NUMERICAL AND GEOMETRIC OPTIMIZATIONS FOR SURFACE AND TOLERANCE MODELING

A Dissertation

by

SONGGANG XU

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2015

Major Subject: Computer Science

ABSTRACT

Optimization techniques are widely used in many research and engineering areas. This dissertation presents numerical and geometric optimization methods for solving geometric and solid modeling problems.

Geometric optimization methods are designed for manufacturing process planning, which optimizes the process by changing dependency relationships among geometric primitives from the original design diagram. Geometric primitives are used to represent part features, and dependencies in the dimensions between parts are represented by a topological graph. The ordering of these dependencies can have a significant effect on the tolerance zones in the part. To obtain tolerance zones from the dependencies, the conventional parametric method of tolerance analysis is decomposed into a set of geometric computations, which are combined and cascaded to obtain the tolerance zones in the geometric representations. Geometric optimization is applied to the topological graph in order to find a solution that provides not only an optimal dimensioning scheme but also an optimal plan for manufacturing the physical part. The applications of our method include tolerance analysis, dimension scheme optimization, and process planning.

Two numerical optimization methods are proposed for local and global surface parameterizations. One is the nonlinear optimization, which is used for building the local field-aware parameterization. Given a local chart of the surface, a two-phase method is proposed, which generates a folding-free parameterization while still being aware of the geodesic metric. The parameterization method is applied in a view-dependent 3D painting system, which constitutes a local, adaptive and interactive painting environment. The other is the mixed-integer quadratic optimization, which is used for generating a quad mesh from a given triangular mesh. With a given cross field, the computation of parametric coordinates is formulated to be a mixed-integer optimization problem, which parameterizes the surface with good quality by adding redundant integer variables. The mixed integer system is solved more efficiently by an improved adaptive rounding solver. To obtain the final quadrangular mesh, an isoline tracing method and a breadth-first traversal mesh generation method are proposed so that the final mesh result has face information, which is useful for further model processing.

ACKNOWLEDGEMENTS

Foremost, I would express my sincere thanks to my advisor, Dr. John Keyser, for his guidance, comments, suggestions and support throughout my Ph.D study and research. Dr. Keyser shared his knowledge and gave endless help to finish this dissertation.

I would like to express my thanks to other committee members, for their ideas, comments and directions on improving the dissertation.

My thanks are extended to other students that I have the opportunity to work with. Shu-Wei Hsu, Ruoguan Huang, Donghui Han, Billy Clack, Jason Smith, Lei He, Josiah Manson. The great time to work with all of you is the most valuable experience of my life.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

Optimization problems arise in many research topics of computer graphics, and are probably even more common in geometric modeling. Any designed optimization problem usually involves optimizing some criteria of merit, such as the distortion of surface parameterization. The aim of solving optimization problems is to figure out the configuration that best achieves the desired criterion among all configurations in the possible design space. Common steps for applying optimization in modeling are: finding a mathematical representation (analytic or numeric) of the geometric problem, including modeling the objective function and constraints of the original problem; finding the appropriate numerical or topological method for solving the problem; reconstructing the optimal solution. Furthermore, optimization in geometric modeling has its own special characteristics.

First, the formulation of optimization is versatile, i.e., the structure of the optimization is neither straight nor unique. Take surface parameterization as an example. The final goal is usually to minimize the distortion after unwrapping the surface. Therefore, the optimization is to to minimize a metric that measures the distortion essentially. Different researchers proposed different metrics, such as angle metric (conformal mapping), distance metric (isometric metric), etc. Though different metrics produce different parametric results, no universal criteria can be applied to order preferences among all those metrics. The formulation of the optimization is perhaps more decided by the the application background, which, in consequence, provides versatility for building the optimization.

Second, because of the complications of the formulation, conventional numeric solvers sometimes cannot be applied directly, i.e., there are requirements of modifying or improving existing numeric algorithms for solving the optimization. Take surface quadrangulation as an example. Essentially, it is a seamless global parameterization, which can be formulated to be a mixed-integer optimization (MIQ) for parameterization, and a quadratic optimization for grid generation. The mixed-integer optimization is an NP-hard problem. We usually pursue an approximated optimal solution. However, the approximated solution must fit the application background, which means the relaxation of the optimum solution is somehow very limited. How to formulate an appropriate numerical solver becomes another critical problem in optimization for geometric modeling.

Third, despite the versatility of formulation and the speciality of the numerical solver, the scale of the geometric problem itself also proposes challenges to the design of optimization. On one hand, it is very common to formulate an optimization that has more than hundreds of thousands of variables. On the other hand, many geometric applications require the optimization to be finished in an acceptable time cost, or even in real time. These contrary goals require us to invent a more efficient optimization process that fulfills requirements from real applications.

Because of all those special characteristics of optimization in geometric modeling, we proposed new optimization formulations for existing modeling problems and propose new numerical and topological methods for solving the optimization with these formulations.

Specifically, we work on three geometric modeling problems utilizing the optimization techniques.

1.1 Local Parameterization

The first problem is local surface parameterization for 3D paintings. 3D modeling systems form the basis for many of the most widely used graphics applications.

The ability for a designer to model not only the geometry but also the color and appearance of a model (which we will refer to by the generic term “painting”) is a critical aspect of any such modeling system. Painting has a long history in computer graphics [24, 51]. Hanrahan et al. [24] proposed an direct painting system. Pedersen et al. [51] proposed a painting technique for implicit surfaces. Igarashi et al. [28] proposed an adaptive paradigm for interactive texture paintings. Although interfaces have long provided the ability to paint directly on a 3D model, the colors are usually stored in a 2D domain, such as in a texture map. Despite its widespread use and several advantages, texture mapping has several shortcomings in the context of 3D painting. 3D painting usually requires only a local surface parameterization, while many parameterization methods are global and have some limitations to be applied directly to 3D painting, such as computing time being too high for real-time parameterization and the presence of obvious texture distortions. In particular, the need to determine a map (including the unfolding process, seams, and dealing with multiple charts), and then for the texture artist to consider the orientation of this mapping while painting something nominally 2D, can be problematic. In recent years, methods have emerged that address these shortcomings. One of the key works is by Schmidt et al. [58], who provided an efficient local parameterization for 3D painting. This work is extended by the stroke parameterization proposed by Schmidt et al. in [57]. They tried to minimize the distortion of textures by approximating the exponential map. Melvael et al. [42] presented an algorithm to approximate the polar coordinates of the exponential map by the geodesic distance on triangular and general polygonal meshes. The geodesic field can be obtained by approximated methods in [59, 64, 14]. Sethian [59] proposed the level set method for computing geodesic distance. Xu et al. [64] improved the performance of the original fast sweeping method for computing geodesic distance. Crane et al. [14] utilized the heat transfer equation to model the geodesic distance field. These incremental unwrapping methods are efficient and easy to implement. These methods distribute the parameterization distortion unevenly in the texture space, so that painting results are more reasonable to users. However, those methods have limitations for 3D paintings. The major problem is that incremental unwrapping methods often fail to give high quality parameterization of the surface. Since they are sensitive to the surface connectivity, they have a very high chance to have triangle foldings even with meshes of good quality. They may also fail easily when applied to surfaces with sharp or complicated local features, so that they are not suitable on diverse surface topologies. Therefore, while those methods work well in some standard situations, these limitations can cause significant texture distortion and error in many other cases.

In our work, we present a two-phase method that not only provides awareness of the geodesic metric but also is insensitive to the local surface connectivity so that triangle foldings can be avoided for most cases and the distortion distribution guided by the geodesic metric is maintained in the texture space.

1.2 Global Parameterization

The second problem is quadrangulation via global surface parameterization. Problems of surface quadrangulation and parameterization have received a lot of attention in the fields of graphics and modeling. The interest comes from the initial motivation of quadrangulation, that is converting raw surface data into high quality, compact, and regular representations. These representations support much further processing, such as texture mapping and surface deformation in graphics, and high precision finite element analysis in engineering. The difficulties of this problem include the distribution and control of distortion which affects the quad mesh quality, the performance of the internal linear solver which defines the performance of the whole quadrangulation process, and the topological correctness of the final mesh extraction.

Most recent methods use two techniques, field aligned optimization and global segmentation, to generate quad meshes. The cross field usually provides two orthogonal directions for each (discrete) point on surfaces. Therefore, the cross field can be defined by the intrinsic smoothed principal curvature field, which uses two principal directions as the guiding directions, or by some field obtained by optimization techniques. Recently, the design of this field has been formulated as a minimization problem for certain types of functions so that the optimal solution generates the aligned smooth cross field. Singular points of the field are usually identified by finding points that violate some geometric property on surfaces. The cross field is then used as input to many parameterization methods for the computation of parametric coordinates. As part of this, the model is typically segmented into several patches according to the positions of singular points. All patches are homeomorphic to a disc so that they can be parameterized by one of several local parameterization methods. This process propagates the surface distortion to those singular points and boundaries of surfaces. Since patches will be combined as part of mesh generation, the parametric coordinates along the seams between patches are required to match on both sides. This requirement is usually treated as a constraint during parameterization. To demonstrate the final quad mesh, many methods only show their results by drawing isolines of integer grids in parametric space. However, it has been shown that the generation of the quad mesh with a correct topological structure is not a trivial task. Thus, mesh extraction is also raised as an interesting topic for research.

We focuses on the parametric coordinates computation and the quad mesh extraction stages of this process. Meanwhile, we propose a solution for extracting a quad mesh after parameterization rather than simply drawing isolines of integer grids on models, which is more appropriate for further model processing.

1.3 Geometric Optimization for Process Planning

The third problem is process planning. In parts manufacturing, the quality of finished parts is determined by both design and manufacturing tolerances, which determine the dimensional and geometric properties of the part. Engineers are required to select appropriate machining processes and equipment so that the requirements of the design tolerances are met. This process is usually called *process planning*. The traditional way to design an appropriate process plan is a time consuming task that requires the engineer to iterate the process of designing, testing, and modifying the solution until all the design requirements are satisfied. Recently, computer aided process planning (CAPP) has drawn considerable attention as a way to simplify this process. Key techniques related to CAPP include tolerance modeling, tolerance analysis, and tolerance allocation. The tolerances determined in the design phase serve as the basis for the manufacturing tolerances, which are used to reflect individual manufacturing operations.

Our work is on design tolerances. We present a new geometric model for tolerance modeling and propagation, geared toward tolerance analysis in process planning. Our aim is to decompose a big chunk of analytic computations of conventional tolerance analysis into a series of geometry computations. We first decompose the part into basic geometric primitives, or features. Because those primitives are to be manufactured in a common part, we know they are related and that there are dependencies between them, which could be represented by a graph structure. We first decouple the primitives into several co-related primitive groups. In each group, the geometric position of a certain primitive (*target primitive*) is decided by the remaining primitives (*reference primitives*). Linear parametric geometric uncertainty model (LPGUM), proposed by Joskowicz et al. [29], approximates tolerance zones with linear precision. This allows us to use the LPGUM model to model the tolerance zone for the target primitive, because all its variations have been obtained. Next, we formulate a method for cascading the decoupled primitive groups, so that the tolerances can be transferred between groups. Using those cascading techniques, we could obtain the tolerance zones for all primitives by traversing the embedded graph structure representing the dependencies. The tolerance zones thus obtained provide a worst case estimation on dimensions, represented as a geometric polytope. Finally, we provide a computational optimization method which can improve the quality of the existing process plan so that the tolerance of the parts could be minimized as much as possible. The optimization problem itself is an NP hard problem. We propose an efficient approximated dynamic programming solver which utilizes the optimal substructure.

2. RELATED WORK

Since we focus on formulating novel optimizations in geometric modeling, we survey previous works related to our research.

2.1 Local Surface Parameterization for 3D Paintings

Since our work focuses on providing a novel local parameterization for 3D painting systems, we survey previous work from two parts, techniques on local parameterization and the design of 3D painting systems.

For local surface parameterization, a very wide range of techniques have been applied over the past decade to address the problems of distortion and performance in surface parameterization. We refer to the work presented by Floater [20] for a literature survey on methods of authalic (area-preserving), conformal (angle-preserving), and isometric (length-preserving) mapping, as well as some hybrid methods. Most methods can be applied to a whole surface or its local chart if they are homeomorphic to a disc. Strictly speaking, though applicable, they are not good for local parameterization, which usually has a center point. Those methods only distribute the distortion as evenly as possible while ignoring the distribution of distortion, which is affected a lot by the shape and the boundary of the local chart, so that high distortions can happen anywhere on the mapped texture. To overcome those problems, Schmidt et al. [58] proposed the decal map, a local method that approximates the exponential map on the surface chart. Melvaer and Reimers [42] approximated the exponential map using the geodesic metric. The center information is considered naturally by the exponential map in both methods. However, some high texture distortions as well as frequent triangle foldings cannot be overcome, and discontinuities of color will appear across the texture atlas. Schmidt's recent work [57], though extending the decal map, still lacks explicit avoidances of the sensitivity to the mesh connectivity and frequent triangle foldings. Besides those methods, the method presented by Liu et al. [41], built on the isometric setting, is related to our work. The as-rigid-as-possible (ARAP) energy is formulated for optimization so as to parameterize texture coordinates, which is a nonlinear parameterization that is solved by a local/global technique. The local/global framework provides the possibility to add constraints that make the system aware of the geodesic metric without breaking the isometric setting.

For 3D painting, several techniques for directly painting on models have been proposed in recent years. These methods allow a user to paint on the 3D view of the model, with the 3D information translated to the 2D texture map. While this is a very useful feature in many cases, often painting on a 2D canvas is more desirable. The main issue is that while painting brushes and strokes are usually defined by 2D patterns, the 3D surface is not typically planar, and thus discontinuities and distortion arise during painting. The parameterization can also be avoided by using 3D structures to represent textures when constructing the 3D painting system. One option is an octree, which provides different texture resolutions over different parts of the 3D model. Benson and Davis[4] used octree for storing textures. Gibbs et al. [21] implemented painting system over octree textures. Another data structure, explored by Lefebvre and Hoppe [35], is a hash table, which packs sparse data into a compact table while retaining efficient random access. Lefebvre et al. [36] also presented the texture mapping method over surfaces. Zwicker et al. [67] implemented a texture mapping system over the structure of point cloud. Carr et al. [11] implemented a real-time texture painting system over surfaces. Kalnins et al. [31] implemented a user interactive texture mapping system. Texture look-ups are very efficient, but they do not allow different resolutions of textures over the surface of the 3D model. Another way to avoid the texture mapping is by associating the color data directly with the polygonal mesh. Color data at polygon edges can be duplicated by techniques in [10]. The color offset can be inferred from the edge by techniques in [9]. Yuksel et al. [65] proposed mesh colors to store textures over polygon faces. Image editing techniques [49] can also be utilized over the structure of mesh colors.

2.2 Global Surface Quadrangulation

Global quadrangulation has been proposed along with the development of parameterization techniques. The trend of research interests has been moving from models that are homeomorphic to a disc to models that are homeomorphic to a sphere or shapes having a more complicated topological structures. The basic idea is to segment the model into several disc-like patches, parametrize each and stitch them together. Dong et al. [18] used the Morse-Smale complex to the generate high level patch layout. Each patch was further refined to generate the quad mesh. This method was improved later by Huang et al. [26] to consider singular point positions, orientations and feature alignments. However, both methods are not fully automatic since high level quad layouts are always required. The concept of the cone singularity was then introduced by Kharevych et al. [33]. This work allowed concentration of the distortion at a small number of singularities. This work was extended by Ben-Chen et al. [3] with a greedy scheme to decide and extract singularities. Both methods produce conformal parameterizations with lower distortion, but do not account for features such as sharp edges. Zhang et al. [66] proposed a method for applying wave functions on surfaces to generate quad meshes while respecting features on the surface.

Recently many methods have been proposed that are based on the idea of two-phase parameterization: the design of a guiding field and then segmentation considering singularities. Ray et al. [53] proposed a non-linear parameterization method which was guided by a low level vector field. Kalberer et al. [30] used the smoothed principal curvature field as the guide field for quad generation. The concept of *branch points* was introduced to segment the model, which was similar to singularities. Bommes et al. [6] proposed a method that contributed to two phases: the guide field design and the parameterization. They formulated the two phases to be two mixed-integer quadratic problems and proposed a greedy numeric solver which generated smooth results for both phases efficiently. Following this idea, Li et al. [38] proposed an implementation on meshless surfaces. Bommes et al. [5] later proposed a method for generating sparse quads on models. Bommes et al. [7] also presented the numeric solver for the internal optimization problem. Ebke et al. [19] provided a robust implementation for the quad mesh extraction. Gunpinar et al. [23] provided a segmentation method for quadrangulation in reverse engineering. This two-phase idea provided possibilities for feature and orientation controls. However, positions of singularities and layouts of segmenting seams affect the distortion distribution. Finding an optimal solution for both phases, which is an NP-hard problem, has become the major focus of quadrangulation research in recent years. Myles et al. [46] proposed a greedy method to approximate the solution for placing singularities and segmenting the model. This method was later extended by Myles et al. [47] for placing cones with feature alignments. Very recently, Peng et al. [52] studied the uniqueness of quadrangulation on several input shapes. However, their method still requires surface segmentation. Meanwhile, while most methods focus on automatic quadrangulation, Tierny et al. [61] presented a system that provides a highly customized solution for subjective quad mesh design.

Since many parameterization methods are field guided, the design of the field, especially the N-symmetry field, is also a core problem in quadrangulation. Ray et al.

[55] proposed a method to design the N-symmetry direction field in a linear manner. Crane et al. [13] provided a smooth connection on the surface to facilitate the design of vector fields. However, both methods require one to specify singularities manually. Bommes et al. [6] used MIQ to train the cross field. Ray et al. [54] used geometry features to identify singularities automatically. Knoppel et al. [34] provided the globally optimal formulation of the N-symmetry field on surfaces.

2.3 Geometric Optimization for Tolerance Modeling

Process planning usually pursues high precision part processing, which can be measured by tolerances in geometric modeling. Tolerances can be represented by analytical functions or geometric zones. A key part of tolerance modeling is representing the zone within which geometric characteristics of a model may vary. Akella et al. [1] represented tolerance zones as simple geometric entities that are guaranteed to bound the features of the model. They can be thought of as a representation of the uncertainty in the geometric position.

Geometric variations can also be modeled as higher dimensional geometric objects, such as polytopes or dual-cones, which represent the region as intervals of the coefficients of their algebraic parameterization. Roy et al. [56] modeled variations as polytopes. Mujezinovic et al. [44] proposed techniques for modeling variations over polygon faces. Davidson et.al. [15] proposed techniques for modeling variations over round faces.Tolerances of part features such as form, orientation, and size can be represented in this way, though further computation on such models can be very complicated. Tolerance analysis covers techniques that compute the variations of tolerances for worst case estimation or statistical expectation. Hong et al. [25] presented broad reviews of the area are available. Shen et al. [60] presented recent updates on tolerance analysis methods. Brost et al. [8] presented an automatic design paradigm for the 3D part assembly with the consideration of tolerances. Though many techniques can be employed for tolerance analysis in the design phase, tolerance propagation is the core part of process planning. Tolerance propagation refers to the determination of one tolerance zone based on others. When manufacturing one feature of a part, engineers have to use other features as the geometric or dimensioning references. Thus, tolerances in the references must be propagated to the feature referring to it. Typically propagation involves a series of computational geometry operations. However, because the tolerances are modeled in different spaces (world space and local part-centric space), the representation and the transformation of tolerances is not straightforward. For high-dimensional representations, the computation of tolerance propagation relies on computational techniques in high dimension that are notorious for difficulty with implementation and robustness. Several lines of prior research have proposed methods addressing propagation.

Tolerance charting methods, which evaluate tolerance propagation based on engineers' experience, are a traditional way of handling tolerances. Computer-aided tolerance charting methods [37] have been developed to reduce the iteration of physical trial-and-error runs. Shortcomings of tolerance charting are that it cannot deal with complex spatial tolerance propagation issues or geometrical tolerances. To overcome this problem, methods for modeling tolerance propagation in higher dimensions have been presented. One kind of method uses the small displacements torsor (SDT), presented by Clement et al. [12]. Villeneuve et al. [62] proposed tolerance models for manufacturing 3D parts by SDT. Another kind of method, Technologically and Topologically Related Surfaces (TTRS), forms any part as a tree representing the succession of surface associations, presented by Desrochers et al. in [17]. Tolerance information can be tracked along the stacking chain in the graph. The stacking up of parts could be simulated by a Monte Carlo method to estimate tolerance propagation. Huang et al. [27] proposed techniques to evaluate tolerance-based process planning by random simulation. Barkallah et al. [2] proposed techniques to evaluate manufacturing tolerances using statistical tolerance models. This has the advantage of simplicity and flexibility, however the drawbacks of such methods are that it can be very time consuming and have poor computational accuracy at small to medium sample sizes. A third kind of method tries to find a feasible assembly plan on a graph structure by representing the related parts with the consideration of tolerances [50]. This method bridges the gap between its generalized tolerance model and previous models so that it could be incorporated with previous assembly planning methods. Though this method formulates a general framework for tolerance estimation, its contribution to optimal planning is limited.

Regardless of the representation of the zones themselves, the most common and straightforward implementations represent the tolerance zones independently. Unfortunately, as tolerances are propagated the dependencies between zones is then lost, causing over-estimation of the zones. One way of exploiting dependencies between the tolerance zones is by analyzing sensitivity to parametric variations using the technique proposed by Giordano et al. [22]. Ostrovsky et al. [50] computed relative positions for parts during the assembly planning by analyzing parametric sensitivity. Recently, a new method for describing dependencies of geometric uncertainties has been proposed by Oskowicz et al. in [29]. The original model is extended to more geometric objects by Myers et al. in [45]. This method, called LPGUM (Linear Parametric Geometric Uncertainty Model), uses a first-order approximation of the uncertainty zones of geometric primitives. The dependencies of uncertainties are derived from sensitivity matrices. Despite the promise of LPGUM and similar methods, they have to this point been shown to handle only very simple operations on very basic primitives, and thus have not been practical for modeling tolerance zones of real parts. Further, no tolerance propagation model has been designed, and thus tolerance zones cannot be cascaded. As a result, these methods were not (yet) suitable for tolerance analysis during the process planning phase.

3. NONLINEAR OPTIMIZATION FOR FIELD-AWARE PARAMETERIZATION*

Field-aware parameterization is a technique designed for composing strokes in 3D painting systems. Xu et al. [63] proposed a field-aware parameterization. We propose a field-aware local parameterization utilizing the nonlinear optimization. The first section discusses this original motivation and contributions of our work. The next two sections cover method details, including the mechanism and the implementation. In the last section, we demonstrate the application of our method by some painting results in our 3D painting system.

3.1 Parameterization for 3D Painting Systems

In our work, we present a two-phase method that not only provides awareness of the geodesic metric but also is insensitive to the local surface connectivity so that triangle foldings can be avoided for most cases and the distortion distribution guided by the geodesic metric is maintained in the texture space. The features of our system fall into four major areas. First, we provide an efficient incremental unwrapping method, of which the result provides approximate geodesic polar coordinates for each vertex. Note that the parameterization distortion, guided by the geodesic field, distributes unevenly in the texture space. Second, the geodesic field can be represented by affine transformations of geodesic gradients, which acts as the reference for alignments in the second parameterization phase. We incorporate the geodesic field into an isometric parameterization framework to remove the sensitivity to the

*Part of this section is reprinted with permission from Texture mapping for 3D painting using geodesic distance by Songgang Xu and John Keyser. Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pages 163–163, San Francisco, CA, USA, 2014. Copyright 2014 by ACM. DOI 10.1145/2556700.2566650.

surface connectivity and the triangle foldings for most cases. Third, we derive two methods on the gradient field of geodesic distance, which requires less information over the geodesic field and produces comparable results. Last, we incorporate our local parameterization technique into a novel 3D painting system, which allows users to paint on the surface directly.

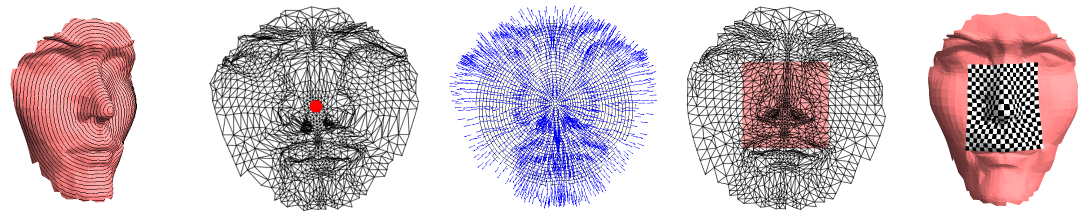

(a) Geodesic field (b) Incremental (c) Gradient field (d) Folding Removal (e) Result

Figure 3.1: Field-aware parameterization pipeline.

We demonstrate the pipeline of our parameterization method in Figure 3.1. The first phase obtains the approximated polar coordinates for the local charts. In phase two, we inherit the isometric framework while trying to reflect the geodesic field. Though geodesic distance can be constrained in existing parameterization methods on optimization, those types of constraints are usually nonlinear, which implies the solution is not globally optimal and the computing complexity is increased. In our method, we represent the geodesic field using a set of affine transformations, which are blended into an isometric framework. We also represent the geodesic field using its intrinsic gradients, upon which we build two methods aligned with those gradients. Those formulations allow us to build linear systems, which are efficient for computation. The isometric setting of the optimization allows us to remove triangle foldings for most general cases. However, if the folding still happens, we use a post processing method to resolve it. Utilizing this novel two-phase parameterization technique, we build our 3D painting system that provides a local adaptive and interactive painting environment. We explain our method following this pipeline in next sections.

3.2 Local Chart Parameterization

In this section, we explain our parameterization techniques in two phases: incremental unwrapping and field-aware parameterization.

3.2.1 Incremental Unwrapping

Most local parameterization methods utilizing incremental unwrapping are based on the approximation of the local chart by the exponential map, which builds correspondences between points on the continuous surface and the tangent plane at a given initial center point. The geodesic metric or its approximation are usually used for measuring the distance between the center point and other points on the surface in order to map points from the surface to the tangent plane. In exact algorithms for computing geodesic distance, roots for different vertices are maintained during the computation, which is ignored in fast marching. As presented by Joseph et al. [43], for a convex manifold model, we know all the geodesic paths share the same root, the initial point. This property means the geodesic path on a convex manifold has a simple structure such that it goes through the starting point followed by a sequence of edges. This inspired us to find an approximated method for geodesic distance on general mesh models by assuming the property always holds. For each vertex other than the starting point, we can associate it with its geodesic distance and the polar angle on the mapping field. This allows us to build an exponential map using geodesic paths on the model.

We follow the traditional incremental unwrapping technique to build our unwrapping paradigm. Vertices are tagged with different states with respect to the computation process: **Alive** if the vertex has been updated with a fixed geodesic value, **Close** if the vertex has been accessed but the geodesic value is not fixed, and **Far** if the vertex has not been accessed. The whole unwrapping paradigm is divided into three steps, initialization, propagation and construction of polar coordinates.

