공학박사학위논문

Hausdorff Distance Computation Between Triangle Mesh and Quad Mesh

삼각형 메시와 사각형 메시 사이의 하우스도르프 거리 계산

2018년 8월

서울대학교 대학원

전기·컴퓨터공학부

강 윤 구

Hausdorff Distance Computation Between Triangle Mesh and Quad Mesh

삼각형 메시와 사각형 메시 사이의 하우스도르프 거리 계산

지도교수 김 명 수

이 논문을 공학박사 학위논문으로 제출함

2018년 4월

서울대학교 대학원

전기·컴퓨터공학부

강 윤 구

강윤구의 공학박사 학위논문을 인준함

2018년 6월

위 원 장 신 영 길 (인)

부위원장 김 명 수 (인)

위 원 이 제 희 (인)

위 원 경 민 호 (인)

위 원 윤 승 현 (인)

Abstract

We present various geometric algorithms to facilitate the employment of semi-regular quad meshes. First, a new bounding volume hierarchy—data structure that is used to accelerate geometric operations—is introduced, designed specifically for quad meshes. With this, we can execute rapid point projection, i.e., minimum distance computation from a point to a quad mesh. Point projection is a crucial operation, a main component of our novel algorithms for Hausdorff distance computation and quad mesh reconstruction.

The Hausdorff distance between a triangle mesh and its quad mesh approximation is a measure that can evaluate the quality of the approximation. But computing the Hausdorff distance, which can also be used for shape matching, has always been an extremely difficult problem for mesh models and was never studied specifically for quad meshes. Our algorithm iteratively narrows down where the Hausdorff distance occurs and simultaneously the lower and upper bounds for the Hausdorff distance, by repeatedly partitioning the query model and discarding irrelevant parts. Point projection is heavily involved in updating the lower bound and identifying the inconsequential model parts. We achieve interactive speed and high accuracy that is only capped by the machine's precision. This is true even for the case where the Hausdorff distance is near zero, which is usually the desired outcome of the approximation or shape matching and also, unfortunately, the most challenging configuration

to handle.

In addition, we provide an interface that can assist in creating quad meshes. It converts an existing mesh model or a point cloud into a quad mesh by first letting the user sketch a desired quad layout. Then our algorithm automatically assigns the input data to each patch in the layout, which is then approximated by the quad mesh that is also generated automatically. Point projection is again a large component of the iterative refinement of the approximation. This quad mesh can be evaluated by the Hausdorff distance calculated by our proposed algorithm.

Keywords: Hausdorff Distance, Quad Mesh, Reconstruction, Bounding Volume Hierarchy, Point Projection, Minimum Distance Computation, Collision Detection

Student Number: 2011-20779

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In de facto standard for manufacturing, geometry is represented in analytic form, e.g., spline-based surfaces [29]. It comes with helpful properties such as continuity and compact representation among others. Being parameterized, it can be evaluated at any parameters, meaning unlimited resolution. But performing geometric operations is complicated, spawning an entire field that is computer-aided geometric design. For instance, Kim et al. have developed various geometric algorithms for freeform surfaces [46, 48, 49]. These surfaces' smoothness is sometimes also a drawback because creating sharp features is not as straightforward a task as it is for discrete representation [13, 30].

On the other hand, the triangle mesh, often used for rendering purposes, is perhaps the simplest form of 3-dimensional surface representation. Certain

designers may find it more intuitive, as it is edited directly via the vertices, whereas NURBS surfaces are controlled indirectly through control points defining the surface, causing some frustration when the resulting surface is not as expected. GPUs have evolved to rapidly handle a vast number of triangles, vertices, pixels, etc. in parallel. However, simple doesn't always mean efficient. Even though triangle mesh processing has been studied extensively [33, 63], the unorganized nature of the triangles is and has long been a huge obstacle in developing efficient geometric algorithms regarding triangle meshes.

Meanwhile, the quadrilateral mesh, or quad mesh, can be said to combine the advantages of both representations. As a form of discrete representation, from a designer's point of view, editing is simple, and its structural regularity makes it much preferable to the triangle mesh, able to better represent or preserve principal curvature and feature curves. The structure can be exploited to efficiently create bounding volume hierarchies [48], data structure used to accelerate many geometric operations. The regularity can also be regarded as a form of parameterization, which further means that it can easily be converted into parameterized surfaces. In finite element analysis, hexahedral meshes, whose boundary is a quad mesh, are preferable to tetrahedral meshes [12, 25].

That is not to say that these properties have been fully utilized. In this thesis, to promote modeling with quad meshes, we will exploit this regularity, like with freeform surfaces, to perform and accelerate various geometric algorithms on quad meshes.

Of course, to be precise, only the semi-regular quad mesh, as will be discussed in Chapter 2, fully satisfies these properties. It is a class of quad mesh

that is well-structured, but allows some irregularities at the same time. As such, for the majority of this thesis, we will say "quad mesh," when we are in fact mostly referring to semi-regular ones.

1.2 Research Objectives and Contributions

Calculating distances from a point to a model, or between models, has been one of the most fundamental operations in geometric modeling. Even in static settings, one would often wish to measure distances for many reasons. Finding the footpoint—where the minimum distance happens—is also frequently needed. The same can be said for the detection of collisions between models or self-collisions within a model itself. Their existence is checked, so that they can be removed to obtain natural, collision-free models. And then their fast resolution become more important in dynamic situations including physics

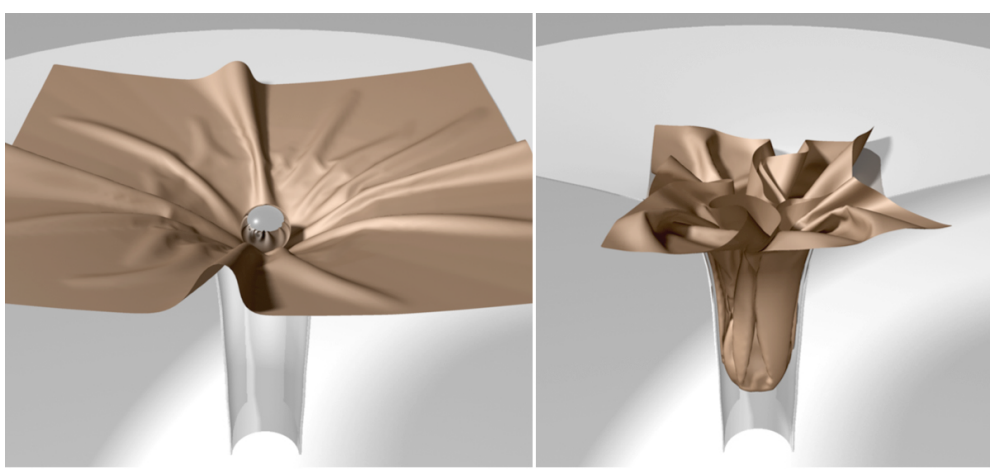


Figure 1.1: Triangle mesh cloth simulation involving a funnel—part of UNC's Dynamic Scene Benchmarks collection.

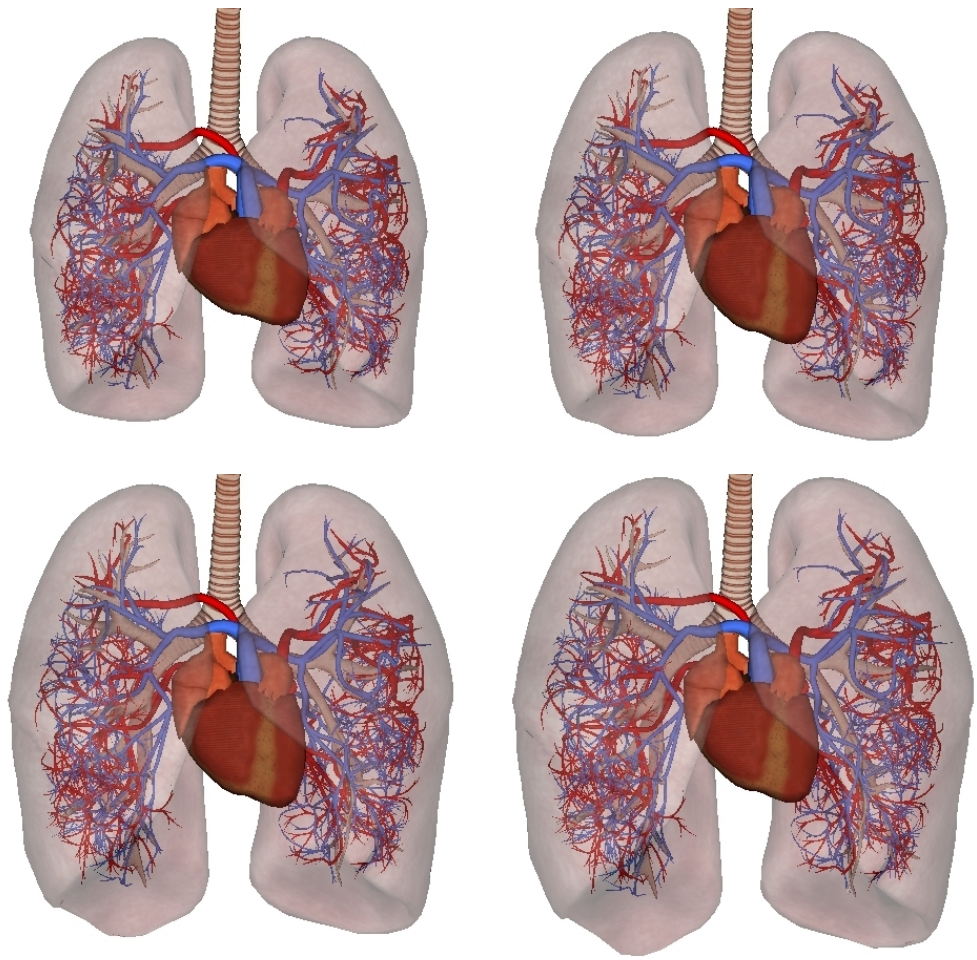


Figure 1.2: Collision-free simulation of the respiratory cycle of the human lung system modeled with Bézier surfaces and sweep surfaces.

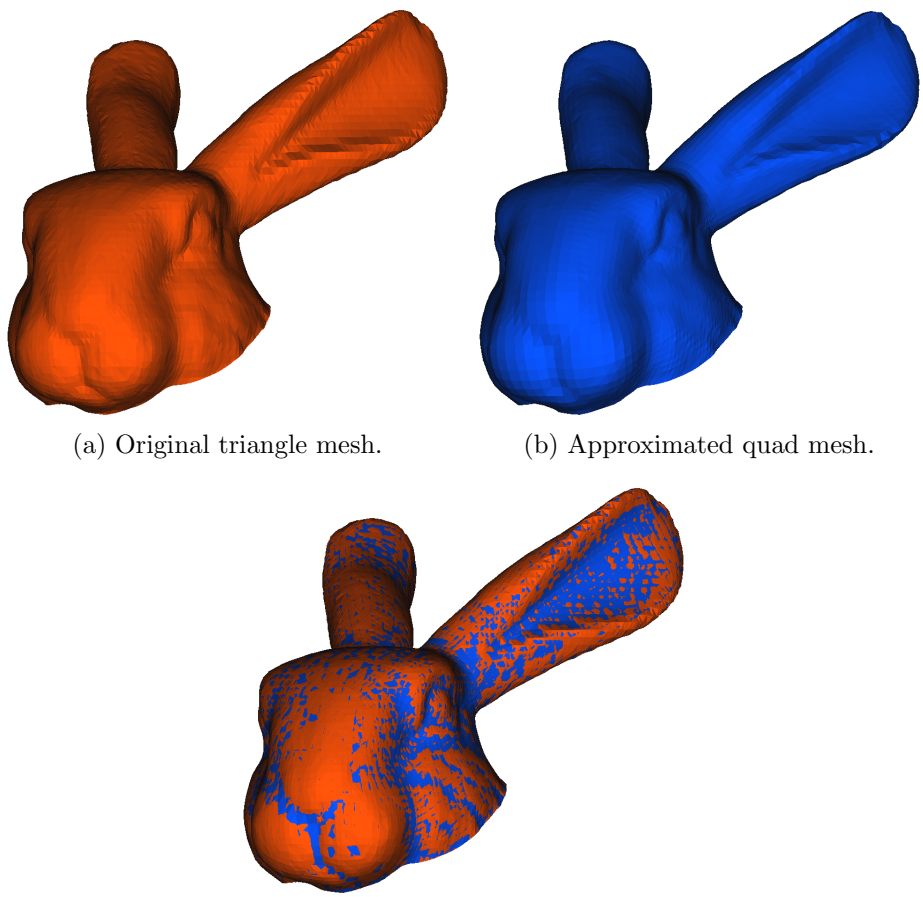
simulation and realistic animation, where interactive-time or even real-time speed may be desired.

Over the years, these problems have been thoroughly studied for models such as triangle meshes and analytic surfaces. Numerous methods have been proposed to speed up the process. But there has been little development on

geometric operations specialized for quad meshes. We aim to utilize the parametric properties to create robust geometric algorithms for the quad mesh that are faster than those for the triangle mesh. We can exploit the structural regularity to create a bounding volume hierarchy whose structure remains unchanged under deformation of the quad mesh; only the shape of each volume is updated to properly contain the assigned mesh part within it.

On a different note, approximating geometric models with quad meshes, due to their benefits, has become a widely researched subject, as surveyed by Bommes et al. [17]. The Hausdorff distance is the measure most often used to determine the approximation quality of such quad meshes.

The Hausdorff distance (HD) between two geometric models is an interesting measure with various application possibilities. It becomes zero only when the two models are identical and positioned exactly the same, which implies that the HD can be used for shape matching and as an approximation error [4]. However, the computation itself is a challenging matter, especially when the HD is near zero, as eliminating parts of the models that do not contribute to the HD becomes harder. Unfortunately, near-zero cases are when HD computation is most relevant, as one would aim to minimize the HD when creating an approximation or finding a match. Such cases also require continuously growing number of iterations and memory space, threatening the proper execution of HD computation algorithms. Previous computation algorithms for HD between mesh models [8, 11, 24, 38, 64] fail in that regard, with even the most recent results occasionally ending up crashing in near-zero cases.



(a) Original triangle mesh.

(b) Approximated quad mesh.

(c) Superposed.

Figure 1.3: An example of triangle mesh approximation by a quad mesh. Measuring the small Hausdorff distance is a way to assess the quality of the approximation.

For NURBS surfaces, the algorithm of Kim et al. [49] is one example that successfully computes near-zero HD in real time. But the situation is much more complex for mesh models, which are unlike smooth NURBS surfaces. This may be surprising, since they are the simplest form of surface representation. Consider offsetting one of the input models in the surface normal direction by a continuously increasing radius. When the radius reaches the HD is the point where the offset first envelops the other model. But the discreteness of meshes leads to complex offset representation; vertices become spheres and edges become cylinders. The increased number of combinatorial possibilities complicates the analysis of the offset's relationship with the models.

This hinders HD computation for mesh models, but we are able to streamline the candidate sifting process and present a fast and robust algorithm for the computation of the two-sided HD between a triangle mesh and a quad mesh, guaranteed within a very small error bound, even for near-zero HD cases. Experimental results show that this error bound can be arbitrary, up to machine precision without any significant performance drawback.

Lastly, compared to general triangle meshes, creation of quad mesh that is semi-regular is not as straightforward. Determining a valid yet pleasing quad layout for the model that faithfully depicts its features can be difficult, as it is something that can only be done manually for satisfying results. Therefore, we present a framework to easily convert an existing model, possibly a point cloud, into a quad mesh via user input. It first aids in designing an appropriate quad layout and then automatically generates an accurate quad mesh approximation.

Our main contributions can be summarized as follows:

- We introduce a simple bounding volume hierarchy (BVH) data structure specifically designed for the quad mesh that allows us to perform fast geometric algorithms over quad meshes. It can handle deformation efficiently, as the hierarchy itself is fixed and does not need restructuring, only requiring updating of each bounding volume's size, which is calculated in an instant.

- Employing the BVH, we have developed fast geometric algorithms regarding quad meshes, including minimum distance and Hausdorff distance computation. Hausdorff distance computation has been a rarely explored matter, especially less for cases involving mesh models, and we provide a quick measurement algorithm, returning the Hausdorff distance that is guaranteed to fall within a very small error bound. Point projection that is accelerated by the BVH and the uniform grid greatly helps this algorithm to achieve interactive speed. The Hausdorff distance between triangle meshes and quad meshes can be used to evaluate quad mesh approximations of triangle meshes.

- To facilitate the creation of quad meshes, we have developed a framework that can convert a given triangle mesh or a point cloud into a quad mesh. After the user sketches a quad layout over the model with the tools we offer for assistance, our algorithm assigns input points to each quad patch of the layout, and then automatically generates the quad mesh approximating the input data, by gradually refining a single quad

that commences each patch's approximation. Fast point projection is also a significant factor in this process. Its approximation error can be calculated with our own Hausdorff distance computation algorithm.

1.3 Thesis Organization

In this chapter, we have stated our main motivation and objectives. Chapter 2 will overview some basics of the quad mesh, and Chapter 3 reviews previous work on Hausdorff distance computation and quad mesh reconstruction. In Chapter 4, we introduce our bounding volume hierarchy for the quad mesh, and how it, along with the uniform grid, can be used to perform rapid point projection. Collision detection between two quad meshes is also addressed for good measure. Hausdorff distance computation will be discussed extensively in Chapter 5, and Chapter 6 describes our framework for creating quad meshes from an existing model. Lastly, in Chapter 7, we summarize this thesis.

Chapter 2

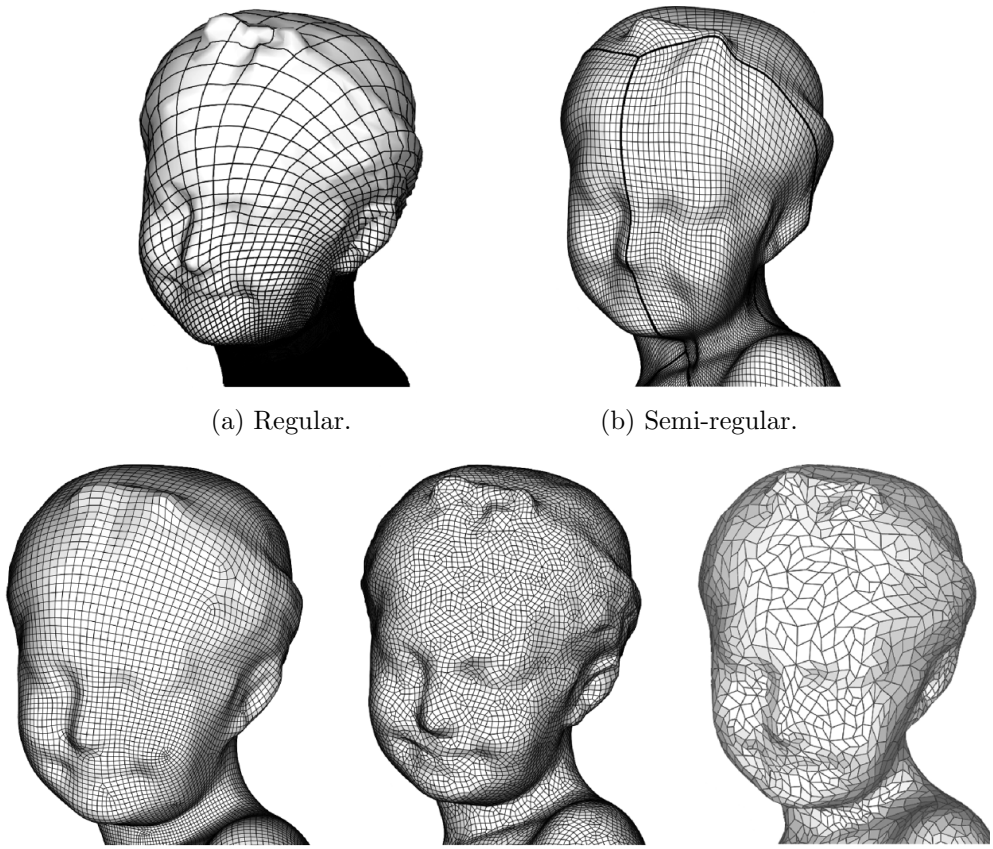
Preliminaries

2.1 Quad Mesh

A polygonal mesh consists of facets. A *facet* is a polygon bounded by edges, and an *edge*, in turn, is a straight line segment connecting two vertices. A *vertex* is a point—in $\mathbb{R}^3$, in this thesis. A quadrilateral mesh, or a *quad mesh*, is a polygonal mesh whose facets are all quadrilaterals, bounded by four edges.

In their survey on quad mesh reconstruction, Bommes et al. [17] classify quad meshes into four categories, based on regularity (Figure 2.1). An internal vertex of a quad mesh is *regular* if its valence is 4; in other words, four edges are connected to the vertex.

- A quad mesh is *regular* if all of its internal vertices are regular.
- If a quad mesh can be partitioned into several regular quad submeshes, it is *semi-regular*, and each partition is called a *patch*.



(a) Regular. (b) Semi-regular.

(c) Valence semi-regular. (d) Unstructured. (e) Unstructured.

Figure 2.1: The four categories of quad mesh.

- A *valence semi-regular* quad mesh, like semi-regular ones, has few irregular vertices, but it cannot be partitioned into regular patches without fragmenting too much. It is hard to travel from one irregular point to another without "making turns."

- A quad mesh is *unstructured* if too many of its vertices are irregular.

While at first glance the regular quad mesh may seem ideal, it generally cannot represent models without showing distortion in area distribution, principal curvature, etc. The semi-regular quad mesh is the best suited, as patch boundaries can be placed along feature curves. Valence semi-regular and unstructured ones are obviously not of much use. Thus, this thesis will focus on semi-regular quad meshes. Most of the times, by "quad mesh," we mean semi-regular quad mesh.

2.2 Interpretation of Quadrilateral Facets

Since we are dealing in $\mathbb{R}^3$, a quadrilateral is in general non-planar, and is therefore, strictly speaking, a *skew quadrilateral* whose interior surface is up to interpretation. One popular solution in graphics is to split the quad into two triangles, by adding a diagonal edge. As there are two diagonals, this is once again a matter of choice, possible criteria for which include diagonal length and dihedral angle. As long as a split is given for each quad, our algorithms can handle them.

One may also consider each quad to be a bilinear surface interpolating its four vertices (Figure 2.2). For generalization, we cover both forms of interpretation with differences in algorithm details. The bilinear surface defined by four vertices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$ can be expressed as:

$$B(u,v) = \mathbf{a}(1-u)v + \mathbf{b}(1-u)(1-v) + \mathbf{c}u(1-v) + \mathbf{d}uv \qquad (2.1)$$

where $0 \leq u, v \leq 1$, and $\mathbf{a} = B(0,1)$, $\mathbf{b} = B(0,0)$, $\mathbf{c} = B(1,0)$, and $\mathbf{d} = B(1,1)$.

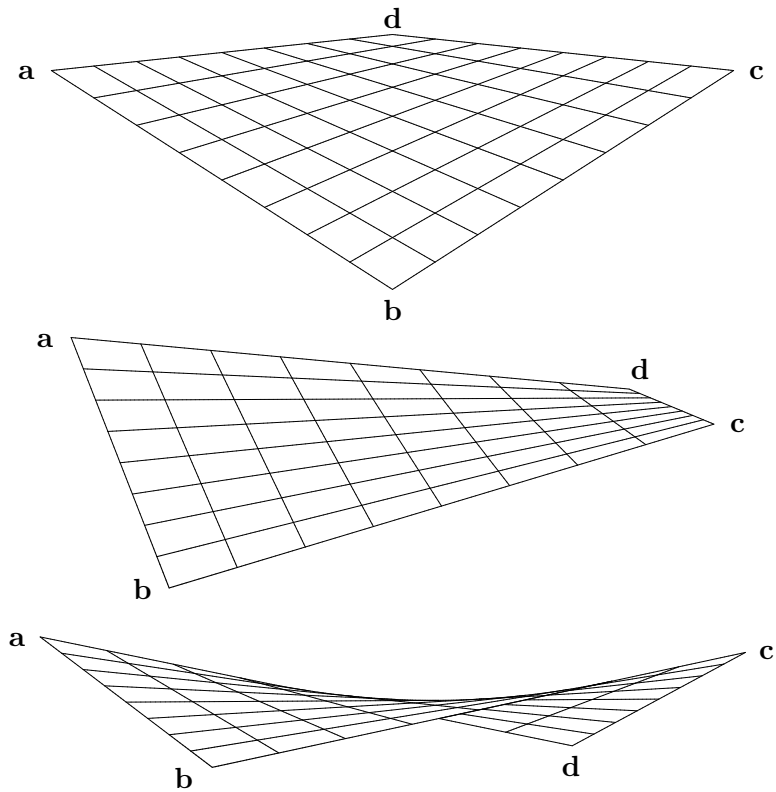


Figure 2.2: A bilinear interpretation of a quad viewed from three directions.

