Modeling and processing of physically based dynamic t-splines

Ma, Yuewen

2013

Ma, Y. (2013). Modeling and processing of physically based dynamic t-splines. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/59539>

<https://doi.org/10.32657/10356/59539>

NANYANG TECHNOLOGICAL UNIVERSITY

MODELING AND PROCESSING OF PHYSICALLY BASED
DYNAMIC T-SPLINES

BY
YUEWEN MA

A THESIS SUBMITTED TO THE NANYANG TECHNOLOGICAL UNIVERSITY IN
FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

2013

Abstract

This research explores how to incorporate physical quantities such as mass distributions, internal deformation energies, damping, and forces into the T-spline geometric substrate and its related geometric processing algorithms. The research is inspired by the successful physics-based design paradigm of dynamic non-uniform rational B-splines (D-NURBS) and T-spline technology. D-NURBS introduce physical quantities into the NURBS geometric formulation through the application of Lagrangian mechanics and provide a new design paradigm that integrates standard geometric design and physics-based design. However, there exist serious weaknesses within the D-NURBS scheme. D-NURBS do not support local refinement which is very important in geometric design and engineering computation, D-NURBS generally do not have closed-form solutions and need numerical methods, and currently D-NURBS lack very effective surface editing tools. T-splines are a free-form surface technology that solve most of the limitations inherent in NURBS representation. In particular, T-splines allow local refinement and are well-suited for geometric modeling, and engineering computation and analysis. Therefore, it is tempting to develop physically-based dynamic T-splines (or PD-T-splines) and related algorithms to overcome the weaknesses of D-NURBS.

In this thesis, we first present the formulation of PD-T-splines. The dynamic equation that governs the motion of the PD-T-splines is derived based on the work-energy version of Lagrangian Dynamics. The highly non-linear equation motivates us to further propose Efficient PD-T-splines (or E-PD-T-splines) by freezing the weights, whose dynamic equation becomes a linear ordinary differential equations (ODE). A closed form solution is derived based on the physical point of view that a vibration system can be viewed as a coupling of individual vibrant modes. Based on this physical insight, a modal reduction technique is developed to reduce computational complexity. We also consider incorporating the local refinement property of T-splines into PD-T-splines, which means the change of the generalized coordinates as well as the dynamic equation. We present an efficient method to achieve

local update of these physical matrices along with local refinement.

Second, we present a new modeling tool based on general curve handles. Curve based modeling techniques become prevalent and are shown more effective than point based. We propose to deform the T-spline surface by adjusting the sketched curve handles on the surface. The curve handle is defined by the composition between a 2D domain curve and the incident T-spline surface. To facilitate the adjustment of the composite curve, physically based dynamic composite (PBDC) curve is developed directly based on the composite representation, which supports arbitrary type of 2D domain curves. To avoid the possible rank deficiency of its dynamic equation, we carefully examine the relation between the dimension of the composite blending functions and the rank of their corresponding construction matrix. We present an approach to identify a basis for the vector space spanned by those blending functions, and compute their dependency. After filtering out the deficient DOFs, we obtain a regularized PBDC curve, which serves as the main tool to adjust the general curve handle on the surface. Finally, the surface deformation is achieved through introducing the curve force into PD-T-splines, which connects the handle curve and the 3D target curve in space.

Third, to explore geometric processing of PD-T-Splines such as morphing and rendering, we consider the problem of constrained mapping. We reveal the relation between a smooth mapping and its induced transformation and propose a refinement strategy based on the longest edge bisection to guarantee that the induced transformation is locally foldover free if the smooth mapping is locally bijective. To make the RBF-based mapping locally bijective, we propose to find non-intersecting trajectories to guide the warping and derive a bound for the warping stepsize. Furthermore, to make the warping process more effective, we derive the truly foldover free condition for the induced transformation. Based on this condition and the bound for the local bijective smooth mapping, we determine whether we perform local mesh refinement or warping and the actual warping stepsize as well. Integrating all these technical components provides a constrained mesh warping algorithm that is effective, provably foldover free, able to handle a large number of constraints, and able

to output a visually pleasing result without extra smoothing optimization.