Initialization. We first compute the geodesic distance and the polar angle for vertices on the 1-ring of the center point **o**. If **o** is a vertex, we unwrap the 1-ring of the initial point to its discrete tangent plane decided by the discrete normal at this vertex. If the point is interior to a triangle, the tangent plane is the triangle that contains the point. All vertices adjacent to the initial point are marked **Alive**.

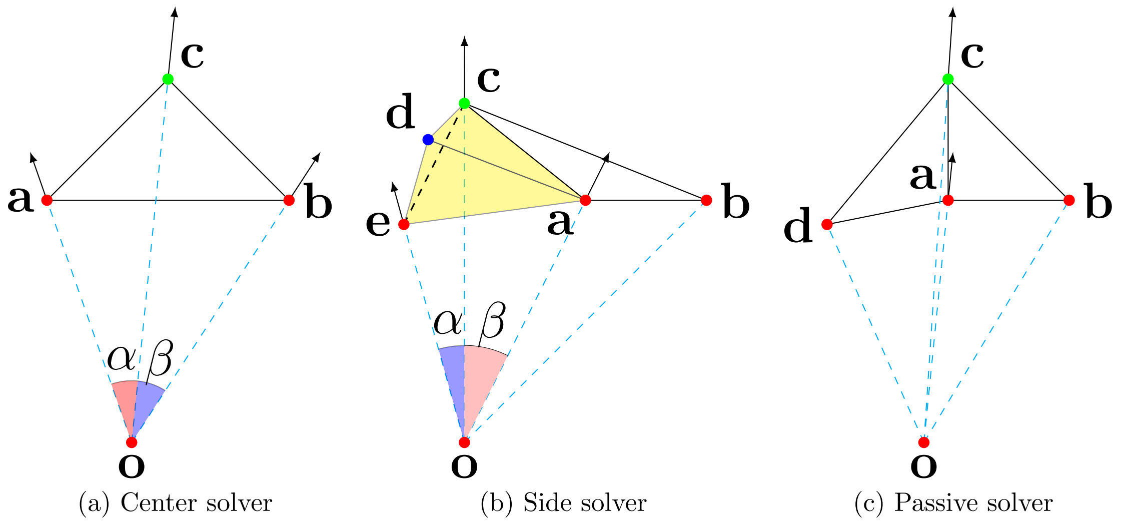


Figure 3.2: Three local solvers.

Propagation. We compute geodesic distance for vertices beyond the 1-ring using the **Alive** vertices. In detail, for one vertex, we try to find two **Alive** vertices in the same triangle by traversing its neighbors. Figure 3.2 shows three basic solvers for propagation. For the center solver in Figure 3.2a and side solver in Figure 3.2b, assume we are updating vertex $\mathbf{c}$. We find two **Alive** vertices $\mathbf{a}$ and $\mathbf{b}$. Then we use the geodesic distance and the polar angles of $\mathbf{a}$ and $\mathbf{b}$ to compute that of $\mathbf{c}$. If the vertex has only one **Alive** neighbor, then we use the passive solver in Figure 3.2c to update it. The center solver in Figure 3.2a deals with the case where the geodesic path of $\mathbf{c}$ goes through the edge $\mathbf{ab}$. In $\triangle\mathbf{abc}$, $\mathbf{a}$ and $\mathbf{b}$ are tagged **Alive**, i.e. we have their geodesic distance $\delta(\mathbf{a})$ and $\delta(\mathbf{b})$ as well as the polar angles $\tau(\mathbf{a})$ and $\tau(\mathbf{b})$ We assume that the three vertices share the same root $\mathbf{o}$. We can obtain $\delta(\mathbf{c}) = |\mathbf{oc}|$ as $\sqrt{|\mathbf{bc}|^2 + |\mathbf{bo}|^2 - 2 \cdot |\mathbf{bc}| \cdot |\mathbf{bo}| \cdot \cos \angle\mathbf{cbo}}$. $\tau(\mathbf{c})$ can be obtained by rotating $\tau(\mathbf{b})$ by the angle $\angle\mathbf{boa} \cdot \frac{\beta}{\beta+\alpha}$. Note that we should try each adjacent pair of **Alive** vertices of neighbor $\mathbf{c}$ and use the result from the pair that generates the minimal geodesic distance.

For a general triangulation, the geodesic path of $\mathbf{c}$ might not go through the edge $\mathbf{ab}$. It might be on the left (Figure 3.2b) of vertex $\mathbf{a}$ or on the right of vertex $\mathbf{b}$, in which case we use the "side solver" technique. Whether to use the center solver or the side solver can be decided by examining the angles $\angle\mathbf{obc}$ and $\angle\mathbf{oac}$. If both are less than or equal to π, we apply the center solver. Otherwise, we apply the side solver. We use an unfolding technique to find two **Alive** vertices across a set of triangles. In $\triangle$ $\mathbf{abc}$, though $\mathbf{a}$ and $\mathbf{b}$ are **Alive** (indicated by red dots), the geodesic path through $\mathbf{c}$ does not go across edge $\mathbf{ab}$. Since the angle sum $\angle\mathbf{cab} + \angle\mathbf{bao} > \pi$, we know the path is on the left side of $\mathbf{a}$. Starting from edge $\mathbf{ca}$, we unfold triangles sharing vertex $\mathbf{a}$, until we find one **Alive** vertex that neighbors $\mathbf{a}$ where the geodesic path goes across the edge connecting them both. In Figure 3.2b, we first unfold $\triangle$ $\mathbf{cad}$. $\mathbf{d}$ is not **Alive**, so we continue to unfold $\triangle$ $\mathbf{dae}$. $\mathbf{e}$ is **Alive** and $\mathbf{co}$ crosses $\mathbf{ea}$. We stop unfolding and solve for $\delta(\mathbf{c})$, the length of $\mathbf{oc}$ in $\triangle$ $\mathbf{cea}$. Similar to the center solver we can find $\tau(\mathbf{c})$ by rotating $\tau(\mathbf{a})$. A symmetric approach will unfold triangles around vertex **b**.

The center solver and the side solver assume we always find two adjacent **Alive** neighbors around **c**. But this is not always true especially when **c** is a saddle point. In this case, we use the passive solver in Figure 3.2c, which updates a new vertex using just one **Alive** neighbor. The distance value for **c**, $\delta(\mathbf{c})$, is set to the minimum value of $\delta(\mathbf{a}) + |\mathbf{ac}|$ among all Alive vertices, **a**, adjacent to **c**. **c** also inherits the polar angle $\tau(\mathbf{a})$ from **a**.

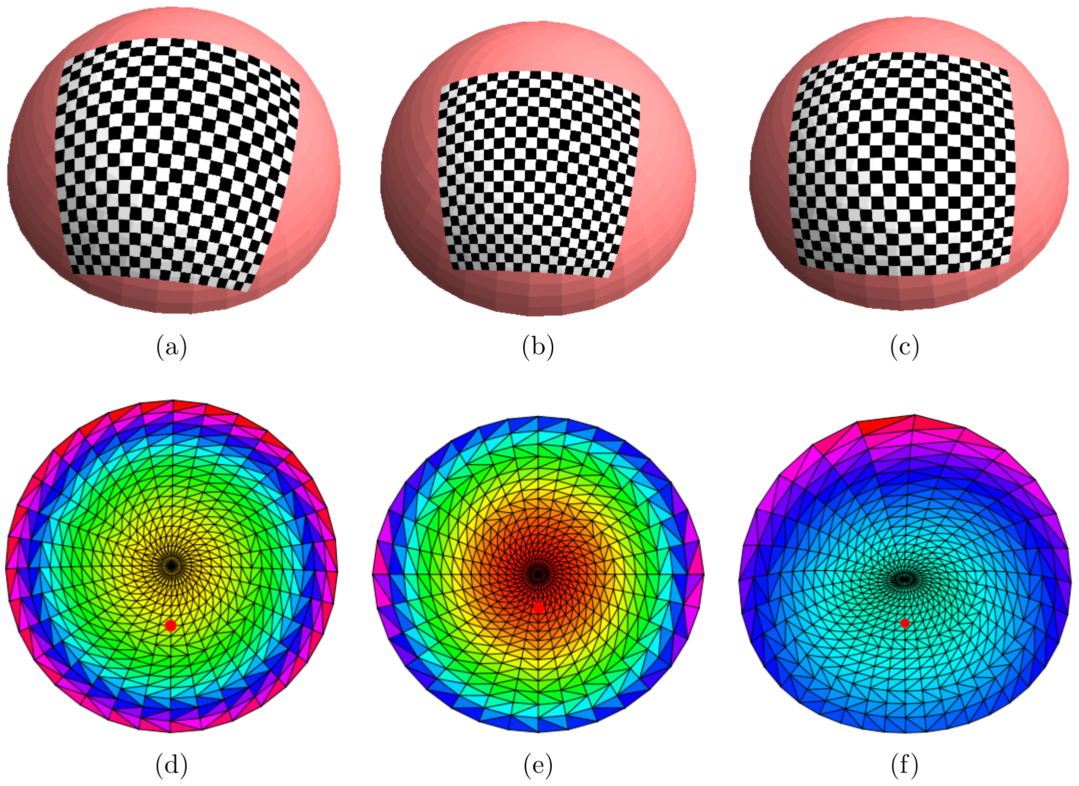
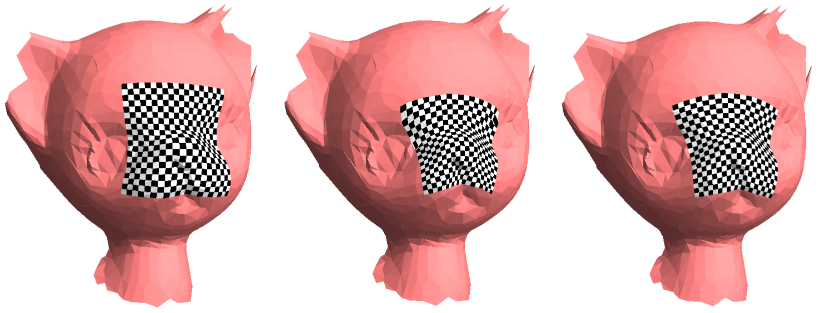
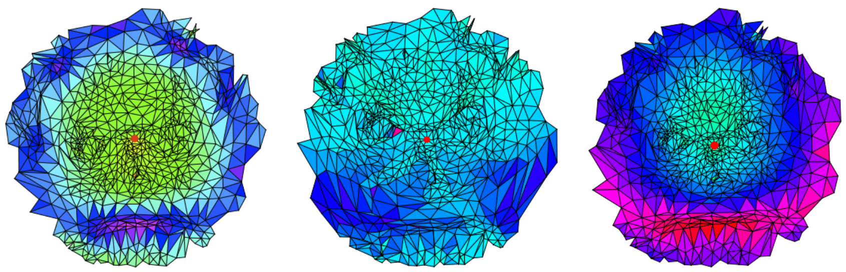


Figure 3.3: Parameterization result comparisons. Incremental unwrapping results (right) compared with other methods on isometric (left) and conformal (middle) settings. Top rows shows the texture mapping result. Bottom line shows the area distortion distribution.

Polar Coordinates Construction. After finishing propagation, we don't have to construct geodesic paths explicitly to obtain polar angles of vertices. Each vertex **v** has been associated with its geodesic distance and its polar angle, i.e, the polar coordinates, which are easily transformed to the texture coordinates. The connectivity of the mesh vertices ensures we can always update a vertex successfully from one of the three solvers, ensuring termination. Figure 3.3 shows comparisons between our incremental unwrapping method, the method (least squares conformal map) in the conformal setting, and the method (as-rigid-as-possible map) in the isometric setting.



(a) ARAP (b) Incremental (c) Hard alignment



(d) ARAP (e) Incremental (f) Hard alignment

Figure 3.4: Mapping result comparisons.

We find that those methods that are adaptive to global parameterization are not

		Our Method			**Fast Marching**			**Exact**
Model	**Faces**	**time (s)**	**max abs**	**ave rel**	**time(s)**	**max abs**	**ave rel**	**time (s)**
Plane	2,048	0.029	0.00%	0.00%	0.029	1.01%	1.10%	0.042
Sphere	3,840	0.062	0.00%	0.00%	0.058	0.21%	0.10%	0.12
Cat	24,192	0.41	1.23%	0.33%	0.38	2.31%	1.65%	1.78
Face	35,568	0.57	1.75%	0.25%	0.53	2.23%	1.04%	1.51
Armadillo	69,184	1.18	1.11%	0.72%	1.12	1.36%	1.18%	2.78
Bunny	69,451	1.12	0.49%	0.18%	1.05	0.91%	0.89%	3.24

Table 3.1: Comparison of our method with fast marching methods for accuracy and performance.

sensitive to the center information. Therefore, the incremental unwrapping method results in better parameterization than those methods despite the triangle foldings. The distortion comparison verifies that global methods try to distribute distortion evenly while incremental unwrapping tries to lower the distortion around the center.

Figure 3.4 shows more mapping result comparisons. Left three sub-figures show that the original ARAP gives out regular grid textures, however, the global texture size is not correct; the incremental one has good metric awareness but has texture defects caused by triangle foldings; the result of our new system has both a regular grid and good awareness of metric. The right three sub-figures shows the area distortions of three methods. The last figure shows that our method has the lowest distortion around the texture center (red dot) while pushing most distortion to the patch boundary. As an approximated method for computing geodesics, we compare our results with the original fast marching method numerically in both accuracy and performance. We use the accurate method proposed in [43] to obtain the precise geodesic distance $\tilde{\delta}(v)$ for each vertex. Table 3.1 reports the maximal absolute difference $|\tilde{\delta}(v) - \delta(v)|$ as the percentage of the object diameter and the average relative difference $|\tilde{\delta}(v) - \delta(v)|/\tilde{\delta}(v)$. As shown in Table 3.1, our method has better accuracy than fast marching while achieving very similar performance. Figure 3.5 shows isolines over surfaces by using our method for computing geodesic distance.

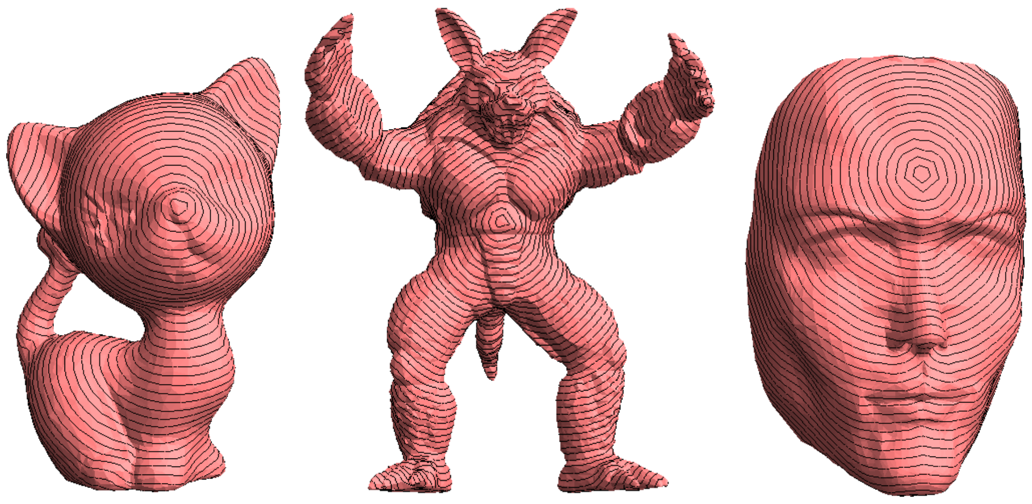


Figure 3.5: Isolines on models.

Similar to many other incremental unwrapping methods, our parameterization results may still have triangle foldings. The field-aware optimization in the next part overcomes this problem while still keeping the geodesic metric information. We note that Crane et al. [14] presented a method computing geodesic distance via heat flow. We didn't adopt this method because it distributes the numerical error evenly in the computing domain. Therefore, there is no guarantee that the chart center has more accurate distance values which might cause higher distortion when mapping the texture.

3.2.2 Field-Aware Parameterization

In this part, we make an isometric parameterization (ARAP) to be aware of the geodesic metric. ARAP assumes that only rotational transformations are allowed for triangle faces during the parameterization. However, this assumption is so strong that it hardly allows us to form a parameterization for a general surface. Therefore, ARAP measures the difference between the real affine transformation matrix and the rotational matrix of each triangle. The energy for optimization is the summation of individual differences as below,

$$E_{\text{ARAP}} = \sum_{t=1}^{|T|} A_t ||J_t - R_t||_F^2, \tag{3.1}$$

where A_t is the area of triangle t, J_t is the affine transformation from the original triangle t to its image in the parameterization space, R_t is the rigid transformation matrix, and $|T|$ is the number of triangles. Since we have the geodesic field which also acts as a parameterization, we can use the corresponding affine transformation matrix to represent the parameterization information naturally. We present a method augmenting ARAP.

Augmentation method. We use Jacobian G_t to indicate the transformation from the original triangle t to its image in the geodesic field. The energy function of our new field-aware system can be formulated by a blending parameter $\lambda_1 > 0$ as below,

$$E_{\text{AUG}} = \sum_{t=1}^{|T|} A_t ||J_t - R_t||_F^2 + \lambda_1 A_t ||J_t - G_t||_F^2. \tag{3.2}$$

For clarity, we call this parameterization method the *augmentation* method. The local-global method, which solves the nonlinear problem in ARAP, can be used again to solve the new field-aware system. In detail, during the local phase, ARAP uses singular value decomposition to obtain R_t; in the global phase, ARAP solves a stitched Poisson equation as below,

$$\begin{aligned} &\sum_{j \in N(i)} [\cot\theta_{i,j} + \cot\theta_{j,i}](\mathbf{u}_i - \mathbf{u}_j) = \\ &\sum_{j \in N(i)} [R_{t(i,j)} \cot\theta_{i,j} + R_{t(j,i)} \cot\theta_{j,i}](\mathbf{x}_i - \mathbf{x}_j), \end{aligned} \tag{3.3}$$

where $i = 1, \ldots, |T|$, $N(i)$ indicates vertex neighbors around vertex i, $\mathbf{x}_i$ is the coordinates of vertex i, and $\mathbf{u}_i$ is the parameterization coordinates of vertex i. Here pair (i, j) indicates the half-edge from vertex i to j, and $\theta_{i,j}$ is the angle opposite to the half-edge (i, j). In the field-aware system, we can compute R_t and G_t in the local phase. In the global phase, the field-aware Poisson equation is formulated by considering G_t and the blending parameter λ_1 for all triangles as,

$$\begin{aligned}(1+\lambda_1)\sum_{j\in N(i)}[\cot\theta_{i,j}+\cot\theta_{j,i}](\mathbf{u}_i-\mathbf{u}_j) = \\ \sum_{j\in N(i)}[(R_{t(i,j)}+\lambda_1 G_{t(i,j)})\cot\theta_{i,j} + \\ (R_{t(j,i)}+\lambda_1 G_{t(j,i)})\cot\theta_{j,i}](\mathbf{x}_i-\mathbf{x}_j).\end{aligned}$$

Hard field alignment. The field-awareness in system (3.2) is achieved by blending the affine transformation matrices, which encapsulate information of the geodesic field. Instead of using this full-fledged information, we can instead align with just the gradient field over the geodesic distance during the global optimization phase, which also produces comparable results. In detail, the Poisson equation in (3.3) can be represented by submatrices as $\begin{bmatrix} L & 0 \\ 0 & L \end{bmatrix} \cdot \begin{bmatrix} U \\ V \end{bmatrix} = \begin{bmatrix} \mathbf{b}_u \\ \mathbf{b}_v \end{bmatrix}$, where L is the discrete Laplace-Beltrami operator matrix, U and V are vectors formed by the u-components and v-components of texture coordinates respectively, and $\mathbf{b}_u$ and $\mathbf{b}_v$ are corresponding right sides of the linear system. Since each vertex on the local chart is associated with a geodesic value after obtaining the geodesic distance, we have defined a scale field M_C on the chart. We map M_C to the texture space by the correspondence of vertices between the surface space and the texture space. The new scale field M_T transfers the metric information from the surface domain to the texture domain.

For one triangle t, assume vertex indices are i, j and k. The gradient on this face is $\mathbf{d}_t = [g_k \cdot (\mathbf{u}_j - \mathbf{u}_i) + g_i \cdot (\mathbf{u}_k - \mathbf{u}_j) + g_j \cdot (\mathbf{u}_i - \mathbf{u}_k)]^{\perp}$, where g_i is the geodesic distance we've obtained during the incremental unwrapping phase. We have $|T|$ equations in this form from which to build a linear system related to gradients over the field. Assume the matrix on the left side of this system is K, formed by g_i, and the right side is $[\mathbf{g}_u, \mathbf{g}_v]^T$, formed by uv-components. Then we obtain a field-aware system as $\begin{bmatrix} K & 0 \\ 0 & K \end{bmatrix} \cdot \begin{bmatrix} U \\ V \end{bmatrix} = \begin{bmatrix} \mathbf{g}_u \\ \mathbf{g}_v \end{bmatrix}$. Let K^T be the transpose of K. We combine the field-aware system with the original ARAP system by another blending parameter $\lambda_2 > 0$,

$$\begin{bmatrix} L + \lambda_2 K^T K & 0 \\ 0 & L + \lambda_2 K^T K \end{bmatrix} \cdot \begin{bmatrix} U \\ V \end{bmatrix} = \begin{bmatrix} \mathbf{b}_u + \lambda_2 K^T \mathbf{g}_u \\ \mathbf{b}_v + \lambda_2 K^T \mathbf{g}_v \end{bmatrix}. \tag{3.4}$$

Relaxed field alignment. Under some conditions, artists might prefer regular grids rather than aligning the texture with the field on both directions and magnitudes. We allow alignment with directions only by adding relaxing variables into face gradients, which allows the system to be field-aware in a mild way. Within the isometric setting, we add s_t for triangle t and align with $s_t \mathbf{d}_t$. Therefore, s_t provides the relaxation for the direction alignment. Assume Λ is a set of diagonal matrices composed by s_t, and $\mathbf{g}_u^D$ and $\mathbf{g}_v^D$ are diagonalized $\mathbf{g}_u$ and $\mathbf{g}_v$ respectively. System (3.4) can be relaxed as

$$\begin{bmatrix} \lambda_2 K^T K + L & 0 & \lambda_2 K^T \mathbf{g}_u^D \\ 0 & \lambda_2 K^T K + L & \lambda_2 K^T \mathbf{g}_v^D \\ \lambda_2 (K^T \mathbf{g}_u^D)^T & \lambda_2 (K^T \mathbf{g}_v^D)^T & \lambda_2 (\mathbf{g}_u^D)^T \mathbf{g}_u^D + \lambda_2 (\mathbf{g}_v^D)^T \mathbf{g}_v^D \end{bmatrix} \begin{bmatrix} U \\ V \\ \Lambda \end{bmatrix} = \begin{bmatrix} \mathbf{b}_u \\ \mathbf{b}_v \\ 0 \end{bmatrix}. \tag{3.5}$$

This system can be obtained from a direct relaxed system as below,

$$\begin{bmatrix} \lambda_2 K^T K + L & 0 & \lambda_2 K^T \mathbf{g}_u^D \\ 0 & \lambda_2 K^T K + L & \lambda_2 K^T \mathbf{g}_v^D \end{bmatrix} \begin{bmatrix} U \\ V \\ \Lambda \end{bmatrix} = \begin{bmatrix} \mathbf{b}_u \\ \mathbf{b}_v \end{bmatrix}. \tag{3.6}$$

Our goal is to augment the left side of the linear system to have 3×3 submatrices. From symmetry of this term, we know the system should be in the form as

$$\begin{bmatrix} \lambda_2 K^T K + L & 0 & \lambda_2 K^T \mathbf{g}_u^D \\ 0 & \lambda_2 K^T K + L & \lambda_2 K^T \mathbf{g}_v^D \\ \lambda_2 (K^T \mathbf{g}_u^D)^T & \lambda_2 (K^T \mathbf{g}_v^D)^T & X \end{bmatrix} \begin{bmatrix} U \\ V \\ \Lambda \end{bmatrix} = \begin{bmatrix} \mathbf{b}_u \\ \mathbf{b}_v \\ 0 \end{bmatrix}. \tag{3.7}$$

We need to figure out what X is.

If we view gradient equations as constraints of the ARAP system, another direct relaxed system can be formed as,

$$\begin{bmatrix} L & 0 & 0 \\ 0 & L & 0 \\ \tilde{\lambda}_2 K & 0 & \tilde{\lambda}_2 \mathbf{g}_u^D \\ 0 & \tilde{\lambda}_2 K & \tilde{\lambda}_2 \mathbf{g}_v^D \end{bmatrix} \begin{bmatrix} U \\ V \\ \Lambda \end{bmatrix} = \begin{bmatrix} \mathbf{b}_u \\ \mathbf{b}_v \\ 0 \\ 0 \end{bmatrix}.$$

This system cannot be solved directly because the left side is not a square matrix. We compute the left side in its quadratic form as

$$\begin{bmatrix} \tilde{\lambda}_2^2 K^T K + L^T L & 0 & \tilde{\lambda}_2^2 K^T \mathbf{g}_u^D \\ 0 & \tilde{\lambda}_2^2 K^T K + L^T L & \tilde{\lambda}_2^2 K^T \mathbf{g}_v^D \\ \tilde{\lambda}_2^2 (K^T \mathbf{g}_u^D)^T & \tilde{\lambda}_2^2 (K^T \mathbf{g}_v^D)^T & \tilde{\lambda}_2^2 (\mathbf{g}_u^D)^T \mathbf{g}_u^D + \tilde{\lambda}_2^2 (\mathbf{g}_v^D)^T \mathbf{g}_v^D \end{bmatrix}. \tag{3.8}$$

Compare (3.7) and (3.8). We have found a possible term for $X = \lambda_2 (\mathbf{g}_u^D)^T \mathbf{g}_u^D + \lambda_2 (\mathbf{g}_v^D)^T \mathbf{g}_v^D$, where $\lambda_2 = \tilde{\lambda}_2^2$. Since this method only aligns with directions over the

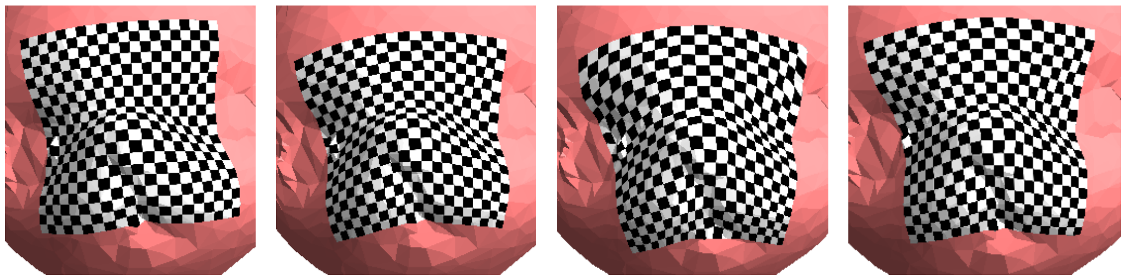

(a) ARAP (b) Augmentation (c) Hard (d) Relaxed

Figure 3.6: Comparisons of different parameterizations.

gradient field approximately, we call it the *relaxed (field) alignment* method.