Finding the minimum distance from a point to the bilinear surface and the corresponding footpoint becomes an important component in various algorithms starting with point-to-quad mesh minimum distance computation. For query point $\mathbf{p}$, we wish to find the parameters (u_f, v_f) for the surface that satisfy the two conditions:

$$\begin{aligned} f_1(u,v) &= (\mathbf{p} - B) \cdot \frac{\partial B}{\partial u} = 0 \\ f_2(u,v) &= (\mathbf{p} - B) \cdot \frac{\partial B}{\partial v} = 0. \end{aligned} \tag{2.2}$$

This guarantees that the line connecting $\mathbf{p}$ and its footpoint $B(u_f, v_f)$ is perpendicular to the surface's tangent plane at the footpoint, assuming $B(u_f, v_f)$ is an interior point, i.e., $0 \leq u_f, v_f \leq 1$. But f_1 is a function with a uv^2 term, and f_2 with a u^2v term. Eliminating a variable results in a ninth degree equation. Instead of solving the high-degree equation, we solve the system numerically using Newton's method. Initial values are chosen by projecting $\mathbf{p}$ to the plane that passes through the four midpoints of the edges. Most of the times, it takes six or less iterations to minimize f_1 and f_2 within 10^{-4} of $\|-\mathbf{b}+\mathbf{c}-\mathbf{a}+\mathbf{d}\|^2$ and $\|-\mathbf{b}-\mathbf{c}+\mathbf{a}+\mathbf{d}\|^2$, respectively. If u_f or v_f falls outside the range $[0, 1]$, the footpoint to the quad—as opposed to the expanded bilinear surface—should be found among the four edges, or boundary line segments. While the iteration may oscillate and fail to converge to a solution, such reported cases usually do not occur when the quad mesh is of quality.

Chapter 3

Related Work

3.1 Hausdorff Distance Computation

Past research on Hausdorff distance (HD) computation that does not merely involve point sets is in general easily traceable, likely due to its difficulty. It was first studied for the case of planar polygons with a linear-time algorithm by Atallah [9], albeit restricted to non-overlapping, convex cases. Planar polygons and polygonal curves were further studied by Alt et al. [2], with an $O(n \log n)$ time complexity algorithm. Godau, Alt et al. [3, 34] have also explored HD computation for n-dimensional simplices. HD computation for planar algebraic curves has been tackled by at least four different groups [5, 6, 10, 43, 47]. Jüttler [43] shows a way to estimate upper bounds of the HD, and Alt and Scharf [5, 6] compute the exact HD. Kim et al. [47] propose a real-time approach that utilizes the GPU's depth buffer.

The subject has been extended to the space curve case [23, 61], while Elber and Grandine [27] cover rational surfaces as well. Also for NURBS surfaces, Krishnamurthy et al. [50] have developed a GPU implementation for real-time computation of the HD. Parallelism over the projection of a vast number of sample points finds the HD in a robust manner, even for near-zero HD, in the modified version of Hanniel et al. [39]. But in the most recent work that solves NURBS surface HD computation, Kim et al. [49] achieve real-time performance in pure CPU implementation. It is based on iterative partitioning of the surfaces and eliminating redundant parts. Their main contribution is the "matching" of parts and the corresponding local upper bound, which is compared to the known lower bound of the HD, the criterion for culling. The algorithm successfully handles near-zero HD cases, which is possible thanks to the smoothness of the surfaces. We borrow from Kim et al. [49] and apply "matching" to triangles and quads.

But HD computation between mesh models is a very complicated matter. Bartoň et al. [11] correctly analyze and handle the many different bisector configurations, but the result is an unwieldy $O(n^4 \log n)$ algorithm. Other previous work on HD computation over triangle meshes is limited in number [8, 24, 38, 64], also demonstrating the difficulty of the problem. Metro [24] and MESH [8] are the earliest results, with the source code made public. Both only approximate the HD from point samples, without guaranteeing any kind of upper bound on the HD; they always return values lower than the actual HD. Moreover, the only way to achieve higher accuracy is to blindly increase the number of samples, quickly escalating the computation time. Tang et al. [64]

improve upon Guthe et al. [38] to compute the HD in real time. However, the inefficiency in the way they manage the mesh pieces under consideration leads to a very large number of candidate piece-pairs being pushed into the queue when the HD is near zero. We achieve interactive-time computation with modest memory requirement and arbitrary-precision error bound under near-zero HD situations.

3.2 Quad Mesh Reconstruction

Bommes et al. [17] have compiled a list of results on quad mesh generation until the year 2013, most of which take triangle meshes as input. They can be roughly classified into three categories: direct triangle-to-quad conversion, patch-based construction, and global parameterization-based quadrangulation. Direct conversion [37, 60, 66], whose main operation is to merge two triangles into a quad, can only output unstructured results, whereas patch-based quadrangulation [16, 22] generate semi-regular quad mesh.

PolyCube maps [65] are one major subclass of patch-based construction. A PolyCube is a generalized cube, whose edges are all axis-aligned. Geometry is mapped onto this PolyCube, which has been proposed to be constructed both manually [68, 69] and automatically [40, 54, 55]. More recently, volumetric PolyCubes are used to create hexahedral mesh for simulation purposes [28, 32, 36, 41, 53, 70]. These results are related to ours in that they are evaluated by the Hausdorff distance between the original model and the quad mesh.

Campen et al. [19, 20, 21] are more interested in creating quality *quad layout*, partitioning of a surface into quadrilateral patches, which can then

be refined into quad mesh. They too have approached quad layouting both automatically [19, 21] and interactively [20].

Meanwhile, parameterization-based results find global parameterization first, which is then used to generate quad mesh [18, 26, 45, 59, 67]. Most of these create valence semi-regular quadrangulation, which cannot be decomposed into a small number of regular patches, unlike semi-regular quad mesh. They are therefore less structured than semi-regular ones and unfit for potential applications such as NURBS conversion and accelerated geometric operations. Quad conversion of range images [58] and point clouds [52] have also been studied.

Works on interactively editing the irregular vertices of quad mesh include Peng et al. [57] and Takayama et al. [62]. While the former discusses modification of existing quad meshes, the latter also incorporates various tools for the creation of quad mesh, based on sketching over an existing triangle mesh. These results also concern themselves with valence semi-regular quadrangulation. Our work focuses on creating a semi-regular quad mesh from an input point cloud by sketching, without intermediary triangulation. To the best of our knowledge, there has been no previous work where quad layouts are drawn directly over existing point data.

Chapter 4

Quad Mesh BVH Operations

This chapter introduces a new kind of *bounding volume hierarchy* (BVH) data structure and minimum distance computation algorithms that make good use of it. Point projection, or minimum distance computation from a point, to a quad mesh is a core component in Hausdorff distance computation and quad mesh reconstruction algorithms discussed in Chapters 5 and 6. We also examine collision detection between two quad meshes, which is essentially an extended form of minimum distance computation between the two meshes.

4.1 Offset Tetrahedron BVH

The BVH is a type of tree where each node is a *bounding volume*, a shape whose purpose is to contain its assigned model part within. Playing a crucial role in certain algorithms, the BVH makes possible fast geometric operations

on surface models. It has been a thoroughly researched subject [35, 51], but we have developed a simple binary tree BVH that is tailored for the quad mesh.

Our BVH uses offset tetrahedra as bounding volumes. An *offset tetrahedron* is simply a tetrahedron that is offset by a certain radius, so the distance to one can be calculated by simply finding the distance to the tetrahedron and subtracting the offset radius. A semi-regular quad mesh consists of multiple regular patches, and for each we construct a single BVH. (These in turn may be combined to form a larger BVH, but the performance gain would be minimal.)

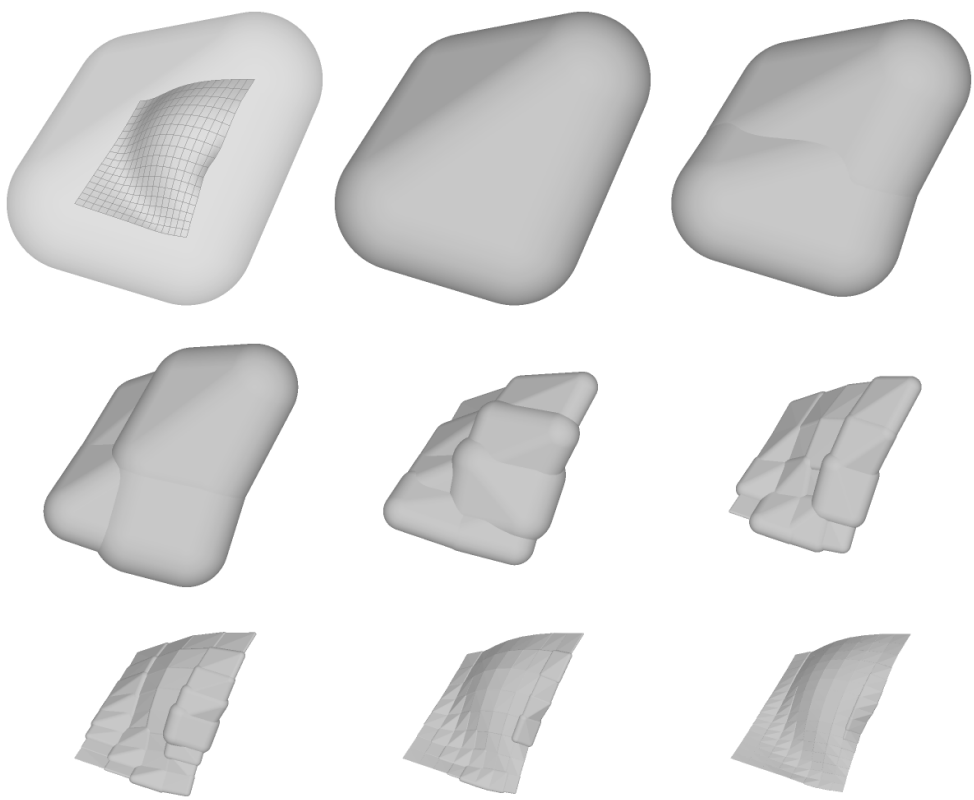
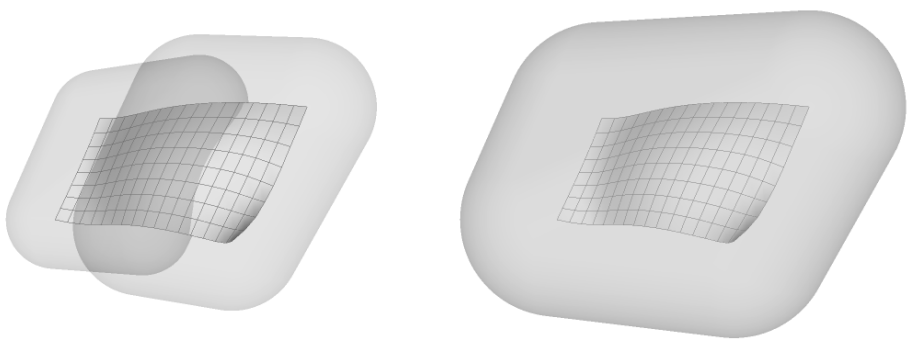


Figure 4.1: The top eight levels of the offset tetrahedron BVH for a regular patch.

We partition a patch into a hierarchy of subpatches, each to be bounded by an offset tetrahedron bounding volume. The four corners of the subpatch define the tetrahedron, but the offset radius that enables the bounding volume to encapsulate the entire subpatch has to be determined later. Partitioning is done top-down, splitting each subpatch into two. A subpatch with more rows of quads than columns is split horizontally, and otherwise, vertically, in a way that the number of rows and columns of the two new subpatches would differ by at most one. The leaves of the hierarchy would be all of the individual quads, although their depth may vary.

Now the offset radii of the tetrahedra are calculated from the deepest nodes up to the root. For a leaf, which is a single quad, whether it is interpreted as two triangles or a bilinear surface, it is already contained within the tetrahedron formed by its corners. So 0 as the offset radius suffices.

But for a node with two children, their radii factor into the parent's radius. Let us call the parent and the two children's tetrahedra T_m, T_a, and T_b, and



(a) Children's offset radii. (b) Parent's offset radius.

Figure 4.2: A parent node's offset radius is calculated from those of its children.

their offset radii r_m, r_a, and r_b. The children share two vertices $\mathbf{v}_1$ and $\mathbf{v}_2$. Then setting r_m by the following equation guarantees that the parent encapsulates its subpatch P_m, as proven in Section 4.1.1.

$$r_m = \max(r_a, r_b) + \max\big(\operatorname{dist}(\mathbf{v}_1, T_m), \operatorname{dist}(\mathbf{v}_2, T_m)\big) \tag{4.1}$$

where $\operatorname{dist}(\mathbf{p}, T) = \min_{\mathbf{q}\in T} \|\mathbf{p} - \mathbf{q}\|$, the distance from point $\mathbf{p}$ to tetrahedron T. We consider T to be a solid, so if $\mathbf{p}$ is inside T, $\operatorname{dist}(\mathbf{p}, T) = 0$. The simplicity of the radius calculation plays a huge part in our algorithms' efficiency.

4.1.1 Proof of the Offset Radius Rule

How Equation (4.1) sets a large enough radius can be proven like an inductive step in an induction proof. The "induction hypothesis" is that the two children bound their subpatches P_a and P_b.

$$\forall \mathbf{p} \in P_a,\ \operatorname{dist}(\mathbf{p}, T_a) \le r_a \tag{4.2}$$

$$\forall \mathbf{p} \in P_b,\ \operatorname{dist}(\mathbf{p}, T_b) \le r_b \tag{4.3}$$

We have to prove that

$$\forall \mathbf{p} \in P_m,\ \operatorname{dist}(\mathbf{p}, T_m) \le r_m. \tag{4.4}$$

Since $P_m = P_a \cup P_b$, if $\mathbf{p} \in P_m$, either $\mathbf{p} \in P_a$ or $\mathbf{p} \in P_b$. We first suppose that the former is true; the latter case can be proven in the same fashion. As T_m and T_a share an edge, the farthest point in T_a from T_m is either $\mathbf{v}_1$ or $\mathbf{v}_2$,

which are not shared with T_m. In other words,

$$\forall \mathbf{q} \in T_a, \;\; \text{dist}(\mathbf{q}, T_m) \leq \max\big(\,\text{dist}(\mathbf{v}_1, T_m), \text{dist}(\mathbf{v}_2, T_m)\big). \tag{4.5}$$

Now consider the minimum distance footpoint from $\mathbf{p} \in P_a$ to T_a, denoted by $\mathbf{f}_a^{\mathbf{p}}$, and then again, the footpoint from $\mathbf{f}_a^{\mathbf{p}}$ to T_m.

$$\|\mathbf{p} - \mathbf{f}_a^{\mathbf{p}}\| = \text{dist}(\mathbf{p}, T_a) \tag{4.6}$$

$$\|\mathbf{f}_a^{\mathbf{p}} - \mathbf{f}_m^{\mathbf{f}_a^{\mathbf{p}}}\| = \text{dist}(\mathbf{f}_a^{\mathbf{p}}, T_m) \tag{4.7}$$

Figure 4.3 illustrates the relationship between $\mathbf{p}$, $\mathbf{f}_m^{\mathbf{p}}$, $\mathbf{f}_a^{\mathbf{p}}$, and $\mathbf{f}_m^{\mathbf{f}_a^{\mathbf{p}}}$, with the three tetrahedra drawn simply as quads.

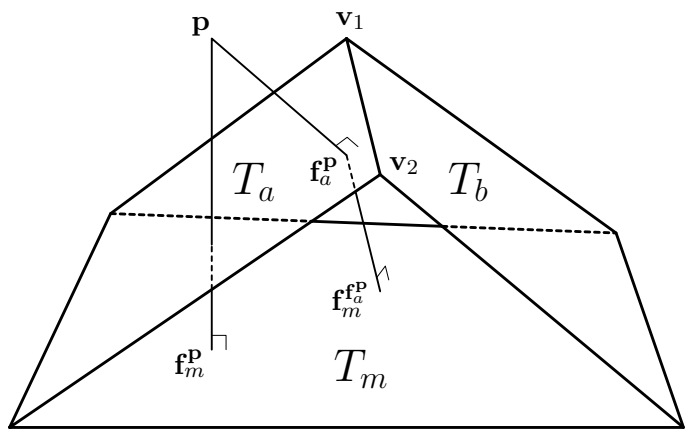


Figure 4.3: From $\mathbf{p}$, two footpoints to T_m and one to T_a.

By definition, $\text{dist}(\mathbf{p}, T_m)$ cannot be greater than the distance between $\mathbf{p}$ and $\mathbf{f}_m^{\mathbf{f}_a^{\mathbf{p}}} \in T_m$. Applying to this the triangle inequality and then (4.2) and (4.5)

proves (4.4).

$$\begin{aligned}
\operatorname{dist}(\mathbf{p}, T_m) &= \|\mathbf{p} - \mathbf{f}_m^{\mathbf{p}}\| \\
&\leq \|\mathbf{p} - \mathbf{f}_m^{\mathbf{f}_a^{\mathbf{p}}}\| \\
&\leq \|\mathbf{p} - \mathbf{f}_a^{\mathbf{p}}\| + \|\mathbf{f}_{\mathbf{a}}^{\mathbf{p}} - \mathbf{f}_m^{\mathbf{f}_a^{\mathbf{p}}}\| \\
&\leq r_a + \max\big(\operatorname{dist}(\mathbf{v}_1, T_m), \operatorname{dist}(\mathbf{v}_2, T_m)\big) \\
&\leq \max(r_a, r_b) + \max\big(\operatorname{dist}(\mathbf{v}_1, T_m), \operatorname{dist}(\mathbf{v}_2, T_m)\big) \\
&= r_m
\end{aligned} \tag{4.8}$$

4.1.2 Handling Deformation

The offset tetrahedron BVH can handle the quad mesh's deformation efficiently by keeping the subpatch hierarchy fixed and only updating the offset radii bottom-up, once again. When only part of a regular patch is deformed, we begin with only the leaves that have changed. They are added to a queue of nodes that need updating. The queue is kept sorted by depth to make sure every node is updated only after its children are. We pop the first node in the queue, update its radius, and add its parent to the queue. Eventually, the root will be updated, and the queue will be empty.

During this process, if self-collision detection were to be performed (Section 4.5), we also mark some of these nodes for its acceleration. First, all of the deformed leaves are marked. Then when a node is popped and updated, we mark it only if both of its children have been marked. If only one child is marked, we add that child to a list of nodes that can be later used as a starting point in collision detection. After the entire BVH is updated and marked, we

should end up with roots of non-overlapping subtrees whose leaves consist of, and only of, all of the deformed quads.

4.2 Point Projection with the BVH

With our offset tetrahedron BVH, the traditional BVH-based minimum distance computation, or point projection, can be performed. As we search the bounding volumes, or nodes, of our BVHs iteratively, we manage a queue of nodes that are up for consideration, kept sorted by the distance from the query point $\mathbf{p}$. This enables us to examine the closest bounding volume first, which is most likely to contain the desired footpoint. Since we only wish to know the closest one at any given time, complete sorting is unnecessary, so a priority queue, also called a heap, suffices.

Pushing a node V_i into the queue means that it potentially has the footpoint. Thus, we check the distance d_i from $\mathbf{p}$ to the bounding volume—by subtracting the offset radius from the distance to the tetrahedron—and if it is greater than the current known distance d, V_i is discarded and no longer considered, because the subpatch it encloses is known to be separated from $\mathbf{p}$ by a distance of at least d_i. Otherwise, we insert it into the queue with d_i being its sorting key. We do not enqueue leaf nodes, each of which contains a single quad, but we still measure the distance to update d. But now the precise distance to the quad is used, as opposed to the distance to the bounding volume. Its computation differs depending on how the quad is interpreted, as discussed in Section 2.2.

The known distance d is initially set to a very large placeholder value and

then updated throughout the iteration to a smaller one. When we measure the distance to a non-leaf V_i to determine whether to enqueue it, we also measure the distances to its four corners. These corners are actual vertices of the quad mesh, and the true minimum distance should be less than or equal to those distances. Therefore, if their minimum is less than d, we can assign it as d's new value. If V_i is a leaf with a smaller distance, d is updated to be the exact distance to the quad. Whenever d's value changes, we also remember the corresponding footpoint as $\mathbf{f}$.

Enqueueing starts with the root nodes—recall that there is a BVH for each regular patch, so there are multiple roots. Then the closest node is popped, and we decide whether or not to push its two children, by comparing to d their distances from $\mathbf{p}$. Since leaves are never inserted, the popped node always has children. Popping and pushing is repeated until the queue is eventually empty. At that point, the algorithm exits and returns d and $\mathbf{f}$ as the minimum distance from $\mathbf{p}$ and the corresponding footpoint.

4.3 Point Projection with the Uniform Grid

While the BVH can be used for the projection of any point, if the point were very close to the mesh, traversing the BVH is still relatively time-consuming when compared to point projection based on a uniform grid, or bucketing [7, 31]. This is extremely relevant during near-zero Hausdorff distance computation, where the footpoint is likely to be very close to the query point, so directly searching only the neighborhood is much more efficient. Thus, we employ a mixture of the two, for both triangle mesh and quad mesh.

We preprocess and assign the faces into a 3D grid with uniform cell size. Each face is assigned to every cell that it passes through, and each cell may hold multiple faces. To make sure that not too many faces are assigned to each cell, the cube cell's edge length is set to be proportional to the cube root of the mesh's bounding box volume and inversely proportional to the cube root of the number of faces.

When a query point is given for projection to the mesh, we first check if the query point is in its cell with other faces. If there are none, we fall back to BVH-based projection. Otherwise, we exhaustively find the minimum distance d_m to the faces in the cell. But d_m is not guaranteed to be the minimum distance to the entire mesh. Thus, we also check neighboring cells that are not farther away from the query point than d_m. Even though some faces may be unnecessarily assigned to some cells, this only results in extra time cost, but not inaccuracy in point projection.

4.4 Collision Detection

The traditional BVH-based collision detection is possible by what can be regarded as a slight variation of the point projection algorithm. In fact, finding collisions between two quad meshes is basically collecting the locations where the minimum distance 0 occurs. To calculate the minimum distance between two quad meshes, the priority queue now consists of *pairs* of bounding volumes, one from each of the two BVHs for the two quad meshes. It is partially sorted by the distance between the two bounding volumes, calculated by subtracting the two offset radii from the distance between the two tetrahedra.

We begin by enqueueing all possible root node pairs of the two BVHs. If one BVH has m roots and the other n, at most mn pairs can be enqueued, or less if some pairs happen to be farther than the current known distance d at the time of insertion. As was the case with point projection, d should be the distance between actual points on the two quad meshes. (If we are strictly interested in *only* the collisions, d does not need to be kept, and we always compare the distances to 0, i.e., any non-overlapping pairs are discarded.)

When a node pair V_i and V_j is popped, we consider the children of the node that has more leaves. If V_i contains more leaves, we pair each of its children with V_j, creating two pairs. For each new pair, if the distance between the two nodes is greater than d, (or 0, if only collisions are of interest,) the pair is discarded. Otherwise, it is enqueued for later consideration. If both nodes are leaves, the pair is not inserted, but we check if they overlap, so that we can report collision. If all collisions need to be found, we iterate until the queue is empty.

4.5 Self-Collision Detection

To find self-intersections within a single quad mesh, we can simply apply the above algorithm to the same two quad meshes. Unfortunately, every single non-leaf node will be paired with itself and enqueued, extremely bogging down the performance. Even if we tweak the implementation so that they don't have to be enqueued per se and duplicate computations, i.e., checking both pair V_i and V_i and pair V_j and V_i, are avoided, we still need to essentially execute collision detection over all non-leaf sibling pairs.

But if the mesh is under partial deformation, we can accelerate subsequent self-collision detections. Instead of starting with all root pairs, we can pair the roots of the BVH with the roots of non-overlapping, all-marked subtrees that we have found while updating the BVH as discussed in Section 4.1.2. This allows us to skip unnecessary distance comparisons and check only the deformed parts. For non-deformed quads, the previous collision reports between them are still valid; we can carry over the results without rechecking. But the collision results involving deformed parts need to have been cleared during the BVH update.

