting plane of this section. On either side, this image face is overlaying either another domain of the same color, in which case the two are eventually connected, or another domain of a different color, in which case triangles that close up the boundary between the domains are created. Therefore, any domain is closed.

Every reconstructed domain is also a 2-manifold, resulting from the construction of the triangles for every overlay face. Mesh triangles are either on the overlay plane, or between an original edge and its projection, and in any case have only one adjacent triangle of the same triangle of the same color. Within one skeletal cell of the arrangement of planes, there are no intersections between the domains, because of the proper stretching step, and due to the orthogonal projection in the monotone skeletal cell. Since the creation of mesh portions within one cell is completely independent of the reconstruction within other cells, and the fact that the mesh portion lies completely inside the cell, no two reconstructions of mesh portions of two cells interfere with each other.

Second, we argue why no gaps (portions of space associated with the null color) are created inside a cell of the arrangement of planes, whose boundary is completely covered by non-null colors. When all the faces of a cell of the arrangement of planes are covered by images, none of which is associated with the null color, the images are stretched along the obtuse walls so as to fully cover them, as proved in Section 3.4. Therefore, the orthogonal projection always covers the entire cells of the three-dimensional skeleton. Every full overlay (having no null color faces) creates two portions of back-to-back surfaces without gaps between them. Therefore, every skeletal cell and, consequently, every arrangement cell with full images, do not contain any gaps.
3.6 Optional Improvements

3.6.1 Mesh Refinement

Our algorithm produces consistent meshes. However, because of the projective method, and creating a triangulation in the overlay plane, the mesh may exhibit sharp angles and long, skinny triangles, which denote a poor mesh. We follow the coarse-to-fine method established in [56], following the remeshing method of [58], and the mesh smoothing method of [50]. We use smoothing parameters similar to those used in [56].

3.6.2 User-Guided Reconstruction

An inherent shortcoming of all projective reconstruction algorithms is that the correspondence between images on planes is mostly dictated by the topology of the projective planes. This results in the image shape being of concern only on overlays of corresponding planes. In order to alter a reconstruction of full planes, one has to introduce new full cross-sections. Our algorithm has the benefit of working with all settings of piecewise-linear cross-sections. Thus, we allow the user to insert guiding patches of arbitrary topology inside arrangement cells, in order to alter the reconstruction, without having to re-sample a new full cross-section. Figure 3.8 depicts such an intervention.

3.7 Experimental Results

Finally, we provide a few examples of running our algorithm on synthetic and real input.

Figure 3.9(a) shows three floating patches (a section of a “3-knuckle,” consisting of contours of the same color (brown). However, each patch is contained in a region of a different color. Figure 3.9(b) shows the different image overlays. Figures 3.9(c,d) show the reconstruction of one pipe and a 3-tunnel, respectively. Figure 3.9(e) shows the full reconstruction of the object.

Figure 3.10 demonstrates the effect of partial slices. The input is seen in (a): one full slice (at the top), containing a green contour surrounded by a blue region. This pattern is also reflected in two partial slices (halfplanes sharing their borders) at the bottom. In addition, on the two horizontal sides there are two partial slices containing red contours. Although the two sections with the red patches
Figure 3.8: Inserting a patch to aid reconstruction. The two blue stripes are expected to connect, but they do not project onto the same overlay. An aiding blue patch is put in the middle to ensure the continuity of the blue domain.

![Figure 3.8](image.png)

Table 3.1: Statistics of experimental results.

<table>
<thead>
<tr>
<th>Object</th>
<th>Sections</th>
<th>Input Vertices</th>
<th>Arrangement Cells</th>
<th>Overlays</th>
<th>Output Vertices</th>
<th>Output Triangles</th>
<th>Running Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat</td>
<td>13</td>
<td>489</td>
<td>166</td>
<td>1,004</td>
<td>2,194</td>
<td>4,336</td>
<td>21</td>
</tr>
<tr>
<td>Hand</td>
<td>50</td>
<td>1,872</td>
<td>625</td>
<td>2,600</td>
<td>7,911</td>
<td>15,435</td>
<td>68</td>
</tr>
<tr>
<td>Color Example</td>
<td>7</td>
<td>250</td>
<td>21</td>
<td>97</td>
<td>1,997</td>
<td>2,456</td>
<td>23</td>
</tr>
<tr>
<td>Organs</td>
<td>34</td>
<td>9,764</td>
<td>1,682</td>
<td>7,873</td>
<td>19,048</td>
<td>37,148</td>
<td>105</td>
</tr>
</tbody>
</table>

seem to “interfere” with the reconstruction of blue/green domains, this is not the case since the former sections are partial. This is manifested in (b), which shows the interaction of the overlays. The reconstructed domains are shown in (c,d,e). It is clearly seen how the green contours connect to each other.

Figure 3.11 shows the reconstruction of the single-label mesh of a cat, with unknown regions, which have been completed naturally. Figure 3.12 shows similar results for a hand, in which much noise is present in the middle fingers in the input. Figure 3.13, shows an example of a multi-labeled reconstruction with noisy regions, in which both regions were reconstructed in a consistent way, and the noisy gaps were filled in an intuitive manner. Finally, we show in Figure 3.14 the most general reconstruction of multiple internal organs (stomach, spleen, pancreas, and a kidney) with unknown regions, which exhibits a natural solution as well. Table 3.1 provides some statistical data about running our algorithm on these examples.
Figure 3.9: Reconstruction of a “3-knuckle.”
Figure 3.10: Reconstruction with partial sections.