Figure 3.6 shows results of ARAP, the augmented field-aware system by blending the affine transformation, the hard alignment system and the relaxed alignment system. Since the relaxed system isn't aware of the magnitude of the field gradients, it is not as sensitive to the geodesic metric as the hard alignment system. However, the awareness of directions of gradients provides a reasonable compromise between ARAP and the incremental unwrapping. As we know, given good initial texture coordinates (such as obtained by the incremental unwrapping), the original ARAP can converge very quickly in a few iterations. In our experiment, we found it usually converged within twenty iterations for those local charts on the surface. After augmenting the original method with the new geodesic information, the linear system is bigger than before. However, the convergence speed is faster than before. It usually took five to ten iterations to converge to a stable status, and the triangle foldings were removed in most cases. We believe that since the ARAP framework always obtained the signed rotational transformation in the local part, this prevented foldings very effectively.

Finally, for most cases where the incremental unwrapping produces triangle foldings, our optimization process removes those foldings. However, if any folding hap-

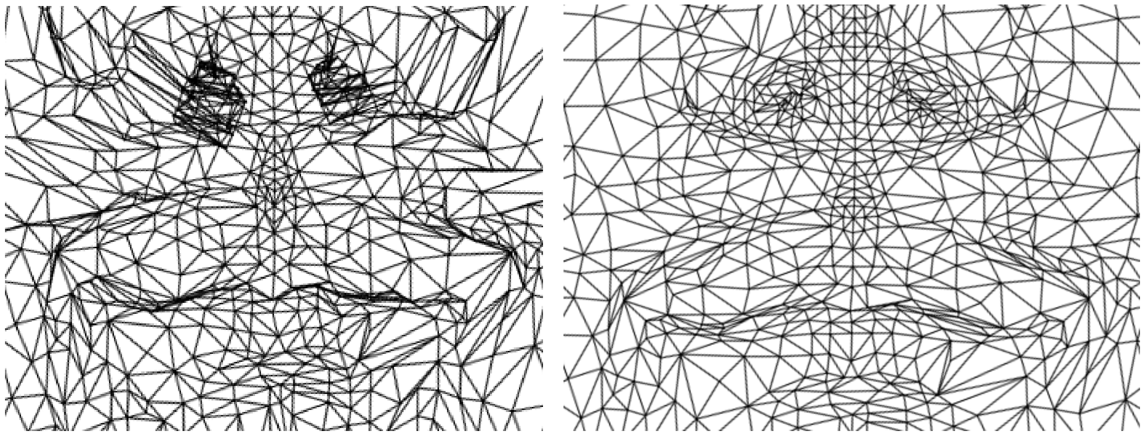

(a) Before (b) After

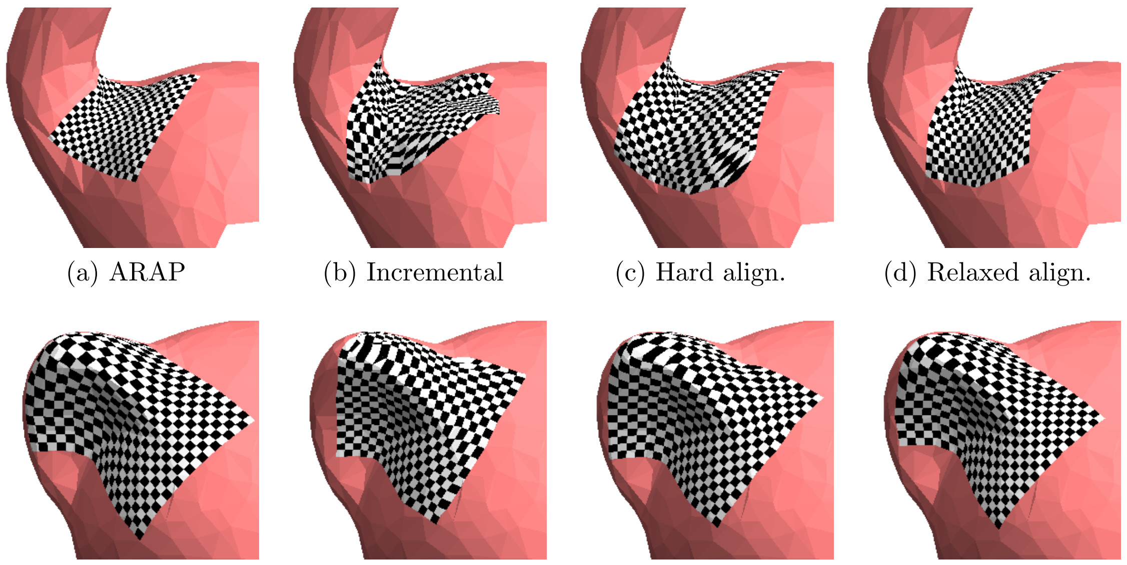
Figure 3.7: Folding removal by applying the alignment system.

pens, a folding-free result can be guaranteed by a post processing presented by Kami et al. [32]. Figure 3.7 shows the result before and after applying our system, i.e., removing the foldings, on the girl face model. We provide more examples on mapping results of different parts of the cat model in Figure 3.8.

3.2.3 Weighted ARAP

Weighted ARAP, which assigns vertices or faces different weights, is one way to make the original ARAP method aware of the geodesic field. In this subsection, we describe several weighted ARAP implementations and compare them with our field-aware parameterization method.

The first method is vertex-weighted ARAP method, which assigns different weights to each vertex. The weight function of vertex i is $w_i = \text{Reg}(1.0/(g_i + \delta))$, where g_i is the geodesic distance of vertex i, and δ is a small value for handling the case that $g_i = 0$. Reg is a function that regularized the weight value into the range of $(0, 1]$. For this vertex-weighted setting, the linear system of ARAP in equation 3.3



(a) ARAP (b) Incremental (c) Hard align. (d) Relaxed align.

(e) ARAP (f) Incremental (g) Hard align. (h) Relaxed align.

Figure 3.8: More examples of comparisons on ARAP, incremental unwrapping, and alignment systems. As we see in those results from alignment, we have maintained sensitivity to the geodesic metric while removing triangle foldings.

is transformed to be

$$\sum_{j\in N(i)} [\cot\theta_{i,j} + \cot\theta_{j,i}](w_i\mathbf{u}_i - w_j\mathbf{u}_j) = \sum_{j\in N(i)} [R_{t(i,j)}\cot\theta_{i,j} + R_{t(j,i)}\cot\theta_{j,i}](\mathbf{x}_i - \mathbf{x}_j). \tag{3.9}$$

This system can also be solved by the local-global method. In the local phase, we still extract the SO(2) matrix for each triangle. In the global phase, we view $w_i\mathbf{u}_i$ as a new variable, *denoted* by $\mathbf{z}_i$, and solve it using the same global method. $\mathbf{u}_i$ can be obtained by $\mathbf{z}_i/w_i$. We apply this method on the face model. Results are shown in Figure 3.9. We find that the vertex-weighted method is not a good solution for field-aware parameterization by using the distance weight function.

Another method is the face-weighted ARAP method, which assigns different

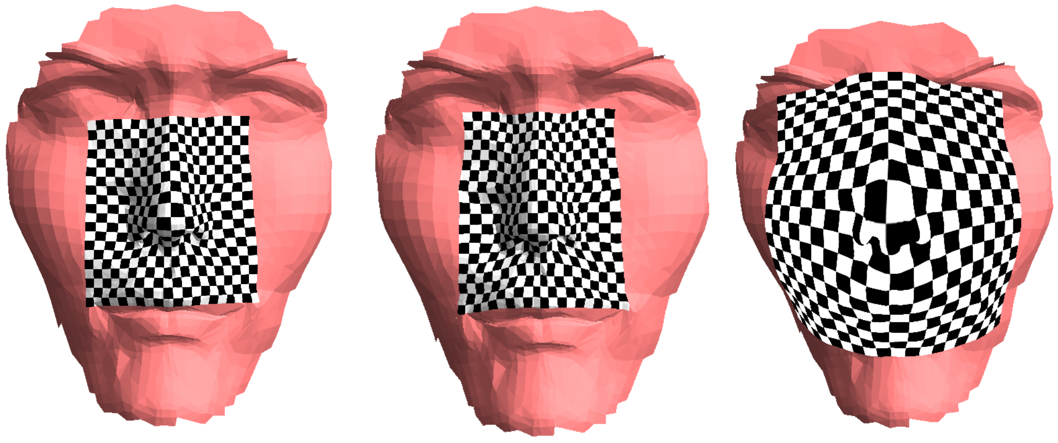

(a) ARAP (b) Hard alignment (c) Vertex-weighted

Figure 3.9: Comparing vertex-weighted ARAP with our previous results.

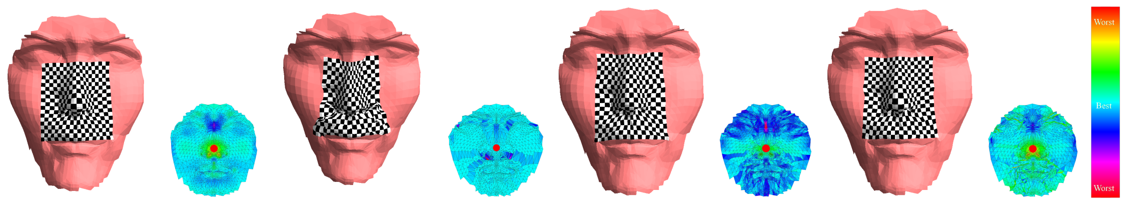


Figure 3.10: Results of field-aware parameterizations.

weights to each face. The weight function of face t is $w_t = \text{Reg}(1.0/(g_i + g_j + g_k))$, where g_i, g_j and g_k are geodesic distances of vertices in triangle t. Reg is a function that regularizes the weight value into the range of $(0, 1]$. For this face-weighted setting, weights of ARAP in equation 3.1 are transformed to be A_t/w_t.

We show distortion distributions in Figure 3.10. The first group of figures shows the result of a global method with an isometric setting. The second group of figures shows the result of the incremental unwrapping considering the geodesic distance. The third group of figures is the field-aware result with (hard) alignments over directions and magnitudes of the geodesic gradient field. The last group of figures

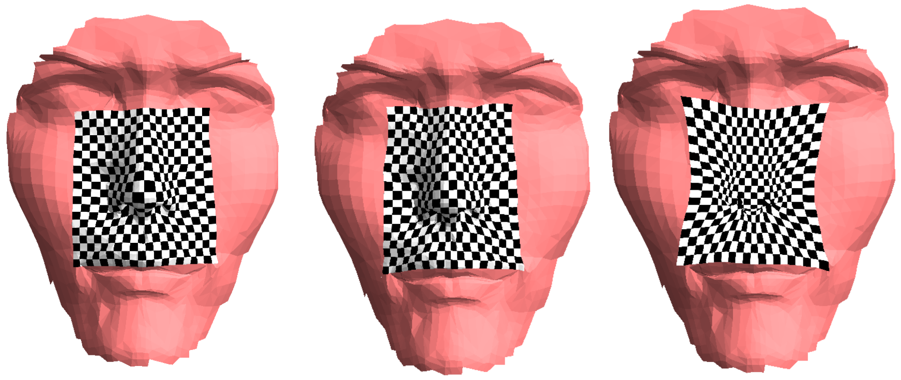

(a) ARAP (b) Hard alignment (c) Face-weighted

Figure 3.11: Comparing face-weighted ARAP with our previous results.

is the field-aware result with (relaxed) alignments only over directions of the field. The small side figures show area distortions for each parameterization. For those results, we conclude that the global isometric method only distributes the distortion evenly while ignoring the texture center and the geodesic metric. Our field-aware results consider that information so that the texture metric is more like that of the incremental unwrapping while generating high quality parameterizations. We provide two levels of field-awareness, hard and relaxed, to provide flexible choices for surface painting.

This system can also be solved by the local-global method. From the result, we find that the face-weighted ARAP is not a good solution for field-aware parameterization by using the distance weight function, as shown in Figure 3.11.

3.3 Painting System Implementation

We developed our field-aware parameterization method in order to improve the quality of real-time 3D painting. We have implemented it as a core part of our 3D painting system. The key features of our system are a view-dependent local parameterization and view-dependent oriented painting.

Our system stores colors using Mesh Colors [65], but would be easily adapted to other surface-based color storage schemes, such as that in [9]. It is worth noting that one of the challenges in using these approaches for color storage is that the standard 2D image manipulation tools artists use to paint texture maps are not readily available for systems that store all color data on the mesh itself. Our method, by interactively generating local 2D maps, makes it possible to apply all the standard 2D image manipulation tools even to such 3D surfaces, without having to worry about issues such as seams and chart boundaries.

3.3.1 View-Dependent Local Parameterization

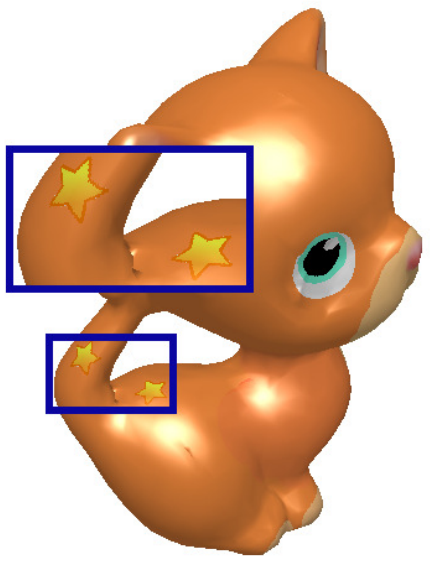
Painting is a local behavior. In 3D systems, the canvas is a model surface. Instead of parameterizing the whole model and unfolding it onto a planar texture space, we want to separate the local surface region being painted and unfold it for mapping the brush strokes. We need to create a local chart from *relevant* triangles falling under the brush in the 3D screen space view. Locally, the surface might be convex, concave or saddle. We use our two-phase method to parameterize the local surface on the fly. Our parameterization adapts to the screen position of the painting tool and the perspective view of the model. In detail, we have three steps to build the local chart by our method.

Center triangle picking. Given the brush center, we pick the center triangle as the one containing the brush center. Note that an AABB or BSP tree for the model can help to accelerate this operation.

Boundary triangles picking. The second step aims to find the visible triangles whose image on the screen intersects with the brush region. Instead of focusing on all triangles covered by the brush, we only identify those intersecting the brush's screen-space boundary. We call these **boundary triangles**, which can be obtained using ray intersection or screen-space picking.

Local chart construction. Given the center triangle and a set of boundary triangles, we construct a local chart that will encompass the region surrounding the center point including both nearby triangles falling under the brush in screen space and hidden triangles that nevertheless need to be painted on by the stroke. We unwrap the surface starting from the picked center, until we reach all boundary triangles. The full set of triangles reached forms the local chart. Starting from the center triangle, a breadth-first traversal of faces is conducted, adding triangles encountered to the local chart. This continues until either all the boundary triangles are found, or some maximum number of visited triangles is reached. The maximum number prevents the excessive growth of the local chart when the brush is too large (e.g. covering components on distant portions of the mesh that are close in screen space). The local chart, which is a continuous triangular 3D surface, is homeomorphic to a disc. We can apply our parameterization on it directly.

With the given center and the local chart we are able to apply our parameterization method. Figure 3.12 gives two parameterization examples from our system that highlight the benefits of our local parameterization. Figure 3.12a is a genus one cat model. In Figure 3.12b, we place the mouse such that the brush area overlaps both the tail and back, which are widely separated on the surface. Our system can identify the portion to be parameterized and painted correctly, based on the focus (center of the brush); the figures show focus on both the tail and the back. The important effect is seen in the results of the star shape painted on both the tail and the back. For the tail, even though the brush overlaps the silhouette of the model, the brush stroke is painted correctly on the model itself, wrapping around the model locally. For the back, the star brush stroke is painted onto the model, even in the


(a) Model

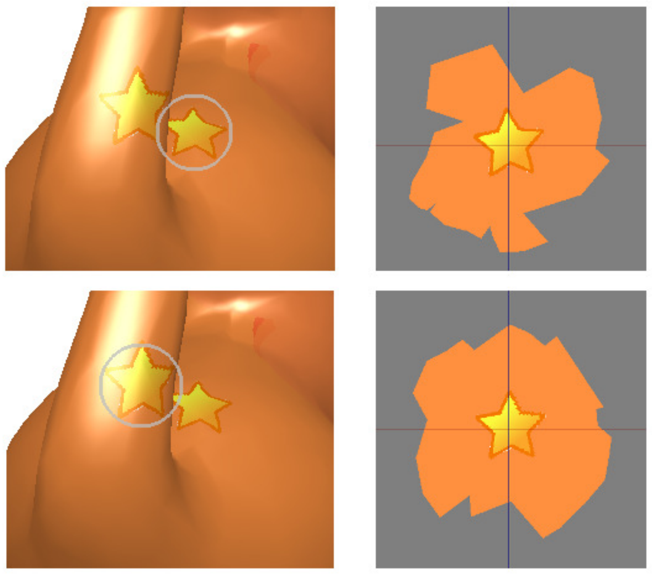

(b) Parameterization Views

Figure 3.12: View-dependent local parameterization. The white circle indicates the brush. Though the two places are very near each other, our system distinguishes the cat back and the cat tail so that correct local charts are unwrapped.

area obscured by the tail. It is worth noting that both stars have no obvious distortion. Compared to [28] these results justify that our parameterization method is not sensitive to the local geometry, and compared to pure projection methods this demonstrates that we can have true 2D operations over the surface.

3.3.2 View-Dependent Oriented Painting

Artists can zoom in or out, translate or rotate the model. These operations change the perspective view of the model surfaces, i.e. the canvas. Many current painting systems require the user to adjust the orientation or the scaling size of textures (strokes). In our system, they are decided by the current view perspective of the model. Our painting implementation adapts to those changes of views and paints the model in a correct way.

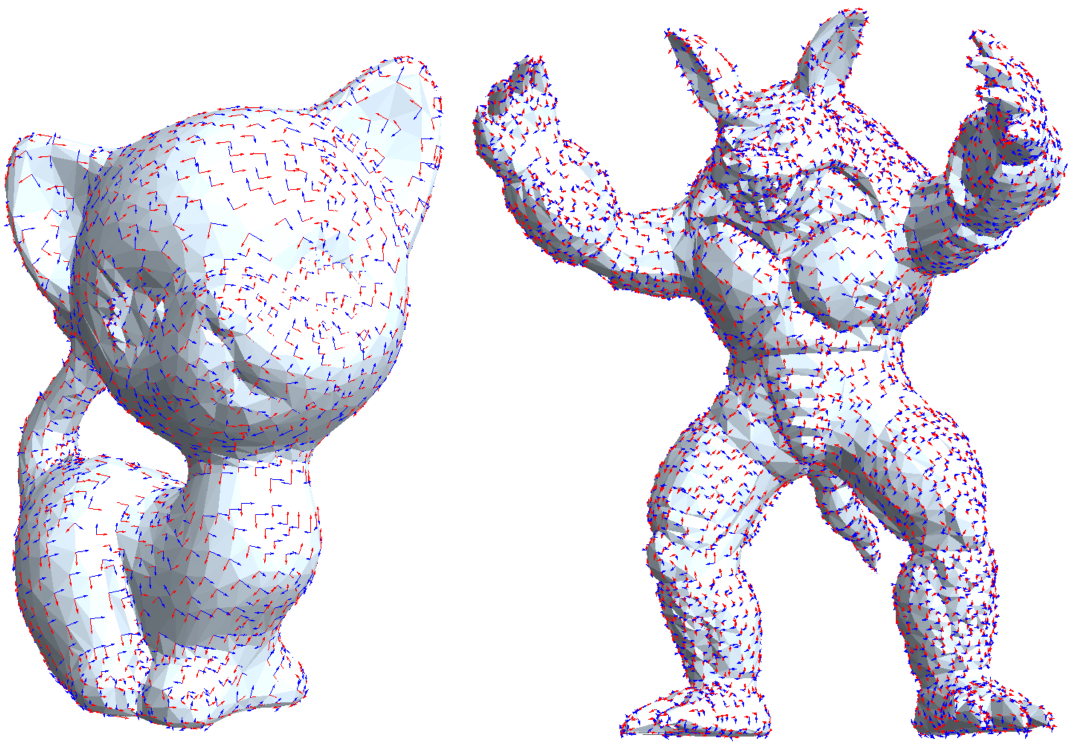


Figure 3.13: Principal direction fields on models are references for stroke orientations.

Orientation. While some strokes are radially symmetric (e.g. solid, Gaussian blur), others have orientation (e.g. stamp). We thus want our local parameterization to have a “good” orientation. We compute the directions of principal curvatures at each vertex, as shown in Figure 3.13. Principal directions form the local frames at each vertex. The texture orientation is decided by the vertex frame nearest to the texture center. There is an angle difference between the local frame and the vertical direction of the perspective. The texture is rotated by the difference angle in the texture field so that it is always aligned with the user view. We also allow the artist to rotate the mapping view manually so that the stroke can be mapped in an arbitrary angle if desired.

Brush Strokes Construction. To illustrate painting operations, we implemented a few different brush operations modeled on typical 2D brushes available in systems such as GIMP and Photoshop. The properties we observed in typical brush strokes include **size** and **pattern**, and we implemented several examples of each in different brushes and stamps. Since we do not have a 2D raster canvas, and parameterizations change rapidly, we do not have the notion of a pixel size for determining scaling, as is typical in 2D paint systems. We can define the size in terms of the 3D view (typical), or the 2D parametric space based on geodesic distance.

Coloring Samples. With the local chart parameterized and oriented, we can map the texture to the surface by projecting the region of strokes onto the (u, v) domain, where the local chart is unfolded. The mesh color sample points of the triangles within the local chart are also mapped into the (u, v) domain, and are assigned the appropriate color based on the stroke and brush parameters.

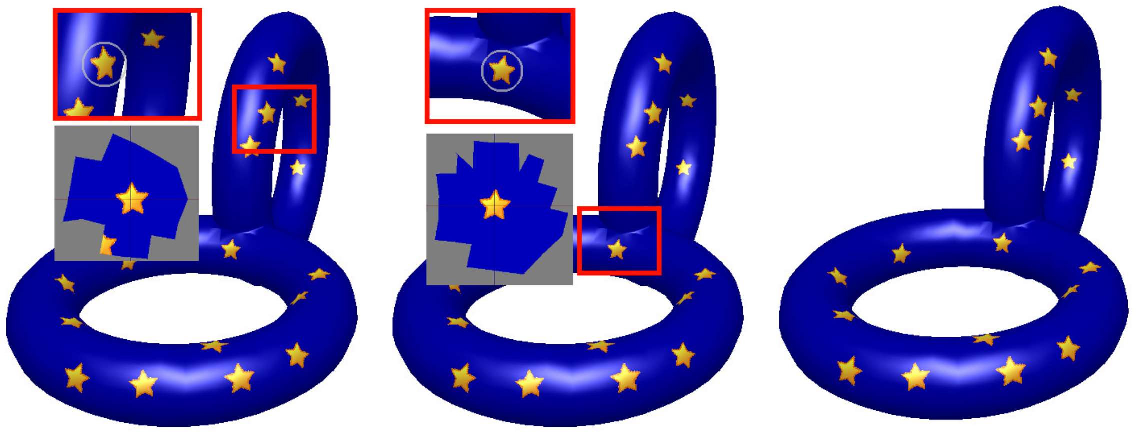

(a) Editing view (b) Editing view (c) Result

Figure 3.14: Painting a genus 2 model. At left are views of two stamping operations, at right is the result. Note that the local parameterization is always homeomorphic to a disc.

3.3.3 Painting Results

Our painting system is implemented on a PC with an Intel Core(TM) i7 3.4GHz processor and 16GB RAM. We show some examples here.

We first test our system on different geometric shapes. Figure 3.14 shows our painting results on a high genus model. We draw strokes on convex, concave and saddle parts of the model. Our system behaves stably. The texture size is also kept the same since the parameterization is sensitive to the geodesic metric. We invited art students to try our paint system. They produced the complete painting results in Figure 3.15.

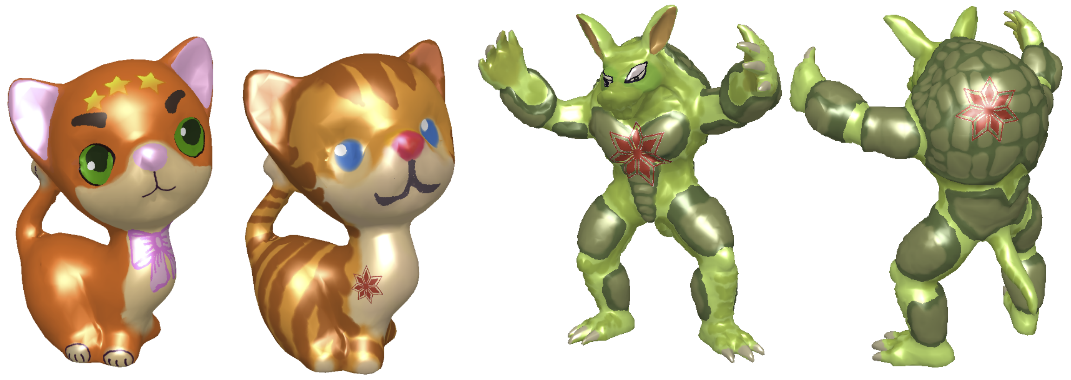


Figure 3.15: Examples of objects painted with our system.

3.4 Discussion on Field-Aware Parameterization

Our two-phase local parameterization is an efficient method for mapping texture on the local surface. The incremental unwrapping flattens the surface by approximating the exponential map, which has a comparable performance to the fast marching method. The optimization obtains texture coordinates with a few iterations. All this makes our method very efficient for constructing the real time parameterizations in

3D painting systems. When constructing the optimization, we provide a novel way to make the gradient in the texture field align with any given field. We use this to make our parameterization result aware of the geodesic metric so that the distortion is redistributed considering the center information. The geodesic field is reflected by blending affine transformations into the isometric setting. Those affine transformations can also be replaced by the intrinsic gradient field over the geodesic distance. Two methods with different gradient alignments are proposed, which allow corresponding numeric systems to have a linear form so that the computing complexity is reduced.

Our method has two limitations that would be avenues for future work. First, the method does not handle holes in a surface ideally, since the geodesic path cannot go over holes using current solvers. While there are methods for filling holes, it is not clear whether these computations could be implemented into a real-time system as our method has been. Note however, that holes are only an issue when painting around the hole border; because our parameterization is local, holes farther away do not matter. Second but related, our method assumes the region being mapped is relatively local, such that it is reasonable for the triangles within to be homeomorphic to a disc. Though we provided painting examples over high genus models, it is likely that this assumption would break down, and the parameterization would fail, if a brush covered a large enough region of a model with high genus. Again, this is only an issue if pushing the painting process to an unlikely extreme.