Chapter 5

Hausdorff Distance Computation

In this chapter, we present an algorithm for the computation of the Hausdorff distance between triangle meshes and quad meshes. For surfaces in general, the one-sided HD from surface A to surface B is defined as:

$$h(A, B) = \max_{\mathbf{p} \in A} \min_{\mathbf{q} \in B} \|\mathbf{p} - \mathbf{q}\|. \tag{5.1}$$

In other words, it is the maximum among points in A of the minimum distance to B. As mentioned in Chapter 1, it can be used as a measure of B's approximation power of A. Having small $h(A, B)$ means good approximation, since for any point on A, there is a point on B within the radius $h(A, B)$. Thus, fast and accurate computation of the HD aids in the evaluation of the quality of an approximation, and we do this for the case where A is a triangle mesh and B is a quad mesh and vice versa.

The two-sided HD, which is necessary to properly evaluate approximations, is simply the maximum of the two one-sided HD's.

$$H(A, B) = \max\big(h(A, B), h(B, A)\big) \tag{5.2}$$

Technically, we need to consider both $h(A, B)$ and $h(B, A)$. But by interchanging the notations for the triangle mesh and the quad mesh, we will rather consider A to always mean the source model, while B represents the target model.

We iteratively split certain relevant pieces of A into smaller pieces, managing them in a queue. Eliminating pieces known to be irrelevant to the HD is a core operation, and their identification (matching) is one of our main contributions. Unlike the approach introduced by Tang et al. [64], the A pieces are the *only* ones enqueued, each by itself, instead of as combinatorial pairs with pieces of B. This greatly reduces the memory requirement necessary for HD computation, resulting in a more robust algorithm, even when the HD is near zero. Meanwhile, assigning the faces of B into a uniform grid allows much faster point projection, which is a bottleneck operation in HD computation.

To simplify illustrating our algorithm, in Section 5.1, we first describe the general framework for finding the HD between two surfaces without any assumption about the properties of the surfaces. Then we apply it to the simple problem of computing the HD from a single triangle to a single quad by specifying its ambiguous elements, in Section 5.2. This is extended to triangle mesh-to-quad mesh HD in Section 5.3, while Section 5.4 deals with quad mesh-

to-triangle mesh HD. These are two one-sided HD's in the opposite direction, and Section 5.5 briefly discusses merging the two algorithms to fully compute the two-sided HD. Section 5.6 shows some experimental results, demonstrating the algorithm's speed and other characteristics.

5.1 General Framework

This section will discuss the one-sided Hausdorff distance computation algorithm for general surfaces, without mention of a mesh, or a triangle, quad, etc. As such, elements of the algorithm will be left vague, which will be summarized in Section 5.1.4. This was the approach used by many others [38, 49], and we have only taken what can be applied to our situation.

For surfaces A and B, calculating the HD can be done with an iterative algorithm that is based on the following observation where $\underline{A}$, $\overline{A}$, $\underline{B}$, and $\overline{B}$ are any sets that satisfy $\underline{A} \subset A \subset \overline{A}$ and $\underline{B} \subset B \subset \overline{B}$:

$$h(\underline{A}, \overline{B}) \leq h(A, B) \leq h(\overline{A}, \underline{B}). \tag{5.3}$$

This naturally follows from the definition. Removing elements from A may remove $\mathbf{p}$ that maximizes $\min_{\mathbf{q} \in B} \|\mathbf{p} - \mathbf{q}\|$, decreasing $h(A, B)$, while adding to A can only increase $h(A, B)$. Similarly, adding elements to B may introduce $\mathbf{q}$ that further minimizes $\|\mathbf{p} - \mathbf{q}\|$, resulting in smaller $h(A, B)$, while removing from B may do the opposite. We exploit this fact to find a *lower bound* $\underline{h}$ and an *upper bound* $\overline{h}$ for the HD. During the iterative process, we partition A into smaller pieces, and update $\underline{h}$ to greater values and $\overline{h}$ to smaller values. This

is repeated until $\underline{h}$ and $\overline{h}$ eventually converge within a certain error bound, giving a precise estimate of the HD. They are also used to eliminate pieces of A that are no longer relevant, speeding up the algorithm.

5.1.1 Lower Bound

From $h(\underline{A}, \overline{B}) \leq h(A, B)$, we specify $\underline{A}$ to be A', a finite set of points in A, and $\overline{B}$ to be, simply, B. Then $h(A', B)$ is a calculation that we can manage. We simply need to find $\min_{\mathbf{q} \in B} \|\mathbf{p} - \mathbf{q}\|$ for all $\mathbf{p}$ in A' and take their maximum. So $\underline{h}$ is initialized with a certain number of points in A. Then, during the iterative process of surface partitioning, the distance to B from more points is calculated, updating $\underline{h}$ by their maximum. In the end, the point that causes the final update may be returned as the point that realizes the HD within the given error bound.

5.1.2 Upper Bound

Suppose A is partitioned into $A_1, \cdots, A_n$. Then the HD from A is the maximum of the local HDs from A_k's.

$$h(A, B) = \max_k h(A_k, B) \tag{5.4}$$

By applying (5.3), we get

$$h(A, B) \leq \max_k h(A_k, B_k), \tag{5.5}$$

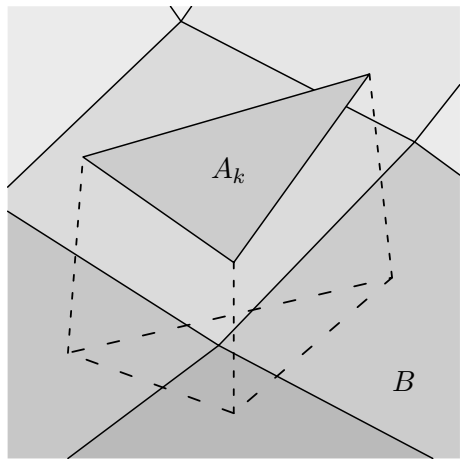


Figure 5.1: Choosing a match on B, a quad mesh, for A_k, which is a triangle in this case. The projection of A_k on B is a good candidate, but as will be addressed in Section 5.1.4, we employ a substitute that is much simpler to compute.

where $B_k \subset B$ is a *match* chosen appropriately for each A_k so that finding an overestimation for $h(A_k, B_k)$ that is sufficiently small is a plausible task. In other words, for each point on A_k, we assign a match point on B, resulting in B_k. We call the overestimation the *local upper bound* for A_k, or $\overline{h}_k$. It should shrink smaller as A_k is partitioned over and over. The maximum among $\overline{h}_k$ acts as $\overline{h}$, which may also be called the *global upper bound* in contrast. Figure 5.1 shows one example for matching, which will be discussed in more detail later.

5.1.3 Iterative Partitioning

In short, we have

$$\underline{h} = h(A', B) \leq h(A, B) \leq \max_k h(A_k, B_k) \leq \max_k \overline{h}_k = \overline{h}. \tag{5.6}$$

Until $\underline{h}$ and $\overline{h}$ converge, we repeatedly split A, and its pieces are either discarded or collected in a queue for further partitioning, based on the relationship between the local upper bound $\overline{h}_k$ and the current lower bound $\underline{h}$. For every piece A_k, we overestimate the local HD and get $\overline{h}_k$. If $\overline{h}_k < \underline{h}$, the piece is no longer considered. Otherwise, it is pushed into the queue, which sorts the pieces by $\overline{h}_k$, so that the piece with the greatest local upper bound will be considered next. For example, in the beginning, assuming the lower bound is initialized to 0, the queue is empty, and A is inserted in its entirety, regardless of its local upper bound. If A were inherently a collection of smaller pieces, it may be necessary to start with those. As was the case with BVH-based algorithms in Chapter 4, since we only need to know the greatest $\overline{h}_k$, thorough sorting is not needed, and thus a priority queue will do.

After all pieces have been either discarded or pushed into the queue, we pop the queue to obtain the next piece to consider, whose $\overline{h}_k$ becomes the current $\overline{h}$, possibly decreasing, but never increasing it. It is at this point that the decision is made to continue the iteration. If it is to be continued, the piece is split into smaller pieces, during which more points are used to possibly increase $\underline{h}$. For each new piece, its local upper bound is calculated from its match, and the process reiterates. In the end, the queue is not necessarily empty, but we have terminated the iteration since the difference between the lower bound and upper bound is less than the given error bound; we have found the HD within the error bound.

When the HD is near zero, pieces are not easily discarded, and the queue continues to grow for numerous iterations, not only immensely delaying the

termination of the algorithm, but also expanding the memory requirement. But the fact that we only enqueue pieces of A (with corresponding matches on B) makes our approach much more robust than previous HD computation algorithms on meshes, where pieces from both A and B are enqueued pairwise. This results in the size of the queue being roughly squared, leading to memory consumption that can easily crash the system in near-zero HD cases.

5.1.4 Unspecified Elements

The following is a list of all elements from the general framework above that have been described inexplicitly, as specifics wouldn't apply to general HD computation circumstances. They need to be filled in according to the individual details of the input surfaces.

- **Partitioning.** How to divide A is unlikely to be a difficult choice, but nevertheless depends on what kind of surface A actually is.
- **Sampling.** It has to be decided which points on A will be used to update $\underline{h}$, both in the beginning and during iteration. Partitioning A likely requires evaluation of some points, which would be a good choice, since they are newly evaluated points.
- **Point projection.** There has to be a way of measuring the minimum distance from a point (sample) on A to B. For the case where B is a quad mesh, this was already addressed in Chapter 4.
- **Matching and local upper bound.** Choosing a good match B_k and local upper bound $\overline{h}_k$ for each A_k is the most challenging task in the

algorithm. While projecting A_k to B, which is not a simple job itself, to obtain B_k may seem the instinctive option, finding a reasonable $\overline{h}_k$ is also hard; it has to be both a simple computation and a tight estimate of $h(A_k, B_k)$. Some variation of A_k's projection on B may be more feasible.

5.2 Triangle-to-Quad HD

Now we apply the above framework to the computation of the one-sided Hausdorff distance where A is a single triangle and B a single quad. Here, we will only treat each quad to be a bilinear surface, since the other case—each quad as two triangles—is rather *too* simple. Below is a list of the specified versions of the elements in Section 5.1.4.

- **Partitioning.** Perhaps intuitively, we split the triangle into four smaller subtriangles, by connecting the midpoints of the three edges, and do the same for those when the occasion arises, i.e., when the enqueued triangle is popped.

- **Sampling.** The three corners of the triangle are used to set the initial $\underline{h}$. During iteration, when a triangle is split, three midpoints are evaluated. The minimum distance to B from those points is measured, updating $\underline{h}$.

- **Point projection.** Since B is a bilinear surface, we can use Newton's method discussed Section 2.2.

- **Matching and local upper bound.** The three corners have already been projected to B, but it is not an easy task to find the proper projec-

tion of a triangle to a bilinear surface, nor is it necessary. (As illustrated in works of Bo et al. [14, 15], even the projection of a line segment is generally a space curve.) Instead, based on the parameters of the footpoints on B, we find a subsurface that is close in shape to the proper projection, as detailed below.

Suppose the three corners $\mathbf{p}_i$ of the triangle A_k are projected to $B(u_i, v_i)$, where $i = 1, 2, 3$ and $0 \leq u_i, v_i \leq 1$ (Figure 5.2). We connect these in B's parameter domain and form a triangle, which creates a subsurface of B to be used as B_k. While this surface is *not* the projection of A_k on B, we can yet use it as a match for A_k. We also have to find a nice local upper bound $\overline{h}_k$ that satisfies

$$\overline{h}_k \geq h(A_k, B_k) = \max_{\mathbf{p} \in A_k} \min_{\mathbf{q} \in B_k} \|\mathbf{p} - \mathbf{q}\|. \tag{5.7}$$

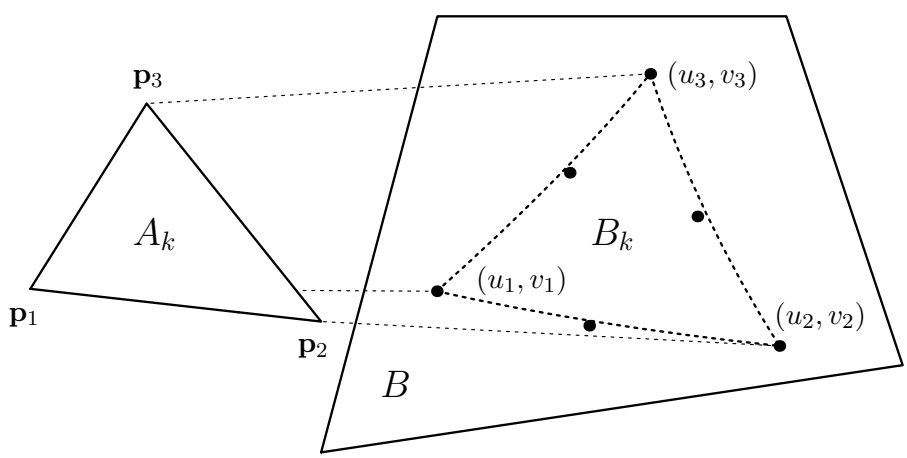


Figure 5.2: Cutting a bilinear surface along a triangle in its parameter domain produces a triangular quadratic Bézier surface, whose control points are marked in the figure. The resulting dotted triangle does *not* represent the actual projection of A_k to B, but only that of the corners.

If we match some $\mathbf{q}$ for each $\mathbf{p}$, denoted as $\mathbf{q} = \mathbf{q_p}$, the maximum distance over $\mathbf{p}$ should be greater than or equal to the HD, as long as $\mathbf{q_p} \in B_k$.

$$\max_{\mathbf{p}\in A_k} \min_{\mathbf{q}\in B_k} \|\mathbf{p}-\mathbf{q}\| \le \max_{\mathbf{p}\in A_k} \|\mathbf{p}-\mathbf{q_p}\| \tag{5.8}$$

Assuming both A_k and B_k are parameterized in barycentric coordinates, we can pair $\mathbf{p} = A_k(\bar{u},\bar{v},\bar{w})$ with $\mathbf{q_p} = B_k(\bar{u},\bar{v},\bar{w})$, where $\bar{u}+\bar{v}+\bar{w}=1$.

$$h(A_k, B_k) \le \max_{\bar{u},\bar{v},\bar{w}} \|A_k - B_k\| \tag{5.9}$$

Now let us find such parameterization. We can express B_k as a reparameterization of $B(u,v)$ with $u = u_1\bar{u}+u_2\bar{v}+u_3\bar{w}$ and $v = v_1\bar{u}+v_2\bar{v}+v_3\bar{w}$. Then one can easily check that B_k is actually a triangular quadratic Bézier surface.

$$\begin{aligned}
&B_k(\bar{u},\bar{v},\bar{w}) = B(u,v)\\
&= \mathbf{a}(1-u)v + \mathbf{b}(1-u)(1-v) + \mathbf{c}u(1-v) + \mathbf{d}uv\\
&= \quad \mathbf{a}\big((1-u_1)\bar{u}+(1-u_2)\bar{v}+(1-u_3)\bar{w}\big)(v_1\bar{u}+v_2\bar{v}+v_3\bar{w})\\
&\quad + \mathbf{b}\big((1-u_1)\bar{u}+(1-u_2)\bar{v}+(1-u_3)\bar{w}\big)\\
&\qquad \big((1-v_1)\bar{u}+(1-v_2)\bar{v}+(1-v_3)\bar{w}\big)\\
&\quad + \mathbf{c}(u_1\bar{u}+u_2\bar{v}+u_3\bar{w})\big((1-v_1)\bar{u}+(1-v_2)\bar{v}+(1-v_3)\bar{w}\big)\\
&\quad + \mathbf{d}(u_1\bar{u}+u_2\bar{v}+u_3\bar{w})(v_1\bar{u}+v_2\bar{v}+v_3\bar{w})\\
&= B(u_1,v_1)\bar{u}^2 + B(u_2,v_2)\bar{v}^2 + B(u_3,v_3)\bar{w}^2\\
&\quad + \big(B(u_1,v_2)+B(u_2,v_1)\big)\bar{u}\bar{v}\\
&\quad + \big(B(u_2,v_3)+B(u_3,v_2)\big)\bar{v}\bar{w} + \big(B(u_3,v_1)+B(u_1,v_3)\big)\bar{w}\bar{u}
\end{aligned} \tag{5.10}$$

The triangle A_k and its difference from B_k can also be expressed as Bézier surfaces.

$$\begin{aligned} A_k(\bar{u},\bar{v},\bar{w}) &= \mathbf{p}_1\bar{u} + \mathbf{p}_2\bar{v} + \mathbf{p}_3\bar{w} \\ &= \mathbf{p}_1\bar{u}^2 + \mathbf{p}_2\bar{v}^2 + \mathbf{p}_3\bar{w}^2 \\ &\quad + (\mathbf{p}_1+\mathbf{p}_2)\bar{u}\bar{v} + (\mathbf{p}_2+\mathbf{p}_3)\bar{v}\bar{w} + (\mathbf{p}_3+\mathbf{p}_1)\bar{w}\bar{u} \end{aligned} \tag{5.11}$$

$$A_k - B_k = \mathbf{h}_1\bar{u}^2 + \mathbf{h}_2\bar{v}^2 + \mathbf{h}_3\bar{w}^2 + 2\mathbf{h}_{12}\bar{u}\bar{v} + 2\mathbf{h}_{23}\bar{v}\bar{w} + 2\mathbf{h}_{31}\bar{w}\bar{u} \tag{5.12}$$

where

$$\begin{aligned} \mathbf{h}_1 &= \mathbf{p}_1 - B(u_1,v_1) & \mathbf{h}_{12} &= \frac{\mathbf{p}_1+\mathbf{p}_2}{2} - \frac{B(u_1,v_2)+B(u_2,v_1)}{2} \\ \mathbf{h}_2 &= \mathbf{p}_2 - B(u_2,v_2) & \mathbf{h}_{23} &= \frac{\mathbf{p}_2+\mathbf{p}_3}{2} - \frac{B(u_2,v_3)+B(u_3,v_2)}{2} \\ \mathbf{h}_3 &= \mathbf{p}_3 - B(u_3,v_3) & \mathbf{h}_{31} &= \frac{\mathbf{p}_3+\mathbf{p}_1}{2} - \frac{B(u_3,v_1)+B(u_1,v_3)}{2}. \end{aligned}$$

The convex hull property of Bézier surfaces allows us to guarantee an upper bound for a Bézier surface's norm with quick computation. We can exploit this property to set our $\overline{h}_k$.

$$\max_{\bar{u},\bar{v},\bar{w}} \|A_k - B_k\| \le \max(\|\mathbf{h}_1\|, \|\mathbf{h}_2\|, \|\mathbf{h}_3\|, \|\mathbf{h}_{12}\|, \|\mathbf{h}_{23}\|, \|\mathbf{h}_{31}\|) = \overline{h}_k \tag{5.13}$$

5.3 Triangle Mesh-to-Quad Mesh HD

With the configuration of A being a triangle mesh and B a quad mesh, we carry over many of the elements from Section 5.2. Finding the match and its local upper bound is still the most complex issue.

- **Partitioning.** Now the entire triangle mesh is not enqueued as a whole in the beginning, but as individual triangles instead. Then when a triangle is popped but not eliminated, we continue to split it into four smaller ones.

- **Sampling.** Now that there is an array of triangles, every vertex is first projected to the quad mesh. They all figure into the initial $\underline{h}$. Otherwise, updating during iteration remains the same.

- **Point projection.** The offset tetrahedron BVH and uniform grid are used for minimum distance computation, as detailed in Sections 4.2 and 4.3. In near-zero HD cases, most point projection is done using the grid, and its computation time is sped up by more than ten times. This mitigates the fact that the number of point projection necessary for HD computation dramatically increases as the two meshes overlap more closely.

- **Matching and local upper bound.** The three corners of a triangle are no longer guaranteed to be projected onto a single quad. Even if triangles are split and become smaller, there are a few unavoidable cases of the relationship between the footpoint quads. Each must be handled differently, but also differently depending on whether to think of a quad

as a bilinear surface or two adjoining triangles (Sections 5.3.1 and 5.3.2).

5.3.1 Matching to Quads as Bilinear Surfaces

When quads are interpreted as bilinear surfaces, matching the triangle $A_k = \triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ boils down to four cases listed below. For each A_k, we find the first applicable case in the order listed. Sometimes, the local upper bound may not seem ideal, but the overestimation is mitigated when the triangle is later subdivided. Also, $\mathbf{p}_i$'s true closest footpoint on the quad mesh may not be used; it may be better to project $\mathbf{p}_i$ to a different quad.

a. The simplest case where the three footpoints are on the same quad was already discussed in Section 5.2. We will call the local upper bound for this case $\overline{h}_a(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q)$, where the three corners $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ are all projected to the same quad Q.

b. The three footpoints are on two quads that share an edge. Without loss of generality, let us assume that $\mathbf{p}_1$ is projected to Q_1, and $\mathbf{p}_2$ and $\mathbf{p}_3$ are projected to Q_2. We can also reparameterize Q_1 and Q_2 so that the footpoints are positioned as in Figure 5.3. (u_i, v_i) is the parameter of $\mathbf{p}_i$'s footpoint on its quad. To match the triangle, we split it into three and match them separately. We first evaluate internal divisions of $\overline{\mathbf{p}_1\mathbf{p}_2}$ and $\overline{\mathbf{p}_1\mathbf{p}_3}$ roughly in the ratios $1 - u_1 : u_2$ and $1 - u_1 : u_3$.

$$\mathbf{q}_2 = \frac{u_2\mathbf{p}_1 + (1-u_1)\mathbf{p}_2}{1 - u_1 + u_2}, \mathbf{q}_3 = \frac{u_3\mathbf{p}_1 + (1-u_1)\mathbf{p}_3}{1 - u_1 + u_3} \tag{5.14}$$

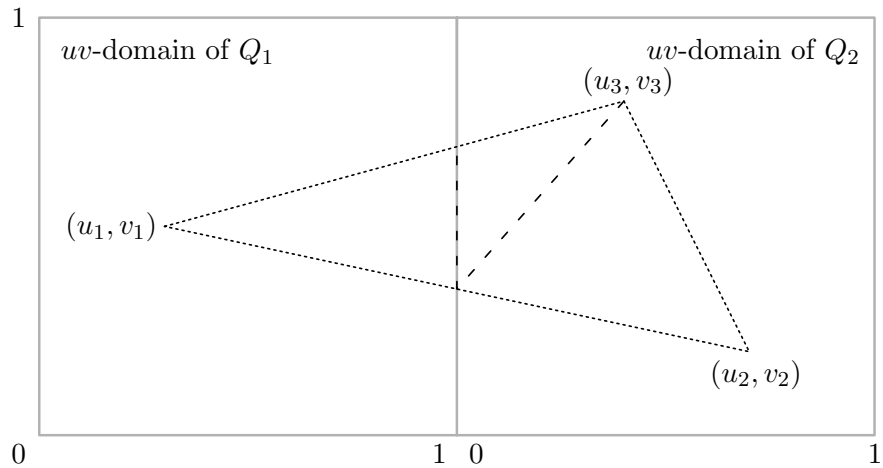


Figure 5.3: When the three corners of the triangle are projected onto two neighboring quads, it is split into three smaller triangles and matched individually, the final local upper bound being the maximum of the three local upper bounds.