Fourth, we present a novel solution to the T-spline morphing problem. It is well known that morphing involves two basic steps: vertex correspondence and vertex path. We solve the vertex correspondence problem by consistent approximation technique, which aligns important feature points using our proposed constrained mapping technique and approximates the two input surfaces using the same preimage of control T-mesh. Observing that PD-T-splines naturally produce the dynamic evolution from one pose to another, the vertex path problem can be readily solved by introducing PD-T-splines into morphing. In this case, we impose the boundary conditions into the ODE, which is also analytically solvable. Furthermore, we also introduce extra DOFs into the physical model to enable speed control.

Finally, we consider the rendering of the dynamic evolution in both modeling and morphing. We present an efficient method to map a static image onto the evolution sequence. Here we would like to achieve dynamic rendering in realtime along with dynamic evolution. Rather than applying the technique on every frame separately, we propose to establish the constrained map making use of the existing map. We first compute a satisfactory map for the initial frame. Then we generate the subsequent maps from the first map, which only involves the calculation of the barycentric coordinates. In this way, the proposed method can render the dynamic evolution in realtime.

Acknowledgements

I would like to express great appreciation to my supervisor Dr. Jianmin, Zheng for his constant help and guidance in this research. I would never forget the numerous discussions with him, which really benefit me a lot. And the invaluable experiences he has shared with me has profound impact on my attitude toward life.

I want to thank Nanyang Technological University and the School of Computer Engineering for granting me this study opportunity. Besides, my appreciation goes to the gameLAB which provides me with the excellent working environment and the high performance equipments.

I feel extremely fortunate to work with so many excellent current and former members in gameLAB. Especially, I would like to thank Dr. Yimin Wang, and Kai Wang for their help and concern in both my life and study. I also wish to thank my friends Hailing Zhou, Xiaoqun Wu, Yusha Li, William Fu, Xiaoqi Yan, Min Meng, Xianbin Liu, Jun Liu, Lei Huang, with whom I have shared some great time in Singapore.

Finally, I would like to express my deepest love to my wife and parents, who are the strongest backup for me. Without their persistent encouragement and support, this thesis could not have been possible.

Contents

Abstract	i
Acknowledgements	v
Table of Contents	vii
List of Figures	xii
1 Introduction	1
1.1 Background and motivation	1
1.2 Research problems	4
1.3 Research contribution	7
1.4 Thesis organization	9
2 Prior Work	11
2.1 NURBS	11
2.2 T-splines	13
2.2.1 Definition	13
2.2.2 T-spline Local Refinement	17
2.3 Spline manipulation	22
2.3.1 Problem of traditional methods	22
2.3.2 Direct manipulation	24
2.4 Energy minimization models	25
2.5 D-NURBS and Lagrangian dynamic models	27
2.6 Constrained parameterization	30

2.7	Shape morphing	35
3	Physically based Dynamic T-spline	39
3.1	Introduction	39
3.2	Physically-based dynamic T-splines	40
3.2.1	Dynamic T-spline surfaces	40
3.2.2	Physically-based motion for dynamic T-splines	42
3.2.3	Efficient PD-T-splines	45
3.3	Forces and Constraints	48
3.3.1	Applied forces	48
3.3.2	Constraints	51
3.4	Dynamic equation in unconstrained DOFs	53
3.5	Analytical solution	56
3.5.1	Modal analysis	57
3.5.2	State-space approach	59
3.5.3	Phase synchronization	60
3.5.4	Analytical solution	62
3.5.5	Modal reduction	63
3.6	Implementation	65
3.6.1	Extracting Bézier patches	65
3.6.2	Data structure	68
3.6.3	Calculation of elementary matrices	68
3.6.4	Dynamic update of local refinement	69
3.7	Applications	75
3.7.1	Solid rounding	75
3.7.2	Adaptive optimal T-spline fitting	77
4	Physically based Composite Curve for T-spline Editing	81
4.1	Introduction	81