4. MIXED-INTEGER OPTIMIZATION FOR GLOBAL PARAMETERIZATION

Global parameterization usually requires to solve a mixed-integer optimization problem. We focus on parametric coordinate computation and quad mesh extraction. Unlike previous methods, we plan to avoid segmenting the model, so the distortion can be distributed more evenly over the model. Meanwhile, we will propose a solution for extracting a quad mesh after parameterization rather than simply drawing isolines of integer grids on models, which is more appropriate for further model processing.

4.1 Surface Parameterization

Before presenting techniques on surface parameterization, we first list symbols that are used throughout this work.

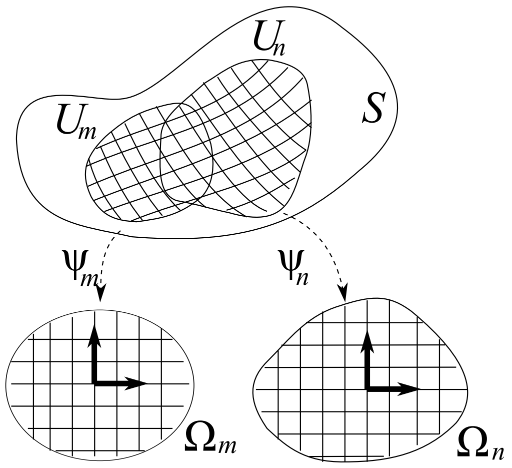
- S is the surface from which we extract the quad mesh. We assume S is a 2-dimensional manifold of triangles.
- U is the local chart on manifold S. ψ is the map that maps U to a 2-dimensional domain Ω. We build the local chart for each vertex, which allows us to use indices of vertices to index charts.
- V is the set of vertices on S in the discrete setting. We use $m, 1 \leq m \leq |V|$, to denote the index of one vertex and use $\mathbf{p}_m$ to denote its geometric coordinates.
- E is the set of edges on S. We use $(m, n) \in E, 1 \leq m, n \leq |V|$ to denote each edge.
- $\nabla \mathbf{t}$ is the gradient over texture coordinate $\mathbf{t}$.
- $\Pi_{m,n}$ is the matching matrix of the transformation from chart m to chart n.

- $\mathbf{g}_{m,n}$ is the translation vector of the transformation from the chart m to chart n. For seamless parameterization, we require all components of $\mathbf{g}$ to be integers.
- N_m is the vertex index set of neighbors of vertex m.
- $C_{m,n}$ is the set of common neighbors of vertex m and vertex n, i.e. $C_{m,n} = N_m \cap N_n$.
- $\mathbf{t}_{m,n}$ are parametric coordinates for vertex n within the local chart of vertex m.

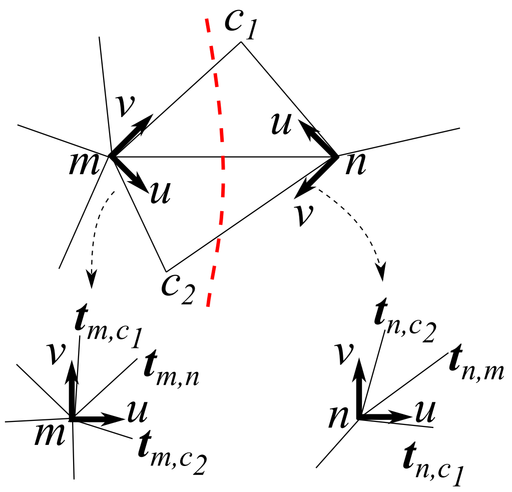
For a given cross field, f, obtaining the field aligned parameterization means finding the best parametric coordinates, $\mathbf{t}$, that minimize the energy

$$\int_S \|h \cdot f - \nabla \mathbf{t}\| ds, \tag{4.1}$$

where h controls the size of grids and ds is an element of surface S.



(a) Continuous setting



(b) Discrete setting

Figure 4.1: Chart matching.

4.1.1 Parameterization Setting

In this section, we first describe the continuous setting of the parameterization model. We borrow the concept of local charts on a manifold to reveal the essential idea of the field aligned parameterization. Then we move our discussion to the discrete setting so that concepts and operations on the continuous setting can be formulated on the discrete form of the manifold, the triangular mesh.

Chart matching. In the continuous setting, we assume we have a 2-manifold S with local mapsψ $: U \subset S \rightarrow \Omega_m \in \mathcal{R}^2$. The local parameterization can be designed within each chart U_m. Within a chart, integer grids are formed by intersections of isolines $l_u \in \mathcal{Z}\times\mathcal{R}$ and $l_v \in \mathcal{R}\times\mathcal{Z}$. Figure 4.1 shows the basic idea of chart matching. The grid lines from two charts should be matched if they share a common region, as shown in Figure 4.1a. To ensure the global continuity of isolines on the surface, the local mapsψ_m andψ_n should be selected to make sure that the grid isolines from two overlapped charts match in the common region $U_m \cap U_n$. In addition, each isoline follows the direction of the gradient of the parameterization functionψ . We use the **matching** concept from [30] to define the transformation between overlapped local charts, which is formulated as

$$\nabla\psi_n(\mathbf{p}) = \Pi_{m,n}\nabla\psi_m(\mathbf{p}) + \mathbf{g}_{m,n}, \quad \mathbf{p} \in U_m \cap U_n. \tag{4.2}$$

Π is the Jacobian connecting U_m and U_n. Since we are matching isolines on integer grids, the Π's are required to be matrices that match coordinates from axis to axis. Therefore, Π can have four different forms that each describe a valid orientation (each being a 90 degree rotation). $\mathbf{g}_{m,n}$ is an integer vector that allows the chart to translate an integer amount. We translate this continuous setting into the triangulated discrete setting to build our parameterization model.

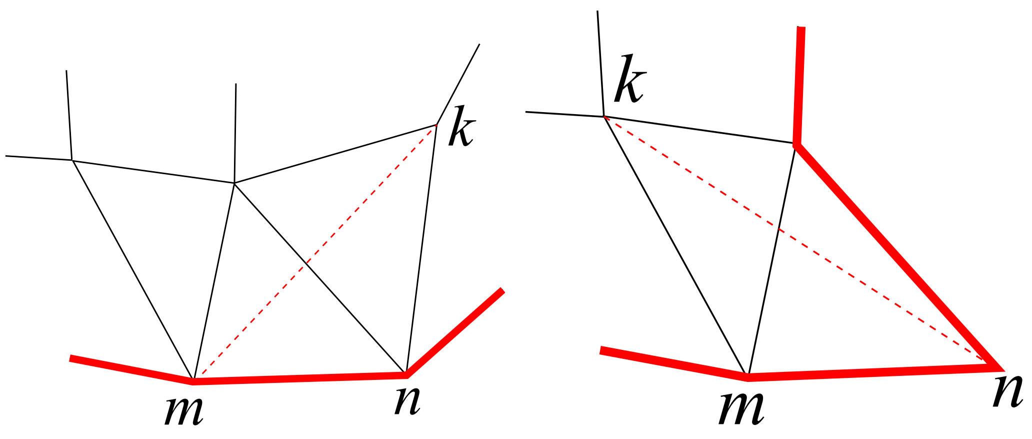
Discrete chart. In the discrete setting, surface S is a triangulated 2-manifold. We define the local chart of one vertex to be its one-ring region. The local frame for each vertex is built on the given field, which is designed by the method in [34]. Each vertex m is associated with a pair of orthogonal vectors that are viewed as two axes of the frame. In this frame, we have coordinates $\mathbf{t}_{m,m}$ for vertex m itself, and coordinates $\mathbf{t}_{m,n_i}$ for neighbors of m, i.e., $n_i \in N_m$. Thus the chart transformation from m to n in the discrete setting, as shown in Figure 4.1b, can be built on the common neighbors of m and n using local coordinates, which is formulated as below.

$$\mathbf{t}_{n,c} = \Pi_{m,n}\mathbf{t}_{m,c} + \mathbf{g}_{m,n}, \tag{4.3}$$

where $c \in C_{m,n}$. The transformation is invalid if $c \notin C_{m,n} \cup \{m, n\}$ because c is not in the common region of neighborhoods of m and n. We associate this transformation to edge (m, n) since m and n are adjacent. In addition, note that for edge (m, n), $\mathbf{t}_{m,n}$ and $\mathbf{t}_{n,m}$, defined in different local charts, have different meanings.

Global parameterization. Current field-guided methods usually require segmenting the model so that singularities can be placed on boundaries. Those seams are usually placed along model edges. Parts on two sides of edges only share seam edges. In other words, seam edges have two copies in the texture space. In our discrete setting, two adjacent charts share a common region formed by their common neighbors. Note that our method does not require us to segment the model before parameterization. The only implicit segmentation implied by the chart matching associated with each edge, shown as the red dashed line in Figure 4.1b, is orthogonal to the corresponding edge.

Since the surface is a 2-dimensional triangular manifold, we have $|C_{m,n}| = 2$ for all edges except those on boundaries. We can enforce the property $|C_{m,n}| \geq 2$ to



(a) Extra edge (b) Special case

Figure 4.2: Extra edge (dashed lines) for boundaries (solid red lines).

all edges by adding extra edges to boundary vertices. Figure 4.2 shows method for boundary edges. As shown in Figure 4.2a, in detail, if $|C_{m,n}| = 1$, we pick vertex $k \in N_n$ and $k \notin N_m$, and add one new edge (k, m). For the special case shown in Figure 4.2b, we find $k \in N_m$ and $k \notin N_n$, and add the new edge (k, n). If the mesh model is not a single triangle, we can always find the vertex k for building the extra edge. This operation enforces the property that $|C_{m,n}| \geq 2$ on all edges, including newly added edges.

4.1.2 Parameterization Model

In this part, we build the optimization function and related constraints for computing parametric coordinates.

Optimization Objective Function. The energy function in Equation 4.1 can be discretized as in [38] as below.

$$\sum_{m \in V} \sum_{n \in N_m} ||\tilde{\mathbf{t}}_{m,n} - \tilde{\mathbf{t}}_{m,m} - h \cdot f_m(\mathbf{p}_n - \mathbf{p}_m)||^2, \tag{4.4}$$

where $\tilde{\mathbf{t}} = (\mathbf{t}, 0)^T$ is a three dimensional vector. f_m is a 3×3 matrix augmented from $\nabla\psi_m = [v_{m_1}, v_{m_2}]$ by $f_m = [v_{m_1}, v_{m_2}, v_{m_1} \times v_{m_2}]$, where v_{m_1} and v_{m_2} are two local frames associated with vertex m. We use the product of this field tensor and the difference of coordinates between m and its neighbors to approximate the gradient. The optimization term implies that the local parametric coordinates of all neighbors of vertex m align with the gradient field over the vertex m. Since the chart transformation in Equation 4.3 is bound to each edge, we transform the energy function above equivalently to the representation on edges as below.

$$\sum_{\forall(m,n)\in E} \sum_{\forall c \in C_{m,n}} (||\tilde{\mathbf{t}}_{m,c} - \tilde{\mathbf{t}}_{m,m} - h \cdot f_m(\mathbf{p}_c - \mathbf{p}_m)||^2 + ||\tilde{\mathbf{t}}_{n,c} - \tilde{\mathbf{t}}_{n,n} - h \cdot f_n(\mathbf{p}_c - \mathbf{p}_n)||^2). \tag{4.5}$$

For each edge (m, n), the first term implies local parametric coordinates of common neighbors of vertex m and n align with the gradient field over the vertex m. Similarly, the second term implies the local parametric coordinates of common neighbors align with the gradient field over the vertex n. Next we define constraints that enforce the chart matching or other properties on edges and vertices.

C1. Chart Matching Constraint

$$\Pi_{m,n}\mathbf{t}_{m,c} + \mathbf{g}_{m,n} = \mathbf{t}_{n,c}, \tag{4.6}$$

$\forall c \in C_{m,n}$ and $\forall(m,n) \in \tilde{E}$, where $\mathbf{g}_{m,n}$ is an integer vector and where $\tilde{E} \subset E$. $\Pi_{m,n}$ is the Jacobian for the transformation from chart m to n. Note that, though it has a similar form as in [30], C1 is enforced on our new discrete setting, which connects local charts using common neighbors. C1 can be reduced to C2 which eliminates the integer vector $\mathbf{g}_{m,n}$.

C2. Reduced Chart Matching Constraint

$$\Pi_{m,n}(\mathbf{t}_{m,c_1} - \mathbf{t}_{m,c_i}) = \mathbf{t}_{n,c_1} - \mathbf{t}_{n,c_i}, \tag{4.7}$$

$\forall c \in C_{m,n}$ and $\forall (m,n) \in \tilde{E}$, where $i \neq 1$ and where $\tilde{E} \subset E$. This constraint is obtained by plugging c_1 and c_i into Equation 4.6 respectively and subtracting two equations. This reducing technique was first proposed by Nieser et al. [48] to reduce the size of the linear system. We use the same idea to eliminate integer variables so that the linear system can be simplified. Since we have $|C_{m,n}| \geq 2$ for all edges, we can always reduce C1 to C2 so that the translation vector $\mathbf{g}_{m,n}$ is eliminated if necessary.

C3. Continuity Constraint

$$\Pi_{m,n}\mathbf{t}_{m,n} + \mathbf{g}_{m,n} = \mathbf{t}_{n,n}, \tag{4.8}$$

$$\Pi_{m,n}\mathbf{t}_{m,m} + \mathbf{g}_{m,n} = \mathbf{t}_{n,m}, \tag{4.9}$$

$\forall (m,n) \in \tilde{E}$, where $\mathbf{g}_{m,n}$ is an integer vector and where $\tilde{E} \subset E$. C3 ensures the continuity of the parametric coordinates under the transformation operation. We will prove those properties when discussing the quad mesh extraction.

C4. Interpolation Constraint

$$\sum_{c \in N_m} (\tilde{\mathbf{t}}_{m,c} - \tilde{\mathbf{t}}_{m,m} - h \cdot f_m(\mathbf{p}_c - \mathbf{p}_m)) = 0, \tag{4.10}$$

$\forall m \in \tilde{V}$, where $\tilde{V} \subset V$. This constraint ensures that the mesh edges will go through point m if it has an integer parametric coordinate component. This constraint is enforced to ensure that the vertex m will be evaluated (in mesh extraction) to point

$\mathbf{p}_m$. This property also relates to the mesh extraction which will be discussed in detail later.

Each translation $\mathbf{g}$ is an integer vector. In addition, the parametric coordinates of singularities are integers, and one component of the local frame, orthogonal to the boundary, of each boundary vertex is also integer.

4.1.3 Parameterization Methods

The energy function and four constraints indicates that this optimization requires solving a mixed-integer quadrangulation system. Since it is an NP-hard problem, different researchers have used different methods for approximating the optimal solution. We adapt two popular methods to our parameterization model using the objective function in Equation 4.5 and proper combinations of constraints listed previously.

Direct Rounding Method. (DRM) is addressed in [48] to obtain the global parameterization for tetrahedral models. This method divides the parameterization process into two phases. In phase one, it solves a linear system without enforcing any integer constraints so that it obtains the continuous solution for all variables, i.e., translation vectors and coordinates of all vertices, with singularities and boundary points identified. In phase two, variables are rounded to their nearest integers and are fixed during the optimization. By solving the system again, this method generates all the parametric coordinates for the remaining points.

Using our parameterization model, we build our direct rounding method for the MIQ problem as follows:

1) Use the objective function in Equation 4.5 for both phases. For preprocessing, partition E into two disjoint subset E_T and E_N so that E_T contains edges of a certain traversing tree on the triangular mesh and E_N contains the remaining

edges.

2) In phase one, solve Equation 4.5, that satisfies C1, where $\tilde{E} = E_T$ and $\forall \mathbf{g} = 0$, and C2 where $\tilde{E} = E_N$. $\mathbf{g}_{m,n}, (m, n) \in E_N$ can be obtained using Equation 4.3 after obtaining the solution of phase one. Round them and other integer variables.

3) In phase two, fix all integer variables and solve Equation 4.5 that satisfies C1 where $\tilde{E} = E$ to obtain the local parametric coordinates for all vertices.

In this method, the linear system is reduced by eliminating integer variables using C2, which improves the system performance. However, since it rounds and fixes all integer variables together, this method has a larger numerical error which might affect the final quad mesh quality.

Adaptive Rounding Method. (ARM) was first proposed by Bommes et al. [6] for quadrangulation. It is used again by Li et al. [40] to obtain hexahedral meshes from tetrahedral models. The major difference between this method and the previous method is that it rounds and fixes one integer variable during each solving iteration, which is called adaptive rounding. Therefore, integer variables are rounded in sequence during the solving process. Though there is no formal proof, this greedy strategy usually leads to better optimization results.

Since the initial motivation of this method is driven by quadrangulation, we can easily build our method incorporating the idea of ARM as follows.

1) The pre-processing step where E is divided into E_T and E_N is similar to DRM. Though it is not necessary, this division reduces the number of variables in the system so that performance is improved.

2) Solve Equation 4.5 that satisfies C1 where $\tilde{E} = E_T$ and $\forall \mathbf{g} = 0$, and C2 where

$\tilde{E} = E_N$. Just round the integer variables associated with boundaries and singularities using ARM while leaving translation vectors to the next step.

3) Fix all rounded variables in the previous step. Again use ARM to solve Equation 4.5 that satisfies C1 where $\tilde{E} = E$ to obtain the parametric coordinates and translation vectors.

It is easy to see that DRM has better performance than ARM. However, for arbitrary 2-dimensional manifolds, ARM has lower numerical error than DRM. Since we introduced local coordinates within charts, each vertex has several copies in the parametric space. This increases the number of variables in the system, which might make ARM quite slow for optimization. In our implementation, we use an improved rounding technique that overcomes the performance disadvantages while still maintaining a good optimization for most test cases.

We use the technique in [34] to generate the guiding field. Since each vertex is associated with a local frame on the field, each singularity in the field should result in a triangular face (a result of the discrete Poincaré-Hopf theorem such that the field circulates around its three vertices). For each singularity (a triangle), we use constraint C4 to control the geometric coordinates of its vertices. The constraint makes all three vertices converge to a single position decided by the optimization. The singularity thus converges to a single point in the quad mesh. Note that the parametric coordinates of singularities are integers.

Next, we show how to improve the performance of ARM.

4.1.4 Numerical Solvers

Since the original problem is an MIQ optimization, we need numerical solvers to approximate the optimal solution. To date, ARM has been widely used in graphics and modeling. The merit is that it generates better results than other methods by rounding one variable each time. However, we have a large number of integer variables in optimization so that we need to improve the performance further rather than simply adopt this adaptive rounding paradigm. Meanwhile, to our knowledge, there is no theoretical proof given in [6] or that can be found in other literature to guarantee the convergence of Gauss-Seidel iterations given these rounding perturbations.

Algorithm 1: Progressive Rounding Solver.

```
Give a linear system $Ax = b$, where $A$ is a sparse matrix, $x = (x_1, x_2, ..., x_m)$ is
the vector of variables and $b$ is a vector. We assume $x_i, i = 1..k$ are integer
variables.
Set a threshold $\epsilon$, an incremental factor $\beta$, and a minimum number of rounded
variables $t$.
while System still has free integer variables do
    Solve the system with a direct solver.
    Fix all integer variables of which rounding errors are less than $\epsilon$.
    if The number of fixed variables is less than $t$ then
        Increment $\epsilon$ to $\epsilon(1 + \beta)$.
```

To overcome concerns of convergence and performance, we propose an improved MIQ solver. We first replace the iterative solver in ARM with a direct solver, such as superlu [39] or suitesparse [16] etc. For each iteration, we solve the system using the direct solver, and then fix one integer variable that has the least rounding error. We repeat this iteration until all integer variables are fixed. The convergence is ensured obviously. However, this naive new solver has even lower performance than ARM, since it solves the whole linear system for each iteration. The improvement on performance is the key part for this new solver. Instead of fixing one variable, we fix variables for which the rounding error is less than a given threshold. After deploying this improvement, we noticed that in the first several iterations of the optimization, a large number of variables were fixed. In later iterations, fewer variables were fixed. We also observed that the minimum rounding error during each iteration increases with the growing number of fixed variables. Therefore, we increase the threshold if it fails to fix a certain number of variables in one iteration. We call our numerical solver progressive rounding solver (PRS), with which the large linear system is solved very efficiently. We give out the pseudo-code for PRS in Algorithm 1.

4.2 Quad Mesh Extraction

Given a parameterized surface, the quad mesh is extracted by identifying integer grids formed by intersections of isolines. As we know, isolines that form integer grids within a local chart have the form $l_u \in \mathcal{Z} \times \mathcal{R}$ and $l_v \in \mathcal{R} \times \mathcal{Z}$. This idea led to the following design of an isoline tracing paradigm, where we fix one integer component of the parametric coordinates and trace the other component in the continuous domain. During the tracing process, once we meet an integer coordinate in the tracing direction, we generate an integer node using the evaluation of geometric coordinates within a local chart, as described below.

We assume we have coordinates $\mathbf{t}$ in chart m. To be consistent with the parameterization model, we use a local quadratic optimization to obtain geometric coordinates $\mathbf{p}$ for $\mathbf{t}$ as follows.

$$\min \sum_{c \in N_m} (\tilde{\mathbf{t}}_{m,c} - \tilde{\mathbf{t}} - h \cdot f_m(\mathbf{p}_c - \mathbf{p}))^2 = 0, \tag{4.11}$$

where $\mathbf{p}_c$'s are coordinates of neighbors in chart m. Since all variables except $\mathbf{p}$ in Equation 4.11 have been finalized, the optimal value can be reached when $\mathbf{p}$ is

$$\mathbf{p} = \frac{f_m^{-1} \sum_{c \in N_m} (\tilde{\mathbf{t}}_{m,c} - \tilde{\mathbf{t}} - h \cdot f_m \mathbf{p}_c)}{h \cdot |N_m|}. \tag{4.12}$$

Equation 4.12 means, given parametric coordinates $\mathbf{t}$ within the local chart m, we use the coordinates (both parametric and geometric) of m's neighbors, to evaluate geometric coordinates induced by $\mathbf{t}$. Since f_m is a rotational matrix that is invertible, we know $\mathbf{p}$ has a unique solution. Note that Equation 4.11 has the same form as constraint C4 if we replace $\mathbf{t}$ with $\mathbf{t}_{m,m}$ and fix $\mathbf{p}$ to $\mathbf{p}_m$, which means we can fix the geometric coordinates of vertex m to $\mathbf{p}_m$ if C4 is enforced at vertex m. In addition, if $\mathbf{t}_{m,m}$ has one or more integer components, then an isoline will go through $\mathbf{p}_m$, the geometric coordinates of m, thus making vertex m be interpolated by one of the traced lines. For this reason, we call C4 the interpolation constraint. This local evaluation is the basic operation for constructing the isoline tracing paradigm and the quad mesh extraction.

Since we compute geometric coordinates within local frames, we also build the paradigm for isoline tracing within local charts. When the tracing process leaves one chart and enters an adjacent chart, we use the frame transformation in Equation 4.3 between two charts to transform current local coordinates to new local coordinates. In detail, starting from one point m and parametric coordinates $\mathbf{t}$ (which have integer component(s) for convenience) within the chart m, we add a small offset $\mathbf{d}$ onto $\mathbf{t}$ to form new coordinates $\mathbf{t}'$. We measure the distance from $\mathbf{t}'$ to $\mathbf{t}_{m,m}$ and m's neighbors $\mathbf{t}_{m,n_i}, n_i \in N_m$. If m is the nearest point, we continue to add the small offset $\mathbf{d}$. Otherwise, we pick the nearest neighbor n as the new center and use the frame transformation to transform coordinates and the offset in chart m to those in chart n. In this way, the tracing is continued from point n. The form of the offset $\mathbf{d}$, which can be either $(0, \pm\delta)$ or $(\pm\delta, 0)$, decides the tracing direction. Note that $1/\delta$ should be an integer, which allows us to generate integer nodes, as we will see shortly.

During the tracing process, the transformation from m to neighbor n must be reliable, i.e., it won't return to m right after the transformation. Mathematically, the condition in Equation 4.13 advances the isoline tracing from m to n,

$$\|\mathbf{t} - \mathbf{t}_{m,m}\| > \|\mathbf{t} - \mathbf{t}_{m,n}\|. \tag{4.13}$$

After transforming into local chart n, we want the distance from $\mathbf{t}$ to $\mathbf{t}_{n,m}$ to still be larger than that from $\mathbf{t}$ to $\mathbf{t}_{n,n}$, i.e., we have

$$\|\Pi_{m,n}\mathbf{t} + g_{m,n} - \mathbf{t}_{n,m}\| > \|\Pi_{m,n}\mathbf{t} + g_{m,n} - \mathbf{t}_{n,n}\|. \tag{4.14}$$

Since the parameterization satisfies constraint C3, we obtain Equation 4.15 by plugging $\mathbf{t}_{n,n}$ and $\mathbf{t}_{n,m}$ from Equation 4.8 and Equation 4.9 into Equation 4.14.

$$\|\Pi_{m,n}(\mathbf{t} - \mathbf{t}_{m,m})\| > \|\Pi_{m,n}(\mathbf{t} - \mathbf{t}_{m,n})\|. \tag{4.15}$$

Matrix $\Pi_{m,n}$, which belongs to $SO(2)$, does not change the magnitude of a vector. Thus, Equation 4.13 and Equation 4.15 are equivalent. C3 is named the continuity constraint because it ensures the continuity of the distance measurement under the transformation operation.

In addition, we want n to be the best candidate among all neighbors of m if it is nearest to current coordinates $\mathbf{t}$. We assume n is the only neighbor whose distance from $\mathbf{t}$ is less than that of m, i.e.,

$$\|\mathbf{t} - \mathbf{t}_{m,l}\| > \|\mathbf{t} - \mathbf{t}_{m,m}\| > \|\mathbf{t} - \mathbf{t}_{m,n}\|, \tag{4.16}$$

where $l \neq n$. We want to have

$$\|\Pi_{m,l}\mathbf{t} + \mathbf{g}_{m,l} - \mathbf{t}_{l,l}\| > \|\Pi_{m,n}\mathbf{t} + \mathbf{g}_{m,n} - \mathbf{t}_{n,m}\| > \\ \|\Pi_{m,n}\mathbf{t} + \mathbf{g}_{m,n} - \mathbf{t}_{n,n}\|. \tag{4.17}$$

Similarly, we can obtain Equation 4.16 by plugging in the terms from C3 into Equation 4.17 and removing Π's, which concludes Equation 4.16 and Equation 4.17 are equivalent.