Assuming that $u_2 > u_3$, we use triangles $\triangle \mathbf{p}_1\mathbf{q}_2\mathbf{q}_3$, $\triangle \mathbf{q}_2\mathbf{p}_2\mathbf{p}_3$, and $\triangle \mathbf{p}_3\mathbf{q}_3\mathbf{q}_2$. Otherwise, we introduce $\overline{\mathbf{p}_2\mathbf{q}_3}$ instead of $\overline{\mathbf{p}_3\mathbf{q}_2}$ as the new edge. In the same way as described in Section 5.2, we match the first triangle with a subsurface of Q_1, while the latter two are matched with subsurfaces of Q_2. We use the maximum of the three resulting local upper bounds as the local upper bound of the triangle $\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$.

$$\begin{aligned} \overline{h}_{b'}(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q_1, Q_2) = \max \big(&\overline{h}_a(\triangle \mathbf{p}_1\mathbf{q}_2\mathbf{q}_3, Q_1), \\ &\overline{h}_a(\triangle \mathbf{q}_2\mathbf{p}_2\mathbf{p}_3, Q_2), \overline{h}_a(\triangle \mathbf{p}_3\mathbf{q}_3\mathbf{q}_2, Q_2)\big) \end{aligned} \tag{5.15}$$

However, there are situations where $\overline{h}_{b'}$ is not the best possible choice. Without splitting, projecting the three $\mathbf{p}_i$'s to the same quad—either Q_1 or Q_2—instead of the actual closest quad from each, may actually result

in smaller local upper bound. Therefore, we explore all options and take the minimum.

$$\begin{aligned}\overline{h}_b(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q_1, Q_2) = \min\big(&\overline{h}_{b'}(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q_1, Q_2),\\ &\overline{h}_a(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q_1), \overline{h}_a(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q_2)\big)\end{aligned} \tag{5.16}$$

c. $\mathbf{p}_i$'s are projected to three different quads, Q_i's, that share a single common vertex. An example is shown in Figure 5.1. For this case, we consider three overlapping subtriangles that cover A_k. (Every point on A_k has to be matched.) Let $\mathbf{q}_{ij}$ be the internal division point of $\overline{\mathbf{p}_i\mathbf{p}_j}$ in the ratio $2:1$. We evaluate all six possible $\mathbf{q}_{ij}$'s where $i \neq j$. The three triangles $\triangle \mathbf{p}_1\mathbf{q}_{12}\mathbf{q}_{13}$, $\triangle \mathbf{p}_2\mathbf{q}_{23}\mathbf{q}_{21}$, and $\triangle \mathbf{p}_3\mathbf{q}_{31}\mathbf{q}_{32}$ cover A_k and meet at its centroid. We match each triangle as in Section 5.2 with a subsurface of Q_1, Q_2, and Q_3, respectively.

$$\begin{aligned}\overline{h}_c(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q_1, Q_2, Q_3) = \max\big(&\overline{h}_a(\triangle \mathbf{p}_1\mathbf{q}_{12}\mathbf{q}_{13}, Q_1),\\ &\overline{h}_a(\triangle \mathbf{p}_2\mathbf{q}_{23}\mathbf{q}_{21}, Q_2), \overline{h}_a(\triangle \mathbf{p}_3\mathbf{q}_{31}\mathbf{q}_{32}, Q_3)\big)\end{aligned} \tag{5.17}$$

d. The last case is when none of the above is satisfied. We would rather enqueue the triangle without hesitation for further consideration without putting too much effort to find a small local upper bound. But at least we can quickly try projecting to a single quad, like in case b.

$$\begin{aligned}\overline{h}_d(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q_1, Q_2, Q_3) = \min\big(&\overline{h}_a(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q_1),\\ &\overline{h}_a(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q_2), \overline{h}_a(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, Q_3)\big)\end{aligned} \tag{5.18}$$

5.3.2 Matching to Quads as Adjoining Triangles

Assuming each quad to be two adjoining triangles, we handle three cases, not four, differently. Now we consider the triangles, not bilinear surfaces, to which each corner $\mathbf{p}_i$ is projected. Since A and B both consist of linear elements, choosing a match is much simpler, and the local upper bound can be much tighter, speeding up the algorithm by roughly two times, compared to the bilinear surface interpretation. We will call $\mathbf{p}_i$'s footpoint on the quad mesh $\mathbf{r}_i$.

a. $\mathbf{p}_i$'s are all projected onto the same triangle T. T's subtriangle that connects the footpoints is taken as the match. By linearly assigning points on $\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ to points on $\triangle\mathbf{r}_1\mathbf{r}_2\mathbf{r}_3$, so that $\mathbf{p}_i$ is assigned to $\mathbf{r}_i$, the local upper bound can be immediately computed from the corners' coordinates.

$$\overline{h}_a(\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, T) = \max_i \|\mathbf{p}_i - \mathbf{r}_i\| \tag{5.19}$$

This is also the exact HD from $\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ to $\triangle\mathbf{r}_1\mathbf{r}_2\mathbf{r}_3$.

b. The three footpoints are on two triangles T_1 and T_2 that share an edge. Unlike in Section 5.3.1, we can easily find a very tight local upper bound. We wish to divide A_k into two pieces so that every point on one piece is closer to T_1 than T_2, and every point on the other is closer to T_2 than T_1. This can be done by splitting by the bisector surface of T_1 and T_2. Nevertheless, as discussed in Barton et al. [11], the bisector surface may include some quadratic surface patches as well as planar ones. To simplify the construction of matching, instead of the exact bisector surface, we

use the bisector plane of the two planes that contain T_1 and T_2. It most likely intersects with two edges of A_k, e.g., $\overline{\mathbf{p}_1\mathbf{p}_3}$ and $\overline{\mathbf{p}_2\mathbf{p}_3}$ in Figure 5.4. Let $\mathbf{b}_1$ and $\mathbf{b}_2$ be the intersection points, but in the very rare case where the bisector plane does not intersect with A_k, we simply set them to be the midpoints of $\overline{\mathbf{p}_1\mathbf{p}_3}$ and $\overline{\mathbf{p}_2\mathbf{p}_3}$. We compute their footpoints on T_1 and T_2, called $\mathbf{r}_{11}$, $\mathbf{r}_{12}$, $\mathbf{r}_{21}$, and $\mathbf{r}_{22}$. Assuming $\triangle\mathbf{p}_3\mathbf{b}_1\mathbf{b}_2$ is closer to T_1, we match it with $\triangle\mathbf{r}_3\mathbf{r}_{11}\mathbf{r}_{21}$, and $\square\mathbf{p}_1\mathbf{p}_2\mathbf{b}_2\mathbf{b}_1$ with $\square\mathbf{r}_1\mathbf{r}_2\mathbf{r}_{22}\mathbf{r}_{12}$. The local upper bound can once again be calculated from corresponding vertex pairs.

$$\begin{aligned}\overline{h}_b(\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, T_1, T_2) = \max(&\max_i \|\mathbf{p}_i - \mathbf{r}_i\|,\\ &\|\mathbf{b}_1 - \mathbf{r}_{11}\|, \|\mathbf{b}_1 - \mathbf{r}_{12}\|, \|\mathbf{b}_2 - \mathbf{r}_{21}\|, \|\mathbf{b}_2 - \mathbf{r}_{22}\|)\end{aligned} \tag{5.20}$$

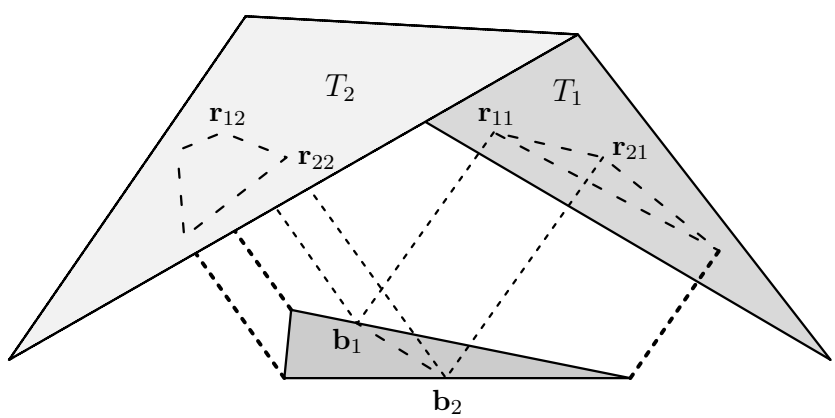


Figure 5.4: When the three corners of the triangle are projected onto two neighboring triangles, the triangle is split into a triangle and a quadrangle and matched individually, the final local upper bound being the maximum of the two local upper bounds.

Remark: We may further analyze the case into separate subcases and reduce the number of terms to compare, which is currently seven in Equation (5.20). For instance, for configurations such as that in Figure 5.4, where T_1 and T_2 "meet concavely" from A_k's point of view, we would only need to compare the four terms that involve $\mathbf{b}_1$ or $\mathbf{b}_2$. However, considering that A_k may intersect with T_1 and T_2, determining the numerous possibilities is quite intricate. Rather than taking the time to pinpoint the exact configuration, it is much quicker to simply evaluate all seven terms.

c. $\mathbf{p}_i$'s are projected to different triangles T_i's, two among which may be the same. Instead of figuring out all of the complicated bisector possibilities, we choose the simple route: split A_k into three quadrilaterals by connecting the edges' midpoints to the centroid $\mathbf{c}$. Let $\mathbf{m}_i$ be the midpoint of $\overline{\mathbf{p}_i\mathbf{p}_{i+1}}$. We project each $\mathbf{m}_i$ onto T_i and T_{i+1}, calling the footpoint $\mathbf{r}_{ia}$ and $\mathbf{r}_{i+1b}$, while $\mathbf{c}$ is projected onto all T_i's, the footpoints being $\mathbf{r}_{ic}$'s. We match each $\square\mathbf{p}_i\mathbf{m}_i\mathbf{c}\mathbf{m}_{i+2}$ with $\square\mathbf{r}_i\mathbf{r}_{ia}\mathbf{r}_{ic}\mathbf{r}_{ib}$ as in Figure 5.5. The local upper bound becomes the maximum distance among vertex-footpoint pairs.

$$\begin{aligned}&\overline{h}_{c'}(\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, T_1, T_2, T_3)\\ &= \max_i \max(\|\mathbf{p}_i - \mathbf{r}_i\|, \|\mathbf{m}_i - \mathbf{r}_{ia}\|, \|\mathbf{c} - \mathbf{r}_{ic}\|, \|\mathbf{m}_{i+2} - \mathbf{r}_{ib}\|)\end{aligned} \tag{5.21}$$

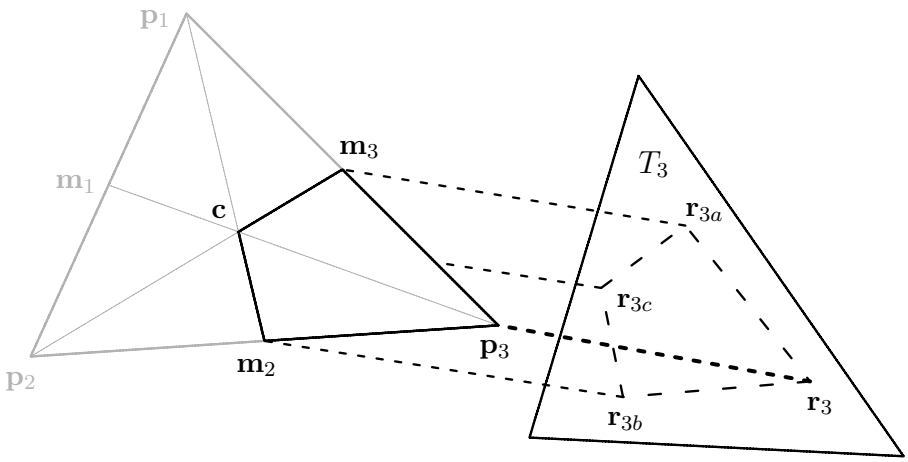


Figure 5.5: An example of a quadrangular part of a triangle being matched to a quadrangle.

However, like the second case in Section 5.3.1, projecting A_k in its entirety onto one of T_i's may be the better choice, so we also check for those possibilities.

$$\begin{aligned}&\overline{h}_c(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, T_1, T_2, T_3)\\ &= \min\left(\overline{h}_{c'}(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, T_1, T_2, T_3), \min_i \overline{h}_a(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3, T_i)\right)\end{aligned} \tag{5.22}$$

5.4 Quad Mesh-to-Triangle Mesh HD

Here, the meshes switch roles; A is a quad mesh and B is a triangle mesh. Matching again has to be done differently depending on the interpretation of quads. Regardless, one common element is that in the beginning, every quad is split into two triangular pieces for enqueueing. This is straightforward for the adjoined-triangles interpretation—a quad is divided along a predetermined

diagonal into two triangles, and then each triangle is repeatedly split into four identical ones, like in Section 5.3.

But for bilinear surfaces, we do something similar to reduce the number of vertices of each piece from four to three. Since we will analyze their projections to determine the matching, we wish to simplify the cases. $B(u,v) = \mathbf{a}(1-u)v + \mathbf{b}(1-u)(1-v) + \mathbf{c}u(1-v) + \mathbf{d}uv$ can be split into two triangular Bézier surfaces S_1 and S_2, by cutting along either of the two diagonals in the parametric domain: $u = 1-v$ or $u = v$. To avoid elongated pieces, we choose $u = 1-v$ if $\|\mathbf{a}-\mathbf{c}\| < \|\mathbf{b}-\mathbf{d}\|$ and $u = v$ otherwise.

Splitting along $u = 1-v$ can be regarded as a reparameterization of $B(u,v)$. For S_1, we wish to have $S_1(1,0,0) = \mathbf{a}$, $S_1(0,1,0) = \mathbf{b}$, and $S_1(0,0,1) = \mathbf{c}$. We can achieve this by substituting u, $1-u$, v, and $1-v$ with $\bar{w}$, $\bar{u}+\bar{v}$, $\bar{u}$, and $\bar{v}+\bar{w}$, respectively.

$$\begin{aligned} B(u,v) &= \mathbf{a}(1-u)v + \mathbf{b}(1-u)(1-v) + \mathbf{c}u(1-v) + \mathbf{d}uv \\ &= \mathbf{a}(\bar{u}+\bar{v})\bar{u} + \mathbf{b}(\bar{u}+\bar{v}) + (\bar{v}+\bar{w}) + \mathbf{c}\bar{w}(\bar{v}+\bar{w}) + \mathbf{d}\bar{w}\bar{u} \\ &= \mathbf{a}\bar{u}^2 + \mathbf{b}\bar{v}^2 + \mathbf{c}\bar{w}^2 + (\mathbf{a}+\mathbf{b})\bar{u}\bar{v} + (\mathbf{b}+\mathbf{c})\bar{v}\bar{w} + (\mathbf{b}+\mathbf{d})\bar{w}\bar{u} \\ &= S_1(\bar{u},\bar{v},\bar{w}) \end{aligned} \tag{5.23}$$

In other words, S_1 is a quadratic Bézier triangle defined by the six control points $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\frac{\mathbf{a}+\mathbf{b}}{2}$, $\frac{\mathbf{b}+\mathbf{c}}{2}$, and $\frac{\mathbf{b}+\mathbf{d}}{2}$. Similarly, S_2 is defined by the control points $\mathbf{c}$, $\mathbf{d}$, $\mathbf{a}$, $\frac{\mathbf{c}+\mathbf{d}}{2}$, $\frac{\mathbf{d}+\mathbf{a}}{2}$, and $\frac{\mathbf{b}+\mathbf{d}}{2}$. Notice that the new control points are the midpoints of the quad's four edges and the diagonal $\overline{\mathbf{bd}}$. If we were to choose $u = v$ as the cut, we would need the midpoint of $\overline{\mathbf{ac}}$.

When a Bézier triangle $S(u, v, w)$ needs to be split, we cut along three lines $u = 0.5$, $v = 0.5$, and $w = 0.5$, resulting in four smaller Bézier triangles. This requires the evaluation of $S(0.5, 0.5, 0)$, $S(0, 0.5, 0.5)$, and $S(0.5, 0, 0.5)$, which we can project onto B to update $\underline{h}$, this time using the BVH introduced by Garland et al. [33] and the uniform grid.

If quads are split into triangles, the same matching described in Section 5.3.2 can be used. Thus, we only need to figure out for the case where A_k's are Bézier triangles. The classification into three cases remains the same, but the matching and upper bounding are fairly different. Let $\mathbf{q}_i$'s be the control points that are not $\mathbf{p}_i$'s—the three vertices—meaning that A_k can be expressed as the following, where $0 \leq u, v, w \leq 1$ and $u + v + w = 1$.

$$\begin{aligned} A_k &= \triangle\!\!\!\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3\mathbf{q}_1\mathbf{q}_2\mathbf{q}_3 \\ &= \mathbf{p}_1u^2 + \mathbf{p}_2v^2 + \mathbf{p}_3w^2 + 2\mathbf{q}_1uv + 2\mathbf{q}_2vw + 2\mathbf{q}_3wu \end{aligned} \tag{5.24}$$

a. If all $\mathbf{r}_i$'s are on the same triangle T, we match A_k with $\triangle\mathbf{r}_1\mathbf{r}_2\mathbf{r}_3$. To find a good local upper bound, we represent B_k (a subtriangle of T) as a Bézier triangle.

$$\begin{aligned} B_k(u, v, w) &= \mathbf{r}_1u + \mathbf{r}_2v + \mathbf{r}_3w \\ &= (\mathbf{r}_1u + \mathbf{r}_2v + \mathbf{r}_3w)(u + v + w) \\ &= \mathbf{r}_1u^2 + \mathbf{r}_2v^2 + \mathbf{r}_3w^2 \\ &\quad + (\mathbf{r}_1 + \mathbf{r}_2)uv + (\mathbf{r}_2 + \mathbf{r}_3)vw + (\mathbf{r}_3 + \mathbf{r}_1)wu \end{aligned} \tag{5.25}$$

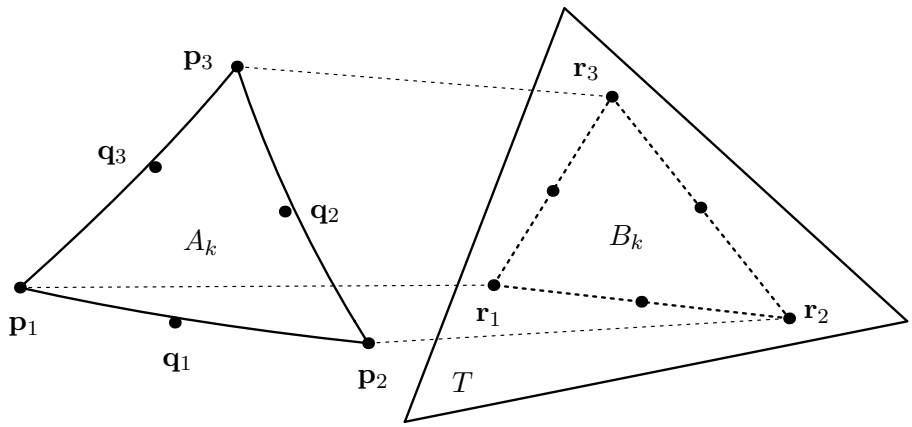


Figure 5.6: We can give one-to-one correspondence between Bézier triangle A_k and triangle B_k, and quickly bound the norm of the difference using the control points.

B_k's control points become $\mathbf{r}_i$'s and $\mathbf{h}_i$'s, where $\mathbf{h}_i = \frac{\mathbf{r}_i + \mathbf{r}_{i+1}}{2}$. With the convex hull property of Bézier surfaces, we can bound the difference between A_k and B_k by simply taking the maximum among the distances between each the six control point pairs (Figure 5.6). This upper bound will be reused in other cases.

$$\begin{aligned} \max_{u,v,w} \|A_k - B_k\| &\leq \max_i (\max \|\mathbf{p}_i - \mathbf{r}_i\|, \|\mathbf{q}_i - \mathbf{h}_i\|) \\ &= \overline{h}_a(\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3\mathbf{q}_1\mathbf{q}_2\mathbf{q}_3, T) \end{aligned} \tag{5.26}$$

b. When $\mathbf{p}_i$'s are projected onto two neighboring triangles T_1 and T_2, we split A_k into three Bézier triangles (Figure 5.7). Let us once again assume that only $\mathbf{r}_3$ is on T_2. For some s and t, we cut A_k along two lines in the parameter domain: $\overline{\mathbf{b}_s\mathbf{b}_t}$ and $\overline{\mathbf{b}_s\mathbf{b}_2}$, where $\mathbf{b}_s = (s, 0, 1-s)$, $\mathbf{b}_t = (0, t, 1-t)$, and $\mathbf{b}_2 = (0, 1, 0)$. We are subdividing two boundary curves

$\widehat{\mathbf{p}_1\mathbf{p}_3}$ and $\widehat{\mathbf{p}_2\mathbf{p}_3}$ by the ratios $1-s:s$ and $1-t:t$. Without diving into detail, we obtain the following Bézier triangles, each with six control points.

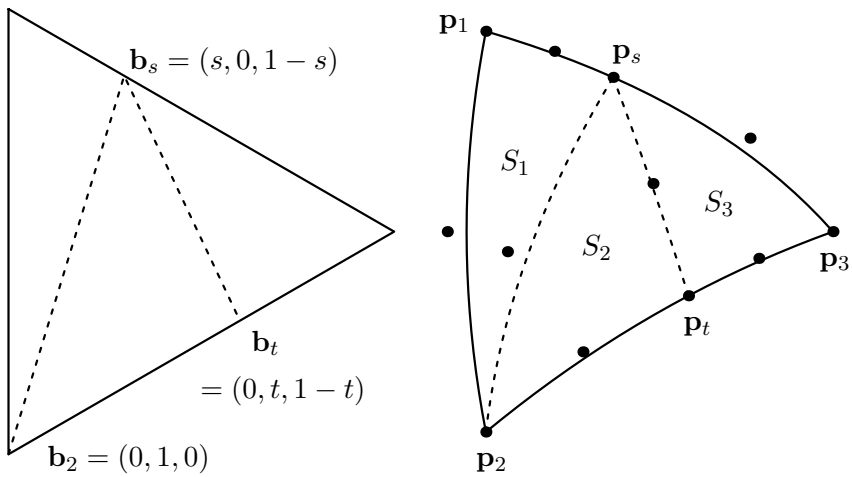
$$\begin{aligned} S_1 &= \triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_s\mathbf{q}_1\mathbf{q}_{2s}\mathbf{q}_{1s} \\ S_2 &= \triangle\mathbf{p}_2\mathbf{p}_t\mathbf{p}_s\mathbf{q}_{2t}\mathbf{q}_{st}\mathbf{q}_{2s} \\ S_3 &= \triangle\mathbf{p}_3\mathbf{p}_s\mathbf{p}_t\mathbf{q}_{3s}\mathbf{q}_{st}\mathbf{q}_{3t} \end{aligned} \tag{5.27}$$

where

$$\begin{aligned} \mathbf{p}_s &= \mathbf{p}_3(1-s)^2 + 2\mathbf{q}_3 s(1-s) + \mathbf{p}_1 s^2 \\ \mathbf{p}_t &= \mathbf{p}_3(1-t)^2 + 2\mathbf{q}_2 t(1-t) + \mathbf{p}_2 t^2 \\ \mathbf{q}_{1s} &= \mathbf{q}_3(1-s) + \mathbf{p}_1 s \\ \mathbf{q}_{2s} &= \mathbf{q}_2(1-s) + \mathbf{q}_1 s \\ \mathbf{q}_{3s} &= \mathbf{p}_3(1-s) + \mathbf{q}_3 s \\ \mathbf{q}_{2t} &= \mathbf{q}_2(1-t) + \mathbf{p}_2 t \\ \mathbf{q}_{3t} &= \mathbf{p}_3(1-t) + \mathbf{q}_2 t \\ \mathbf{q}_{st} &= \mathbf{p}_3(1-s)(1-t) + \mathbf{q}_2(1-s)t + \mathbf{q}_3 s(1-t) + \mathbf{q}_1 st. \end{aligned}$$

As for s and t, since we hope $\mathbf{p}_s$ and $\mathbf{p}_t$ to be close to the bisector of T_1 and T_2, we first measure the distance d_i from $\mathbf{p}_i$ to the bisector plane of the two planes containing T_1 and T_2. Then we set s and t to be

$$s = \frac{d_3}{d_1 + d_3}, \quad t = \frac{d_3}{d_2 + d_3}. \tag{5.28}$$



(a) Parameter domain. (b) Bézier triangles.

Figure 5.7: A_k is subdivided into three Bézier triangles along lines in the parameter domain for separate matching.