Figure 3.11: Reconstruction of a cat from nonparallel sparse slices with noisy regions. Gray regions indicate the missing information.
Figure 3.12: Reconstruction of a hand.

Figure 3.13: A multi-labeled reconstruction with noisy regions.
Figure 3.14: A simultaneous reconstruction of several internal organs.
3.8 Conclusion

This method was the first to consider the reconstruction of an unknown object (or objects) from cross-sections, with all the following degrees of freedom supported: (a) nonparallel sections; (b) multi-colored contours within sections; and (c) partial sections. We used the straight skeleton of polyhedral objects, with some nontrivial generalizations for degenerate edges, and new special algorithmic components needed to support all the required features simultaneously. Although the mesh interpolates the cross-sections, the resulting surface can still suffer from jagged edges and sometimes poor triangulation, as most algorithms. For such cases, we applied a fairing scheme to the mesh. In addition, we cannot yet give an accurate bound on the theoretical time complexity of the algorithm since the complexity of the three-dimensional straight skeleton of a polyhedron, is still an open question. Furthermore, the complexity of the method grows with the complexity of the unknown regions, which is unwanted behavior.
Chapter 4

A Multi-Resolution Approach to Heat Kernels on Discrete Surfaces

In this chapter, we describe our method to compute the heat kernel on discrete surfaces, which importance and which properties were established in Chapter 1. The input to our problem is a triangulated mesh of \( n \) vertices. We take advantage of the multi-scale property of the heat kernel 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 \( t \ll 1 \) - 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.
4.1 Previous Work

4.1.1 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 dataset (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 [79]. 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 [50], based on a progressive mesh structure [47]. More recent work adapts algebraic multigrid approaches to surface meshes [3], for parameterization [71], and deformation [73].

A completely different approach to multi-resolution in the context of heat diffusion is taken by the concept of Diffusion Wavelets [26]. 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.

4.1.2 Matrix Exponential Approximations

As mentioned above, the HK is related to the matrix exponential of the Laplacian, thus efficient computation of $e^{-tL}$ may be useful in approximating $k_0$. 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 [61] for a thorough survey of existing algorithms.
The matrix exponential is defined as the power series:

\[ e^X \equiv S^T(X) \equiv \sum_{k=0}^{\infty} \frac{1}{k!}X^k \tag{4.1} \]

A naive approach to compute an approximation to \( e^X \) is to truncate the series 4.1 after the first \( M \) terms. This is known to be one of the worst methods for computing the exponential [61], 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:

\[ e^X = U e^D U^{-1} \tag{4.2} \]

Since \( D \) is diagonal, \( (e^D)_{ii} \) is simply \( e^{D_{ii}} \). 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:

\[ f(x) \approx \frac{\sum_{k=0}^{N} a_k x^k}{\sum_{k=0}^{M} b_k x^k} \tag{4.3} \]

The coefficients are computed by requiring that this approximation agrees with the Taylor expansion, up to the highest coefficient. It is well known [4] 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:

\[ e^X = (e^{X/m})^m \tag{4.4} \]

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 \( e^{X/m} \). This algorithm - combining scaling and squaring with the Pade approximation (for \( e^{X/m} \) - is one of the best available, and is implemented in MATLAB’s \texttt{expm} function [61]. 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:

$$\psi(t) = e^{-tX}\psi(0)$$ (4.5)

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 [45], 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 $e^{-tX}$, as this will boil down to computing the entire matrix.

### 4.2 Overview of the Algorithm

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 $k_i$ 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 $t \ll 1$. As the heat kernel is local at this resolution level, the resulting matrix $e^{-tL_i}$ 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.

4.3 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:

$$K^M_t(u, v) = \sum_{i=1}^{n} e^{-\lambda_i t} \varphi_i(u) \varphi_i(v)$$  \hspace{1cm} (4.6)

where $n = |V|$ and $u, v \in V$. If, in addition, $L$ is symmetric, then the eigenvectors are orthogonal, implying $K^M_t = e^{-tL}$.

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. Note that we only use one such iteration, as opposed to the conventional multi-grid algorithm.

**Definition 4.3.1** Given a mesh $M$, and constants $d, C \in \mathbb{Z}$, 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 $M_{i+1}$. 