With the technique that allows the tracing between adjacent charts, we can build an isoline tracing paradigm, called **one step tracing**, that traces from one integer node to one of its neighbor integer nodes : starting from an initial point m with parametric coordinates $\mathbf{t}_{m,m}$, we add an offset $\mathbf{d}$ onto $\mathbf{t}_{m,m}$ for $1/||\mathbf{d}||$ times. During this process, if we need to switch to a new chart we transform the current parametric coordinates and the offset using Equation 4.3 to those in the new chart. Note that the offset doesn't need the translation vector for transformation. After advancing $1/||\mathbf{d}||$ times, we evaluate the geometric coordinates of a new node in the current local frame using Equation 4.12. This **one step tracing** allows us to build a general isoline tracing paradigm easily: we repeat this step until we reach the surface boundary or the initial point. Since one axis of the texture coordinates of each boundary vertex, which is orthogonal to the boundary, is an integer coordinate, we generate integer nodes right at the boundary during the isoline tracing. Next, we build the quad mesh extraction with the help of one step tracing paradigm.

Since the form of $\mathbf{d}$ decides the tracing direction, for a given node i and its geometric coordinates $\mathbf{p}_i$, we can trace in four directions using offsets $(\pm\delta, 0)$ and $(0, \pm\delta)$ for one step. Once we obtain a new node j and its geometric coordinates $\mathbf{p}_j$, we connect i and j to form an edge in the quad mesh. Then we make j the

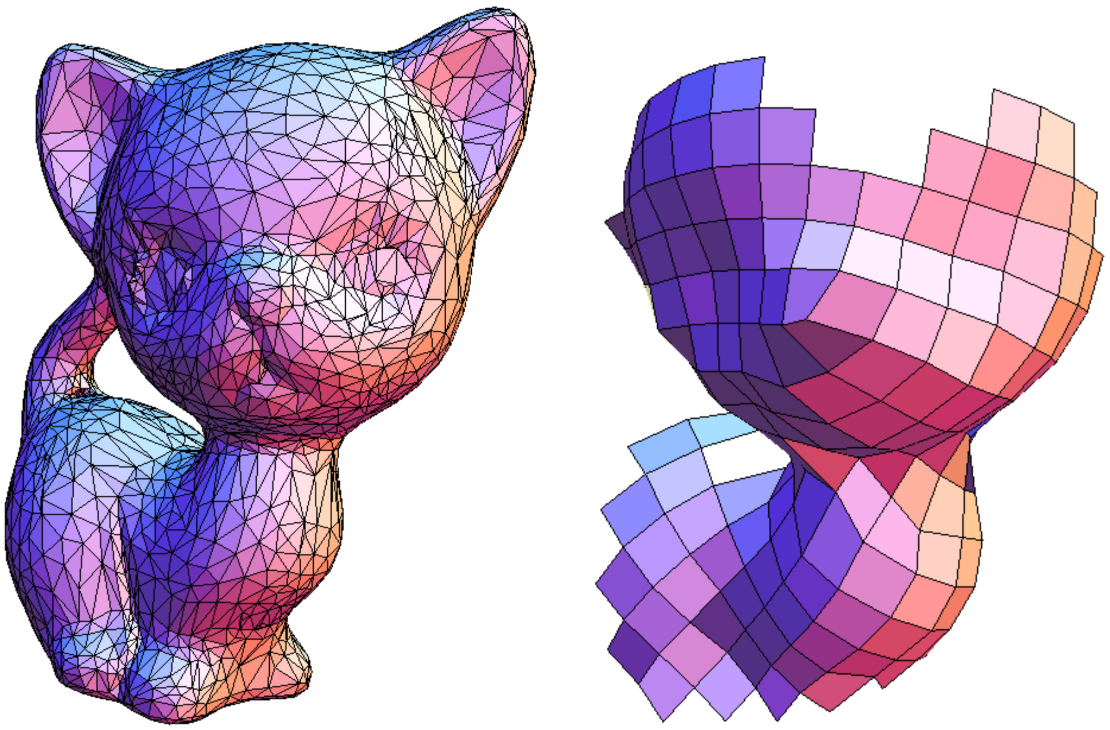

(a) Original triangular model (b) Intermediate state of the mesh extraction

Figure 4.3: Quad generation on manifolds.

new center and continue to trace in four directions. We maintain new nodes with a queue data structure, which allows us to build the quad extracting paradigm using a breadth-first traversal. Note that after advancing one step, we may encounter an existing ("old") node. We also connect that old node and current node since this forms an edge in the quad mesh. The output of this traversal is a general graph structure $G = (V', E')$, where V' is the set of integer nodes and E' is the set of edges connecting nodes in V'. This graph structure can be easily converted to a vertex-face representation by figuring out all cycles of size four in G. We show this traversal technique in pseudo-code in Algorithm 2. Figure 4.3 shows the mesh extraction on the cat model. Our method works well on this model having many sliver triangles.

Algorithm 2: Mesh extraction.

```
Given a point $m$ and its geometric coordinates $\mathbf{p}_m$.
Create a general graph $G$ and create the node for $m$ in $G$. Store point $m$ in queue $Q$.
while $Q$ is not empty do
    Extract the head of $Q$ and call it $n$.
    Using point $n$ as the center, advance in four directions (using one step tracing).
    For each of the four positions, check whether any node exists there and if not, create a new node in $G$ for the new positions.
    Build edges connecting $n$ to four nodes (including new and old) in $G$.
Traverse $G$ to obtain all cycles of size four.
```

Note that Algorithm 2 requires us to decide whether a node is old or new. We achieve this by using a spatial searching structure. Once we reach a node j, we first evaluate its geometric coordinates. Then we do spatial searching to see if there is an existing node i that is in a given neighborhood of node j. If so, this is an old node and $\mathbf{p}_i = \mathbf{p}_j$. Otherwise, we create a new node for j in the graph. The threshold of this neighborhood is set to be much smaller than the length of quad mesh edges. The algorithm terminates because it only generates nodes of integer coordinates, which is a finite set, and because it stops propagating when meeting existing nodes.

As we know, the cross field at singularities is not well defined. Therefore, when we traverse to a singularity, we don't want it to become a center for new tracing. Note that, the neighbor nodes of a singularity will all advance to it when they are taken as centers for tracing, which forms edges connecting neighbor nodes and the singularity. This allows us to generate the correct topology at singularities.

Our method is also adaptive to the feature-aligned field. Those fields can be obtained by smoothing the principal curvature field [48] or by constraining the model features during the field design [34]. We provide result comparisons in the experimental part.

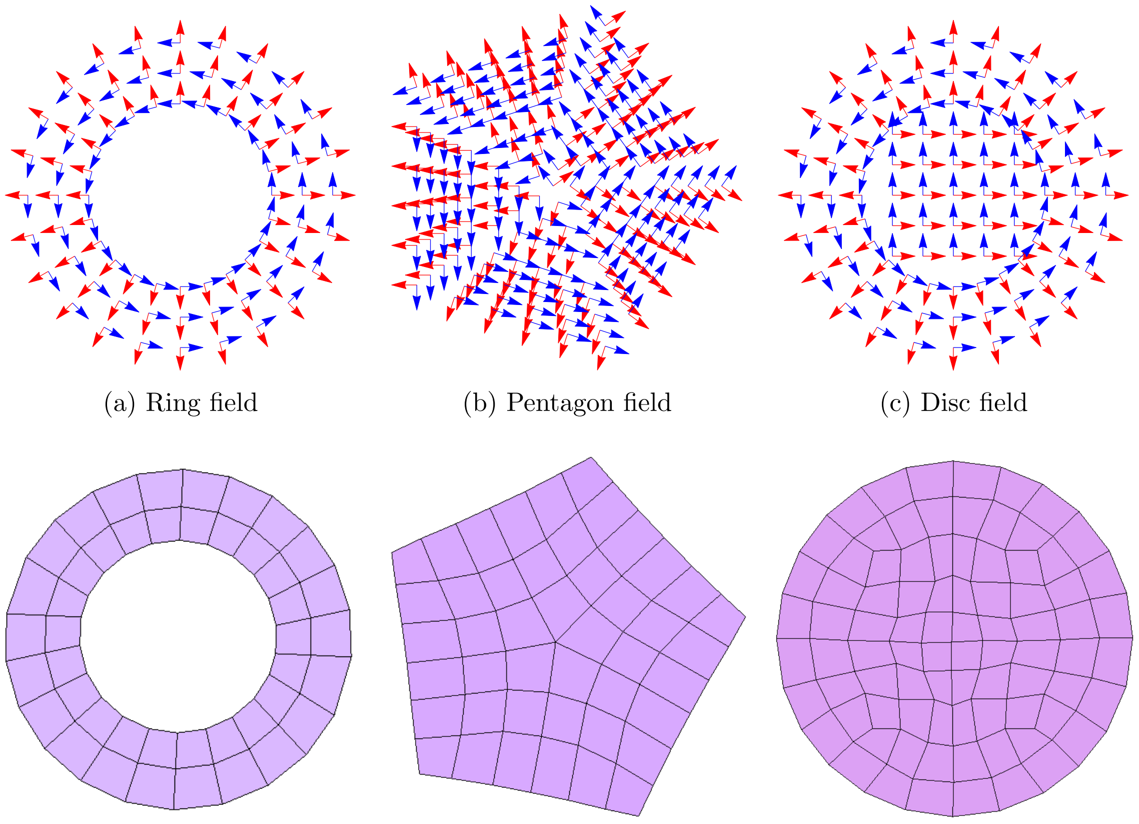
4.3 Experimental Results

We show the quad mesh result using our parameterization model and the performance comparison on the MIQ solvers. The hardware configuration is a desktop computer with one Core i7 3.4G CPU and 16GB RAM. We implemented our work using g++ 4.8 on a 64bit Linux system.

Model	**Sys. size**	**#Int.**	**ARM**	**PRM**	**Extraction**
Sphere	$30k \times 13k$	$5.7k$	2.43s	2.31s	1.02s
Torus	$70k \times 28k$	$12k$	22.94s	7.57s	1.57s
Cube	$73k \times 30k$	$13k$	30.17s	8.14s	1.55s
Cat	$96k \times 40k$	$19k$	52.77s	23.93s	1.74s
Bunny	$169k \times 706k$	$31k$	2m03s	52.27s	2.32s
Hand	$365k \times 152k$	$68k$	5m57s	2m20s	2.17s
Rockerarm	$480k \times 200k$	$90k$	9m52s	3m58s	2.08s

Table 4.1: Performance comparison.

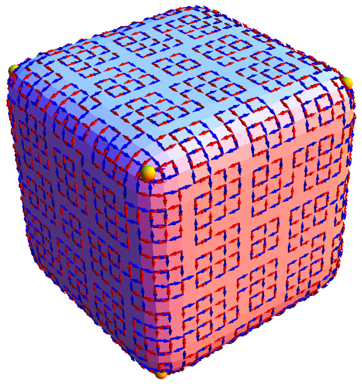
For constructing our MIQ solver, we wrap the Eigen library for linear algebra computations. The internal linear system solver is superlu. We show the comparison of performance between PRM and ARM in Table 4.1. The second column shows the size of the sparse linear system. The third column shows the number of integer variables in the system. Column ARM shows the time for parameterization using ARM. Column PRM shows the time using PRM. Column extraction shows the time for generating quad meshes. According to our experiments, PRM has consistently better performance than ARM by a factor of around 2.2 to 2.9. The extraction time doesn't change too much for different models, because it is mainly related to the number of integer nodes in meshes. Since we introduced many more integer variables than the method in [6] in our optimization, our method needs more time for the whole quadrangulation.

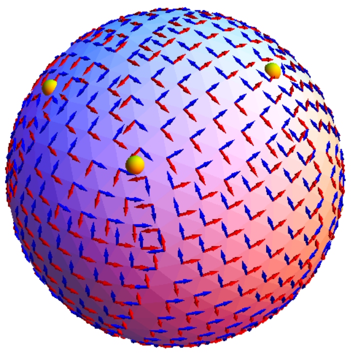

(a) Ring field (b) Pentagon field (c) Disc field

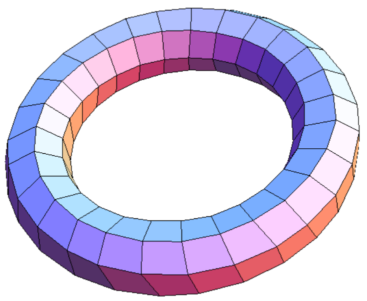
(d) Ring quad (e) Pentagon quad (f) Disc quad

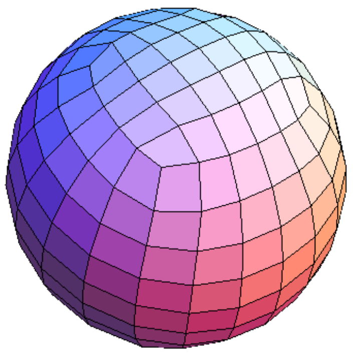
Figure 4.4: Quad meshes on 2D models.

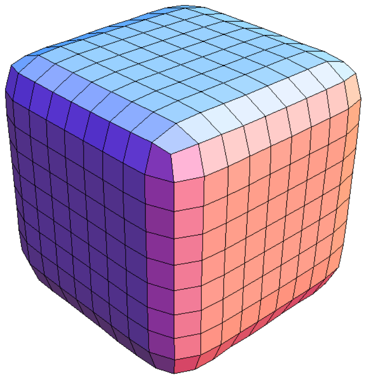
Next we show results on some simple 2D geometric shapes in Figure 4.4. The guide field in these instances is designed manually to reflect a user's intention. For those models, the quad meshes follow the guiding field very well. For the pentagon model, there is a singularity at the model center. Our method identifies it correctly. The generated mesh has the correct valence for this singularity. The disc model has four singularities located where the inside square field meets with the outside round field. Our method identifies those singularities and generates correct topological structures as well.


(a) Cube field


(b) Sphere field


(c) Sparse grid


(d) Sphere quad


(e) Cube quad

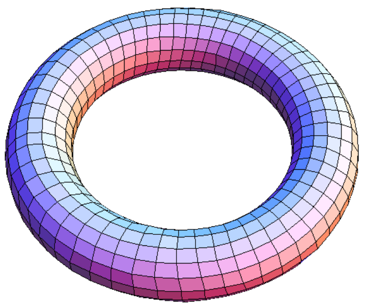
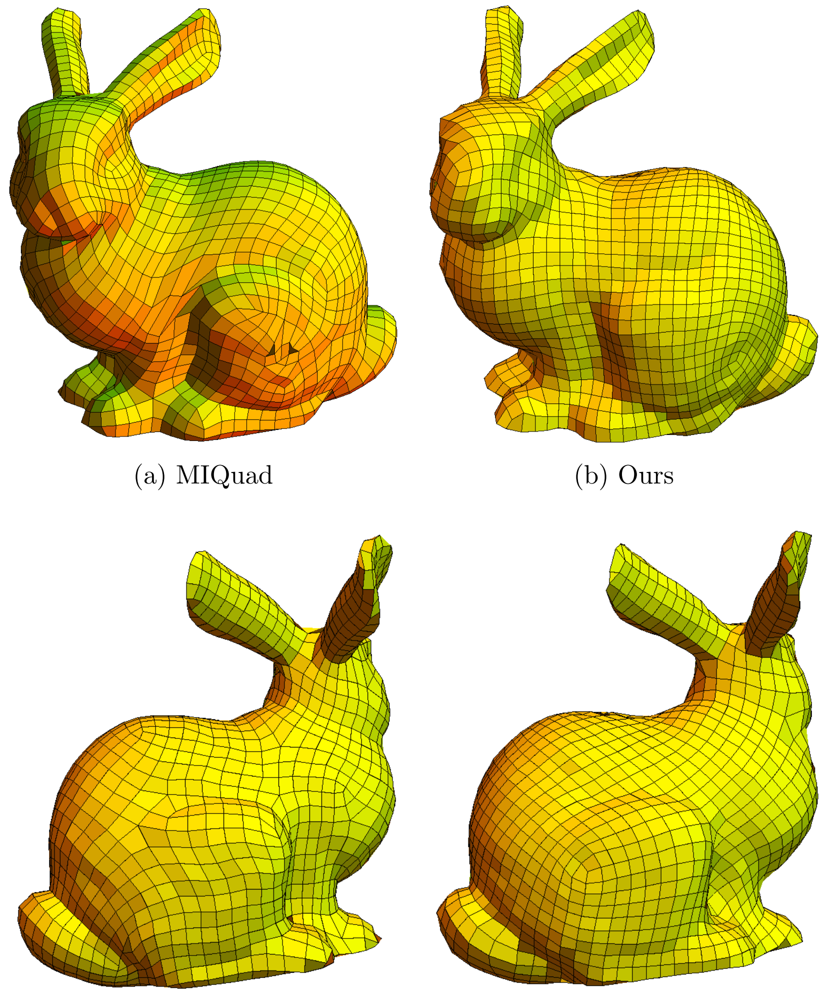

(f) Dense grid

Figure 4.5: Quad meshes on manifolds.

For 2-dimensional manifolds in Figure 4.5, we train the cross field on models using the technique in [34]. The cube and sphere models both have 8 singular points, which are successfully identified by our method. The topological structures generated are also correct. There are no singular points on the torus model. We generated two quad meshes with different edge sizes, which show that our method can control the size of quads.

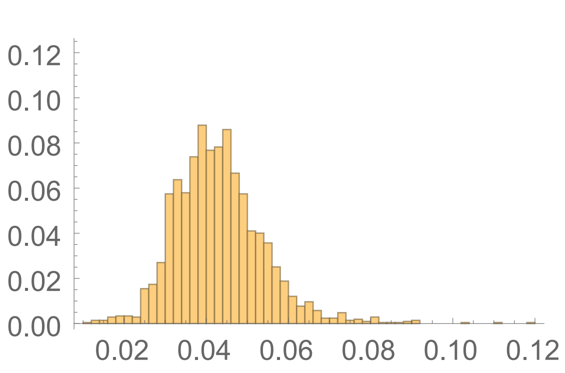


(a) MIQuad (b) Ours

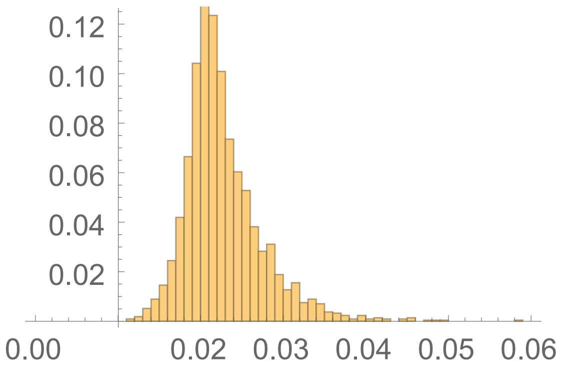
(c) MIQuad (d) Ours

Figure 4.6: Quadrangulation on bunny.

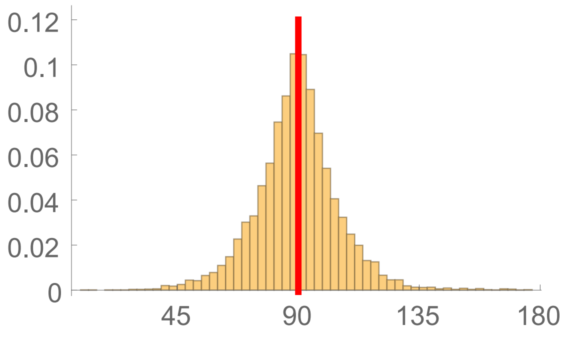
Figure 4.6 shows results from two methods. Though the mesh quality is largely decided by the guiding field, our method, which generates the quad mesh over the vertex field, produces comparable results to those of existing methods built on face fields. We compare the mesh quality between our method and that generated from [6] in terms of the distribution of areas and the distribution of angles. We show one example, but the results were consistent across other examples, and the distributions are very similar for a wide range of quad sizes. For clarity, we call the method of [6] MIQuad.



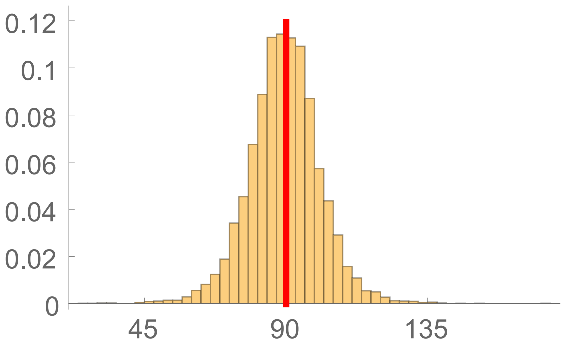
(a) MIQuad area distribution



(b) Our area distribution



(c) MIQuad angle distribution



(d) Our angle distribution

Figure 4.7: Distributions on areas and angles of quads.

Figure 4.7 shows comparisons of distributions on areas and angles of quads generated from the bunny model. Figure 4.7a and 4.7b shows the comparison on the distribution of areas. Figure 4.7c and 4.7d shows the comparison on the distribution of angles within the quads. Our result is comparable to that of MIQuad, which can also be easily noted by comparing the two meshes visually in Figure 4.6. This indicates that our method generates quad meshes with in good quality over vertex fields. In the next part, we discuss contributions of our method. Since our method works on vertex fields, we also discuss the limitations implied by field settings.

4.4 Discussion on Global Parameterization

We present a novel method for quadrangulation that utilizes local charts defined on triangular mesh models. With a given cross field, our method formulates the computation of parametric coordinates as a mixed-integer optimization problem. Our method also adds flexibility to the system by redundant integer variables. The quality of the generated quadrangular mesh depends only on the optimization in the numeric solver. Meanwhile, we improve the performance of the adaptive rounding solver, which is a widely used numerical method for solving MIQ, so that our method generates parametric coordinates more efficiently. To generate the quadrangular meshes, we propose an isoline traversal method and a breadth-first traversal node generating method so that the quadrangulation result is extracted efficiently. To summarize the characteristics of our work, we note the following:

- We build the quadrangulation method over smoothed vertex fields. So far, field based quadrangulation methods are all built over face fields. We solve difficulties in generating quads over vertex fields. One is that singularities are triangle faces rather than points. We design constraints to allow consistent convergence of isolines around each singularity. The other is local frames of vertices are usually undetermined rather than those of faces that are determined. We define a local frame for each vertex and generate parameterization coordinates with respect to the local frame, which reduces effects from other neighbor vertices as much as possible.
- Since our parametric coordinates are defined within local charts, we avoid identifying consistent global base vectors for vertices. The redundant integer vari-

ables increase the flexibility of the optimization system. Therefore, we need no post processing on the cross field before computing parameterization coordinates.

- Since each vertex has more than one copy of parametric coordinates, the number of variables in the optimization system is relatively large. We improve the numerical solver for optimizing the MIQ problem so that the system can be solved efficiently.
- A method for generating quad meshes is also proposed so that our mesh results have the face information rather than simple isolines. This output can be used directly for further processing such as texture mapping or finite element analysis on surfaces.

There are some limitations of our method which provide potential for future research. First, the quad mesh quality is largely decided by the guiding field. Though we proposed the method over vertex fields, it's not easy to compare the mesh quality generated by our method with that generated by existing methods built on face fields. The conversion from the vertex field to the face field or vise versa requires the field smoothing operation, which is likely to change characteristics of the source field, including number of singularities, positions of singularities and feature alignments, etc. In our future work, we aim to find more applications that can show advantages of our vertex field method. Second, we introduce multiple copies of each vertex for parameterization, which increases the number of variables in optimization. We could possibly find a method that would require fewer variables while still recording all parametric information. Overcoming this problem can be a direct extension of our work in the future. Last, we also assume surfaces are smooth everywhere except at boundaries, which allows us to associate each vertex with one local chart. However, vertices on sharp features may require multiple local charts, which will be a focus of attention in our future work.

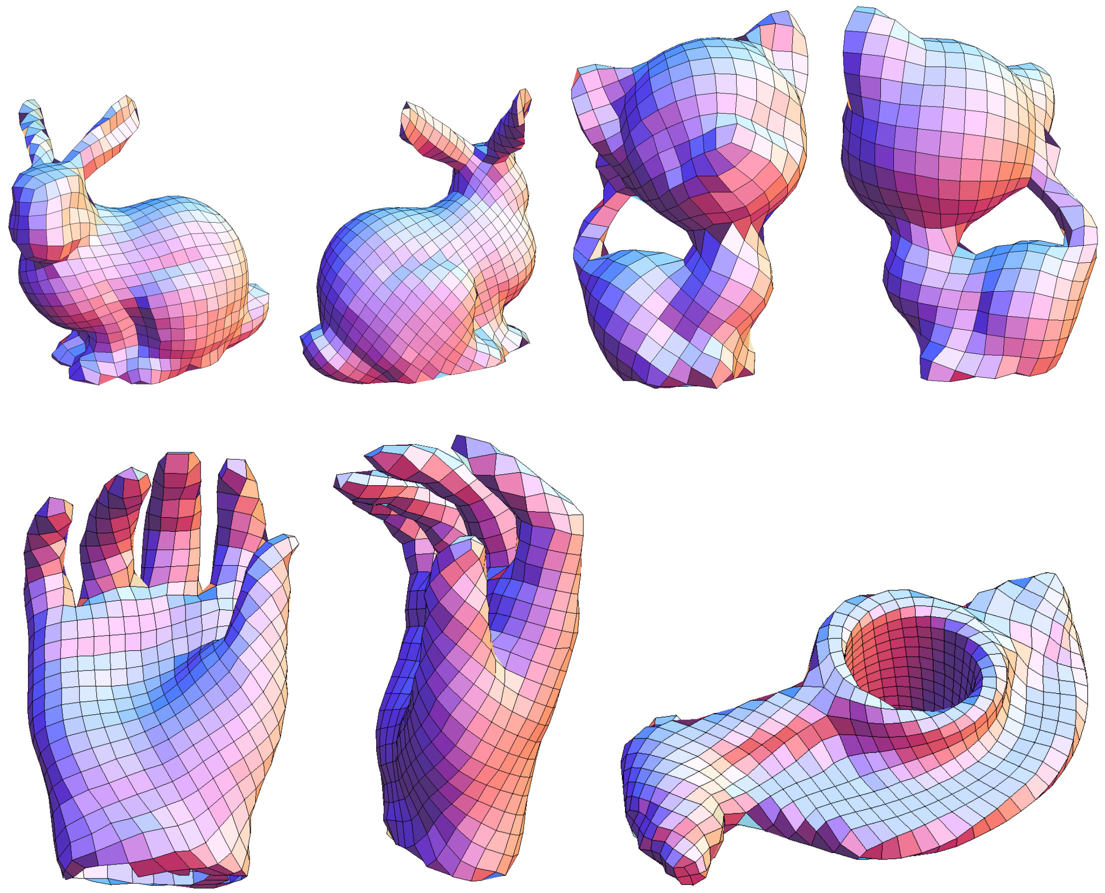


Figure 4.8: Quad meshes on arbitrary manifolds guided by global optimal fields.

At the end of this work, we present quad results obtained by our method on arbitrary manifolds. Results guided by global optimal fields are shown in Figure 4.8. Results guided by feature-aligned fields are shown in Figure 4.9.