We apply $\overline{h}_a$ for each S_i and take the maximum.

$$\begin{aligned}\overline{h}_b(\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3\mathbf{q}_1\mathbf{q}_2\mathbf{q}_3, T_1, T_2) = \max\big(&\overline{h}_a(\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_s\mathbf{q}_1\mathbf{q}_{2s}\mathbf{q}_{1s}, T_1),\\ &\overline{h}_a(\triangle\mathbf{p}_2\mathbf{p}_t\mathbf{p}_s\mathbf{q}_{2t}\mathbf{q}_{st}\mathbf{q}_{2s}, T_1), \overline{h}_a(\triangle\mathbf{p}_3\mathbf{p}_s\mathbf{p}_t\mathbf{q}_{3s}\mathbf{q}_{st}\mathbf{q}_{3t}, T_2)\big)\end{aligned} \tag{5.29}$$

c. For the fallback case that can be applied to any situation, we consider three subtriangles of A_k that cover it and overlap; S_1 takes the subsurface defined over the range $\frac{1}{3} \leq u \leq 1$, while S_2 and S_3 do the same for v and w.

$$S_i = \triangle\mathbf{p}_i\mathbf{p}_{ia}\mathbf{p}_{ib}\mathbf{q}_{ia}\mathbf{q}_{ic}\mathbf{q}_{ib} \tag{5.30}$$

where

$$\begin{aligned}
\mathbf{p}_{ia} &= \frac{1}{9}\mathbf{p}_i + \frac{4}{9}\mathbf{q}_i + \frac{4}{9}\mathbf{p}_{i+1} \\
\mathbf{p}_{ib} &= \frac{1}{9}\mathbf{p}_i + \frac{4}{9}\mathbf{q}_{i+2} + \frac{4}{9}\mathbf{p}_{i+2} \\
\mathbf{q}_{ia} &= \frac{1}{3}\mathbf{p}_i + \frac{2}{3}\mathbf{q}_i \\
\mathbf{q}_{ib} &= \frac{1}{3}\mathbf{p}_i + \frac{2}{3}\mathbf{q}_{i+2} \\
\mathbf{q}_{ic} &= \frac{1}{9}\mathbf{p}_i + \frac{2}{9}\mathbf{q}_i + \frac{2}{9}\mathbf{q}_{i+2} + \frac{4}{9}\mathbf{q}_{i+1}.
\end{aligned}$$

We match S_i to T_i and take the maximum of the resulting upper bounds. But similarly to the adjoining triangles case, we also consider matching A_k as a whole to a T_i. This is another scenario where the computational cost is justified by the immense reduction of the number of pieces pushed into the queue.

$$\begin{aligned}
&\overline{h}_c(\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3\mathbf{q}_1\mathbf{q}_2\mathbf{q}_3, T_1, T_2, T_3) \\
&= \min\big(\max_i \overline{h}_a(\triangle\mathbf{p}_i\mathbf{p}_{ia}\mathbf{p}_{ib}\mathbf{q}_{ia}\mathbf{q}_{ic}\mathbf{q}_{ib}, T_i), \min_i \overline{h}_a(\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3\mathbf{q}_1\mathbf{q}_2\mathbf{q}_3, T_i)\big)
\end{aligned} \tag{5.31}$$

5.5 Two-Sided HD

The full two-sided HD between a triangle mesh and a quad mesh is the maximum of the two one-sided HD's discussed in Sections 5.3 and 5.4. So while consecutive execution of the two algorithms is a valid option for computing the two-sided HD, blending the two can achieve faster performance, because

one of the two directions will benefit from having an increased lower bound, removing candidate pieces even more effectively.

The lower bound is initialized by projecting the vertices of the triangle mesh to the quad mesh, and also the vertices of the quad mesh to the triangle mesh. Pieces of both meshes are pushed to the same queue. Depending on whether the popped piece belongs the triangle mesh or the quad mesh, we apply the appropriate matching and local upper bounding.

The performance gain can be seen in the tables at the end of the chapter. The number of iterations taken in the end for the merged algorithm is always less than the number it would have taken if each one-sided HD were computed separately, the only exception being the sphere example, where the two lower bounds are the same, in which case the number of iterations remain unaffected. For explanations on the tables, refer to the next section.

5.6 Experimental Results

To demonstrate the performance and characteristics of our algorithm, we list the gradual HD computation progress for ten pairs of mesh models. Table 5.1 summarizes the number of vertices and faces in the models used. The first example is a sphere tessellated so that π is divided into 50 segments, but one is rotated along the polar axis by $\frac{\pi}{100}$. Naturally, the one-sided HD in either direction is the same. The next four pairs are our quad mesh conversion results. Figure 1.3 shows one example; Figure 1.3(b) is a quad mesh approximation of Figure 1.3(a). Other models are shown in Figure 5.8. The latter five pairs,

	A (Triangle mesh)		B (Quad mesh)	
Model	Verts	Faces	Verts	Faces
Spheres	4900	9600	4900	4800
Bunny Head	9090	17877	7481	7424
Dental Crown	10143	19826	90000	89401
Monster	29430	58614	40000	39601
John	32512	64684	40000	39601
Bimba	74764	149524	53248	46800
Gargoyle	85558	171112	340114	315136
Kitten	134448	268896	32950	28800
Bust	255358	510712	196761	183552
Ramesses	826266	1652528	112654	98496

Table 5.1: The number of vertices and faces of our example models.

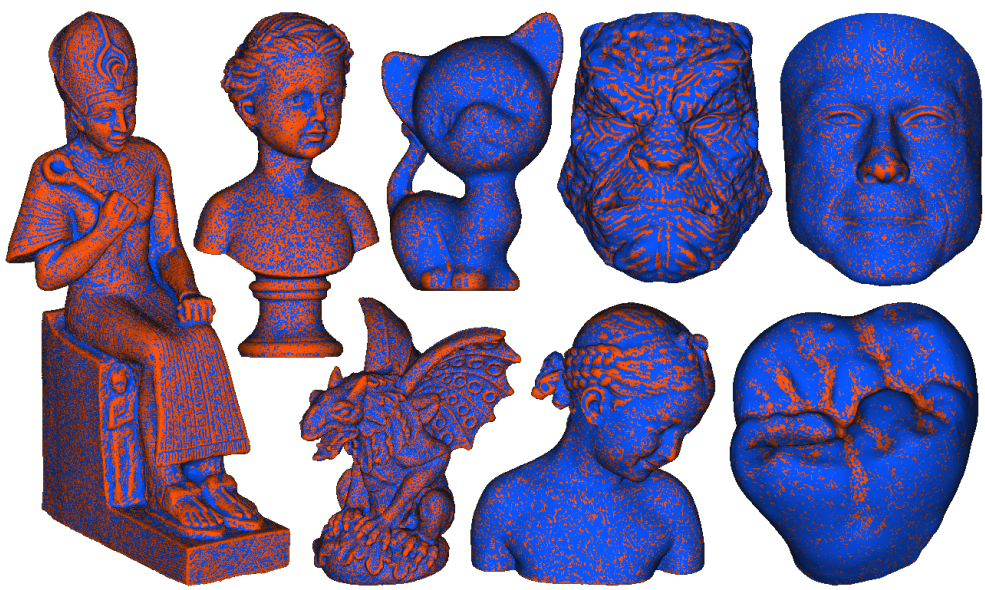


Figure 5.8: Eight superposed triangle mesh and quad mesh model pairs used for our HD algorithm tests.

acquired from the AIM@SHAPE-VISIONAIR Shape Repository[1], are laser scan models quadrangulated by the PGP method introduced by Ray et al. [59]. (We have added additional faces to the gargoyle quad mesh model to fill in the hole in the bottom.)

For all examples, we compute three kinds of HD between the red triangle mesh and the blue quad mesh and show some details for every time a major smaller HD error bound is first reached in logarithmic steps. Tables 5.2 and 5.3 show the results for the same examples, but each quad is treated either as two triangles or as a bilinear surface, respectively. Naturally, the two interpretations lead to slightly different HDs. For each example pair, the triangle mesh-to-quad mesh HD is first computed, followed by the quad mesh-to-triangle mesh HD and the two-sided HD. These three tables are listed on the same page.

In each table, the first column lists 10^{-l}, which means that the difference between the lower bound and the upper bound of the HD has fallen below 10^{-l} of d_A, the diagonal length of A's bounding box. The next two columns show the current lower bound and upper bound at that point, also relative to d_A. The fourth column shows the computation time in milliseconds of when the error bound was reached, averaged over a 100 executions on an Intel i7-8700 machine. The fifth column represents the number of iterations the algorithm has taken, which is the same as the number of times the priority queue has been popped. The first row always shows zero iterations, meaning that the row's error bound has already been reached just after enqueueing the input

[1] http://visionair.ge.imati.cnr.it/ontologies/shapes/

triangles intact, without subdivision. The last column lists the number of triangles enqueued at each moment. Preprocessing (uniform grid construction) time and maximum queue size are also listed.

The results show that we can compute near-zero HD in pure CPU up to very high precision usually within hundreds of milliseconds, thousands for complex models. Taking into account the fact that for most of our examples the HD is less than 10^{-2} of the model size, we can easily claim interactive-time performance for more loose circumstances. Furthermore, the tendency is that after a major bottleneck, reaching smaller error bound does not take too much time, meaning that since most of the candidates are eliminated, the algorithm now approaches the precise HD more quickly. (Excessive rows have been omitted in the tables.) Thus, we can choose an arbitrarily small error bound for the HD as much as is allowed by machine precision. This can also be verified from the changes in the queue size. Meanwhile, the maximum queue size never reaches seven digits, which would not cause any run-time memory-related complications.

In some cases, the initial lower bound is never updated to a greater value. The names of such models are emboldened in the tables. It happens because the HD occurs at a vertex in A, not at an interior point, and hence it is found even before the subdivision iteration, although we are unaware of it until the algorithm is terminated. As such, the computation time tends to be shorter, as pieces are discarded relatively early. The bimba and kitten models are two extreme examples where none of the triangles are ever inserted for the triangle mesh-to-quad mesh HD case.

An interesting observation is that for seven model pairs, the queue for the quad mesh-to-triangle mesh HD computation starts much larger than the triangle mesh-to-quad mesh HD or the two-sided HD case and does not shrink easily. This is because the initial $\underline{h}$ is exceptionally small, since—the quad mesh being an approximation of the triangle mesh—the vertices of the quad mesh were generated very close to the triangle mesh. In fact, a vertex being the occurrence of the HD happens for none of the nine quad mesh approximations. However, regardless of the queue size, the algorithm still returns in a quick manner, because only a relatively small number of iterations are required. While it is possible to check and purge the unnecessarily inserted pieces, it would only add to the computation time and rather cause slowdown.

Comparing the computation time for the two quad interpretations, the bilinear surface case takes up to roughly twice as much time as the two triangles case. (The dental crown example, for which the algorithm by chance finds a close lower bound early in the process, is the only exception where the computation times are comparable.) The disadvantage of the former case is twofold. First, due to non-linearity and curved shape, local upper bounding is not as tight as the latter. Secondly, Newton's method used for point projection takes its toll. This is evident in the dental crown example; even though the former case requires less iterations and smaller queue size, it takes longer computation time than the latter. Nevertheless, we are still able to compute the HD within a very small error bound.

5.7 Summary

In this chapter, we have presented an interactive-time algorithm for the computation of the two-sided Hausdorff distance between a triangle mesh and a quad mesh that successfully handle near-zero cases without GPU acceleration, which has never been done for mesh models, due to their combinatorial complexity. This is possible by keeping the candidate queue lightweight, finding a tight upper bound for candidates, exploiting the structural properties of the quad mesh, and accelerating point projection with a uniform grid.

It should be noted that we have not covered other interpretations of the quad mesh, such as the LN-surface [42, 44], which has the characteristic of being G^1 continuous. Due to their higher degree, adapting our approach to such smooth surfaces would be a challenging task [49] and a possible future work.

Lastly, we would like to shortly add how it may be possible to tweak our general framework for the computation of the Fréchet distance [56], which can roughly be described as the infimum of the maximum distance between two simultaneously parameterized objects. By keeping and updating an upper bound for the Fréchet distance and examining smaller pieces of the models, if a piece in its entirety is known to have greater distance than the current upper bound, it can be removed from further consideration. Of course, this is only speculative and requires much more detail.

Spheres			Prep(ms) 18		Max 18951
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0001426	0.0090776	11	0	9600
10^{-3}	0.0001426	0.0011424	72	13300	18800
10^{-4}	0.0001426	0.0002422	114	23048	11851
10^{-5}	0.0001426	0.0001510	149	32302	2900
10^{-6}	0.0001426	0.0001430	159	34802	800
10^{-7}	0.0001426	0.0001427	161	35402	200
10^{-8}	0.0001426	0.0001426	162	35602	0

(a) Triangle mesh-to-quad mesh HD.

Spheres			Prep(ms) 18		Max 19123
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0001426	0.0090776	10	0	9600
10^{-3}	0.0001426	0.0011424	71	14372	18973
10^{-4}	0.0001426	0.0002422	115	25337	12270
10^{-5}	0.0001426	0.0001515	151	35203	3189
10^{-6}	0.0001426	0.0001435	161	38020	934
10^{-7}	0.0001426	0.0001427	164	38783	200
10^{-8}	0.0001426	0.0001426	165	38983	0

(b) Quad mesh-to-triangle mesh HD.

Spheres			Prep(ms) 38		Max 38074
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0001426	0.0090776	22	0	19200
10^{-3}	0.0001426	0.0011424	152	27672	37773
10^{-4}	0.0001426	0.0002422	244	48385	24121
10^{-5}	0.0001426	0.0001515	319	67505	6089
10^{-6}	0.0001426	0.0001435	339	72822	1734
10^{-7}	0.0001426	0.0001427	345	74185	400
10^{-8}	0.0001426	0.0001426	346	74585	0

(c) Two-sided HD.

Table 5.2: HD computation progress where each quad is two adjoined triangles.

Bunny Head			Prep(ms) 11		Max 8039
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0038088	0.0123961	23	0	8031
10^{-3}	0.0038088	0.0048087	43	3597	4538
10^{-4}	0.0038088	0.0039086	63	7666	472
10^{-5}	0.0038118	0.0038217	65	8079	65
10^{-6}	0.0038195	0.0038199	65	8092	53
10^{-7}	0.0038197	0.0038198	65	8098	54
10^{-8}	0.0038198	0.0038198	65	8100	53

(a) Triangle mesh-to-quad mesh HD.

Bunny Head			Prep(ms) 11		Max 12133
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0029455	0.0109272	18	0	11796
10^{-3}	0.0034294	0.0044293	54	7847	5530
10^{-4}	0.0036700	0.0037699	65	10402	3018
10^{-5}	0.0037088	0.0037180	66	10630	2801
10^{-6}	0.0037112	0.0037120	66	10664	2777
10^{-7}	0.0037118	0.0037118	66	10679	2781
10^{-8}	0.0037118	0.0037118	66	10690	2779

(b) Quad mesh-to-triangle mesh HD.

Bunny Head			Prep(ms) 24		Max 17760
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0038088	0.0123961	40	0	17718
10^{-3}	0.0038088	0.0048087	95	10120	8235
10^{-4}	0.0038088	0.0039086	134	17468	907
10^{-5}	0.0038118	0.0038217	138	18259	124
10^{-6}	0.0038195	0.0038199	138	18283	101
10^{-7}	0.0038197	0.0038198	138	18289	102
10^{-8}	0.0038198	0.0038198	138	18291	101

(c) Two-sided HD.

Dental Crown			Prep(ms) 118		Max 54915
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0011498	0.0272993	39	0	19593
10^{-2}	0.0017492	0.0117476	46	757	21864
10^{-3}	0.0021256	0.0031256	369	36512	41563
10^{-4}	0.0021981	0.0022706	656	70548	7777
10^{-5}	0.0022149	0.0022217	676	72930	5399
10^{-6}	0.0022214	0.0022217	676	72936	5401
10^{-7}	0.0022216	0.0022217	676	72946	5402
10^{-8}	0.0022217	0.0022217	676	72947	5401

(a) Triangle mesh-to-quad mesh HD.

Dental Crown			Prep(ms) 26		Max 9832
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0019490	0.0058701	139	0	9832
10^{-3}	0.0019490	0.0029484	147	1598	8288
10^{-4}	0.0021265	0.0022067	171	6445	3512
10^{-5}	0.0022027	0.0022067	171	6448	3512
10^{-6}	0.0022065	0.0022067	171	6452	3512
10^{-7}	0.0022067	0.0022067	171	6456	3512
10^{-8}	0.0022067	0.0022067	171	6459	3511

(b) Quad mesh-to-triangle mesh HD.

Dental Crown			Prep(ms) 140		Max 64271
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0019490	0.0272993	178	0	29043
10^{-2}	0.0019490	0.0119476	186	715	31188
10^{-3}	0.0021256	0.0031256	503	37665	49876
10^{-4}	0.0021981	0.0022706	810	76336	11490
10^{-5}	0.0022149	0.0022217	833	79222	8608
10^{-6}	0.0022214	0.0022217	833	79228	8610
10^{-7}	0.0022216	0.0022217	833	79238	8611
10^{-8}	0.0022217	0.0022217	833	79239	8610

(c) Two-sided HD.

Monster			Prep(ms) 46		Max 15
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0101105	0.0297328	82	0	11
10^{-2}	0.0101105	0.0194350	82	3	15
10^{-3}	0.0101105	0.0110357	82	13	5
10^{-6}	0.0101105	0.0101114	82	17	1
10^{-8}	0.0101105	0.0101105	82	18	0

(a) Triangle mesh-to-quad mesh HD.

Monster			Prep(ms) 31		Max 1840
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0035145	0.0079671	73	0	1836
10^{-3}	0.0035145	0.0045121	75	260	1620
10^{-4}	0.0038128	0.0039127	80	801	1119
10^{-5}	0.0038585	0.0038679	80	882	1052
10^{-6}	0.0038633	0.0038637	81	900	1045
10^{-7}	0.0038635	0.0038636	81	909	1048
10^{-8}	0.0038636	0.0038636	81	917	1048

(b) Quad mesh-to-triangle mesh HD.

Monster			Prep(ms) 75		Max 15
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0101105	0.0297328	152	0	11
10^{-2}	0.0101105	0.0194350	152	3	15
10^{-3}	0.0101105	0.0110357	152	13	5
10^{-6}	0.0101105	0.0101114	152	17	1
10^{-8}	0.0101105	0.0101105	152	18	0

(c) Two-sided HD.

John			Prep(ms) 44		Max 81
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0077911	0.0093646	74	0	81
10^{-3}	0.0077911	0.0087466	74	4	77
10^{-4}	0.0077911	0.0078767	74	58	23
10^{-5}	0.0077911	0.0077993	75	80	1
10^{-8}	0.0077911	0.0077911	75	81	0

(a) Triangle mesh-to-quad mesh HD.

John			Prep(ms) 32		Max 27825
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0017710	0.0085868	85	0	27356
10^{-3}	0.0024637	0.0033307	97	1811	26579
10^{-4}	0.0026584	0.0027092	123	5845	22568
10^{-5}	0.0026734	0.0026818	127	6291	22136
10^{-6}	0.0026762	0.0026769	127	6364	22091
10^{-7}	0.0026765	0.0026765	128	6382	22093
10^{-8}	0.0026765	0.0026765	128	6394	22087

(b) Quad mesh-to-triangle mesh HD.

John			Prep(ms) 72		Max 93
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0077911	0.0093646	144	0	93
10^{-3}	0.0077911	0.0087466	145	4	89
10^{-4}	0.0077911	0.0078782	145	64	29
10^{-5}	0.0077911	0.0077993	145	92	1
10^{-8}	0.0077911	0.0077911	145	93	0

(c) Two-sided HD.

Bimba			Prep(ms) 62		Max 0
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-8}	0.0083136	0.0083136	210	0	0

(a) Triangle mesh-to-quad mesh HD.

Bimba			Prep(ms) 95		Max 92574
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0000007	0.0197031	154	0	92548
10^{-2}	0.0065921	0.0151946	154	3	92557
10^{-3}	0.0068473	0.0078386	156	151	92458
10^{-4}	0.0071101	0.0072076	160	424	92235
10^{-5}	0.0071180	0.0071273	163	553	92229
10^{-6}	0.0071209	0.0071219	165	627	92248
10^{-7}	0.0071212	0.0071213	168	701	92263
10^{-8}	0.0071212	0.0071212	170	784	92240

(b) Quad mesh-to-triangle mesh HD.

Bimba			Prep(ms) 156		Max 63
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0083136	0.0197031	305	0	56
10^{-2}	0.0083136	0.0182074	306	1	59
10^{-3}	0.0083136	0.0092779	306	30	42
10^{-4}	0.0083136	0.0084100	307	65	7
10^{-5}	0.0083136	0.0083209	307	71	1
10^{-8}	0.0083136	0.0083136	307	72	0

(c) Two-sided HD.

Gargoyle			Prep(ms) 318		Max 1210
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0083843	0.0492421	300	0	814
10^{-2}	0.0083843	0.0183631	302	124	1175
10^{-3}	0.0083843	0.0093835	308	866	493
10^{-4}	0.0083843	0.0084823	311	1303	56
10^{-5}	0.0083843	0.0083930	311	1353	6
10^{-6}	0.0083843	0.0083851	312	1358	1
10^{-8}	0.0083843	0.0083843	312	1359	0

(a) Triangle mesh-to-quad mesh HD.

Gargoyle			Prep(ms) 127		Max 80310
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0009520	0.0293209	586	0	80299
10^{-2}	0.0098079	0.0193221	586	4	80310
10^{-3}	0.0107733	0.0114654	587	65	80265
10^{-4}	0.0109463	0.0110453	588	81	80265
10^{-5}	0.0109571	0.0109648	588	98	80272
10^{-6}	0.0109598	0.0109605	589	109	80274
10^{-7}	0.0109601	0.0109601	589	125	80288
10^{-8}	0.0109601	0.0109601	589	136	80283

(b) Quad mesh-to-triangle mesh HD.

Gargoyle			Prep(ms) 434		Max 1164
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0083843	0.0492421	826	0	940
10^{-2}	0.0098079	0.0197802	827	93	1163
10^{-3}	0.0107733	0.0114654	831	387	896
10^{-4}	0.0109463	0.0110459	832	457	842
10^{-5}	0.0109571	0.0109648	832	485	838
10^{-6}	0.0109598	0.0109605	833	496	840
10^{-7}	0.0109601	0.0109601	833	512	854
10^{-8}	0.0109601	0.0109601	834	523	849

(c) Two-sided HD.

Kitten			Prep(ms) 57		Max 0
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-8}	0.0052897	0.0052897	287	0	0

(a) Triangle mesh-to-quad mesh HD.

Kitten			Prep(ms) 152		Max 57737
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0000008	0.0245412	101	0	57600
10^{-2}	0.0047498	0.0146965	101	13	57639
10^{-3}	0.0047498	0.0057490	111	1211	56673
10^{-4}	0.0050437	0.0051436	123	3083	54834
10^{-5}	0.0050872	0.0050970	125	3264	54667
10^{-6}	0.0050933	0.0050942	126	3308	54680
10^{-7}	0.0050937	0.0050938	127	3334	54682
10^{-8}	0.0050937	0.0050937	128	3366	54697

(b) Quad mesh-to-triangle mesh HD.

Kitten			Prep(ms) 206		Max 2417
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0052897	0.0245412	348	0	2331
10^{-2}	0.0052897	0.0146965	348	13	2370
10^{-3}	0.0052897	0.0062882	352	474	2035
10^{-4}	0.0052897	0.0053896	364	2153	360
10^{-5}	0.0052897	0.0052996	366	2470	43
10^{-6}	0.0052897	0.0052906	366	2510	3
10^{-8}	0.0052897	0.0052897	366	2513	0

(c) Two-sided HD.

Bust			Prep(ms) 223		Max 2156
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0018266	0.0074853	624	0	2125
10^{-3}	0.0018266	0.0028241	624	37	2135
10^{-4}	0.0018266	0.0019265	631	826	1347
10^{-5}	0.0018266	0.0018365	640	1975	198
10^{-6}	0.0018266	0.0018276	642	2151	22
10^{-7}	0.0018266	0.0018266	642	2172	1
10^{-8}	0.0018266	0.0018266	642	2173	0

(a) Triangle mesh-to-quad mesh HD.

Bust			Prep(ms) 234		Max 352270
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0003887	0.0072438	669	0	351949
10^{-3}	0.0016799	0.0026799	679	1419	351468
10^{-4}	0.0017371	0.0018371	774	14706	338248
10^{-5}	0.0017958	0.0017991	790	16690	336278
10^{-6}	0.0017964	0.0017970	791	16840	336150
10^{-7}	0.0017968	0.0017969	792	16851	336152
10^{-8}	0.0017969	0.0017969	792	16860	336150

(b) Quad mesh-to-triangle mesh HD.

Bust			Prep(ms) 454		Max 16960
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0018266	0.0074853	1034	0	16775
10^{-3}	0.0018266	0.0028264	1042	1091	16273
10^{-4}	0.0018266	0.0019266	1121	11767	5630
10^{-5}	0.0018266	0.0018366	1160	16715	684
10^{-6}	0.0018266	0.0018276	1166	17336	65
10^{-7}	0.0018266	0.0018267	1166	17395	6
10^{-8}	0.0018266	0.0018266	1166	17401	0

(c) Two-sided HD.

Ramesses			Prep(ms) 259		Max 2
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-4}	0.0254824	0.0255508	1954	0	2
10^{-6}	0.0255416	0.0255426	1955	7	2
10^{-7}	0.0255417	0.0255418	1956	20	2
10^{-8}	0.0255418	0.0255418	1957	27	1

(a) Triangle mesh-to-quad mesh HD.

Ramesses			Prep(ms) 1004		Max 196601
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0008285	0.0130428	567	0	196443
10^{-2}	0.0014685	0.0108792	567	4	196455
10^{-3}	0.0039470	0.0049444	575	313	196381
10^{-4}	0.0044145	0.0045129	580	560	196145
10^{-5}	0.0044359	0.0044404	582	630	196104
10^{-6}	0.0044368	0.0044377	583	640	196106
10^{-7}	0.0044371	0.0044371	583	649	196115
10^{-8}	0.0044371	0.0044371	584	656	196111

(b) Quad mesh-to-triangle mesh HD.