57
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 \( i+1 \), \( g^{i+1} : V_{i+1} \rightarrow \mathbb{R} \). The prolongation of \( g^{i+1} \) to the level \( i \) is:

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

(4.7)

where \( w_l \) are the barycentric coordinates of \( p \) in \( t_p \) with respect to \( u_l \). We can write 4.7 in matrix notation, using a prolongation matrix \( P_{i+1}^i \):

\[
g^i = P_{i+1}^i g^{i+1}
\]  

(4.8)

By recursively applying 4.8, we can infer the values at the vertices of the finest level from the values on the vertices on any resolution level \( h \):

\[
g^M = g^1 = P_1^1 P_3^2 \ldots P_{h-1}^h g^h = P_h^1 g^h
\]  

(4.9)

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

\[
A^i = P_{i+1}^i A^{i+1} P_{i}^{i+1},
\]  

(4.10)

which is the natural extension of 4.7 to functions of two vertices. Note that if \( A^{i+1} \) were symmetric, then \( A^i \) will be symmetric as well. Finally, we define the multi-resolution (MR) heat kernel on \( M \):

58
Definition 4.3.2 Given \( t > 0 \) and \( \varepsilon > 0 \), let \( r_\varepsilon(t) \) be the numerical rank of \( K^M_t \), i.e. the number of eigenvalues \( \lambda_i \) of the Laplacian matrix, such that \( e^{-t\lambda_i} > \varepsilon \). In addition, let \( h \) be the coarsest resolution level, such that \( cn_h > r_\varepsilon(t) \), for some constant \( 0 < c \leq 1 \). Then the multi-resolution heat kernel on \( M \) is:

\[
\hat{K}^M_t = P^1_h K^h_t P^h_1
\]  

(4.11)

In the special case that \( h = 1 \), we obtain \( \hat{K}^M_t = K^M_t \)

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 \( e^{-t\lambda_i} > \varepsilon \) — 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 \( K^M_t \) is \( r_\varepsilon(t) = r \), we may approximate 4.6 by:

\[
K^M_t(u, v) \approx \sum_{i=1}^{r} \exp(-\lambda_i t) \varphi_i(u) \varphi_i(v)
\]  

(4.12)

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 \( L_h \). Rewriting 4.12 in matrix notation, we get:

\[
K^M_t \approx \tilde{U} \exp(-t\tilde{D})\tilde{U}^T
\]  

(4.13)

Using the same formulation for \( K^h_t \), we have:

\[
\tilde{K}^M_t = P^1_h K^h_t P^h_1 \approx P^1_h \tilde{U}^h \exp(-t\tilde{D}_h)\tilde{U}^T P^h_1
\]  

(4.14)

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 \( \tilde{K}^M_t \approx K^M_t \). 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.
Figure 4.1: (left) The spectrum of different resolutions levels of the gargoyle model, and (right) exponential for different values of $t$.

However, [66] showed that, in the continuous case at least, the heat kernel is resilient to these changes. This leads to the conjecture that even though the eigenvectors and their prolonged versions may not be exactly the same, this effect is canceled 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 4.3.1. We used the “quadric edge collapse” method [41] as implemented in MeshLab [21]. Although we cannot reach the theoretical minimum as defined in Property 3 of Definition 4.3.1, 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 [43], which is based on the algorithm of [76].
Figure 4.1 shows $\lambda^h_i$ and $e^{-t\lambda^h_i}$ as functions of $\frac{i}{n_1}$ 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 spectra are very similar, for the appropriate values of $t$.

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 $\tilde{R}^M_t = \tilde{U} \exp(-0.5t\tilde{D})$ — the first $r$ axes of the diffusion map [55] of $M$, and respectively $\tilde{R}^h_t = \tilde{U}_h \exp(-0.5t\tilde{D}_h)$. $R_t$ is the square root of the heat kernel matrix: $K_t = R_t R^T_t$.

Figure 4.2 shows the effect of $t$ on our approximation. For visualization purposes, we plotted $\tilde{U}$ vs. $P^1_h \tilde{U}_h$, and $\tilde{R}^M_t$ vs. $P^1_h \tilde{R}^h_t$, 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 $e^{-t}$ expression attenuates that, and makes $\tilde{R}^M_t$ very similar to $P^1_h \tilde{R}^h_t$, 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 $t$. Figure 4.3 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 Figure 4.4, this is a reasonable approximation.

Thus, given a range of timescales $[t_{min}, t_{max}]$ for which we would like to com-
Figure 4.3: The HKS and its prolonged version, for $t = 8, 64$. Note how noisy the prolonged version is at $t = 8$.

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 Figure 1.4. 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.

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 $P_h$ 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 $v$ [45] 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:

\[
(HKS_t^M)_{ii} = p_i K_t^h p_i^T ,
\]

(4.15)
where \( p_i \) is the \( i \)-th row of the prolongation matrix \( P_{h^*}^1 \).

4.4 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 \( L \) is still quite problematic, since 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 counterintuitive, 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 \( \varepsilon \), 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 [16, 60, 67, 69, 82], 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 [66]:

$$K^M_t = e^{-tL}A^{-1}$$  \hspace{1cm} (4.16)

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 $K^M_t$ are close to 0.

### 4.4.1 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 4.4.1** Let $K^M_t$ be the heat kernel matrix of mesh $M$ at time $t$. The $\varepsilon$-sparsified heat kernel matrix of $M$ is:

$$SK^M_{t,\varepsilon})_{ij} = \begin{cases} (K^M_t)_{ij} & (K^M_t)_{ij} > \varepsilon \\ 0 & otherwise \end{cases}$$  \hspace{1cm} (4.17)

Now, assume we can efficiently compute $K^M_t$ for a very small $t = t_0$. Then we can compute the heat kernel matrix for $t = 2t_0$, using: $K^M_{2t} = (SK^M_{t,\varepsilon}A)^2A^{-1}$. Note that although, in general, sparse matrix multiplication might result in many non-zero entries, the number of significant entries in $K^M_{2t}$ 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 $2^n t_0$, given the heat kernel at time $t_0 \ll 1$.