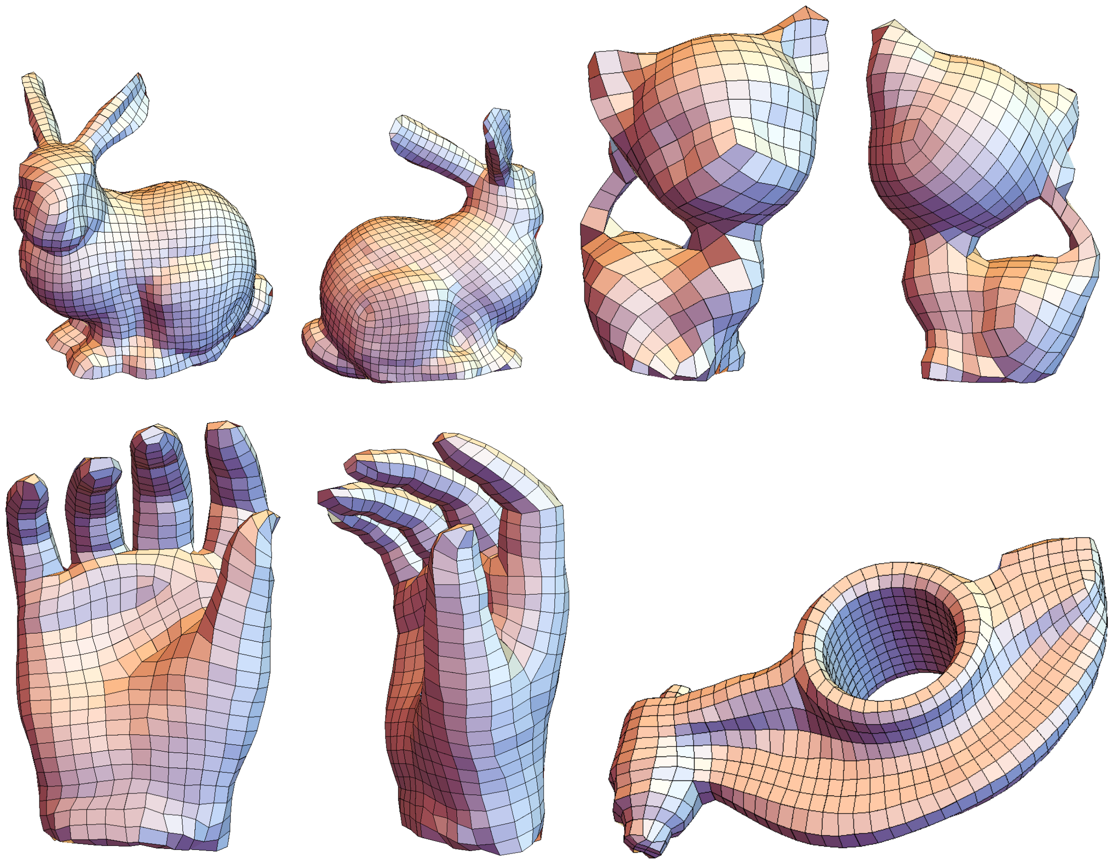


Figure 4.9: Quad meshes on arbitrary manifolds guided by feature-aligned fields.

5. GEOMETRIC OPTIMIZATION FOR PROCESS PLANNING*

Geometric optimization deals with problems of computing geometric objects that are optimal subject to certain criteria and constraints. Different from numeric optimization, geometric optimization models the optimization problem using geometric properties and solves the problem by exploiting topological properties of geometric elements such that the original problem can be decomposed into numerical problems.

In this work, we present a new geometric model for tolerance modeling and propagation, geared toward tolerance analysis in process planning. Our aim is to decompose a big chunk of analytic computations of conventional tolerance analysis into a series of geometry computations. Our method has three steps. We first decompose the part into basic geometric primitives, or features. Because those primitives are to be manufactured in a common part, we know they are related and that there are dependencies between them, which could be represented by a graph structure. We first decouple the primitives into several co-related primitive groups. In each group, the geometric position of a certain primitive (*target primitive*) is decided by the remaining primitives (*reference primitives*). This allows us to use the LPGUM model [29] to model the tolerance zone for the target primitive, because all its variations have been obtained. Second, we formulate a method for cascading the decoupled primitive groups, so that the tolerances can be transferred between groups. Using those cascading techniques, we could obtain the tolerance zones for all primitives by traversing the embedded graph structure representing the dependencies. The tolerance zones thus obtained provide a worst case estimation on dimensions, represented

*Part of this section is reprinted with permission from Geometric computation and optimization on tolerance dimensioning by Songgang Xu and John Keyser. Journal of Computer-Aided Design, vol 46, pages 129-137, 2014. Copyright 2014 by Elsevier. DOI 10.1016/j.cad.2013.08.025.

as a geometric polytope. Last, we provide a computational optimization method which can improve the quality of the existing process plan so that the tolerance of the parts could be minimized as much as possible. The optimization problem itself is an NP hard problem. We propose an efficient approximated dynamic programming solver which utilizes the optimal substructure.

5.1 Relative Tolerance

When computing relative tolerances, we assume the references $\{\mathbf{m}_i\}$ have an exact position. Thus, the tolerance zone of the target $\mathbf{t}$ is only caused by variations of dimensions connecting $\mathbf{t}$ and $\{\mathbf{m}_i\}$. When $i = 1$, we name the dependency $\{\mathbf{m}_1\} \rightarrow \mathbf{t}$ *single dependency*, which indicates that the target $\mathbf{t}$ is only affected by one primitive. If $i > 1$, we name the dependency $\{\mathbf{m}_i\} \rightarrow \mathbf{t}$ *multiple dependency*, which implies that the target $\mathbf{t}$ is affected by two or more primitives. We present tolerance evaluation techniques for both cases.

5.1.1 Single Dependency

We assume a point and line segment are sufficient for representing all features. This is always true within some tolerance (which we in fact have), and given that we linearize our nonlinear constraints, this is even more appropriate. The relative tolerance zone $\mathcal{R}(\mathbf{t}, \{\mathbf{m}_i\})$ could be computed analytically via an explicit formulation on a certain parameterization or approximated by the linearization of variances [29]. We adopt the second approach since it allows us to convert the analytical computation to the corresponding geometric computation. Though the linearization is a first-order approximation of the exact tolerance zone, this minor accuracy trade-off can be neglected because of the tiny values of tolerances. We compute tolerance zones for combinations of the point, denoted by $\mathbf{p}$, and the segment, denoted by its two ends $\mathbf{ab}$. We use $\mathbf{p}'$ to denote another point and use $\mathbf{a}'\mathbf{b}'$ to denote another segment.

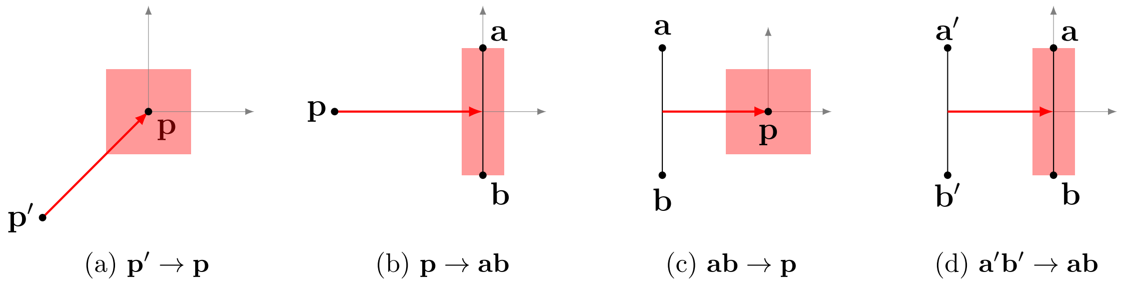

p
p'
p
a
b
a
b
p
a'
a
b'
b
(a) $\mathbf{p}' \rightarrow \mathbf{p}$
(b) $\mathbf{p} \rightarrow \mathbf{ab}$
(c) $\mathbf{ab} \rightarrow \mathbf{p}$
(d) $\mathbf{a}'\mathbf{b}' \rightarrow \mathbf{ab}$

Figure 5.1: Single dependency tolerance zones.

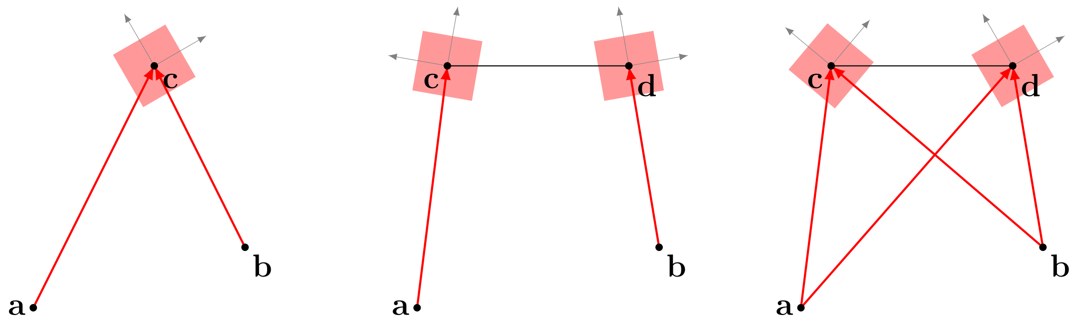
The linearization model in [29] is built on the variational zone of a point. For one fixed point $\mathbf{p}$, the tolerance zone is approximated by

$$\mathcal{Z}(\mathbf{p}) \approx \mathcal{Z}(\bar{\mathbf{p}}) + J\delta_i, \tag{5.1}$$

where J is the (Jacobian) sensitivity matrix of uncertainties and where $\delta_i, 1 \leq i \leq n$ define variations of n tolerances. When considering $\mathcal{R}(\mathbf{p}, \{\mathbf{p}'\})$, $\mathbf{p}'$ tries to lock all degrees of freedom (DOF) of $\mathbf{p}$. Figure 5.1 shows single dependency tolerance zones. We have $n = 2$ for 2D and the zone is a quadrangle as shown in Figure 5.1a. Similarly, we can obtain tolerance zones for the remaining combinations. For $\mathcal{R}(\mathbf{p}, \{\mathbf{ab}\})$, the tolerance zone would be a band along the segment as shown in Figure 5.1b. For $\mathcal{R}(\mathbf{ab}, \{\mathbf{p}\})$, the local tolerance frame still has two DOFs, so the tolerance zone, in Figure 5.1c, is similar to that of $\mathcal{R}(\mathbf{p}, \{\mathbf{p}'\})$. For $\mathcal{R}(\mathbf{ab}, \{\mathbf{a}'\mathbf{b}'\})$, we can apply several kinds of geometric constraints, such as parallelism or perpendicularity. Figure 5.1d shows the tolerance zone for parallelism.

5.1.2 Multiple Dependency

When the references set $\{\mathbf{m}_i\}$ has more than one primitive, we have a multiple dependency. We examine three cases where the target is a point or a segment as shown in Figure 5.2.



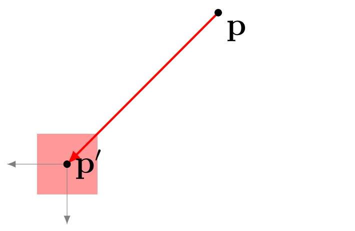
(a) Pt. target (b) Seg. independent (c) Seg. interleaved

Figure 5.2: Multiple dependencies tolerance zones.

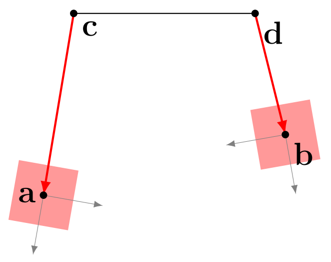
Figure 5.2 shows multiple dependencies tolerance zones. Figure 5.2a shows the point target case. Points $\mathbf{a}$ and $\mathbf{b}$ could be two point primitives or points on other primitives. DOFs of $\mathbf{c}$ are locked by both $\mathbf{a}$ and $\mathbf{b}$. We can obtain $\mathcal{R}(\mathbf{c})$ by applying the linearization in (5.1) using the variations of dimensions between $\mathbf{a}, \mathbf{c}$ and $\mathbf{b}, \mathbf{c}$. The number of vertices of $\mathcal{R}(\mathbf{c})$ is decided by the corresponding sensitivity matrix. In Figure 5.2b, points $\mathbf{a}$ and $\mathbf{b}$ are also two point primitives or points on other primitives. DOFs of $\mathbf{c}$ are locked by $\mathbf{a}$ and DOFs of $\mathbf{d}$ are locked by $\mathbf{b}$. We can decompose this case into two independent groups of point dependencies: $\{\mathbf{a}\} \to \mathbf{c}$ and $\{\mathbf{b}\} \to \mathbf{d}$. The tolerance zone for segment $\mathbf{cd}$ is the convex hull of $\mathcal{R}(\mathbf{c})$ and $\mathcal{R}(\mathbf{d})$. In Figure 5.2c, points $\mathbf{c}$ and $\mathbf{d}$ are both decided by $\mathbf{a}$ and $\mathbf{b}$ together. We decompose this relation into two interleaved dependencies $\{\mathbf{ab}\} \to \mathbf{c}$ and $\{\mathbf{ab}\} \to \mathbf{d}$ and treat $\mathbf{c}$ and $\mathbf{d}$ as point targets.

5.1.3 Dependency Inverse

For single dependency, the relative tolerance operation is not commutative, i.e., $\mathcal{R}(\mathbf{t}, \{\mathbf{m}\}) \neq \mathcal{R}(\mathbf{m}, \{\mathbf{t}\})$. Therefore, we define the inverse operation $\mathcal{R}^{-1}(\mathbf{t}, \{\mathbf{m}\}) = \mathcal{R}(\mathbf{m}, \{\mathbf{t}\})$. The computation rule is that we view $\mathbf{t}$ as the reference and build the tolerance zone around $\mathbf{m}$. We could easily use the models in Figure 5.1 to obtain the inverse dependency for each case of the single dependency. For example, $\mathcal{R}^{-1}(\mathbf{p}, \{\mathbf{ab}\}) = \mathcal{R}(\mathbf{ab}, \{\mathbf{p}\})$, which indicates that operations in Figure 5.1b and Figure 5.1c are inverses of each other. Figure 5.3 shows dependency inversions. The inverse for Figure 5.1a is in Figure 5.3a. We can obtain the inverse operation for the two-segment case similarly.



(a) Single dependency inverse



(b) Multi. dependency inverse

Figure 5.3: Dependency inversions.

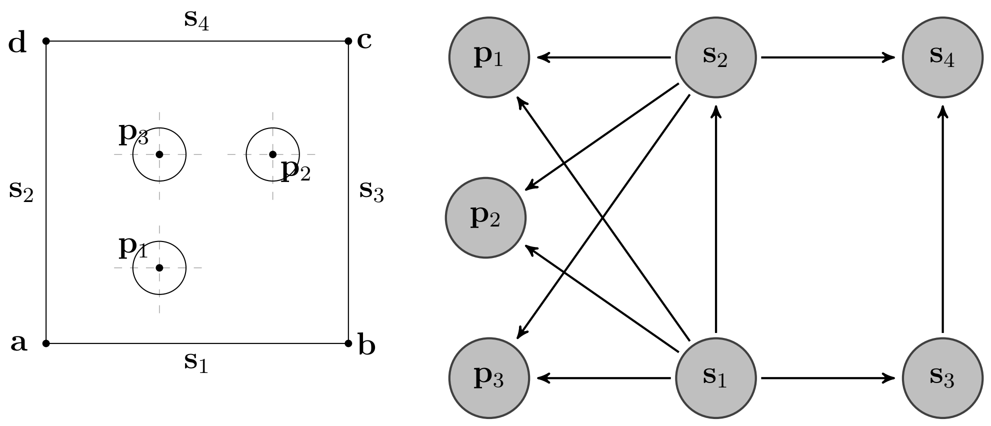
If a multiple dependency can be decomposed into a set of independent single dependencies, we define the inverse $\mathcal{R}^{-1}(\mathbf{t}, \{\mathbf{m}_i\}) = \cup_i \mathcal{R}(\mathbf{m}_i, \{\mathbf{t}\})$. The case in Figure 5.2b can be inverted by inverting $\mathcal{R}(\mathbf{a}, \{\mathbf{c}\})$ and $\mathcal{R}(\mathbf{d}, \{\mathbf{b}\})$ respectively and computing their union as in Figure 5.3b. The case in Figure 5.2a usually cannot be inverted: because **c**'s DOFs are locked by two different objects, we cannot invert the dependency, just as we cannot invert a multivariate function. The same applies to the case of Figure 5.2c.

Notice that inverting dependencies changes the tolerance zones even though the dimensioning scheme itself is unchanged. This noncommutative nature of dependencies and tolerances provides the potential that we could improve an existing dimensioning scheme to a more optimal one by selectively inverting dependencies. Related techniques will be presented in section 5.3 in detail.

5.2 Tolerance Cascading

While relative dependencies assumed that references had exact positions, in reality the references themselves may have their own tolerance zones. These variations in the references might need to transfer to any target they lock. Given an existing dimensioning scheme, we can define a directed acyclic graph (DAG) of dimension dependencies based on the single and multiple dependencies on the individual primitives. The graph nodes are primitives. Each edge of the graph starts from the references and points to the target. Note that there might be many possible dependency graphs for the same physical part.

To begin, we use a dependency graph that captures the intention of tolerance design and that can be easily interpreted during manufacturing. Usually this will have been provided by the designer. But if no design is provided, we can use some simple heuristics to order the dependencies into a graph. In section 5.3, the dependency graph will be augmented so that it covers a larger portion of the space of dependency graphs and an optimal dependency graph will be chosen from that subspace. We show an example for building a dependency graph in Figure 5.4. Fig-



(a) A simple part

$$\begin{tikzcd}
\mathbf{p}_1 & \mathbf{s}_2 \arrow[l] \arrow[r] \arrow[dl] \arrow[ddl] & \mathbf{s}_4 \\
\mathbf{p}_2 & & \\
\mathbf{p}_3 & \mathbf{s}_1 \arrow[l] \arrow[r] \arrow[uu] \arrow[ul] \arrow[uul] & \mathbf{s}_3 \arrow[uu]
\end{tikzcd}$$

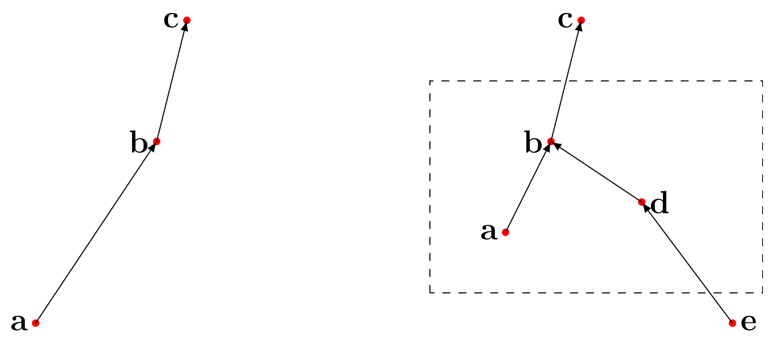
(b) Dependency graph

Figure 5.4: Dependency graph for a part. Notice that $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ are all defined relative to $\mathbf{s}_1$ and $\mathbf{s}_2$, that $\mathbf{s}_2$ and $\mathbf{s}_3$ are defined as being perpendicular to $\mathbf{s}_1$, and that $\mathbf{s}_4$ is defined relative to both $\mathbf{s}_2$ and $\mathbf{s}_3$.

ure 5.4a is a simple rectangular part with three position constraints. One possible dependency graph is that in Figure 5.4b. The graph includes single dependency, such as $\{\mathbf{s}_1\} \rightarrow \mathbf{s}_2$ and $\{\mathbf{s}_1\} \rightarrow \mathbf{s}_3$, and multiple dependencies such as $\{\mathbf{s}_1, \mathbf{s}_2\} \rightarrow \mathbf{p}_1$ and $\{\mathbf{s}_2, \mathbf{s}_3\} \rightarrow \mathbf{s}_4$. This graph is acyclic since it has no back edge. There are tolerance chains such as $\mathbf{s}_1 \rightarrow \mathbf{s}_3 \rightarrow \mathbf{s}_4$, which means the tolerance zone of $\mathbf{s}_4$ also partially depends on $\mathbf{s}_1$ through $\mathbf{s}_3$. We classify dependency chains into one of two types as shown in Figure 5.5. If the entire chain consists of single dependencies (e.g. the chain $\mathbf{a} \rightarrow \mathbf{b} \rightarrow \mathbf{c}$ in Figure 5.5a), we call it *single dependency cascading*, while if there are multiple dependencies in the chain (e.g. the chain in $\mathbf{d} \rightarrow \mathbf{b} \rightarrow \mathbf{c}$ Figure 5.5b) we call it *hybrid dependency cascading.*

5.2.1 Single Dependency Cascading

In single dependency cascading, each primitive is only decided by only one reference along the dimension chain. The basic chain pattern is $\mathbf{m}_1 \rightarrow \mathbf{m}_2 \rightarrow \mathbf{m}_3$, where $\mathbf{m}_i$ are primitives. The key problem is how to transfer the tolerance of $\mathbf{m}_1$ to $\mathbf{m}_3$



(a) Single dependency cascading (b) Hybrid dependency cascading

Figure 5.5: Two types of tolerance zone cascading.

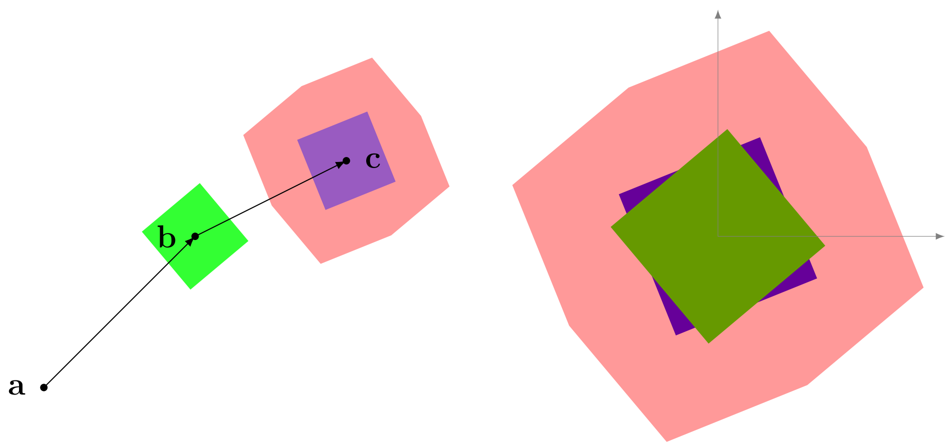
through $\mathbf{m}_2$.

Transferring tolerances through a point, as in Figure 5.6, is straightforward. For each level, $\mathcal{R}(\mathbf{b})$ and $\mathcal{R}(\mathbf{c})$ can be evaluated. Assume the chain root $\mathbf{a}$ is exact (i.e. $\mathcal{Z}(\mathbf{a}) = \mathbf{a}$ and $\mathcal{Z}(\mathbf{b}) = \mathcal{R}(\mathbf{b})$). Since the variations of dependencies $\{\mathbf{a}\} \to \mathbf{b}$ and $\{\mathbf{b}\} \to \mathbf{c}$ are independent, $\mathcal{Z}(\mathbf{c})$ is the cascading of $\{\mathbf{a}\} \to \mathbf{b}$ and $\{\mathbf{b}\} \to \mathbf{c}$, which is equal to the Minkowski sum of $\mathcal{R}(\mathbf{b})$ and $\mathcal{R}(\mathbf{c})$.

Cascading through a segment is based on cascading through points:

Definition 1. *The tolerance zone of the segment is the convex hull of the tolerance zones of its two endpoints.*

Assume we have a point $\mathbf{c}$ and segment $\mathbf{ab}$. We map $\mathbf{c}$ to $\mathbf{ab}$ to get a new point $\mathbf{d}$ on the segment. We compute $\mathcal{Z}(\mathbf{d})$. Then $\mathcal{Z}(\mathbf{c})$ is obtained by the Minkowski sum of $\mathcal{Z}(\mathbf{d})$ and $\mathcal{R}(\mathbf{c}, \{\mathbf{d}\})$.



(a) Point cascading (b) Minkowski sum

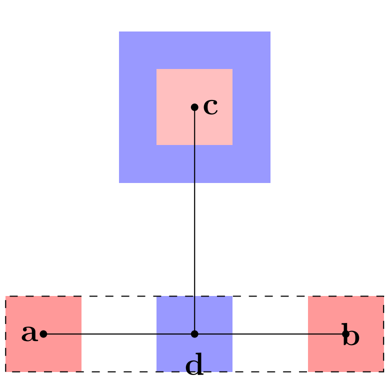
Figure 5.6: Point-point cascading.

Following this idea, as shown in Figure 5.7, we can describe perpendicularity or angularity constraints, as shown in Figure 5.7a and Figure 5.7b. Note that $\mathcal{Z}(\mathbf{d})$ can be obtained by interpolating $\mathcal{Z}(\mathbf{a})$ and $\mathcal{Z}(\mathbf{b})$. We use a linear interpolation:

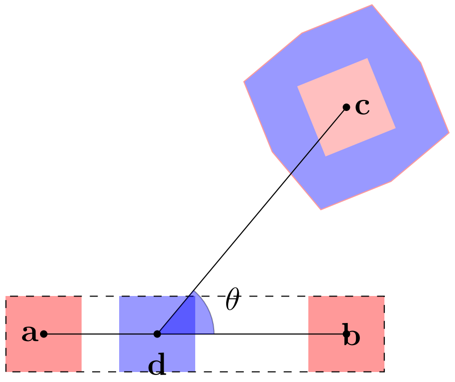
$$\mathcal{Z}(\mathbf{d}) = \text{Minkowski sum}\left(\mathcal{Z}(\mathbf{a})\frac{|\mathbf{bd}|}{|\mathbf{ab}|}, \mathcal{Z}(\mathbf{b})\frac{|\mathbf{ad}|}{|\mathbf{ab}|}\right), \tag{5.2}$$

where $|\cdot|$ indicates the segment length, and the scalar multiplication means scaling one tolerance zone along directions starting from the origin and pointing to their vertices.

We want to comment on the definition of the segment and related computations on tolerance zones. Notice that if we use two ends of the segment to define a line [29], then the tolerance zone of the line will fully cover the tolerance zone the segment. This means that our definition of the segment tolerance is compatible with that of the line it embeds. Furthermore, since tolerance zones of the two ends are both convex, it is not hard to deduce that the tolerance zone for any segment point is



(a) Perpendicularity



(b) Angularity

Figure 5.7: Tolerance zones for perpendicularity and angularity.

also convex and would also be covered by the zone of the segment if using linear interpolation. Therefore, the tolerance zones of the segment point and the segment are compatible, which provides the theoretical support for our cascading techniques. We can thus cascade the geometric primitives along a dimension chain having single dependencies.

5.2.2 Hybrid Dependency Cascading

Multiple dependencies are an important part of the dependency DAG. If a multiple dependency can be decoupled into several single dependences, such as in Figure 5.2b, we can apply single dependency cascading on each so that the tolerance can be transferred through those primitives. Otherwise, multiple dependency cascading has to be applied. We examine the case in Figure 5.2a since it is the basic building block for this problem. We need some theoretical preparations for deriving this type of cascading.