Ramesses			Prep(ms) 897		Max 2
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-4}	0.0254824	0.0255508	2220	0	2
10^{-6}	0.0255416	0.0255426	2221	7	2
10^{-7}	0.0255417	0.0255418	2222	20	2
10^{-8}	0.0255418	0.0255418	2223	27	1

(c) Two-sided HD.

Spheres			Prep(ms) 18		Max 18951
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0001426	0.0124781	19	0	9600
10^{-2}	0.0001426	0.0085280	108	9200	13800
10^{-3}	0.0001426	0.0010670	432	49200	13600
10^{-4}	0.0001426	0.0002414	665	76800	11600
10^{-5}	0.0001426	0.0001523	761	88400	4000
10^{-6}	0.0001426	0.0001435	789	91600	1400
10^{-7}	0.0001426	0.0001427	797	92600	400
10^{-8}	0.0001426	0.0001426	863	101000	0

(a) Triangle mesh-to-quad mesh HD.

Spheres			Prep(ms) 16		Max 11056
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0001426	0.0090776	12	0	9600
10^{-3}	0.0001426	0.0011376	82	13253	10447
10^{-4}	0.0001426	0.0002422	107	17987	9332
10^{-5}	0.0001426	0.0001510	175	30810	2313
10^{-6}	0.0001426	0.0001430	184	32519	832
10^{-8}	0.0001426	0.0001426	191	33831	0

(b) Quad mesh-to-triangle mesh HD.

Spheres			Prep(ms) 59		Max 24856
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0001426	0.0124781	32	0	19200
10^{-2}	0.0001426	0.0090776	121	9200	23400
10^{-3}	0.0001426	0.0011376	522	62453	24047
10^{-4}	0.0001426	0.0002422	784	94787	20932
10^{-5}	0.0001426	0.0001523	953	119210	6313
10^{-6}	0.0001426	0.0001435	991	124119	2232
10^{-7}	0.0001426	0.0001427	1007	126431	400
10^{-8}	0.0001426	0.0001426	1072	134831	0

(c) Two-sided HD.

Table 5.3: HD computation progress where each quad is a bilinear surface.

Bunny Head			Prep(ms) 8		Max 8649
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0038850	0.0183101	42	0	8642
10^{-2}	0.0038850	0.0136974	42	9	8648
10^{-3}	0.0038850	0.0048850	100	5519	3407
10^{-4}	0.0038850	0.0039850	130	8575	394
10^{-5}	0.0039350	0.0039383	132	8773	201
10^{-6}	0.0039350	0.0039357	132	8790	197
10^{-7}	0.0039354	0.0039354	132	8798	195
10^{-8}	0.0039354	0.0039354	132	8804	194

(a) Triangle mesh-to-quad mesh HD.

Bunny Head			Prep(ms) 12		Max 13736
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0029455	0.0131963	22	0	12732
10^{-2}	0.0029455	0.0125425	22	2	12738
10^{-3}	0.0034294	0.0044292	78	9727	7033
10^{-4}	0.0036700	0.0037699	92	12300	4484
10^{-5}	0.0037117	0.0037211	94	12562	4237
10^{-6}	0.0037139	0.0037149	94	12608	4222
10^{-7}	0.0037140	0.0037141	94	12635	4227
10^{-8}	0.0037140	0.0037140	94	12662	4235

(b) Quad mesh-to-triangle mesh HD.

Bunny Head			Prep(ms) 20		Max 19342
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0038850	0.0183101	63	0	19213
10^{-2}	0.0038850	0.0136974	63	9	19219
10^{-3}	0.0038850	0.0048850	175	13790	6911
10^{-4}	0.0038850	0.0039850	226	19926	822
10^{-5}	0.0039350	0.0039383	229	20320	433
10^{-6}	0.0039350	0.0039360	229	20346	420
10^{-7}	0.0039354	0.0039354	229	20363	409
10^{-8}	0.0039354	0.0039354	229	20370	407

(c) Two-sided HD.

Dental Crown			Prep(ms) 82		Max 34974
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0012746	0.0396296	58	0	17655
10^{-2}	0.0021803	0.0121790	75	1186	20478
10^{-3}	0.0021803	0.0031803	474	32527	24930
10^{-4}	0.0022222	0.0023222	749	53981	7187
10^{-5}	0.0022475	0.0022515	783	56693	4743
10^{-6}	0.0022499	0.0022503	784	56739	4712
10^{-7}	0.0022500	0.0022501	784	56753	4708
10^{-8}	0.0022501	0.0022501	784	56761	4707

(a) Triangle mesh-to-quad mesh HD.

Dental Crown			Prep(ms) 26		Max 13023
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0019490	0.0070856	175	0	13021
10^{-3}	0.0019490	0.0029485	196	3162	10172
10^{-4}	0.0021265	0.0022067	234	9434	4020
10^{-5}	0.0022027	0.0022067	234	9440	4020
10^{-6}	0.0022065	0.0022067	234	9444	4020
10^{-7}	0.0022067	0.0022067	234	9448	4020
10^{-8}	0.0022067	0.0022067	234	9451	4019

(b) Quad mesh-to-triangle mesh HD.

Dental Crown			Prep(ms) 108		Max 46394
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0019490	0.0396296	230	0	29181
10^{-2}	0.0021803	0.0121790	246	1186	31934
10^{-3}	0.0021803	0.0031803	660	34658	34379
10^{-4}	0.0022222	0.0023222	988	62068	10715
10^{-5}	0.0022475	0.0022515	1030	65570	7486
10^{-6}	0.0022499	0.0022503	1030	65625	7446
10^{-7}	0.0022500	0.0022501	1030	65639	7442
10^{-8}	0.0022501	0.0022501	1030	65647	7441

(c) Two-sided HD.

Monster			Prep(ms) 31		Max 215
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0101105	0.0188441	140	0	210
10^{-3}	0.0101105	0.0110712	140	72	149
10^{-4}	0.0101105	0.0102074	142	187	34
10^{-5}	0.0101105	0.0101148	142	216	5
10^{-8}	0.0101105	0.0101105	142	221	0

(a) Triangle mesh-to-quad mesh HD.

Monster			Prep(ms) 31		Max 4965
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0035145	0.0069647	92	0	4965
10^{-3}	0.0035145	0.0045139	100	758	4281
10^{-4}	0.0038197	0.0039197	111	2273	2793
10^{-5}	0.0038585	0.0038684	113	2526	2562
10^{-6}	0.0038633	0.0038643	114	2556	2552
10^{-7}	0.0038635	0.0038636	114	2576	2552
10^{-8}	0.0038636	0.0038636	114	2593	2548

(b) Quad mesh-to-triangle mesh HD.

Monster			Prep(ms) 61		Max 215
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0101105	0.0188441	224	0	210
10^{-3}	0.0101105	0.0110712	225	72	149
10^{-4}	0.0101105	0.0102074	226	187	34
10^{-5}	0.0101105	0.0101148	227	216	5
10^{-8}	0.0101105	0.0101105	227	221	0

(c) Two-sided HD.

John			Prep(ms) 29		Max 681
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0077911	0.0117977	138	0	681
10^{-3}	0.0077911	0.0087875	140	274	407
10^{-4}	0.0077911	0.0078910	144	625	56
10^{-5}	0.0077911	0.0078008	144	675	6
10^{-8}	0.0077911	0.0077911	144	681	0

(a) Triangle mesh-to-quad mesh HD.

John			Prep(ms) 34		Max 36308
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0017710	0.0085903	108	0	35047
10^{-3}	0.0022905	0.0032904	141	4378	32959
10^{-4}	0.0026366	0.0027097	201	12459	24897
10^{-5}	0.0026455	0.0026553	209	13559	23806
10^{-6}	0.0026499	0.0026508	210	13657	23725
10^{-7}	0.0026503	0.0026504	210	13677	23719
10^{-8}	0.0026503	0.0026504	211	13688	23717

(b) Quad mesh-to-triangle mesh HD.

John			Prep(ms) 59		Max 720
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0077911	0.0117977	222	0	720
10^{-3}	0.0077911	0.0087875	225	274	446
10^{-4}	0.0077911	0.0078910	228	659	61
10^{-5}	0.0077911	0.0078008	229	712	8
10^{-6}	0.0077911	0.0077917	229	719	1
10^{-8}	0.0077911	0.0077911	229	720	0

(c) Two-sided HD.

Bimba			Prep(ms) 43		Max 0
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-8}	0.0087925	0.0087925	375	0	0

(a) Triangle mesh-to-quad mesh HD.

Bimba			Prep(ms) 95		Max 93509
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0000007	0.0223284	177	0	93486
10^{-2}	0.0075865	0.0171428	177	5	93500
10^{-3}	0.0075865	0.0085800	180	291	93245
10^{-4}	0.0076213	0.0077148	187	982	92607
10^{-5}	0.0076499	0.0076596	190	1137	92594
10^{-6}	0.0076520	0.0076530	195	1290	92628
10^{-7}	0.0076523	0.0076524	199	1407	92666
10^{-8}	0.0076523	0.0076523	203	1530	92631

(b) Quad mesh-to-triangle mesh HD.

Bimba			Prep(ms) 138		Max 188
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0087925	0.0223284	492	0	176
10^{-2}	0.0087925	0.0185828	492	3	184
10^{-3}	0.0087925	0.0097719	493	39	161
10^{-4}	0.0087925	0.0088830	495	159	41
10^{-5}	0.0087925	0.0087970	495	196	4
10^{-8}	0.0087925	0.0087925	495	200	0

(c) Two-sided HD.

Gargoyle			Prep(ms) 226		Max 3202
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0084811	0.0734651	509	0	2876
10^{-2}	0.0084811	0.0184455	512	205	3190
10^{-3}	0.0084811	0.0094777	539	2433	1056
10^{-4}	0.0084811	0.0085810	551	3371	128
10^{-5}	0.0084811	0.0084909	552	3489	10
10^{-6}	0.0084811	0.0084816	552	3498	1
10^{-8}	0.0084811	0.0084811	552	3499	0

(a) Triangle mesh-to-quad mesh HD.

Gargoyle			Prep(ms) 126		Max 104020
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0009520	0.0347890	735	0	104005
10^{-2}	0.0098079	0.0194860	735	9	104020
10^{-3}	0.0107733	0.0117441	737	81	103969
10^{-4}	0.0109463	0.0110444	738	114	103969
10^{-5}	0.0109572	0.0109663	739	151	103983
10^{-6}	0.0109598	0.0109608	740	183	103988
10^{-7}	0.0109601	0.0109602	741	208	104004
10^{-8}	0.0109601	0.0109601	743	243	103987

(b) Quad mesh-to-triangle mesh HD.

Gargoyle			Prep(ms) 337		Max 3314
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0084811	0.0734651	1150	0	3030
10^{-2}	0.0098079	0.0197233	1153	187	3303
10^{-3}	0.0107733	0.0117733	1167	1245	2282
10^{-4}	0.0109463	0.0110444	1171	1567	1997
10^{-5}	0.0109572	0.0109663	1173	1644	1971
10^{-6}	0.0109598	0.0109608	1174	1678	1974
10^{-7}	0.0109601	0.0109602	1175	1704	1989
10^{-8}	0.0109601	0.0109601	1176	1739	1972

(c) Two-sided HD.

Kitten			Prep(ms) 41		Max 0
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-8}	0.0052792	0.0052792	566	0	0

(a) Triangle mesh-to-quad mesh HD.

Kitten			Prep(ms) 154		Max 57747
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0000008	0.0241088	113	0	57600
10^{-2}	0.0047498	0.0143626	113	20	57660
10^{-3}	0.0047797	0.0057797	143	3998	53929
10^{-4}	0.0050730	0.0051274	183	9168	48795
10^{-5}	0.0050800	0.0050866	188	9688	48337
10^{-6}	0.0050809	0.0050817	190	9817	48276
10^{-7}	0.0050810	0.0050811	192	9879	48294
10^{-8}	0.0050810	0.0050811	193	9929	48282

(b) Quad mesh-to-triangle mesh HD.

Kitten			Prep(ms) 193		Max 7406
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0052792	0.0241088	641	0	7315
10^{-2}	0.0052792	0.0150974	641	16	7363
10^{-3}	0.0052792	0.0062792	658	2138	5363
10^{-4}	0.0052792	0.0053783	690	6611	897
10^{-5}	0.0052792	0.0052892	696	7427	81
10^{-6}	0.0052792	0.0052800	697	7499	9
10^{-8}	0.0052792	0.0052792	697	7508	0

(c) Two-sided HD.

Bust			Prep(ms) 172		Max 28128
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0017420	0.0103224	1163	0	28081
10^{-3}	0.0017420	0.0027392	1165	190	27985
10^{-4}	0.0017420	0.0018420	1444	21618	6557
10^{-5}	0.0017420	0.0017520	1516	27400	775
10^{-6}	0.0017420	0.0017430	1524	28112	63
10^{-7}	0.0017420	0.0017421	1525	28163	12
10^{-8}	0.0017420	0.0017420	1525	28175	0

(a) Triangle mesh-to-quad mesh HD.

Bust			Prep(ms) 250		Max 352270
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0003887	0.0078691	751	0	351533
10^{-3}	0.0015824	0.0025823	776	3203	350270
10^{-4}	0.0016454	0.0017089	1148	48009	305529
10^{-5}	0.0016592	0.0016668	1208	54875	298676
10^{-6}	0.0016602	0.0016612	1216	55898	297667
10^{-7}	0.0016604	0.0016605	1218	56036	297563
10^{-8}	0.0016604	0.0016604	1218	56086	297537

(b) Quad mesh-to-triangle mesh HD.

Bust			Prep(ms) 412		Max 70643
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-2}	0.0017420	0.0103224	1639	0	70355
10^{-3}	0.0017420	0.0027416	1658	2370	68965
10^{-4}	0.0017420	0.0018420	2192	52147	19205
10^{-5}	0.0017420	0.0017520	2366	69168	2184
10^{-6}	0.0017420	0.0017430	2386	71107	245
10^{-7}	0.0017420	0.0017421	2388	71319	33
10^{-8}	0.0017420	0.0017420	2388	71352	0

(c) Two-sided HD.

Ramesses			Prep(ms) 215		Max 2
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-3}	0.0254824	0.0256320	3560	0	2
10^{-4}	0.0255078	0.0255508	3561	2	2
10^{-5}	0.0255416	0.0255508	3561	7	2
10^{-6}	0.0255416	0.0255426	3561	8	2
10^{-7}	0.0255417	0.0255418	3563	22	2
10^{-8}	0.0255418	0.0255418	3563	27	1

(a) Triangle mesh-to-quad mesh HD.

Ramesses			Prep(ms) 664		Max 197021
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-1}	0.0008285	0.0131896	517	0	196755
10^{-2}	0.0008285	0.0107296	518	6	196773
10^{-3}	0.0029392	0.0039392	541	1594	195732
10^{-4}	0.0032628	0.0033622	572	4183	193189
10^{-5}	0.0032805	0.0032905	581	4714	192735
10^{-6}	0.0032820	0.0032830	591	4923	192821
10^{-7}	0.0032822	0.0032823	616	5365	193126
10^{-8}	0.0032822	0.0032822	699	6807	192745

(b) Quad mesh-to-triangle mesh HD.

Ramesses			Prep(ms) 856		Max 2
10^{-l}	$\underline{h}$	$\overline{h}$	Time	Iter	Size
10^{-3}	0.0254824	0.0256320	3805	0	2
10^{-4}	0.0255078	0.0255508	3805	2	2
10^{-5}	0.0255416	0.0255508	3806	7	2
10^{-6}	0.0255416	0.0255426	3806	8	2
10^{-7}	0.0255417	0.0255418	3807	22	2
10^{-8}	0.0255418	0.0255418	3808	27	1

(c) Two-sided HD.

Chapter 6

Quad Mesh Reconstruction

3D scanners that are used to acquire 3D models from real life objects typically primarily output range sets or point clouds. To make use of this data, Delaunay triangulation or surface reconstruction using moving least squares [1] are available. But triangulated models, lacking regular structure, are at a disadvantage when applying accelerated geometric operations using BVHs.

Although NURBS surfaces are used in computer-aided design and manufacturing, quad meshes are still valuable in other areas. They are also subject to conversion to analytic surfaces. Thus, we present a novel method for the conversion of point clouds to semi-regular quad mesh that approximates the input mesh within a given error bound. The framework can naturally also be applied for use with triangle meshes as input.

We first manually sketch patch boundaries via 2D input such as a mouse or a digitizer, as described in Section 6.1. Various tools to aid in this process

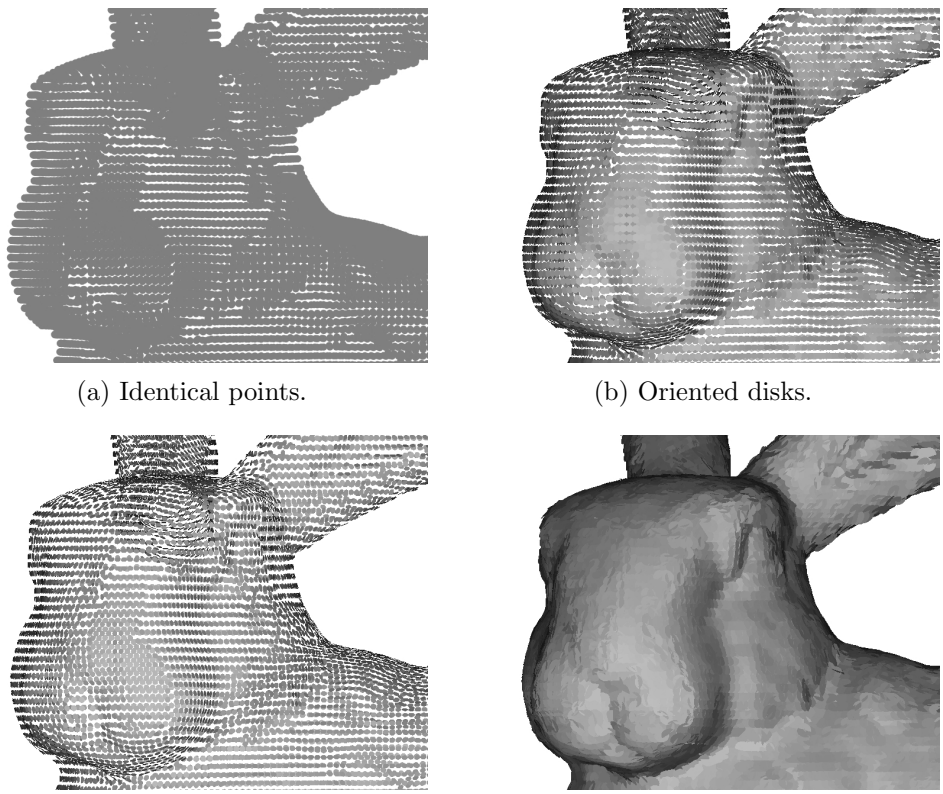
are provided. Then the quad mesh is automatically generated from point data and boundaries, which we assume to be absolute and noiseless (Section 6.3). Section 6.2 discusses point-to-patch assignment that needs to be done in advance.

While most quadrangulation methods require a global parameterization of the entire input mesh beforehand, our method can be described as alternating between parameterization and quadrangle subdivision. Each regular patch is first initialized as a single bilinear surface, which is then subdivided into two repeatedly, with each iteration approximating the input more closely. Parameterization calculated at each step is used for approximation.

6.1 Boundary Curve Sketching

6.1.1 Point Cloud Rendering

To allow ideal placement of quad patches, or quadrangular segmenting of the input data, patch boundaries are sketched over the rendered point cloud. But as shown in Figure 6.1(a), when all points are rendered with the same color and size, it is difficult for the user to discern the shape even when viewed from different angles. Assuming that normal data is obtained from the scanner, we render shaded oriented disks instead. Their radius r_d is critical to the perception of a surface; when it is too small, the disks will still look like points (Figure 6.1(c)), and when it is too large, the disks will overlap, and the shape will appear bloated, hindering correct sketching (Figure 6.1(d)). We suggest $r_d = 0.15 \cdot (\frac{V}{n})^{\frac{1}{3}}$, where V is the volume of the data's bounding box and n is the number of points, but the user is of course allowed to adjust the value.



(a) Identical points. (b) Oriented disks.

(c) Disks too small. (d) Disks too large.

Figure 6.1: Rendering points as oriented disks with appropriate size and shading can greatly aid the designer in grasping the three dimensional shape.

6.1.2 Basic Point Input

From a 2D input on this rendered image, finding the 3D point that lies on the surface is a central element. For triangle meshes, it is a trivial matter; the three corners of a triangle can be interpolated. For other rendered surfaces, with the depth value read from the depth buffer, the 2D point can be "unprojected" into 3D space. However, our reference model is a point cloud, and no correct depth can be inferred from the rendering.

Instead, we find on the fly some triangle to interpolate, assuming the user is trying to draw on the "visible" perceived surface. Even with millions of input points, it can be done in real time by capitalizing on the GPU's capabilities.

1. When window coordinates (x_w, y_w) is given, we first select all of the rendered points in the view, regardless of (x_w, y_w). This is done only once per stroke at the beginning, since the user is not going to change views during a single curve input.

2. We then select the points within a certain radius r_f, in 2D window coordinates' distance, as candidates.

3. Among these candidates, the closest distance d_c to the viewing plane is calculated.

4. Points farther than $d_c + d_f$ from the viewing plane are discarded, and the remaining ones become the final candidates, colored blue in Figure 6.2.

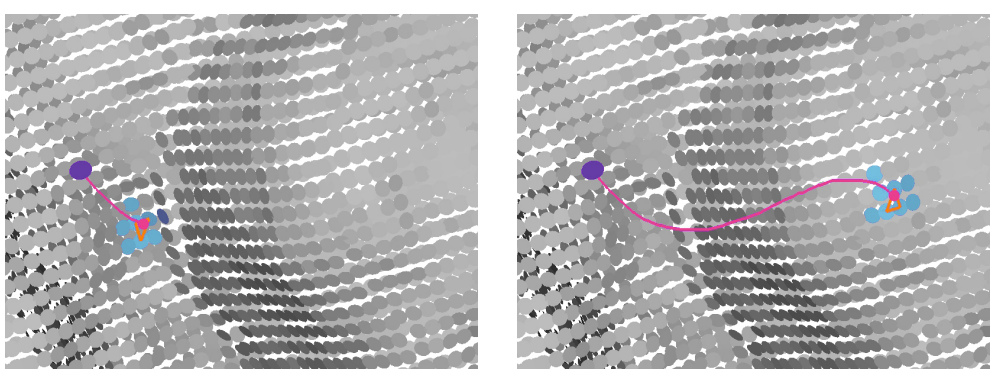


Figure 6.2: Sketching a boundary curve. Gray points located among blue ones are not candidates because they are far away from the viewing direction and not "visible."

r_f and d_f are controlled by the user, so that point data of various density can be handled. Or we may set r_f to be directly proportional to r_d (Section 6.1.1), so that choosing the r_d that best shows the desired surface shape, as in Figure 6.1(b), would inherently make r_f large enough for the method to work correctly, and small enough for it to work efficiently.

Getting back to the candidates, we search these points to find triangles that, when projected to the viewing window, enclose (x_w, y_w). The triangle with the smallest (unprojected original) perimeter is chosen as the desired triangle, drawn in orange in Figure 6.2.

Now suppose its vertices $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ are projected to (x_1, y_1), (x_2, y_2), and (x_3, y_3). There should be barycentric coordinates (u, v, w), where $u+v+w = 1$, that satisfy

$$u(x_1, y_1) + v(x_2, y_2) + w(x_3, y_3) = (x_w, y_w). \tag{6.1}$$

We find these weights (u, v, w) and use the same weights to interpolate the vertices and obtain the final 3D point, shown as a pink dot in Figure 6.2.

$$\mathbf{p}_f = u\mathbf{p}_1 + v\mathbf{p}_2 + w\mathbf{p}_3 \tag{6.2}$$

By connecting a sequence of these input points, we get a boundary curve, represented as a polygonal chain.

6.1.3 Curve Editing

We sketch a set of boundary curves interconnected at endpoints, or *nodes*, to create the quad layout that will be used to generate the quad mesh. Every

irregular vertex in the final quad mesh comes from a node, although not all nodes are irregular. But unless the entire quad layout has been planned out beforehand, simply sketching multiple curves consecutively—perfectly on the first try at that—is not enough to construct a satisfying partitioning of the input data. *This* is the planning, and it requires trial and error, as it is not easy to figure out where to place curve intersections and how to connect them. Thus, we provide the following additional features with our interface to help edit the layout without completely redrawing.