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 Figure 4.5 for an example. If some third vertex $w$ satisfies $k_t(u, w) + k_t(v, w) < \varepsilon$, then
Input: $K_{t_0}^M, \varepsilon, s$
Output: $SK_{t_0}^M, t = s^2 t_0$

$K_t^M = K_{t_0}^M, t = 2^s t_0$

for $i = 1$ to $s$
do
$K_{2^i t}^M = (SK_{t/2}^M A)^2 A^{-1}$
t $= 2t$
end for

**Algorithm 1:** Sparse scale and square

Figure 4.5: The geometric interpretation of sparse scale and square. The sum of $k_t(u, \cdot)$ and $k_t(v, \cdot)$ for times $t = 64, 128$ and $512$. Vertices whose sum is $< \varepsilon$ at time $t$ do not influence the computation of $k_{2t}(u, v)$.

$w$ 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 4.6 shows empirical evidence of the accuracy of our sparse “scale and square” approach. We computed $e^{-tL}$ 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.
Figure 4.6: Comparison of the diagonal of $e^{-tL}$, computed using the scale and square approach (left), with the exact result (middle). $\log_{10}$ of the error per row (right).

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.

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

### 4.4.2 Binomial Approximation

The most basic building block of our algorithm is the computation of $K_t$ for $t \ll 1$, by computing $e^{-tL}$. 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 4.4.2** 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:

$$S^B_N(t, L) = \sum_{m=0}^{N} Q_m(L)(\exp(-t) - 1)^m, \quad Q_m(L) = \binom{L}{m} = \frac{1}{m!} \prod_{k=0}^{m-1} (L - kI)$$
Proposition 4.4.3 Let $S^B_N$ be defined as in 4.18, and $t > 0$. Then:

$$S^B(t, L) \triangleq \lim_{N \to \infty} S^B_N(t, L) = \exp(-tL) \tag{4.19}$$

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:

$$s^x = 1^x + x 1^{x-1}(s-1) + \frac{1}{2!} x(x-1) 1^{x-2}(s-1)^2 + ...$$

$$= \sum_{m=0}^{\infty} \binom{x}{m} (s-1)^m, \quad \binom{x}{m} = \frac{x(x-1)...(x-(m-1))}{m!} \tag{4.20}$$

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 $L$. 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 $Wv = \lambda v$. 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:

$$\frac{1}{m!} \prod_{k=0}^{m-1} (L - kI) = V \left( \frac{1}{m!} \prod_{k=0}^{m-1} (D - kI) \right) V^{-1} = V E_m V^{-1} \tag{4.21}$$

where $E_m$ is a diagonal matrix. Plugging 4.21 into 4.18, we have:

$$S^B_N(t, L) \triangleq \sum_{m=0}^{N} (e^{-t} - 1)^m V E_m V^{-1} = V \left( \sum_{m=0}^{N} (e^{-t} - 1)^m E_m \right) V^{-1} = VF_N V^{-1} \tag{4.22}$$
where $F_N$ is a diagonal matrix. Considering a single element in $F_N$:

$$(F_N)_{ii} = \frac{N}{m!} \sum_{m=0}^{N} \frac{1}{m!} \prod_{k=0}^{m-1} (\lambda_i - k)(e^{-t} - 1)^m = \sum_{m=0}^{N} \left( \frac{\lambda_i}{m} \right) (e^{-t} - 1)^m \quad (4.23)$$

where $\lambda_i$ 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 $\lambda_i$, when $|e^{-t} - 1| < 1$. Since $t > 0$, we have $0 < e^{-t} < 1$, hence the convergence condition holds, and we get:

$$\lim_{N \to \infty} (F_N)_{ii} = \lim_{N \to \infty} \sum_{m=0}^{N} \left( \frac{\lambda_i}{m} \right) (e^{-t} - 1)^m = e^{-t\lambda_i} \quad (4.24)$$

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 4.22, we conclude:

$$\lim_{N \to \infty} S_B^N(t, L) = \lim_{N \to \infty} VF_NV^{-1} = Ve^{-tD}V^{-1} = e^{-tL} \quad (4.25)$$

Which proves Proposition 4.4.3.

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 $e^{-tx}$ around $tx = 0$, and as such converges slowly as $tx$ is farther from 0. Instead, we define a new variable $s = e^{-t}$, 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 Figure 4.7, 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 $e^{-tL}$ 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 $s = e^{-t}$. For large $\alpha$, 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 $\|B_N\|_2 < \varepsilon$, using the following bound:
Figure 4.7: Log of the norm of the residual $R_N(t, L) = e^{-tL} - S_N(t, L)$ of the approximation of $e^{-tL}$ with $N$ elements, using the Binomial series $S^B$ and the Taylor series $S^T$. $t \in \{0.25\text{(blue)}, 1\text{(red)}, 4\text{(green)}\}$. $\|L\|_2 = 30.5$.