As we know, for an equation $f(x_1, \dots, x_n) = 0$ and one of its solutions $\bar{\mathbf{p}}$, the sensitivity matrix associated with $\bar{\mathbf{p}}$ is the Jacobian of f at $\bar{\mathbf{p}}$. The bases of the tolerance zone of $\bar{\mathbf{p}}$ are the column vectors of the sensitivity matrix. We have the following conclusion about the variations of the tolerances: The variations of tolerances could be an explicit scale range or an implicit range decided by some functions. Regardless, the shape, defined by angles of adjacent edges of the tolerance zone, as well as the convexity, is kept. Below is the formal proof.

Theorem 1. *The variations of tolerances could be an explicit scale range or an implicit range decided by some functions. Under both situations, the shape of the tolerance zone, decided by the angle of adjacent edges, and the convexity of the tolerance zone are the same.*

Without loss of generality, we prove that for function $(f(x,y), g(x,y))$ and $f(x(u,v),y), g(x(u,v),y)$, if $x_0 = x_0(u_0,v_0)$, then the bases of two tolerance zones have the same directions.

Proof. Follow the linearization model. The tolerance zone for the first function are represented by

$$\begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} f_0 \\ g_0 \end{pmatrix} + \left.\begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{pmatrix}\right|_{\substack{x=x_0\\y=y_0}} \begin{pmatrix} d_x \\ d_y \end{pmatrix} \tag{5.3}$$

The two columns in the variational matrix show the directions of the two bases for expanding the tolerance zone. The tolerance zone for the second function is

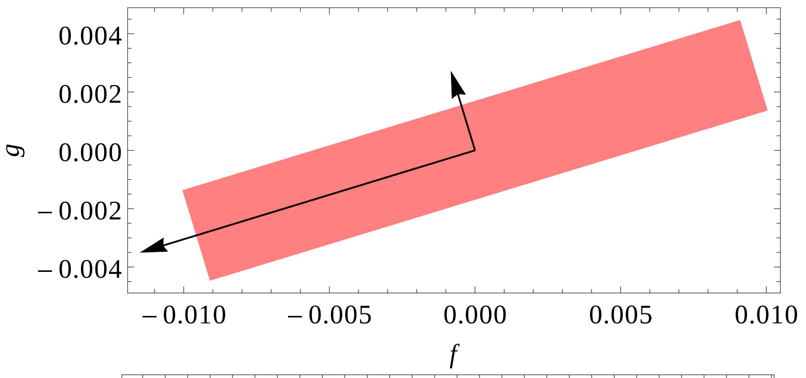
$$\begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} f_0 \\ g_0 \end{pmatrix} + \left.\begin{pmatrix} \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial u} & \frac{\partial g}{\partial v} & \frac{\partial f}{\partial y} \end{pmatrix}\right|_{\substack{u=u_0\\v=v_0\\y=y_0}} \begin{pmatrix} d_u \\ d_v \\ d_y \end{pmatrix} \tag{5.4}$$

The variation for f in (5.3) is $\frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy\Big|_{\substack{x=x_0\\y=y_0}}$.

The variation for f in (5.4) is

$$\frac{\partial f}{\partial u}du + \frac{\partial f}{\partial v}dv + \frac{\partial f}{\partial y}dy\bigg|_{\substack{u=u_0\\v=v_0\\y=y_0}} = \frac{\partial f}{\partial x}\left(\frac{\partial dx}{\partial du}du + \frac{\partial dx}{\partial dv}dv.\right) + \frac{\partial f}{\partial y}dy\bigg|_{\substack{u=u_0\\v=v_0\\y=y_0}}.$$

Since $x_0 = x(u_0, v_0)$, we can view $\frac{\partial dx}{\partial du}du + \frac{\partial dx}{\partial dv}dv$ as d_x. This means that the magnitude of the bases can be changed by the composition, but not the direction, and thus the angle of adjacent edges and the convexity of the tolerance zone are the same. $\square$



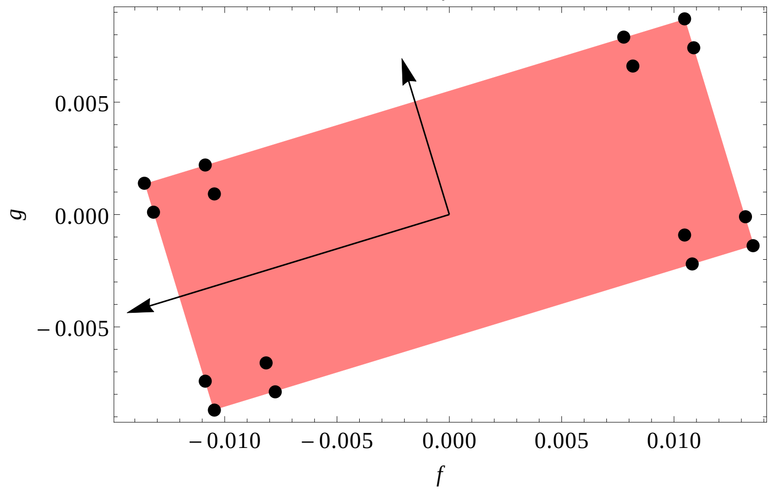


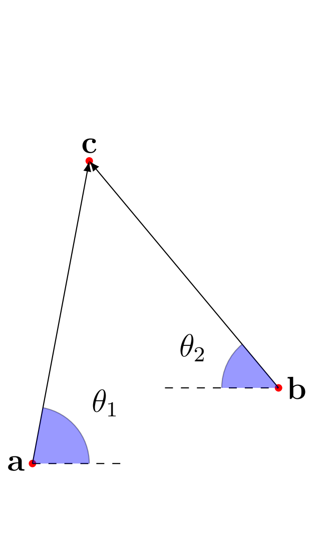
Figure 5.8: Tolerance zones for composite functions.

For completeness, we provide an example to illustrate this proof. Let $(f, g) = (r\cos(\theta), r\sin(\theta))$. In Figure 5.8, the pink region in the top shows the tolerance zone for (f, g). Let $r = r(\cos(u + v) + \sin(v))$, $\theta = e^z + \sin(w)$, which are both nonlinear functions. The pink region in the bottom shows the tolerance zone for the composite function. The black dots show the function values trying all combinations of $\{\pm d_u, \pm d_v, \pm d_z, \pm d_w\}$. Note that black arrows are parallel in the two figures, which supports the theorem.

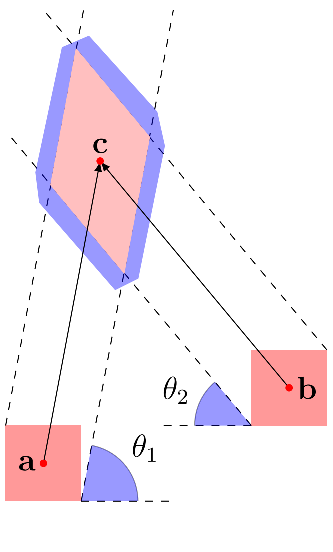
This conclusion indicates that the tolerance zone of the point $\mathbf{p}$ in (5.1) is centered at $\bar{\mathbf{p}}$ and is spanned by the column vectors of the sensitivity matrix. The variations of δ_i only affect the magnitude of the span along the bases. Having this shape consistency allows us to decompose the computation of the tolerance zone of the target in the multiple dependency into a few steps:

- Compute the target shape, i.e. compute the sensitivity matrix at point $\bar{\mathbf{p}}$.
- Compute the transferred tolerance zone in the target by finding the min/max ranges of the reference variations along the bases of the target shape determined earlier.
- Compute the zone formed by independent variations of the individual references.
- Zone $\mathcal{Z}(\mathbf{p})$ is the Minkowski sum between the transferred tolerance zone and the zone spanned by independent variations.

We show an example in Figure 5.9. Point $\mathbf{c}$ is decided by both $\mathbf{a}$ and $\mathbf{b}$ with the angular dimension constraints θ_1 and θ_2 as in Figure 5.9a. The column vectors of the sensitivity matrix indicate the bases are along the directions θ_1 and θ_2. In Figure 5.9b, by estimating the ranges of variations, we know the transferred tolerance



(a) Multiply reference



(b) Tolerance zone

Figure 5.9: Hybrid cascading.

zone is the pink parallelogram spanned by two bases. Angles θ_1 and θ_2 also have variations, which spans another (small) parallelogram. We know $\mathcal{Z}(\mathbf{c})$ is the octagon formed from the Minkowski sum of two parallelograms.

5.3 Tolerance Optimization

The dependency graph described earlier connects all primitives and represents dependencies of dimensions. For each primitive, its global tolerance zone can be obtained by applying the techniques for computing the relative tolerance zones and those for tolerance zone cascading. However, the dependency graph is not unique with respect to one dimensioning scheme. In this section, we present a technique for finding the dependency graph having minimum tolerances. In order to define it formally, we formulate this problem as an optimization problem.

5.3.1 Optimization Formulation

Given an existing dimensioning diagram, we could decompose it into geometric primitives. Each primitive $\mathbf{m}$ is associated with a global tolerance zone $\mathcal{Z}(\mathbf{m})$, which is yielded by a corresponding dependency graph. We define the cost function $\mathcal{E}$ for optimization in terms of the dependency graph G as

$$\mathcal{E}(G) = \sum_{i=1}^{n} w_i \cdot \|\mathcal{Z}(\mathbf{m}_i)\|, \tag{5.5}$$

where w_i is the weight associated with the primitive $\mathbf{m}_i$ and where $\|\cdot\|$ means a scaling function (e.g. the zone area) that maps $\mathcal{Z}(\mathbf{m}_i)$ to a scale value. The w_i will generally be specified by the engineer to indicate (by assigning larger values to the w_i) which primitives are a priority to minimize.

The optimization on tolerancing is usually motivated by cost requirements, functional requirements, assembly requirements, etc. Here we minimize the tolerance associated with one part. For each primitive on the part, we allow the designer to adjust its weight in optimization. The objective function in (5.5) has a two-fold meaning. One is that the tolerance is minimized so that the processing cost could be reduced without requiring more precision engineering machines or processes. The other is that designers could reflect the design intentions by adjusting the weights on primitives, so that the functional or assembly requirements could be met.

5.3.2 Augmented Graph

Though the dependency graph can represent the dependencies between primitives, two factors prevent us from finding an optimal solution on it directly. One is the tolerancing ambiguity. Take Figure 5.4b as an example. The chain $\mathbf{s}_1 \rightarrow \mathbf{s}_2 \rightarrow \mathbf{s}_4$ could be two levels of single dependency cascading or could be a part of a hybrid dependency cascade. The other is reasonableness of the optimized scheme. The solution space of the optimization is the space consisting of all dimensioning schemes. However, many of them are unreasonable in engineering because they are far from an acceptable scheme in practice, though they may seem to have lower cost. Our goal is to find a structure to present schemes that are practical for design and manufacturing without causing any dimensioning ambiguity. Therefore, we build an *augmented graph* from a given dependency graph, which represents the set of reasonable schemes "nearby" the original dependency graph among the whole solution set. We will also see that any ambiguity is resolved in this new representation. By applying the optimization on this subspace, we will have an optimized and reasonable engineering scheme. The augmentation steps are as follows.

- Create new nodes to represent multiple dependencies. Let's color the nodes of the original dependency gray and color the new type of nodes red. For a given multiple dependency $\{\mathbf{m}_1, \mathbf{m}_2\} \to \mathbf{t}$, we first create a red node $\mathbf{m}_{12}$; then we replace the original edges of the multiple dependency with three directed edges $\mathbf{m}_1\mathbf{m}_{12}$, $\mathbf{m}_2\mathbf{m}_{12}$ and $\mathbf{m}_{12}\mathbf{t}$.

- Decompose the multiple dependency, if possible, into several single dependencies. If a given multiple dependency $\{\mathbf{m}_1, \mathbf{m}_2\} \to \mathbf{t}$ is decomposable we add two directed edges $\mathbf{m}_1\mathbf{t}$ and $\mathbf{m}_2\mathbf{t}$ into the graph.

- Add additional edges for inverse dependencies. For a single reference $\{\mathbf{m}\} \to \mathbf{t}$, we add the directed edge $\mathbf{tm}$; for a multiple reference $\{\mathbf{m}_1, \mathbf{m}_2\} \to \mathbf{t}$, if it is invertible we add two directed edges $\mathbf{tm}_1$ and $\mathbf{tm}_2$ into the graph.

- Repeat until we cannot add more node or edge into the graph.

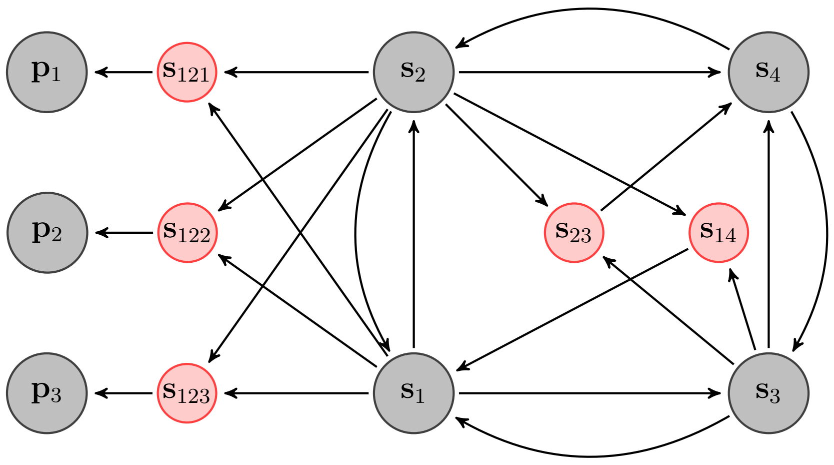


Figure 5.10: Augmented graph of the part.

Note that we don't add any duplicated directed edges into the graph. To illustrate the process, consider the part in Figure 5.4a; we create the augmented graph in Figure 5.10. First, we create red nodes $\mathbf{s}_{23}$, $\mathbf{s}_{121}$, $\mathbf{s}_{122}$ and $\mathbf{s}_{123}$ representing the multiple dependencies, remove the original edges for those dependencies, and add the associated directed edges for the red nodes accordingly. Second, the multiple dependencies of node $\mathbf{s}_{23}$ can be decomposed. Two edges $\mathbf{s}_2\mathbf{s}_4$ and $\mathbf{s}_3\mathbf{s}_4$ should be added (note that these edges existed previously but were removed in the prior step). Third, we invert the dependencies as much as possible. We add edges $\mathbf{s}_2\mathbf{s}_1$, $\mathbf{s}_3\mathbf{s}_1$, $\mathbf{s}_4\mathbf{s}_3$, and $\mathbf{s}_4\mathbf{s}_2$ into the graph. We repeat all steps. The earlier process caused $\mathbf{s}_1$ to have two edges from primitives pointing to it, and we thus create a red node $\mathbf{s}_{14}$ representing a new multiple dependency.

5.3.3 Optimization Solver

The goal of the optimization is to find a dependency graph of which the cost function in (5.5) could be minimized. The input is the augmented graph and the output is a dependency graph which can be translated to a new dimensioning scheme as well as a processing plan.

Since it is a DAG, the dependency graph must have some nodes as roots (no in-edges). For example, $\mathbf{s}_1$ is the sole root for the DAG in Figure 5.4b. The root(s) guarantee *reachability*, meaning that non-root nodes could be reached via the graph that starts from those roots. Our optimization process aims to (1) identify appropriate root nodes, and (2) for those root nodes, determine an order of evaluation of the dependencies. In this way, a dependency DAG is extracted from the augmented graph, and our goal is to find the dependency DAG that minimizes (5.5).

Assume that a root node(s) has been determined in the augmented graph. The root node is assumed to have an exact dimension, and thus the tolerance zone will only be a point. Given root nodes, a dependency chain can be created and tolerance zones propagated in the augmented graph: gray nodes will be determined by the evaluation methods for primitives. Red nodes can be evaluated (i.e. the gray node the red node points to can be evaluated) only once all gray nodes pointing to it have been evaluated.

To find the optimal dependency graph, we will do the following:

- Choose a set of root nodes. Note that a user may also specify the root nodes directly if the situation dictates it (e.g. a fixture point). All other nodes must be reachable from the root(s). We will typically consider using only one or two root nodes, since this is all that is typically allowed in practice. It also makes an exhaustive check feasible. Notice that a trivial "optimal" solution will be to make every node a root node. Our goal is to find an optimal dependency graph, and keeping a minimal set of roots is a part of that.

- For the given root node(s), compute the minimized cost for the remaining nodes. This is the major part of the optimization. We attempt to exploit the optimal substructure for this problem. In (5.5), the cost for node $\mathbf{p}$ is $\|(\mathcal{Z}(\mathbf{p}))\|$. Assume the reference nodes that points to $\mathbf{p}$ are $\mathbf{p}_i, 1 \leq i \leq n$. We have

$$\|(\mathcal{Z}(\mathbf{p}))\| = \min\{\|\text{Cascade}(\mathcal{Z}(\mathbf{p}_i), \mathcal{R}(\mathbf{p}))\|\}. \tag{5.6}$$

If $Z(\mathbf{p}_i)$ also yields the minimal cost for node $\mathbf{p}_i$, then formula (5.6) defines the optimal substructure of the optimization. This means the subproblems are overlapping and therefore we can use dynamic programming to solve this problem efficiently. Whether optimal substructure holds depends on the selection of the scaling function, which will be discussed later.

- Evaluate the overall cost function; if this is a new minimum, determine the associated dependency graph and store it as the current best graph.
- Repeat the process by choosing a new root candidate until all candidates have been determined.

Selecting the scaling function is the key problem for our evaluation method. Ideally, this function will be compatible with the optimal substructure in formula (5.6) so that we can overlap subproblems during optimization. In practice, however, it is usually hard to find such a function which satisfies the requirement strictly. We therefore propose to use the area of the tolerance zone as the scaling function. This selection makes the original optimization become an NP hard problem because the optimal substructure is broken and enumerating the solution set is the only way to compute the optimal solution. However, we show that for most cases, this selection adapts well to the optimal substructure criteria.

First note that points form the basis for most of the computations, both as direct feature positions and as segment endpoints. From experimental observation we found that the cascaded tolerance zones of points tend to approach circularities. This is caused by 1) the convexity of relative tolerance zones and 2) the random distribution of the directions of bases spanning the tolerance zones. We mostly use Minkowski sum to cascade tolerance zones and thus the cascaded zones expand in every direction in a uniform way, and become more circular as the tolerances cascade. We can easily verify that if the tolerance zones are circular, then the optimal structure is always satisfied by using the area as our scaling function. We take the perpendicularity in Figure 5.7a as an example. If $\mathcal{Z}(\mathbf{a})$ and $\mathcal{Z}(\mathbf{b})$ are both circles, then $\mathcal{Z}(\mathbf{d})$ is also a circle. If we decrease the size of $\mathcal{Z}(\mathbf{a})$ or $\mathcal{Z}(\mathbf{b})$ then $\mathcal{Z}(\mathbf{d})$ will also be decreased (as well as $\mathcal{Z}(\mathbf{c})$). The inverse of this procedure also holds. As a result, area tends to be a good scaling function; though not guaranteeing optimality, it is close to doing so, and thus allows us to apply dynamic programming and achieve good results.

We show that our dynamic programming paradigm has a polynomial complexity bounded by $O(n^4)$, where n is the number of nodes in the augmented graph. The complexity for choosing root candidates is $\binom{n}{2}$. For evaluating non-root nodes, we use the bottom up scheme. If one node can be evaluated, we compute its tolerance zone and mark it as evaluated. We try to evaluate all non-evaluated nodes. This could be achieved with complexity $O(n^2)$. Therefore, the complexity of the whole algorithm is bounded by $O\left(\binom{n}{2} \cdot n^2\right) = O(n^4)$.

5.4 Experiments

We first show the tolerance cascading results which are used for estimating the tolerance zone. Then we show an integrated experiment for optimizing a discrete 3D part.

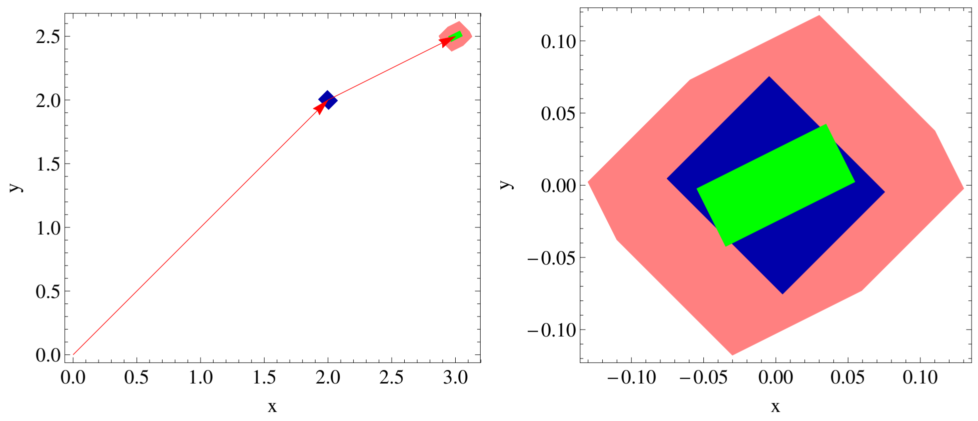


Figure 5.11: Cascaded tolerance zone for point-point cascading.

5.4.1 Cascaded Tolerance Zone

The first experiment demonstrates the result of point cascading as in Figure 5.11. Points $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are placed at $(0,0)$, $(2,2)$ and $(3,2.5)$ and $\mathbf{a}$ is considered to be fixed. We assume a relative angular tolerance of 0.02 (about one degree) and a radius tolerance of 0.05. The blue region shows $\mathcal{R}(\mathbf{b},\{\mathbf{a}\})$. The green region shows $\mathcal{R}(\mathbf{c},\{\mathbf{b}\})$. The pink region shows $\mathcal{Z}(\mathbf{c})$ which is the Minkowski sum of the blue and green regions.

The second experiment shows a result of an angularity formed from two segments as in Figure 5.12. We have two segments $\mathbf{p}_1\mathbf{p}_2$ and $\mathbf{q}_1\mathbf{q}_2$, with $\mathbf{p}_1$ located on $\mathbf{q}_1\mathbf{q}_2$ and an angle specified between the two segments. Their ends are shown with red dots. If we only consider the relative tolerance for $\mathbf{p}_1\mathbf{p}_2$, then $\mathbf{q}_1\mathbf{q}_2$ is the reference. The region of $\mathcal{R}(\mathbf{p}_1\mathbf{p}_2)$ is the convex hull of $\mathcal{R}(\mathbf{p}_2)$ and point $\mathcal{R}(\mathbf{p}_1)$. Now we add the tolerance zone to segment $\mathbf{q}_1\mathbf{q}_2$. Zone $\mathcal{Z}(\mathbf{q}_1)$ is induced by a reference point at $(0,-1)$ with the angle tolerance 0.05 and the radius tolerance 0.05. Zone $\mathcal{Z}(\mathbf{q}_2)$ is induced by a reference chain $(1,-1) \rightarrow (2,-1) \rightarrow \mathbf{q}_2$, with the same angle and radius tolerances as $\mathbf{q}_1$. We can see that $\mathcal{Z}(\mathbf{q}_1)$ is a rectangle while $\mathcal{Z}(\mathbf{q}_2)$ is an octagon;

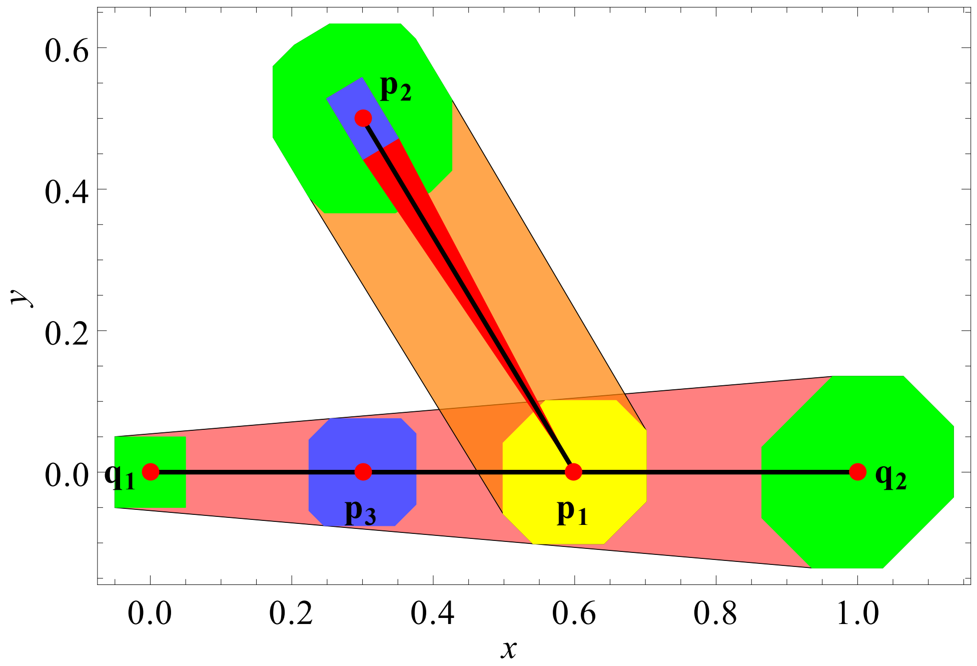


Figure 5.12: Cascaded tolerance zone for segment-segment cascading.

both are shown in Green. We use the technique in section 5.2 to obtain $\mathcal{Z}(\mathbf{p}_1)$ from $\mathcal{Z}(\mathbf{q}_1)$ and $\mathcal{Z}(\mathbf{q}_2)$. Then $\mathcal{Z}(\mathbf{p}_2)$ is the Minkowski sum of $\mathcal{R}(\mathbf{p}_2)$ and $\mathcal{Z}(\mathbf{p}_1)$ and the tolerance zone for the segment is the convex hull of $\mathcal{Z}(\mathbf{p}_1)$ and $\mathcal{Z}(\mathbf{p}_2)$, shown as the orange region.

Notice that the tolerance zones for points on a segment, such as $\mathbf{p}_1$ and $\mathbf{p}_3$, are fully covered by the tolerance zone of the segment, which indicates that the definition of the tolerance zone for the segment and that for the point on it are compatible with each other.