- **Deletion of individual curves.** This basic feature is used to remove unsatisfactory curves and sketch again entirely.
- **Snapping to endpoints.** With human input, two curves will never be able to share endpoints. When an endpoint of a new curve is placed near another, they should be combined by the system to create a node.
- **Merging two curves sharing an endpoint into one.** Due to limited view of the point cloud, not to mention human error, it is often difficult and sometimes impossible to sketch a curve in one stroke. The ability to unify curves created in separate strokes is extremely helpful and a necessity. It can also remove misplaced nodes.
- **Splitting a curve into two at a chosen point.** This makes it possible to insert nodes that are possibly unplanned, but were later deemed necessary, into an existing curve. It can also be used with the deletion feature for partial deletion of a curve.

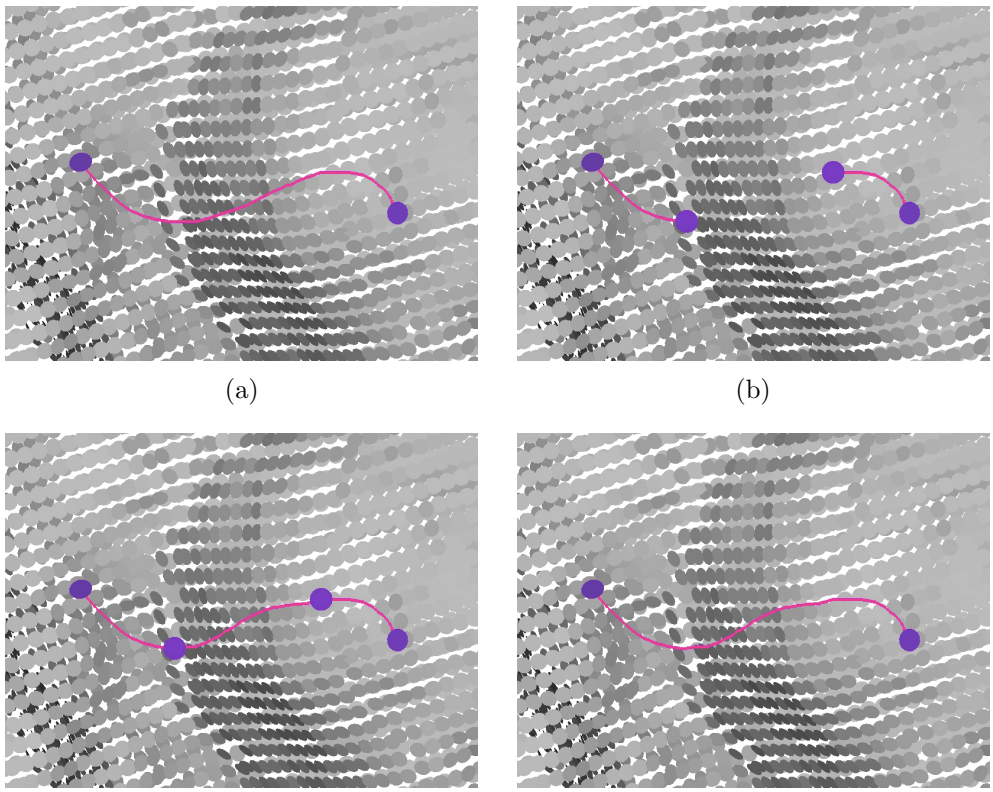
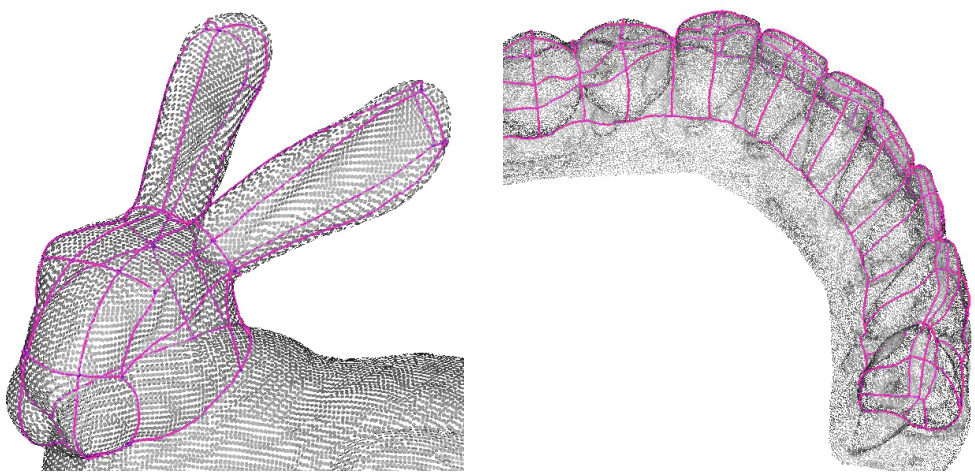


Figure 6.3: Editing a boundary curve. An initial boundary curve (a) is split into three, the second of which is then deleted (b). The missing part is redrawn (c) and merged with the other two to form the final boundary curve (d).

- **Smart merging and splitting.** While splitting and merging can be invoked manually by the user, it is far more convenient for the system to automatically extend a curve as subsequent strokes are added, or insert a node when a user tries to start a new curve in the middle of another curve. Upon creation of two connected curves, if the angle at the node is obtuse enough, i.e., $\cos\theta < -0.85$, we join the two curves into one. Even if it was not the user's intention, it isn't much of a disturbance,

since the user would add another curve to the now deleted node, which we automatically reinsert while snapping to it.

Figure 6.4 shows quad layouts designed with our tool.

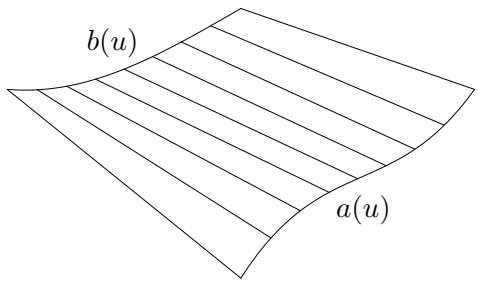


(a) Bunny head. (b) Teeth scan.

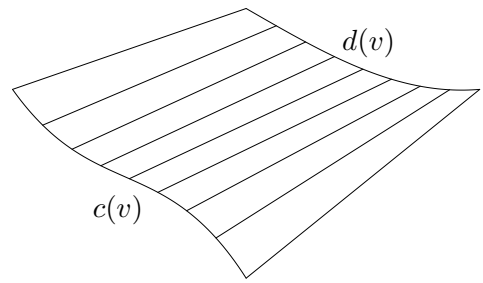
Figure 6.4: Quad layout sketches.

6.2 Patch Assignment

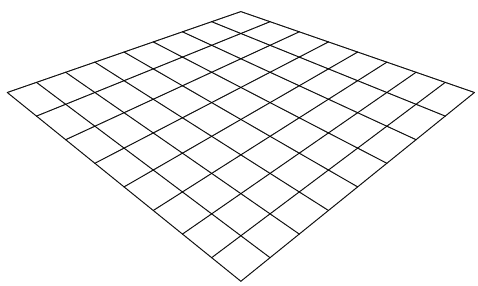
Before quad mesh generation can begin, each input point needs to be assigned to a patch, so that the patches can approximate them. For this purpose, we temporarily create Coons patches. A *Coons patch* is a surface that is completely defined by, and interpolates, its four boundary curves, $a(u)$, $b(u)$, $c(v)$, and $d(v)$, which is all that we currently have. One commonly used form of Coons patch, a bilinearly blended Coons patch, can be described as the sum of two ruled surfaces (linear blend of two curves), subtracted by a bilinear surface $B(u, v)$ defined in Equation (2.1). Assuming the curves satisfy the



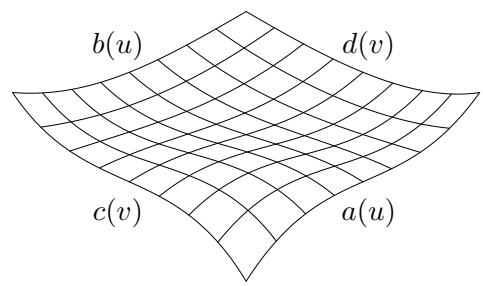
(a) Ruled surface of $a(u)$ and $b(u)$.



(b) Ruled surface of $c(v)$ and $d(v)$.



(c) Bilinear surface $B(u, v)$.



(d) Coons patch $C(u, v)$.

Figure 6.5: The Coons patch can be described as (d)=(a)+(b)-(c).

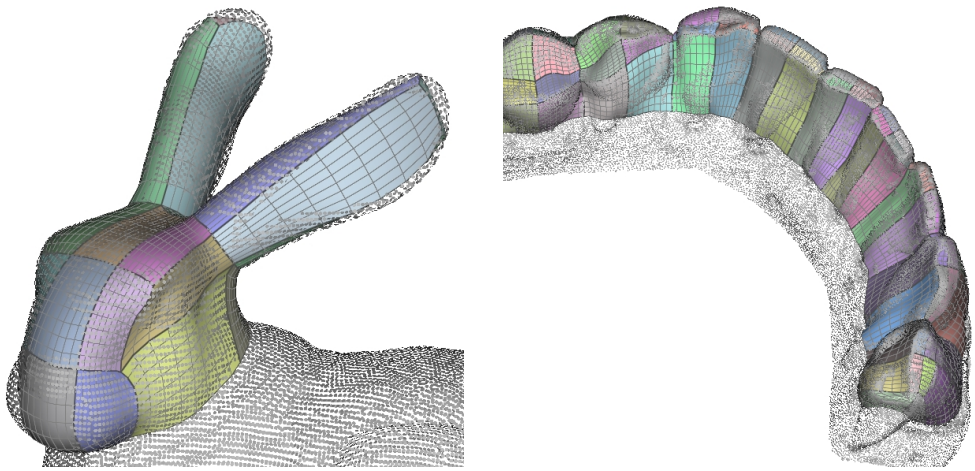
following endpoint conditions,

$$\begin{aligned} b(0) = c(1) = B(0,1) \qquad & a(0) = c(0) = B(0,0) \\ a(1) = d(0) = B(1,0) \qquad & b(1) = d(1) = B(1,1) \end{aligned} \tag{6.3}$$

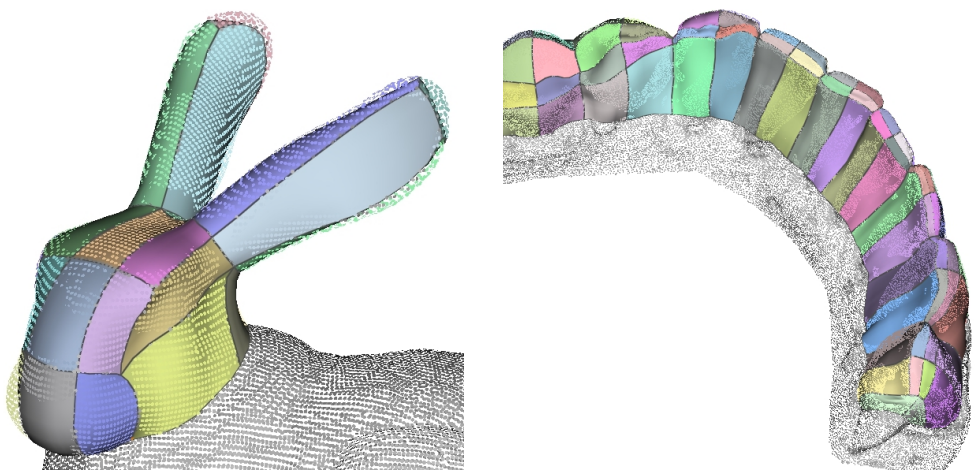
the Coons patch can be expressed as:

$$C(u,v) = (1-v)a(u) + vb(u) + (1-u)c(v) + ud(v) - B(u,v). \tag{6.4}$$

Figure 6.6(a) shows Coons patches generated from our layout in quad mesh form. Notice how they do not approximate the point cloud very well, especially recognizable at the tip of the ears. Yet they are good enough to



(a) Coons patches



(b) Point assignment.

Figure 6.6: Input points are assigned to the closest Coons patch.

find the appropriate patch for each point. We simply perform point projection discussed in Chapter 4 and assign the point to the patch the footpoint belongs to. Each point is colored the same as the patch it is assigned to in Figure 6.6(b). Now the Coons patches are no longer used, and are therefore discarded.

6.3 Iterative Reconstruction

Assuming that the point cloud and the sketched quad layout are absolute, we automatically generate via subdivision a quad mesh that approximates these data. We first describe a case where there is only one patch, as in Figure 6.7. Each subdivision will double either the number of rows n_r or the number of columns n_c. It depends on the length of the boundary curves. If $\frac{L(a(u))+L(b(u))}{n_c} > \frac{L(c(v))+L(d(v))}{n_r}$, where $L(s(t))$ denotes the length of curve $s(t)$, n_c is doubled, and otherwise, n_r.

We begin with a single quad that connects the four corners. It is split into two, with the two new vertices being points on two boundary curves that bisect the curves (Figure 6.7(a)). The second subdivision may double the same n_r or n_c as the first subdivision, in which case new vertices are placed in the same manner. But otherwise, like in Figure 6.7(b), an internal vertex has to be added, along with boundary vertices. To determine its position, we find a parameterization of the input points, as explained below.

For all points assigned to this patch, we recalculate the footpoints to the patch, consisting of two quads at this point. (Previous footpoints to the temporary Coons patch are discarded.) We regard the (u, v) parameters of the footpoint to be the patch parameters of the point itself. Similarly to how a 2D input was converted to 3D in Section 6.1.2, we search the input points for a triangle that contains the parameters $(0.5, 0.5)$, which are the desired parameters in this case, and interpolate it with the same weights that result in $(0.5, 0.5)$ to get the coordinates for the new internal vertex.

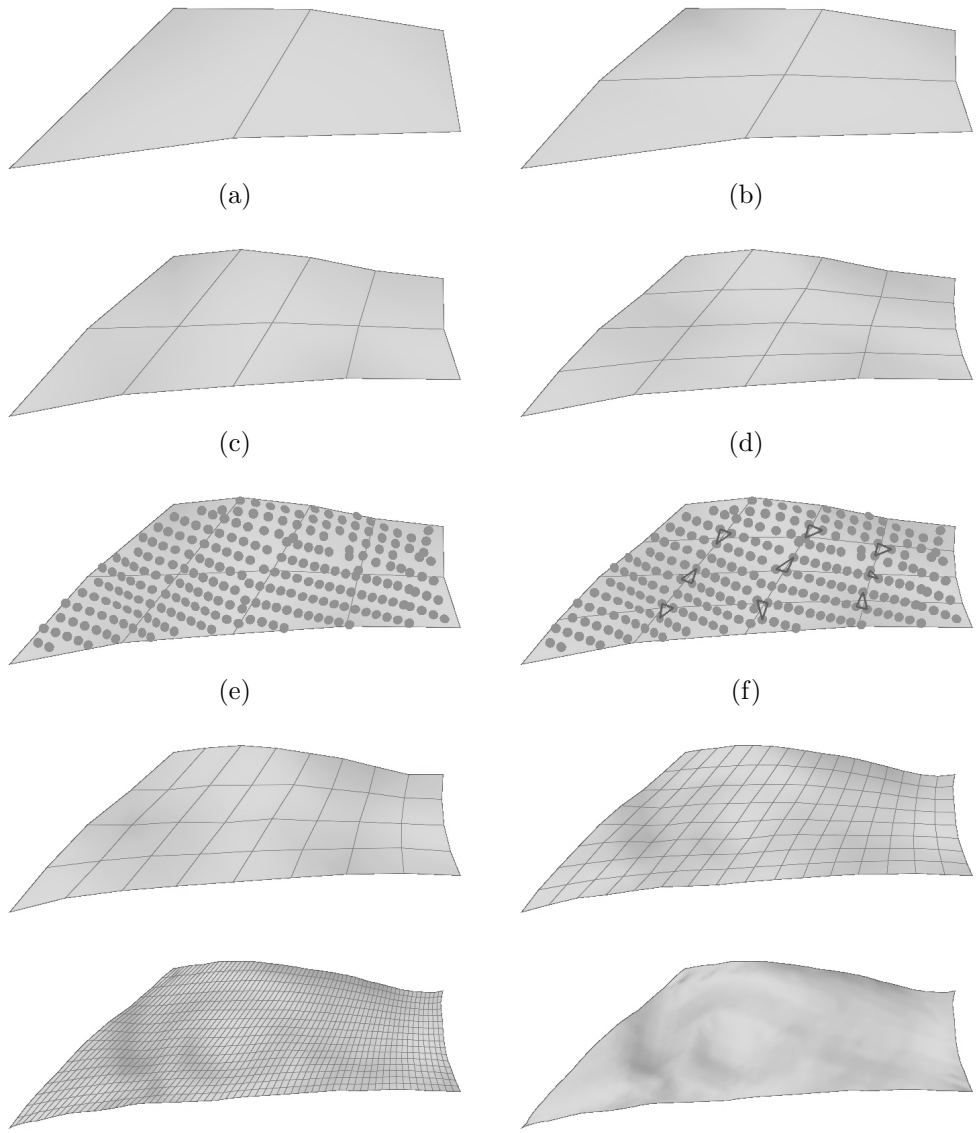


Figure 6.7: Patch subdivision and input approximation process. Subdividing from (c) to (d) searches parameterization of the input points (e) to find triangles containing the desired parameters for all internal vertices, six new and three old (f).

All subsequent subdivisions go through the same process, with updated BVH and an increased number of internal vertices requiring placement, each with different desired parameters. It is not just the new ones, however, since the internal vertices that are already placed also need repositioning; the parameterization changes with each iteration. Otherwise, vertices that were created early will stick out from the rest, diminishing the quality of the quad mesh.

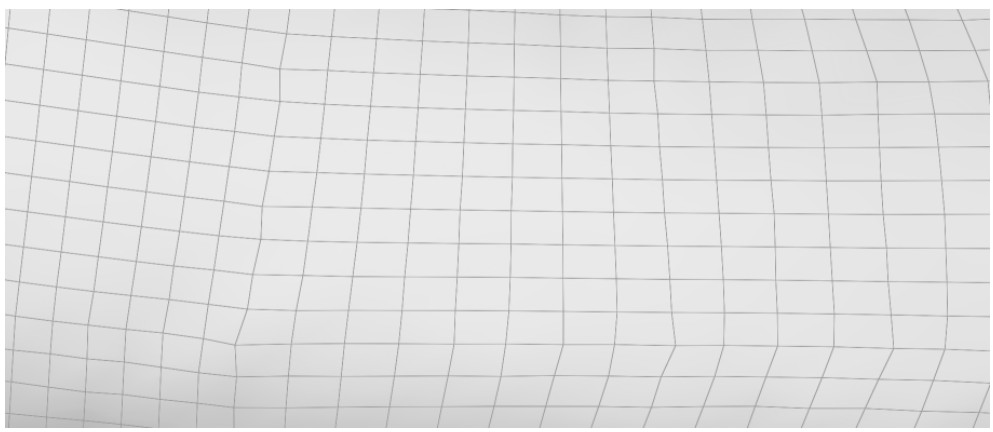


Figure 6.8: An example showing how reconstructed quads can be misaligned when vertices are not updated after initial positioning.

For more complex quad layouts with multiple patches, not all patches are subdivided at the same time. While which patch to split and the direction to split it is decided by the same criterion, subdivision is propagated to neighboring patches along that direction to guarantee the validity of the quad mesh. Vertices at boundary curves will seamlessly share the same coordinates. Subdivision can be iterated until a certain quad size or an approximation error is reached. Hausdorff distance computation discussed in Chapter 5 can be used as an approximation error criterion.

One can see from Figures 6.10 and 6.11 that the final quad mesh shows smoothly positioned vertices and excellent approximation of the input data.

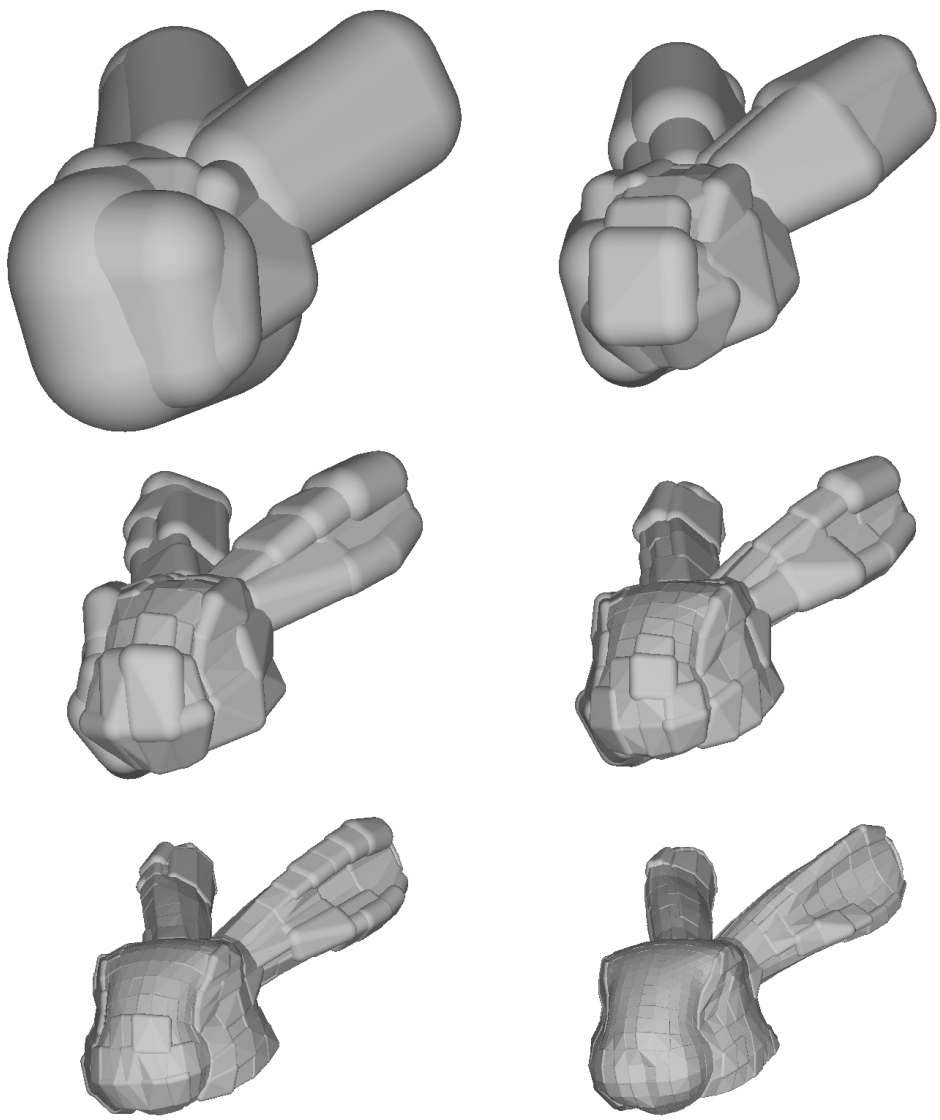


Figure 6.9: The top six levels of the BVH used at one point during quad mesh reconstruction (middle in Figure 6.10) for input point parameterization.