**Proposition 4.4.4** Let $B_m(t, L)$ be the $m$-th element of the binomial series $S^B$. Then:

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

(4.26)

where $\lambda_{\text{max}}$ is the maximal eigenvalue of $L$, and $a_i = A_{ii}$.

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

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

(4.27)

Plugging $L_1$ into 4.27 instead of $L$ we get:
\[
\frac{1}{m!} \prod_{k=0}^{m-1} (L_1 - kI) = \frac{1}{m!} \prod_{k=0}^{m-1} \left( A^{1/2} (L - kI) A^{-1/2} \right) = A^{1/2} \left( \frac{1}{m!} \prod_{k=0}^{m-1} (L - kI) \right) A^{-1/2}
\]

(4.28)

Thus, we have:

\[
B_m(t, L_1) = A^{1/2} B_m(t, L) A^{-1/2}
\]

(4.29)

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

\[
\left\| \frac{1}{m!} \prod_{k=0}^{m-1} (L_1 - kI) \right\|_2 \leq \frac{1}{m!} \prod_{k=0}^{m-1} \| (L_1 - kI) \|_2 = \frac{1}{m!} \prod_{k=0}^{m-1} (\lambda_{\text{max}} - k) = \left( \frac{\lambda_{\text{max}}}{m} \right)
\]

(4.30)

where \(\lambda_{\text{max}}\) is the largest eigenvalue of \(L_1\) (and also of \(L\)), and we assumed that \(\lambda_{\text{max}} > m\). Going back to \(B_m(t, L)\) using 4.29:

\[
\|B_m(t, L)\| = \| A^{-1/2} B_m(t, L_1) A^{1/2} \| \leq \| A^{-1/2} \| \| B_m(t, L_1) \| \| A^{1/2} \| \quad (4.31)
\]

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 4.30 and 4.31, we finally obtain:

\[
\|B_m(t, L)\|_2 \leq \left( \frac{\lambda_{\text{max}}}{m} \right) (e^{-t} - 1)^{m} \sqrt{\frac{\max(a_i)}{\min(a_i)}}
\]

(4.32)

which proves the proposition.

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 \(\frac{t}{2^s} = t_0 < t_1\), compute \(KS_{t_0}\) using \(S_B\), and scale back the result using Algorithm 1. This procedure is summarized in Algorithm 2.
Figure 4.8 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 $\epsilon = 10^{-6}$, and 15 iterations for the series approximation. The figure shows our HKS—denoted \textit{FastHKS}—compared to the true HKS, computed from the spectral decomposition of the mesh, using the algorithm of [76]—denoted \textit{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 4.1 provides the quantitative comparison for these and other meshes. Note that about 12K eigenvectors were needed in order to compute \textit{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 were 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.

4.5 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 \( cn_h > r_{e}(t) \)
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 4.11.

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 \( d = 2 \).
Figure 4.9: Color coding of FastHKS on different meshes. Comparison with EigHKS for one mesh. The timings and errors are given in Table 4.1.
for all our experiments, with $C$ varying from 6K to 20K vertices. As is seen in Table 4.1, 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 [21], decimating the original mesh using the “Quadric edge-collapse decimation” method [41] 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 $K^M_t$ 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 $\varepsilon$, we use $\varepsilon = 10^{-4}$ for the numerical rank computations, and $\varepsilon = 10^{-6}$ for the sparsification process.

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 $5 \times 5$ blocks. We compared our results to those obtained with the EigHKS algorithm, for a variety of meshes and time scales, summarized in Table 4.1, and visualized in Figure 4.9. 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 $e^{-t\lambda_i} < \varepsilon$, 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 (note that these times are not accounted for in the column “Total FastHKS Time” in Table 4.1).

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.