The third experiment shows the cascading result of the hybrid cascading as in Figure 5.13. Point $\mathbf{q}_1$ is at $(0, 0)$ and point $\mathbf{q}_2$ is at $(1, 0)$. We use the distance constraint to decide point $\mathbf{p}$, where the distance to $\mathbf{q}_1$ is $|\mathbf{pq}_1| = 0.7$ and to $\mathbf{q}_2$ is $|\mathbf{pq}_1| = 0.8$. When both $\mathbf{q}_1$ and $\mathbf{q}_2$ have their own tolerance zones, shown in green, the zone of $\mathbf{p}$ is decided only by $|\mathbf{pq}_1|$ and $|\mathbf{pq}_2|$. The green region around $\mathbf{p}$ shows

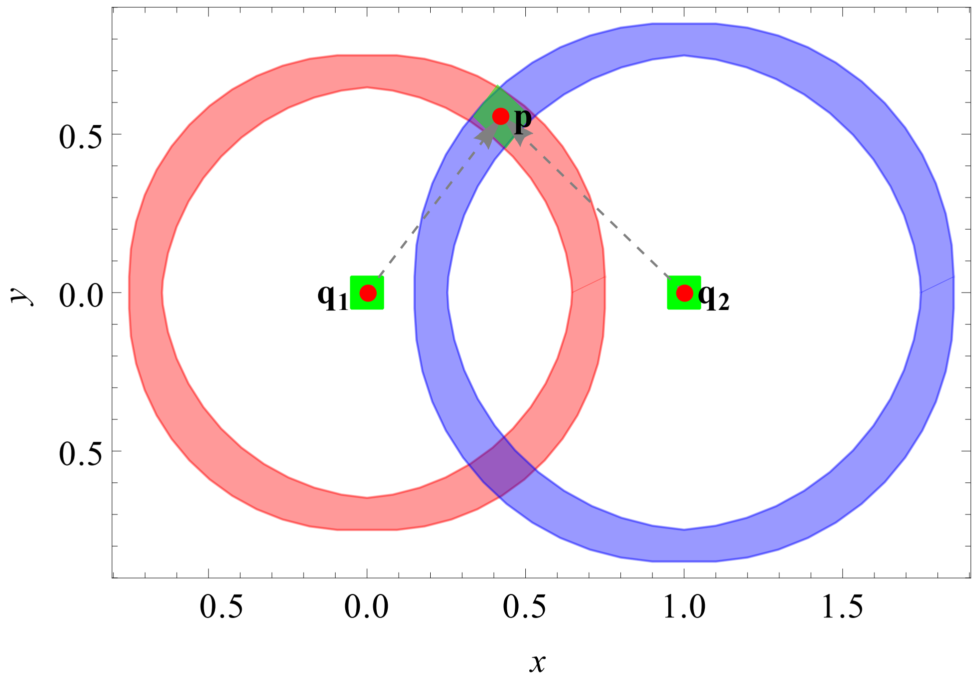


Figure 5.13: Multiple dependency cascading and tolerance zone for curved feature.

the linearized $\mathcal{Z}(\mathbf{p})$. We also paint the exact tolerance zone for $\mathbf{q}_1$ in pink and the same zone for $\mathbf{q}_2$ in blue. We can see that our linearization of $\mathcal{Z}(\mathbf{p})$ covers the intersection of both bands well. In addition, this experiment also shows that the linearization of the tolerance zone on points and segments can be used to construct the curved features such as the circle.

5.4.2 Tolerance Optimization

In this part, we show the results of experiments that use our optimization, thereby improving the tolerance for the part and aiding in process planning.

The first experiment shows two different dimensioning schemes to obtain the tolerance of a segment as in Figure 5.14. Point $\mathbf{p}$ is placed at $(1.0, 1.0)$. Point $\mathbf{q}_1$ is at $(3.0, 1.0)$ and point $\mathbf{q}_2$ is at $(1.0, 2.0)$, which are two ends of segment $\mathbf{q}_1\mathbf{q}_2$. We set an angular tolerance of 0.02 and dimensioning tolerance of 0.05, with only point

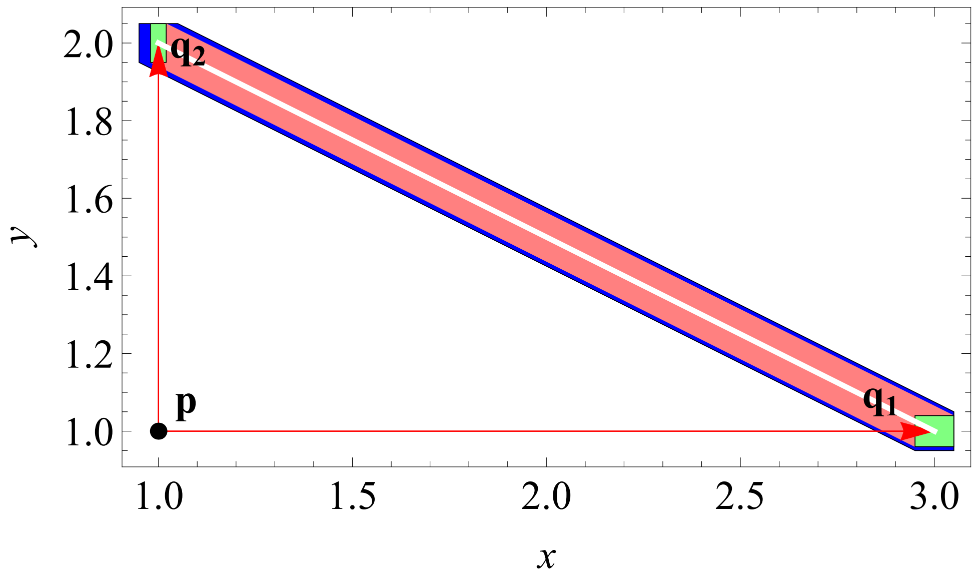


Figure 5.14: Optimized dimensioning scheme.

$\mathbf{p}$ being fixed. The blue region shows $\mathcal{Z}(\mathbf{q}_1\mathbf{q}_2)$ using the conventional tolerancing scheme that takes the $x-y$ axis as the references. In contrast, we use two angular references $\overrightarrow{\mathbf{pq}_1}$ and $\overrightarrow{\mathbf{pq}_2}$ to decide the segment tolerance zone. The two green regions show the tolerance zones of $\mathbf{q}_1$ and $\mathbf{q}_2$ while the pink region is the new zone of the segment, which is smaller than the blue one.

Finally, we show an optimization experiment on a 3D part, a part of a gearbox frame. The 3D view, top and front views are shown in Figure 5.15. As stated earlier, we are limiting ourselves to 2D operations, so we will simplify only tolerances on the side view.

We applied two optimizations. In the first one we set a larger angular tolerance for features. The optimization result, after translation into dimensioning scheme is shown in the top of Figure 5.16. Segments $\mathbf{s}_2$ and $\mathbf{s}_3$ are picked as roots of the references. Circle features use $\mathbf{s}_1$ and $\mathbf{s}_4$ as their references. Notice that all dimensions are specified via dimensioning, which is to be expected since the angular tolerance is

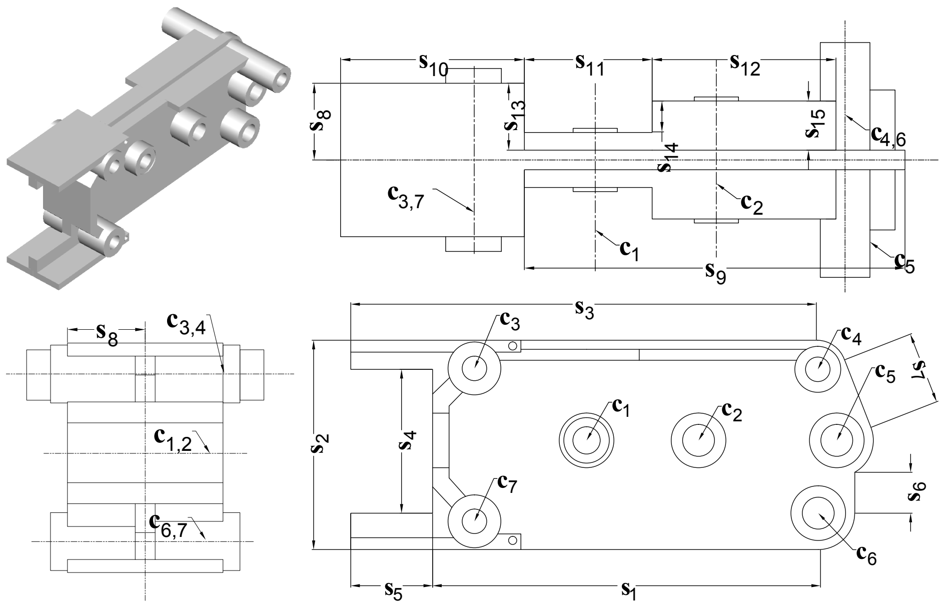


Figure 5.15: Original part.

so high. In the second optimization, we instead set a smaller angular tolerance and weights to give preference to the circle features. The optimization results are shown at the bottom of Figure 5.16. Circle centers $\mathbf{c}_1$ and $\mathbf{c}_2$ are picked as the reference roots. The remaining circle features are associated with $\mathbf{c}_1$ or $\mathbf{c}_2$ via angularity. Segments $\mathbf{s}_2$ and $\mathbf{s}_3$ also depend on $\mathbf{c}_1$ and $\mathbf{c}_2$. For simplicity, we omit some trivial dimensions in the diagram. Notice that $\mathbf{s}_6$ depends on $\mathbf{c}_5$ and $\mathbf{c}_6$, and $\mathbf{s}_7$ depends on $\mathbf{c}_4$ and $\mathbf{c}_5$. Both of them are not invertible so that their dependencies are kept the same during optimization.

The next goal of our work is to extend our method into 3D space. Here we present an idea on how to achieve this when the real part is represented by three or more different 2D views, as shown in Figure 5.15. In different views, though those primitives have different geometric forms, they are not decomposable, i.e., we can always find the geometric forms for each primitive in different views. For example, $\mathbf{c}_3$ is a line in the top view and it is a point in the front view. For different

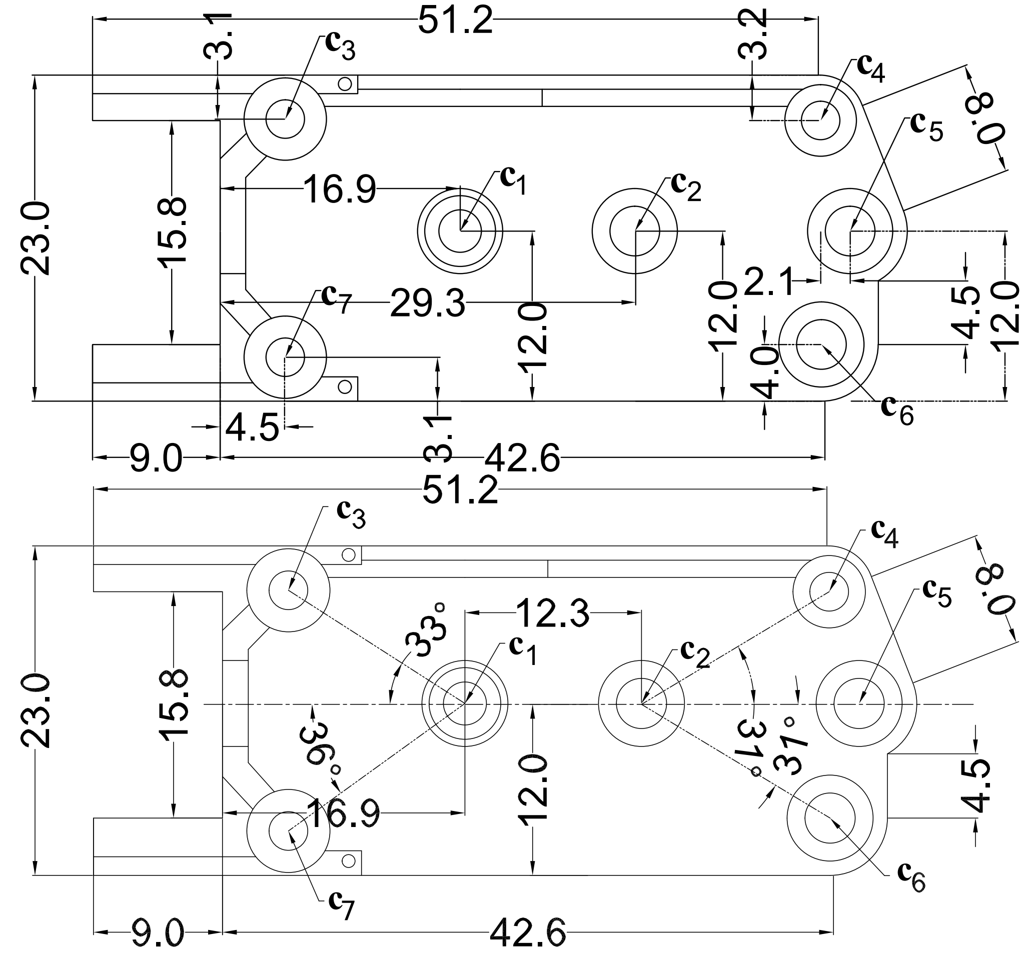


Figure 5.16: Optimization results.

views, we can build different augmentation graphs. We use the same name to mark nodes in different views if they are representing the same primitive, which builds the correlations of features among different views. We associate a weight to the cost function associated with each view and sum them up to obtain a global cost function. This will allow us to compute the optimization on those augmented graphs concurrently so that we will obtain an approximated optimal solution for the 3D part.

5.5 Discussion on Tolerance Optimization

We have presented a geometric method for tolerance analysis for which the results are represented by geometric shapes. Compared with conventional tolerance analysis, this method provides the geometric approximation of the tolerance zone as well as numeric boundaries, such as min/max range. Meanwhile, the dimension dependencies are translated to a general topology graph, which provide the structure for optimizing the dimensioning scheme. Though this geometric optimization is an NP hard problem, we propose an approximation method that allows us to build an efficient dynamic programming solver for the optimization. Our method also demonstrates that the LPGUM approach can serve as an effective method for tolerance modeling and analysis. To our knowledge, this is the first more general application, and first implementation, of the LPGUM model.

Because of its general definition, our method brings several advantages to the designer

- The user can adjust the tolerance preferences. The user can give some features high preferences, so that they will be given precedence during the optimization. The direct result is that the dimensions associated with those features are kept as accuracy as possible.

- Since the dimension scheme will be optimized, our method will generally allow relatively larger individual tolerances while still meeting the same overall tolerance requirements.

- We can allow automatic tolerance reallocation when the tolerances of the same features are kept the same. A designer can set a threshold for the tolerances and check if the target tolerance can be meet for requirements. The optimization will reallocate the tolerance distributions by adjusting the dimension dependencies appropriately among primitives.

There are some limitations of our method which provides potential for future research. First, the tolerance zones are represented by polygons by using the first-order approximation of LPGUM. While LPGUM has a number of advantages we exploit, other methods may provide a more accurate representation of the tolerance zone for nonlinear primitives. Second, along the same lines, a wider range of geometric primitives (such as parametric curves) could be included. Our current approach requires most curves to be linearized. Third, simplification of complicated models might be necessary before optimization. This will shorten the length of the dimension chains so the computing complexity can be reduced. Finally, our examples are demonstrated mainly in 2D space, for which the geometric computations and relations are relatively simple. In the near future, we could attempt to build a direct 3D optimization on parts by following the approach outlined earlier.

6. CONCLUSIONS

In this dissertation, we apply new optimizations for several problems in geometric and solid modeling. In detail, we control the distortion distribution in the local parameterization; we obtain a global parameterization of good quality by adding redundant integer variables into the optimization system and solve the system with an efficient numeric method; we also obtain an optimized plan for parts processing by applying geometric optimization, which improves the processing precision without requiring the enhancement of manufacturing cost or techniques.

In general our work includes contributions as below.

We propose new numeric optimization formulations for existing problems. For local surface parameterization, the control of the distortion provides the possibility for accommodating applications that benefit from an uneven distortion distribution, such as 3D surface painting. For global quadrangulation, we provide quadrangulation methods over optimal and feature-aligned vertex fields. The performance of computation of parameterization coordinates is improved by proposing a new efficient numerical solver.

We propose a new geometric optimization and solve it by utilizing the topological properties between geometric elements. Instead of using the optimization by a group of analytic functions, we represent the optimization by a group of geometric elements, such as point, segment and circle, etc. The criterion to be optimized is represented by polytopes in geometric space. We also present methods that adapt to our representation and allow us to construct the (approximated) optimal solution.

Finally, our work has very strong application. We can see that our optimizations play very important roles in geometric modeling. By modeling the problem using proper numeric or geometric optimizations, we obtained optimized results that benefit wide and extensive applications in art, industry, and life.

REFERENCES

[1] S. Akella and M. T. Mason. Orienting toleranced polygonal parts. *The International Journal of Robotics Research*, 19(12):1147–1170, 2000.

[2] M. Barkallah, J. Louati, and M. Haddar. Evaluation of manufacturing tolerance using a statistical method and experimentation. *International Journal of Machine Tools and Manufacture*, 45(10):1124–1134, 2012.

[3] M. Ben-Chen, C. Gotsman, and G. Bunin. Conformal flattening by curvature prescription and metric scaling. *Computer Graphics Forum*, 27(2):449–458, 2008.

[4] D. Benson and J. Davis. Octree textures. *ACM Transactions on Graphics*, 21(3):785–790, 2002.

[5] D. Bommes, M. Campen, H.-C. Ebke, P. Alliez, and L. Kobbelt. Integer-grid maps for reliable quad meshing. *ACM Transactions on Graphics*, 32(4):98:1–98:12, 2013.

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

[7] D. Bommes, H. Zimmer, and L. Kobbelt. Practical mixed-integer optimization for geometry processing. In *Curves and Surfaces, 7th International Conference*, pages 193–206, Avignon, France, 2012. Springer.

[8] R. C. Brost and R. R. Peters. Automatic design of 3D fixtures and assembly pallets. *The International Journal of Robotics Research*, 17(12):1243–1281, 1998.

[9] B. Burley and D. Lacewell. Ptex: per-face texture mapping for production rendering. *Computer Graphics Forum*, 27(4):1155–1164, 2008.

[10] N. A. Carr and J. C. Hart. Meshed atlases for real-time procedural solid texturing. *ACM Transactions on Graphics*, 21(2):106–131, 2002.

[11] N. A. Carr and J. C. Hart. Painting detail. *ACM Transactions on Graphics*, 23(3):845–852, 2004.

[12] A. Clément and P. Bourdet. A study of optimal-criteria identification based on the small-displacement screw model. *CIRP Annals-Manufacturing Technology*, 37(1):503–506, 1988.

[13] K. Crane, M. Desbrun, and P. Schröder. Trivial connections on discrete surfaces. *Computer Graphics Forum*, 29(5):1525–1533, 2010.

[14] K. Crane, C. Weischedel, and M. Wardetzky. Geodesics in heat: a new approach to computing distance based on heat flow. *ACM Transactions on Graphics*, 32(5):152:1–152:11, 2013.

[15] J. Davidson, A. Mujezinovic, and J. Shah. A new mathematical model for geometric tolerances as applied to round faces. *Journal of Mechanical Design*, 124(4):609–622, 2002.

[16] T. A. Davis. Users guide for suitesparseQR, a multifrontal multithreaded sparse QR factorization package. *http://www.hpc.wm.edu/sciclone/documentation/software/math/suitesparse-3.7.1/SPQR_UserGuide.pdf*, 2012.

[17] A. Desrochers. A CAD/CAM representation model applied to tolerance transfer methods. *Journal of Mechanical Design*, 125(1):14–22, 2003.

[18] 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.

[19] H.-C. Ebke, D. Bommes, M. Campen, and L. Kobbelt. QEx: robust quad mesh extraction. *ACM Transactions on Graphics*, 32(6):168:1–168:10, 2013.

[20] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. *Advances in Multiresolution for Geometric Modelling*, 1(1):157–186, 2005.

[21] J. Gibbs, D. D. Petty, and N. Robins. Painting and rendering textures on unparameterized models. *ACM Transactions on Graphics*, 21(3):763–768, 2002.

[22] M. Giordano, E. Pairel, and S. Samper. Mathematical representation of tolerance zones. In *Global Consistency of Tolerances, Proceedings of the 6th CIRP International Seminar on Computer-Aided Tolerancing*, pages 177–186, Enschede, Netherlands, 1999. Springer.

[23] E. Gunpinar, H. Suzuki, Y. Ohtake, and M. Moriguchi. Generation of bi-monotone patches from quadrilateral mesh for reverse engineering. *Computer-Aided Design*, 45(2):440–450, 2013.

[24] P. Hanrahan and P. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. *Computer Graphics*, 24(4):215–223, 1990.

[25] Y. Hong and T. Chang. A comprehensive review of tolerancing research. *International Journal of Production Research*, 40(11):2425–2459, 2002.

[26] J. Huang, M. Zhang, J. Ma, X. Liu, L. Kobbelt, and H. Bao. Spectral quadrangulation with orientation and alignment control. *ACM Transactions on Graphics*, 27(5):147:1–147:9, 2008.

[27] S. H. Huang, Q. Liu, and R. Musa. Tolerance-based process plan evaluation using monte carlo simulation. *International Journal of Production Research*, 42(23):4871–4891, 2004.

[28] T. Igarashi and D. Cosgrove. Adaptive unwrapping for interactive texture painting. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 209–216, New York, USA, 2001. ACM.

[29] L. Joskowicz, Y. Ostrovsky-Berman, and Y. Myers. Efficient representation and computation of geometric uncertainty: the linear parametric model. *Precision Engineering*, 34(1):2–6, 2009.

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

[31] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein. WYSIWYG NPR: drawing strokes directly on 3D models. *ACM Transactions on Graphics*, 21(3):755–762, 2002.

[32] Z. Kami, C. Gotsman, and S. J. Gortler. Free-boundary linear parameterization of 3d meshes in the presence of constraints. In *International Conference on Shape Modeling and Applications*, pages 266–275, Saarbrucken, Germany, 2005. IEEE.

[33] L. Kharevych, B. Springborn, and P. Schröder. Discrete conformal mappings via circle patterns. *ACM Transactions on Graphics*, 25(2):412–438, 2006.

[34] F. Knöppel, K. Crane, U. Pinkall, and P. Schröder. Globally optimal direction fields. *ACM Transactions on Graphics*, 32(4):59:1–59:10, 2013.

[35] S. Lefebvre and H. Hoppe. Perfect spatial hashing. *ACM Transactions on Graphics*, 25(3):579–588, 2006.

[36] S. Lefebvre, S. Hornus, and F. Neyret. Texture sprites: texture elements splatted on surfaces. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, pages 163–170, Washington DC, USA, 2005. ACM.

[37] E. Lehtihet, S. Ranade, and P. Dewan. Comparative evaluation of tolerance control chart models. *International Journal of Production Research*, 38(7):1539–

1556, 2000.

[38] 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.

[39] X. S. Li, J. W. Demmel, J. R. Gilbert, L. Grigori, M. Shao, and I. Yamazaki. SuperLU users' guide. *http://crd.lbl.gov/˜xiaoye/SuperLU/superlu_ug.pdf*, 2011.

[40] Y. Li, Y. Liu, W. Xu, W. Wang, and B. Guo. All-hex meshing using singularity-restricted field. *ACM Transactions on Graphics*, 31(6):177:1–177:11, 2012.

[41] L. Liu, L. Zhang, Y. Xu, C. Gotsman, and S. J. Gortler. A local/global approach to mesh parameterization. *Computer Graphics Forum*, 27(5):1495–1504, 2008.

[42] E. L. Melvær and M. Reimers. Geodesic polar coordinates on polygonal meshes. *Computer Graphics Forum*, 31(8):2423–2435, 2012.

[43] J. S. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.

[44] A. Mujezinovic, J. Davidson, and J. Shah. A new mathematical model for geometric tolerances as applied to polygonal faces. *Journal of Mechanical Design*, 126(3):504–518, 2004.

[45] Y. Myers and L. Joskowicz. Uncertain lines and circles with dependencies. *Computer-Aided Design*, 45(2):556–561, 2013.

[46] A. Myles and D. Zorin. Global parametrization by incremental flattening. *ACM Transactions on Graphics*, 31(4):109:1–109:11, 2012.

[47] A. Myles and D. Zorin. Controlled-distortion constrained global parametrization. *ACM Transactions on Graphics*, 32(4):105:1–105:14, 2013.

[48] M. Nieser, U. Reitebuch, and K. Polthier. Cubecover–parameterization of 3D volumes. *Computer Graphics Forum*, 30(5):1397–1406, 2011.

[49] B. M. Oh, M. Chen, J. Dorsey, and F. Durand. Image-based modeling and photo editing. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 433–442, Los Angeles, CA, USA, 2001. ACM.

[50] Y. Ostrovsky-Berman and L. Joskowicz. Relative position computation for assembly planning with planar toleranced parts. *The International Journal of Robotics Research*, 25(2):147–170, 2006.

[51] H. K. Pedersen. Decorating implicit surfaces. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 291–300, Los Angeles, CA, USA, 1995. ACM.

[52] C.-H. Peng, M. Barton, C. Jiang, and P. Wonka. Exploring quadrangulations. *ACM Transactions on Graphics*, 33(1):12:1–12:13, 2014.

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

[54] N. Ray, B. Vallet, L. Alonso, and B. Levy. Geometry-aware direction field processing. *ACM Transactions on Graphics*, 29(1):1:1–1:11, 2009.

[55] N. Ray, B. Vallet, W. C. Li, and B. Lévy. N-symmetry direction field design. *ACM Transactions on Graphics*, 27(2):10:1–10:13, 2008.

[56] U. Roy and B. Li. Representation and interpretation of geometric tolerances for polyhedral objects. II.: size, orientation and position tolerances. *Computer-Aided Design*, 31(4):273–285, 1999.

[57] R. Schmidt. Stroke parameterization. *Computer Graphics Forum*, 32(2):255–263, 2013.

[58] R. Schmidt, C. Grimm, and B. Wyvill. Interactive decal compositing with discrete exponential maps. *ACM Transactions on Graphics*, 25(3):605–613, 2006.

[59] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[60] Z. Shen, G. Ameta, J. J. Shah, and J. K. Davidson. A comparative study of tolerance analysis methods. *Journal of Computing and Information Science in Engineering*, 5(3):247–256, 2005.

[61] J. Tierny, J. Daniels II, L. G. Nonato, V. Pascucci, and C. T. Silva. Inspired quadrangulation. *Computer-Aided Design*, 43(11):1516–1526, 2011.

[62] F. Villeneuve, O. Legoff, and Y. Landon. Tolerancing for manufacturing: a three-dimensional model. *International Journal of Production Research*, 39(8):1625–1648, 2001.

[63] S. Xu and J. Keyser. Texture mapping for 3d painting using geodesic distance. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 163–163, San Francisco, CA, USA, 2014. ACM.

[64] S.-G. Xu, Y.-X. Zhang, and J.-H. Yong. A fast sweeping method for computing geodesics on triangular manifolds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(2):231–241, 2010.

[65] C. Yuksel, J. Keyser, and D. H. House. Mesh colors. *ACM Transactions on Graphics*, 29(2):1–11, 2010.

[66] M. Zhang, J. Huang, X. Liu, and H. Bao. A wave-based anisotropic quadrangulation method. *ACM Transactions on Graphics*, 29(4):118:1–118:8, 2010.

[67] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3D: an interactive system for point-based surface editing. *ACM Transactions on Graphics*, 21(3):322–329, 2002.