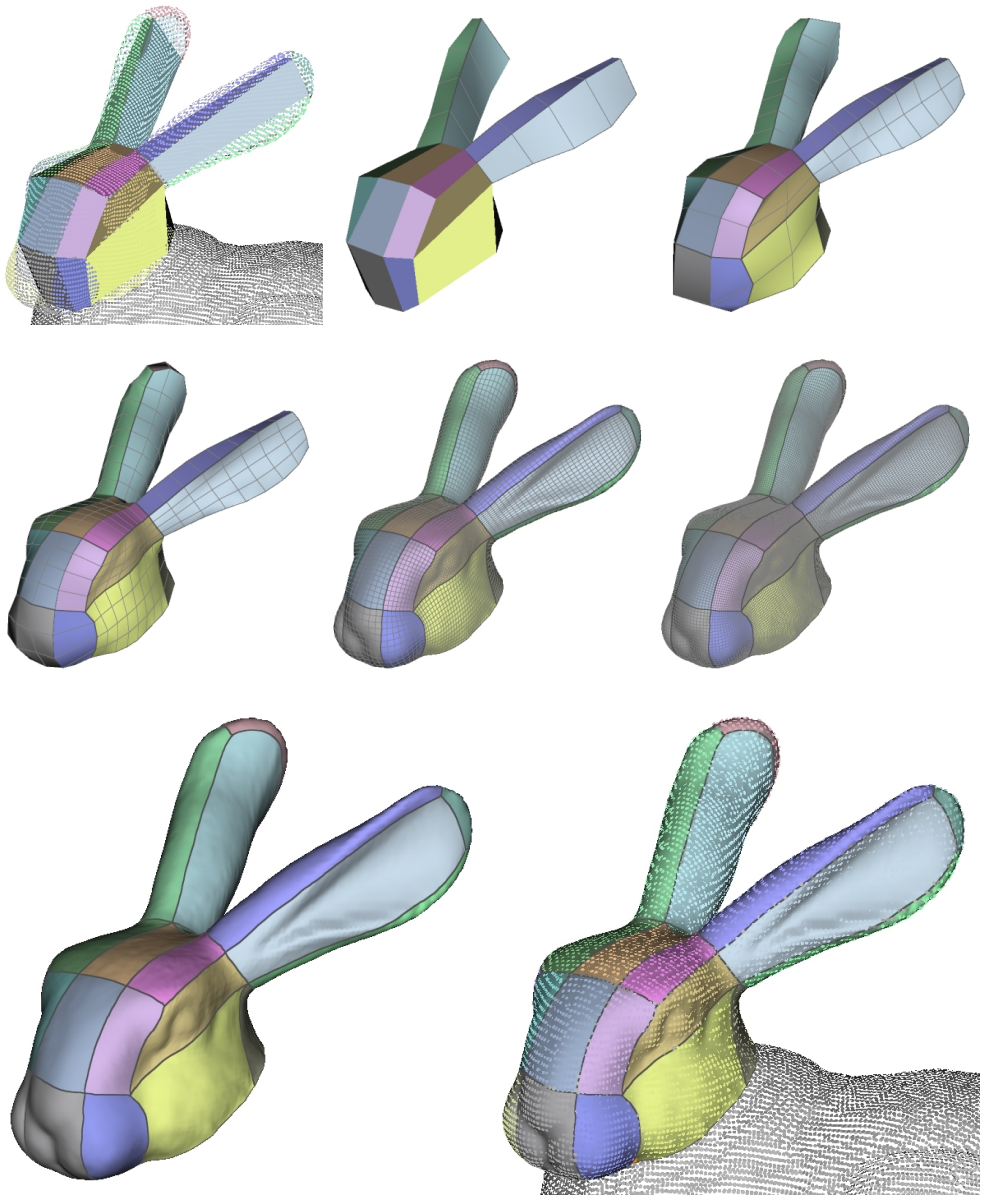


Figure 6.10: A few stages from the quad mesh reconstruction process of the bunny model.

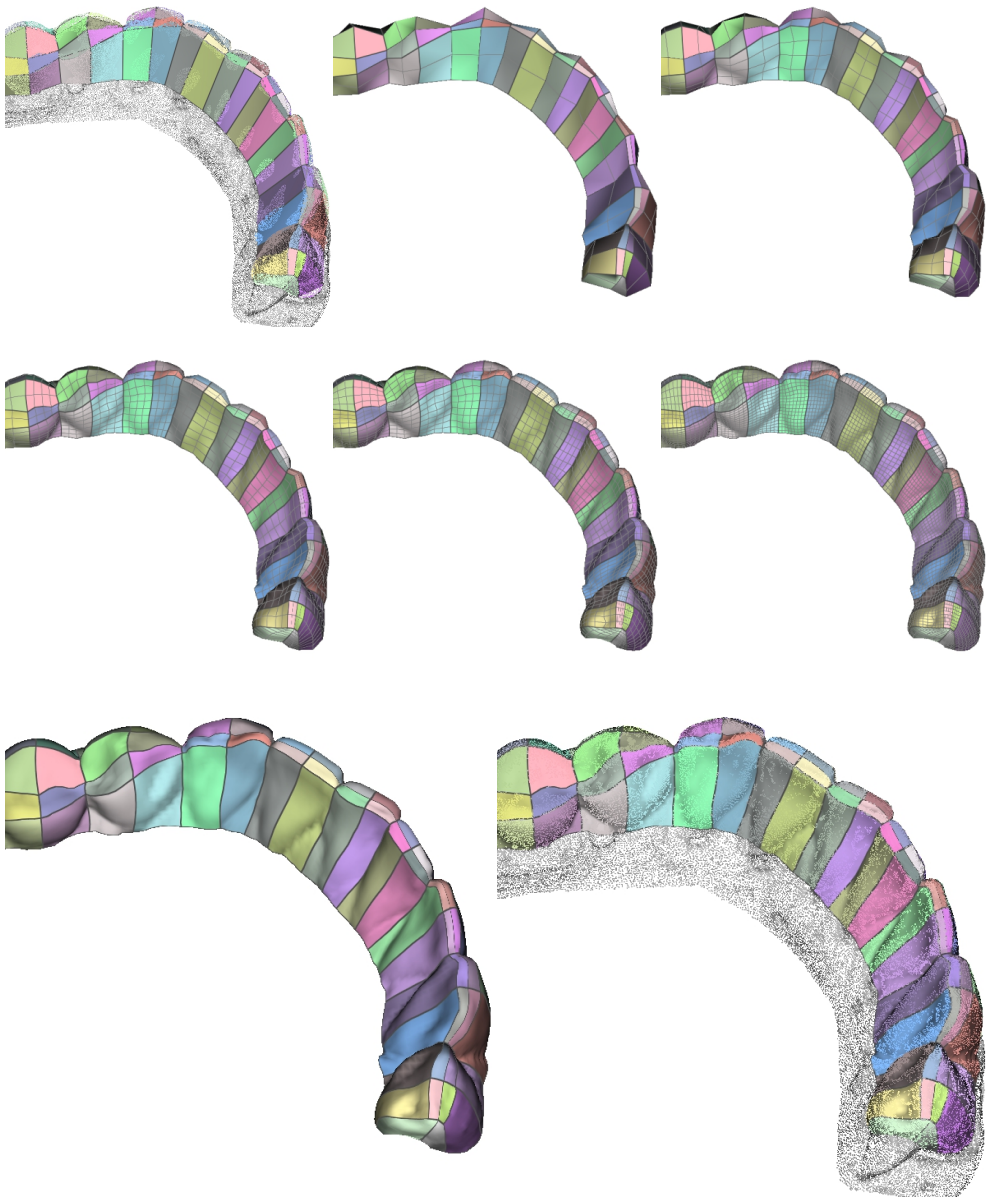


Figure 6.11: A few stages from the quad mesh reconstruction process of the teeth model.

Chapter 7

Conclusion

To encourage widened use of semi-regular quad meshes, we have proposed a simple bounding volume hierarchy for the quad mesh that consists of offset tetrahedra as bounding volumes. It makes possible various geometric operations over the quad mesh in a swift manner. Deformation of the quad mesh is dealt by the BVH with little effort due to the persistence of the hierarchy, only requiring the recalculation of the offset radii. Our recursive formulation of the radii is extremely simple, rendering rapid update of the BVH possible.

At the most basic level, the minimum distance computation from a point, or point projection, has been accelerated with the use of our BVH. This operation is a significant element of our other algorithms: triangle mesh-quad mesh Hausdorff distance computation and quad mesh reconstruction.

For any point on A, the minimum distance to B cannot be greater than A's HD to the B. Thus, we can keep track of a known lower bound for the

HD. We have also introduced a way to estimate a local upper bound for pieces of A. If it is not greater than the lower bound, the piece can be ignored. We partition A and remove redundant parts iteratively to eventually end up with the HD that is accurate within a minuscule error bound.

Lastly, we have presented a workflow to construct a quad mesh from an existing mesh model or a point cloud. By providing our assistive tools and letting the user plan the quad layout, the resulting quad mesh, generated automatically, well preserves feature curves, which also helps to minimize the approximation error. Coons patches are temporarily created to assign input points to patches. We generate the quad mesh by starting with a single large quad for every patch. The current quad mesh structure is considered as a form of parameterization, and the input points are projected to the quad mesh, giving them parameterization. The quad mesh is then subdivided to have double the quads, and all the vertices new and old are positioned to approximate the input data with the help the parameterization. As we repeat this process, the parameterization and the quad mesh is nicely refined.

There is still research potential to be done on the quad mesh, especially regarding the creation of quad meshes. If they were to be generated fully automatically, abundant manual design would have to take place first, perhaps ironically. We hope this work will stimulate the employment of the semi-regular quad mesh.

Bibliography

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.

[2] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 13(3-4):251–265, 1995.

[3] H. Alt, P. Braß, M. Godau, C. Knauer, and C. Wenk. Computing the Hausdorff distance of geometric patterns and shapes. In *Discrete and Computational Geometry*, volume 2 of *Algorithms and Combinatorics 25*, pages 65–76. Springer, 2003.

[4] H. Alt and L. J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In *Handbook of Computational Geometry*, pages 121–153. Elsevier Science B.V., 1999.

[5] H. Alt and L. Scharf. Computing the Hausdorff distance between sets of curves. In *Proceedings of the 20th European Workshop on Computational*

Geometry, pages 233–236, 2004.

[6] H. Alt and L. Scharf. Computing the Hausdorff distance between curved objects. *International Journal of Computational Geometry & Applications*, 18(4):307–320, 2008.

[7] T. Asano, M. Edahiro, H. Imai, M. Iri, and K. Murota. Practical use of bucketing techniques in computational geometry. In G. T. Toussaint, editor, *Computational Geometry*, Machine Intelligence and Pattern Recognition 2, pages 153–195. Elsevier Science Publishers B.V., 1985.

[8] N. Aspert, D. Santa-Cruz, and T. Ebrahimi. MESH: Measuring errors between surfaces using the Hausdorff distance. In *Proceedings of IEEE International Conference on Multimedia and Expo*, pages 705–708. IEEE, 2002.

[9] M. J. Atallah. A linear time algorithm for the Hausdorff distance between convex polygons. *Information Processing Letters*, 17(4):207–209, 1983.

[10] Y.-B. Bai, J.-H. Yong, C.-Y. Liu, X.-M. Liu, and Y. Meng. Polyline approach for approximating Hausdorff distance between planar free-form curves. *Computer-Aided Design*, 43(6):687–698, 2011.

[11] M. Bartoň, I. Hanniel, G. Elber, and M.-S. Kim. Precise Hausdorff distance computation between polygonal meshes. *Computer Aided Geometric Design*, 27(8):580–591, 2010.

[12] S. E. Benzley, E. Perry, K. Merkley, B. Clark, and G. Sjaardama. A comparison of all hexagonal and all tetrahedral finite element meshes for

elastic and elasto-plastic analysis. In *Proceedings of the 4th International Meshing Roundtable*, volume 17, pages 179–191. Sandia National Laboratories Albuquerque, NM, 1995.

[13] H. Biermann, I. M. Martin, D. Zorin, and F. Bernardini. Sharp features on multiresolution subdivision surfaces. *Graphical Models*, 2(64):61–77, 2002.

[14] P. Bo, M. Bartoň, D. Plakhotnik, and H. Pottmann. Towards efficient 5-axis flank CNC machining of free-form surfaces via fitting envelopes of surfaces of revolution. *Computer-Aided Design*, 79:1–11, 2016.

[15] P. Bo, M. Bartoň, and H. Pottmann. Automatic fitting of conical envelopes to free-form surfaces for flank CNC machining. *Computer-Aided Design*, 91:84–94, 2017.

[16] I. Boier-Martin, H. Rushmeier, and J. Jin. Parameterization of triangle meshes over quadrilateral domains. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 193–203. ACM, 2004.

[17] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin. Quad-mesh generation and processing: A survey. *Computer Graphics Forum*, 32(6):51–76, 2013.

[18] D. Bommes, H. Zimmer, and L. Kobbelt. Mixed-integer quadrangulation. *ACM Transactions on Graphics*, 28(3):77:1–77:10, 2009.

[19] M. Campen, D. Bommes, and L. Kobbelt. Dual loops meshing: Quality quad layouts on manifolds. *ACM Transactions on Graphics*, 31(4):110:1–110:11, 2012.

[20] M. Campen and L. Kobbelt. Dual strip weaving: Interactive design of quad layouts using elastica strips. *ACM Transactions on Graphics*, 33(6):183:1–183:10, 2014.

[21] M. Campen and L. Kobbelt. Quad layout embedding via aligned parameterization. *Computer Graphics Forum*, 33(8):69–81, 2014.

[22] N. A. Carr, J. Hoberock, K. Crane, and J. C. Hart. Rectangular multi-chart geometry images. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, pages 181–190. Eurographics Association, 2006.

[23] X.-D. Chen, W. Ma, G. Xu, and J.-C. Paul. Computing the Hausdorff distance between two B-spline curves. *Computer-Aided Design*, 42(12):1197–1206, 2010.

[24] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[25] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. John Wiley & Sons, 2009.

[26] S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J. C. Hart. Spectral surface quadrangulation. *ACM Transactions on Graphics*, 25(3):1057–1066, 2006.

[27] G. Elber and T. Grandine. Hausdorff and minimal distances between parametric freeforms in $\mathbb{R}^2$ and $\mathbb{R}^3$. In *Proceedings of the 5th International Conference on Geometric Modeling and Processing*, pages 191–204. Springer, 2008.

[28] X. Fang, W. Xu, H. Bao, and J. Huang. All-hex meshing using closed-form induced polycube. *ACM Transactions on Graphics*, 35(4):124:1–124:9, 2016.

[29] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Morgan Kaufmann, 5 edition, 2002.

[30] S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics*, 24(3):544–552, 2005.

[31] W. R. Franklin. Nearest point query on 184M points in E3 with a uniform grid. In *Proceedings of the 17th Canadian Conference on Computational Geometry*, pages 239–242, 2005.

[32] X.-M. Fu, C.-Y. Bai, and Y. Liu. Efficient volumetric polycube-map construction. *Computer Graphics Forum*, 35(7):97–106, 2016.

[33] M. Garland, A. Willmott, and P. S. Heckbert. Hierarchical face clustering on polygonal surfaces. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 49–58. ACM, 2001.

[34] M. Godau. *On the complexity of measuring the similarity between geometric objects in higher dimensions*. PhD thesis, Freie Universität Berlin, 1999.

[35] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 171–180. ACM, 1996.

[36] J. Gregson, A. Sheffer, and E. Zhang. All-hex mesh generation via volumetric polycube deformation. *Computer Graphics Forum*, 30(5):1407–1416, 2011.

[37] T. Gurung, D. Laney, P. Lindstrom, and J. Rossignac. SQuad: Compact representation for triangle meshes. *Computer Graphics Forum*, 30(2):355–364, 2011.

[38] M. Guthe, P. Borodin, and R. Klein. Fast and accurate Hausdorff distance calculation between meshes. *Journal of WSCG*, 13(2):41–48, 2005.

[39] I. Hanniel, A. Krishnamurthy, and S. McMains. Computing the Hausdorff distance between NURBS surfaces using numerical iteration on the GPU. *Graphical Models*, 74(4):255–264, 2012.

[40] Y. He, H. Wang, C.-W. Fu, and H. Qin. A divide-and-conquer approach for automatic polycube map construction. *Computers & Graphics*, 33(3):369–380, 2009.

[41] J. Huang, T. Jiang, Z. Shi, Y. Tong, H. Bao, and M. Desbrun. ℓ1-based construction of polycube maps from complex shapes. *ACM Transactions on Graphics*, 33(3):25:1–25:11, 2014.

[42] B. Jüttler. Triangular Bézier surface patches with a linear normal vector field. In *The Mathematics of Surfaces VIII*, pages 431–446. Information Geometers, 1998.

[43] B. Jüttler. Bounding the Hausdorff distance between implicitly defined and/or parametric curves. In *Mathematical Methods for Curves and Surfaces*, pages 223–232. Vanderbilt University, 2001.

[44] B. Jüttler and M. L. Sampoli. Hermite interpolation by piecewise polynomial surfaces with rational offsets. *Computer Aided Geometric Design*, 17(4):361–385, 2000.

[45] F. Kälberer, M. Nieser, and K. Polthier. Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum*, 26(3):375–384, 2007.

[46] Y.-J. Kim, M.-S. Kim, and G. Elber. Precise convex hull computation for freeform models using a hierarchical Gauss map and a Coons bounding volume hierarchy. *Computer-Aided Design*, 46:252–257, 2014.

[47] Y.-J. Kim, Y.-T. Oh, S.-H. Yoon, M.-S. Kim, and G. Elber. Precise Hausdorff distance computation for planar freeform curves using biarcs and depth buffer. *The Visual Computer*, 26(6-8):1007–1016, 2010.

[48] Y.-J. Kim, Y.-T. Oh, S.-H. Yoon, M.-S. Kim, and G. Elber. Coons BVH for freeform geometric models. *ACM Transactions on Graphics*, 30(6):169:1–169:8, 2011.

[49] Y.-J. Kim, Y.-T. Oh, S.-H. Yoon, M.-S. Kim, and G. Elber. Efficient Hausdorff distance computation for freeform geometric models in close proximity. *Computer-Aided Design*, 45(2):270–276, 2013.

[50] A. Krishnamurthy, S. McMains, and I. Hanniel. GPU-accelerated Hausdorff distance computation between dynamic deformable NURBS surfaces. *Computer-Aided Design*, 43(11):1370–1379, 2011.

[51] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical report, TR99-018, Department of Computer Science, University of North Carolina, 1999.

[52] E. Li, B. LéVy, X. Zhang, W. Che, W. Dong, and J.-C. Paul. Meshless quadrangulation by global parameterization. *Computers & Graphics*, 35(5):992–1000, 2011.

[53] X. Li, X. Guo, H. Wang, Y. He, X. Gu, and H. Qin. Harmonic volumetric mapping for solid modeling applications. In *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling*, pages 109–120. ACM, 2007.

[54] J. Lin, X. Jin, Z. Fan, and C. C. L. Wang. Automatic polycube-maps. In *Proceedings of the 5th International Conference on Geometric Modeling and Processing*, pages 3–16. Springer, 2008.

[55] M. Livesu, N. Vining, A. Sheffer, J. Gregson, and R. Scateni. Polycut: Monotone graph-cuts for polycube base-complex construction. *ACM Transactions on Graphics*, 32(6):171:1–171:12, 2013.

[56] E. Neumann. The Fréchet distance of surfaces is computable. 2017.

[57] C.-H. Peng, E. Zhang, Y. Kobayashi, and P. Wonka. Connectivity editing for quadrilateral meshes. *ACM Transactions on Graphics*, 30(6):141:1–141:12, 2011.

[58] N. Pietroni, M. Tarini, O. Sorkine, and D. Zorin. Global parametrization of range image sets. *ACM Transactions on Graphics*, 30(6):149:1–149:10, 2011.

[59] N. Ray, W. C. Li, B. Lévy, A. Sheffer, and P. Alliez. Periodic global parameterization. *ACM Transactions on Graphics*, 25(4):1460–1485, 2006.

[60] J.-F. Remacle, J. Lambrechts, B. Seny, E. Marchandise, A. Johnen, and C. Geuzainet. Blossom-quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. *International Journal for Numerical Methods in Engineering*, 89(9):1102–1119, 2012.

[61] S. L. Rueda, J. Sendra, and J. R. Sendra. Bounding and estimating the Hausdorff distance between real space algebraic curves. *Computer Aided Geometric Design*, 31(3):182–198, 2014.

[62] K. Takayama, D. Panozzo, A. Sorkine-Hornung, and O. Sorkine-Hornung. Sketch-based generation and editing of quad meshes. *ACM Transactions on Graphics*, 32(4):97:1–97:7, 2013.

[63] M. Tang, S. Curtis, S.-E. Yoon, and D. Manocha. ICCD: Interactive continuous collision detection between deformable models using connectivity-based culling. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):544–557, 2009.

[64] M. Tang, M. Lee, and Y. J. Kim. Interactive Hausdorff distance computation for general polygonal models. *ACM Transactions on Graphics*, 28(3):74:1–74:10, 2009.

[65] M. Tarini, K. Hormann, P. Cignoni, and C. Montani. Polycube-maps. *ACM Transactions on Graphics*, 23(3):853–860, 2004.

[66] M. Tarini, N. Pietroni, P. Cignoni, D. Panozzo, and E. Puppo. Practical quad mesh simplification. *Computer Graphics Forum*, 29(2):407–418, 2010.

[67] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. Designing quadrangulations with discrete harmonic forms. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, pages 201–210. Eurographics Association, 2006.

[68] H. Wang, M. Jin, Y. He, X. Gu, and H. Qin. User-controllable polycube map for manifold spline construction. In *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*, pages 397–404. ACM, 2008.

[69] J. Xia, I. Garcia, Y. He, S.-Q. Xin, and G. Patow. Editable polycube map for GPU-based subdivision surfaces. In *Symposium on Interactive 3D Graphics and Games*, pages 151–158. ACM, 2011.

[70] W. Yu, K. Zhang, S. Wan, and X. Li. Optimizing polycube domain construction for hexahedral remeshing. *Computer-Aided Design*, 46:58–68, 2014.

초 록

본 논문에서는 준정규 사각형 메시의 사용성을 증대시키기 위한 다양한 기하 알고리즘을 소개한다. 우선 기하 연산을 가속화하는 데에 쓰이는 bounding volume hierarchy로 사각형 메시에 특화시킨 새로운 구조를 선보인다. 이를 이용하면 사각형 메시로의 점 투영(최단 거리 계산)을 빠르게 수행할 수 있다. 점 투영은 하우스도르프 거리 계산 및 사각형 메시 재건을 위해 본 논문에서 새롭게 제안하는 알고리즘에서 큰 비중을 차지하는 핵심 연산이다.

한 삼각형 메시와 이를 근사하는 사각형 메시 사이의 하우스도르프 거리는 그 근사 품질을 평가할 수 있는 척도이다. 하지만 물체 정합에도 쓰이는 하우스도르프 거리를 계산하는 일은 메시 모델에 대해 늘 매우 어려운 문제이어 왔고 특히 사각형 메시에 대해서는 아예 연구된 바가 없다. 본 논문에서 소개하는 알고리즘은 질의 모델을 반복적으로 분할하고 무관한 조각들을 제거함으로써 하우스도르프 거리가 발생하는 곳과 하우스도르프 거리의 상한과 하한을 점차 좁혀나간다. 하한을 갱신하는 일과 영향력이 없는 모델 조각을 찾아내는 데에 점 투영이 깊게 관여한다. 상호작용 가능한 속도가 달성되었으며 알고리즘의 높은 정확도는 계산 정밀도의 제약만을 받는다. 이는 모델 근사나 물체 정합에 있어서 선호되는 결과인 하우스도르프 거리가 영에 가까운 상황에도 성립하는데, 사실 가장 난이도가 높은 경우이다.

또한, 본 논문은 사각형 메시를 생성하는 데에 도움을 주는 인터페이스를 제안한다. 주어진 메시 모델이나 점 구름을 사각형 메시로 변환하기 위해 먼저 사용자가 원하는 사각 레이아웃을 그려넣도록 한다. 인터페이스의 알고리즘

은 입력 데이터를 레이아웃의 각 패치에 자동으로 할당하고 이를 근사하는 사각형 메시도 자동으로 생성한다. 사각형 메시를 반복적으로 다듬는 과정에서 점 투영은 다시 한 번 큰 비중을 차지한다. 이 사각형 메시는 앞서 제시된 알고리즘을 통해 계산되는 하우스도르프 거리로 평가할 수 있다.

주요어: 하우스도르프 거리, 사각형 메시, 재건, Bounding Volume Hierarchy, 점 투영, 최단 거리 계산, 충돌 감지

학번: 2011-20779

Acknowledgments

First and foremost, my deepest gratitude no doubt goes to my advisor Professor Myung-Soo Kim for his immense patience and guidance throughout the prolonged years. He has provided me with an environment where I could focus on work without too much pressure. I always knew that I was incredibly fortunate to have joined his research group.

I would also like to thank Professors Min-Ho Kyung and Seung-Hyun Yoon for their insightful comments during my research breakthrough, Professor Jehee Lee for inspiring to always pursue the highest goals, and Professor Yeong Gil Shin for chairing the thesis committee. I am grateful for many other academics and researchers who in one way or another have acknowledged and encouraged me along the way, including Professor Gershon Elber, who first introduced me to the concept of Hausdorff distance during his seminar so many years ago before I even officially started graduate school.

Despite what he may think, Dr. Yong-Joon Kim has been truly enlightening during my time with him. Even in his absence, I was able to learn from his past research and come up with my own results. Jaesung Park, Minsub Shim, and Jaewook Lee, assisted by Seon-Young Park, showed great teamwork and participated with me in some of my earliest research. Q Youn Hong has been very helpful with her broader experience and knowledge in my later days. I express my thanks to all of my previous and current labmates, especially

Ahmad Nasikun, Jongin Kim, and Fangda Chen among others for simply being such cheerful friends with me.

Special thanks to my mother, who has always supported me with unending love, my sister, for bravely pushing through and setting out on her own path, and my father, all of my grandparents, and other scholastic members on either side of my extended family, who probably undeniably had an influence on me and my path.

Lastly, major thanks to my dear Moran Kim for putting up with me and helping me out through both of our tumultuous times.

Lastlier, minor thanks to my dead hard drive for holding out and only failing during the holidays!