Our method has some limitations. First, we do not provide theoretical guaran-
<table>
<thead>
<tr>
<th>Model</th>
<th>t</th>
<th>$|V|$ coarse (K)</th>
<th>$|V|$ (K)</th>
<th>Scale Square (sec)</th>
<th>$S^B$ (sec)</th>
<th>Total Fast HKS Time (sec)</th>
<th>RMSE ($\times 10^{-6}$)</th>
<th>$#$ eigens required</th>
<th>Eigens (sec)</th>
<th>HKS given eigens (sec)</th>
<th>Total EigHKS Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buddha</td>
<td>4</td>
<td>112</td>
<td>32</td>
<td>87</td>
<td>6</td>
<td>19</td>
<td>17,971</td>
<td>31,528</td>
<td>252</td>
<td>31</td>
<td>31,780</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>113</td>
<td>21</td>
<td>1.3</td>
<td>72</td>
<td>98</td>
<td>202</td>
<td>3</td>
<td>6</td>
<td>432</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td></td>
<td>32</td>
<td>44</td>
<td>&lt; 1</td>
<td>91</td>
<td>140</td>
<td>242</td>
<td>6</td>
<td>3</td>
<td>432</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td></td>
<td>3.5</td>
<td>20</td>
<td>3</td>
<td>1.4</td>
<td>1000</td>
<td>95</td>
<td>1</td>
<td>96</td>
<td>196</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>3.5</td>
<td>111</td>
<td>31</td>
<td>1.8</td>
<td>113</td>
<td>1000</td>
<td>92</td>
<td>1</td>
<td>96</td>
<td>196</td>
</tr>
<tr>
<td>Chinese lion</td>
<td>153</td>
<td>133</td>
<td>42</td>
<td>95</td>
<td>19</td>
<td>14</td>
<td>313</td>
<td>202</td>
<td>31</td>
<td>406</td>
<td>62,892</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>122</td>
<td>30</td>
<td>95</td>
<td>19</td>
<td>114</td>
<td>21</td>
<td>17,901</td>
<td>32,405</td>
<td>287</td>
<td>32,692</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>112</td>
<td>10</td>
<td>31</td>
<td>5</td>
<td>76</td>
<td>98</td>
<td>661</td>
<td>1,470</td>
<td>10</td>
<td>1,480</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td>18</td>
<td>204</td>
<td>?</td>
<td>576</td>
<td>75</td>
<td>314</td>
<td>603</td>
<td>7</td>
<td>610</td>
</tr>
<tr>
<td>Armadillo</td>
<td>4</td>
<td>122</td>
<td>112</td>
<td>129</td>
<td>32</td>
<td>161</td>
<td>11</td>
<td>204</td>
<td>13</td>
<td>75</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td></td>
<td>16</td>
<td>81</td>
<td>?</td>
<td>139</td>
<td>71</td>
<td>682</td>
<td>1,262</td>
<td>15</td>
<td>1,277</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td></td>
<td>30</td>
<td>131</td>
<td>19</td>
<td>179</td>
<td>61</td>
<td>2,593</td>
<td>4,986</td>
<td>46</td>
<td>5,032</td>
</tr>
<tr>
<td>Garg</td>
<td>3</td>
<td>122</td>
<td>122</td>
<td>129</td>
<td>32</td>
<td>161</td>
<td>11</td>
<td>204</td>
<td>13</td>
<td>75</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td></td>
<td>16</td>
<td>36</td>
<td>?</td>
<td>139</td>
<td>71</td>
<td>682</td>
<td>1,262</td>
<td>15</td>
<td>1,277</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
<td>16</td>
<td>81</td>
<td>?</td>
<td>139</td>
<td>71</td>
<td>682</td>
<td>1,262</td>
<td>15</td>
<td>1,277</td>
</tr>
<tr>
<td>Fish</td>
<td>3</td>
<td>122</td>
<td>100</td>
<td>85</td>
<td>14</td>
<td>110</td>
<td>16</td>
<td>10,068</td>
<td>29,723</td>
<td>235</td>
<td>29,958</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>100</td>
<td>14</td>
<td>&lt; 1</td>
<td>32</td>
<td>150</td>
<td>861</td>
<td>1,594</td>
<td>13</td>
<td>1,607</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td></td>
<td>5</td>
<td>75</td>
<td>&lt; 1</td>
<td>115</td>
<td>114</td>
<td>219</td>
<td>2</td>
<td>2</td>
<td>221</td>
</tr>
<tr>
<td>Horse</td>
<td>3</td>
<td>122</td>
<td>100</td>
<td>85</td>
<td>14</td>
<td>110</td>
<td>16</td>
<td>10,068</td>
<td>9,824</td>
<td>196</td>
<td>10,020</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td></td>
<td>100</td>
<td>14</td>
<td>&lt; 1</td>
<td>32</td>
<td>150</td>
<td>861</td>
<td>1,594</td>
<td>13</td>
<td>1,607</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td></td>
<td>6</td>
<td>32</td>
<td>&lt; 1</td>
<td>71</td>
<td>75</td>
<td>278</td>
<td>286</td>
<td>6</td>
<td>292</td>
</tr>
</tbody>
</table>

Table 4.1: 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.
Figure 4.10: Improving the meshing quality improves the convergence of the Binomial series (and the Taylor series).

teers 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.

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 4.4.4 depends on the maximal eigenvalue of the Laplacian operator, which in turn depends on the meshing quality. Thus, if we repeat the experiment of Figure 4.7 on a badly meshed model, more iterations would be required for convergence (although still less than required by the Taylor series). Figure 4.10 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 4.7. 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 4.10.

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 [66]. Let $T = \{t_1, \ldots, t_k\}$ be the set of interesting time scales, and let $HKST$ be the corresponding set of heat kernel signa-
tures. Two points \( u \) and \( v \) on the surface are considered similar if 
\[
d_{HK}(u,v) = |HK_S_T(u) - HK_S_T(v)|_2^2
\]
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 \([10, \ldots , 256]\), whereas \( T_2 \) also includes smaller times, and is sampled from \([6, \ldots , 256]\). \( K_{T_1} \) can be computed using the eigendecomposition, but \( K_{T_1} \) cannot. Figure 4.11 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”. Figure 4.12 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.

4.6 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.
Figure 4.11: 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 4.12: Separating features from background by clustering the HKS with $k$-means.
Chapter 5

Conclusions and Future Work

5.1 Summary

We have shown three techniques that address problems in reconstruction and morphing of objects. The three-dimensional straight skeleton aids in the creation of a correspondence between linear objects, such as planes, and so the geometric algorithm for reconstruction of multi-labeled cross-sections was directly developed from it. The heat kernel algorithm allows for the fast and robust computation of the heat diffusion on surfaces, and thus allowing algorithms that require this metric information to be used in a more precise manner.

5.2 The Straight Skeleton

We have conjectured in Chapter 2 that the skeleton of a three-dimensional polyhedron has a superquadratic complexity, and that the algorithm to compute it is supercubic. For small inputs, that is reasonable, but it does not scale up well. Also, the algorithm relies heavily on the exact evaluation of predicates, which are computationally expensive. However, since the bulk of the algorithm is the tracking of events, which happen in discrete points in space, this algorithm can greatly benefit from space-partitioning methods, which add a considerable number of events, albeit making the search for events more local, and thus more efficient for complex polyhedra.
5.3 Multi-labeled Reconstruction

As far as we know, our algorithm was the first to offer any treatment of partial information on planes. However, the usage of the straight skeleton makes it cumbersome to compute the interpolation, when the shape of the noise is intricate. Furthermore, the algorithm relies on projections and tiling, which are sensitive numerically. The great advantage of using such a geometric method is the ability to construct the mesh with a minimal space complexity. In an upcoming work [20], an implicit approach is investigated in the same level of generality as we do in this work, and considering the interesting case of on-line reconstruction, where not all slices are given in advance. There are two lines of research we wish to pursue further: The ability to create a reconstruction which preserves the details of the original slices as much as possible (for instance, reproduce rigid motions along the height axis for parallel slices), and the study of what constitutes as a “good” reconstruction solution, in terms of a sampling theory for slices. A topological sampling theory was offered in [2], and a geometric one is still an open question.

5.4 Heat Kernels

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. We would also like to study the possibility to efficiently compute other desirable quantities on the surface, e.g., the discrete Green’s function, or the biharmonic resistance distance.
Bibliography


84

[19] G. Barequet and A. Vaxman. Reconstruction of Multi-Label Do-
 mains from Partial Planar Cross-Sections. Comput. Graph. Forum, 

[20] A. Bermano, A. Vaxman, and C. Gotsman. Online Reconstruction of 
3D Objects from Arbitrary Cross-Sections. ACM Trans. on Graphics. Accepted for Publication.

3D mesh processing system. ERCIM News, 73, 2008.

contour surfaces. Proc. 5th ACM Symp. on Solid Modeling and Appli-


reconstruction of smooth terrain models. Proc. IEEE Visualization, 27–


[27] Y.K. Choi and K.H. Park. A heuristic triangulation algorithm for mul-
tiple planar contours using an extended double branching procedure. 

[28] G. Cong and B. Parvin. Robust and efficient surface reconstruction 

85


בפרק 4: איך מראים להשתמש באנו על זו בתכונה - בצורה החום גרעין את לחשב מנת אפקטיבית קטניםغازיםעבור ישירות החום גרעין את מחשבים אנו עליה המקומית בסביבה מוגדר הוא חדש אלגוריתם באמצעות; אנו יותר גדולים ל지요 החום גרעין את מחשבים המשטח של מוגסת גרסת, زאת בגרסה מקומית יהיה החום גרעין כאשר המשטח של.

_poducts, הרוחות ומיקום ברוס ושתהשכון. אושר ורגית זוחל ממקומי בודרנס זאת עם, זוחל라고 מעבריה שלל היסר של.

IV תגרות ותת-זחלות: זוחל דירוג抑え, האיגריפס שלושה ויד הרוחות שמתחרזות על חבטה

xFFF4

_Description: יגגדל שני והתשובה שלשת ופרטיה של האיגריפס יושר וחלש. ענוי, שאת סיבוכיות תנועה של משל שיש,ーズקבוקה סיבוכיות הזן של האיגריפס ישארת פחתה. העולג בפרק 2 את השמדחט סיבוכיות ה规章制度 של משל שיש א-רביעית (superquadratic).


היתרים את חיתכים על צלו לשישל.

שים דירוגים: למפגש ודרוג, האיגריפטים שלושה והרוחות שלושה חבטה שמתחרזות על חבטה

 Php, אם ה, שאתים משלי שיש, צ祼 אייקון זירה של סיבוכיות (tilings) של פיתוח. בפרט, האיגריפטים מתפקד על המפת הדירוגים ירום ציון זירה. המורך מהי יזכר את ששתיאורית הזן של-פומבקס (supercubic) של פיתוח האיגריפטים משפונר, איגריפיט החלמה פיתוח ההסברה של קיים קיים.

 Php, בועות השיחות שמתחרזות בתוכנה, פיתוח וההתعاطف בואות, (online reconstruction) של קיים קיים, חיתכים של פיתוח האיגריפטים משפונר, איגריפיט החלמה פיתוח ההסברה של קיים קיים.

 Php, חיתכים של בואות של יד תמרון וLiveData של חיתכים, (Rigidity motions) של פיתוח האיגריפטים משפונר, איגריפיט החלמה פיתוח ההסברה של קיים קיים.

 Php, חיתכים של בא_proxy), וחבר של חיתכים בטיה של תענוג, מערער על האורנים זnage של החיתכים.

 Php, חיתכים של בא_proxy), וחבר של חיתכים בטיה של תענוג, מערער על האורנים זnage של החיתכים.

 Php, חיתכים של בא_proxy), וחבר של חיתכים בטיה של תענוג, מערער על האורנים זnage של החיתכים.

 Php, חיתכים של בא_proxy), וחבר של חיתכים בטיה של תענוג, מערער על האורנים זnage של החיתכים.

 Php, חיתכים של בא_proxy), וחבר של חיתכים בטיה של תענוג, מערער על האורנים زmage של החיתכים.

 Php, חיתכים של בא_proxy), וחבר של חיתכים בטיה של תענוג, מערער על האורנים זmage של החיתכים.
The dissertation is handled in a manner that is adapted to the specific requirements of a PhD candidate. It begins with a detailed introduction, followed by a comprehensive literature review. The main body of the dissertation is divided into several chapters, each focusing on a specific aspect of the research. The dissertation concludes with a detailed discussion of the findings and their implications, as well as suggestions for future research.

The dissertation is written in a clear and concise style, with a strong emphasis on the use of formal mathematical notation where appropriate. The dissertation also includes a number of appendices, which provide additional supporting material for the main body of the work.

The dissertation is an important contribution to the field of computer science, and it is hoped that it will be of value to researchers and practitioners alike.
We have shown that each component in the graph of the function \( f \) corresponds to a unique pair of points \((x, f(x))\) in the plane. This correspondence allows us to define a new function \( g \) that maps each point \((x, f(x))\) to its nearest neighbor \((y, f(y))\) in the graph of \( f \). This process is repeated until no further improvements can be made. We have also shown that the resulting function \( g \) is a lower bound for the true function \( f \). In other words, for any point \((x, f(x))\) in the graph of \( f \), the value of \( g(x) \) is less than or equal to \( f(x) \).

Furthermore, we have developed an algorithm that efficiently computes the function \( g \) for a given function \( f \). The algorithm involves iteratively applying a mapping rule to the graph of \( f \), and it converges to the desired function \( g \) in a finite number of steps.

Finally, we have used the constructed function \( g \) to solve a real-world problem, demonstrating its practical applications. The results obtained from using \( g \) as a surrogate model for \( f \) were compared with those obtained from using \( f \) directly, and we found that \( g \) provided significantly better performance in terms of accuracy and computational efficiency.
 setwor, תוגאות המדגימות ושליגה זוורז נשים בצורות של תיאום. ברווחים הנוקשים של הכונן בנגזרם ותיאום מרחבصارו. יוצרו样板ו השגיאuden, אובייקטים של צורה הנושאים ושינויים בעולמית המרכזיות. בתי שגרה התיות נחור רק רבד -כל בחזרה של מציאת האפשרות בשתי צורות. זירות וצורות בתוך סמךיה פנימיות של צורות.

בעובדות ואמגת החולים מתווך לדוגר בתייתם. ראישה, לק מפתיה ממנה חל

מצודת עליר, הנפטרה מצאיה התאמה בין מישורי הלקחים שערורייתם, על בסיס מפרים זה

אוא מפתיה אנגלית שערוריית לדוגר בתייתם, על מציאת מפתיה המרכזיות. לאחר הכן, בוא

מציאת אנגלית השבירה יוצל "גראים ההשכ". אשר משיכי השמיע בל רימ לאתחכית.

בהיות מתחא אוכלי חתכים מתוחות על צורות.

נתאר כתות האתחרים שמעדו לפנינו וואת הפתרונות שציבור

бро מחכים

setwor, אובייקט תהלת-מייסי מאוף של התוכן של התוכן, והא写字楼 תיאום העורף


בחורים וצורות בעיך תהלתית רפואית, בד מפרים CT/MRI שארקנס אובייקט פנימי

ומייצרים התוכנים שלחה. הבועה קרית גמ בחרום הגאולוגית, הנדסה הפוכה, והтекנו

במאגרות מתחכית.

רbservable הפתרון שציבור בみたいה ה- קולט והאוסף של התוכן מיקודליים של

אובייקט אחד בלבד [7, 9, 10, 12, 15, 18, 22, 39, 65, 11, 17, 32, 56, 68].

בלד-חולות (לא נוחצים). מעש עבוריית אלע מתורנות עבר התוכנים כולם, בכרון כולם.

במלד המבנדת של התוכמות [1]
תודה

תודה 먹קר ונאשהתה תחת הדרכתו של פרופ' גל ברקח בפקולטה
למדעי המחשב.

אני מודה לственный לmlin ג'ייקובס-קורהלקום על התמכחות
הכפيفة הנדרשת בשטחשלموت.
שינטואת כלאיות בראונשפרוליציה, שחזור והשוני

צורה של משטחים פוליגאדרליים

היבחר על מחקר

לשם מילוי חלקי של תדרישות להבנת חומר
ודקצור להפרוטופיה

אפרים רכמס

הוות לשנ STUDENT – מרכז טכנולוגיה לישראל

הופה – אדר א', התש"ע א'

פברואר 2011
שיטות כלילות באנטרפולייזה, שחרור והשניה
צורות של מפתות פוליגרדרליים

אמיר חסם