4.2	Theory	83
4.2.1	Notations	83
4.2.2	Composite curve	84
4.2.3	Problem statement	87
4.2.4	Theorems	89
4.2.5	Solution	92
4.3	Regularized physically based dynamic composite curve	94
4.3.1	Problem of PBDC curve	95
4.3.2	Regularized PBDC curve	97
4.4	Curve-based T-spline manipulation	99
4.4.1	Handling curve creation	101
4.4.2	Target curve generation	101
4.4.3	Surface modification	102
4.5	Implementation	103
4.5.1	Finite element	104
4.5.2	Data structure for R-PBDC finite elements	106
4.5.3	Calculation of element matrices	107
4.5.4	Local update of T-spline refinement	108
4.6	Experimental results	111
4.6.1	Modeling a goblet from a cylinder	113
4.6.2	Feature based surface manipulation	114
4.6.3	Creating feature on a Surface	115
5	Foldover Free Reparameterization and Texture Mapping	119
5.1	Introduction	119
5.2	Fundamental Theory	122
5.2.1	Basic observations	123
5.2.2	Longest edge bisection	125

5.3	Algorithm for 2D Mesh Transformation with Positional Constraints	130
5.3.1	Preprocess	131
5.3.2	Construction of non-intersecting warping trajectories	132
5.3.3	Iterative RBF-based mapping	135
5.3.4	Foldover-free induced transformation	138
5.4	Experimental Results	142
5.5	Conclusions	148
6	Physically based T-spline Morphing with Speed Control	149
6.1	Introduction	149
6.2	Correspondence establishment	151
6.2.1	Problem statement	152
6.2.2	Consistent approximation	155
6.3	Vertex path	166
6.3.1	Linear Averaging	167
6.3.2	Physically based morphing	170
6.3.3	Extra force control	174
6.4	Experimental Results	181
6.4.1	More consistent approximation results	181
6.4.2	Eliminating small wiggles	183
6.4.3	Speed control	185
6.4.4	Morphing based modeling	186
6.4.5	Different rest poses	186
6.4.6	Insert key frames	188
7	Dynamic T-spline Rendering	193
7.1	Introduction	193
7.2	Problem statement	193
7.3	Our solution	195

7.3.1	Initial constrained mapping	196
7.3.2	Induced time-dependent map	197
7.4	Results	199
8	Conclusions and Future Work	203
8.1	Conclusions	203
8.2	Future work	204
	References	207
	Author's Publications	223

List of Figures

2.1	A NURBS curve and its control polygon.	12
2.2	B-spline basis functions	12
2.3	A NURBS surface and its control mesh.	13
2.4	A T-spline surface and its T-mesh.	14
2.5	Pre-image of T-mesh in parametric domain	15
2.6	Local refinement example	21
2.7	Original surface	22
2.8	Move control points.	22
2.9	Adjust weights.	23
2.10	Change the knot configuration	23
2.11	Texture mapping and constrained texture mapping.	31
3.1	Possible element types	66
3.2	Failure case by simply extending all the T-junction by two bays	67
3.3	Splitting the false Bézier patch	68
3.4	Another local refinement example. The squares with red boundary are the new inserted points	71
3.5	The support of the blending function associated with P_0 has no intersection with the Bézier patch S_1	71
3.6	the support of the blending function covers the Bézier patch, but the knot lines does not cut it.	72

3.7	the knot lines cut Bézier patch	72
3.8	The associated control point may no long influence the Bézier patch after refinement.	73
3.9	The weight changed case	74
3.10	Rounding a trihedral vertex	76
3.11	Rounding method in [122]	77
3.12	Rounding a bevel joint.	78
3.13	Illustration of adaptive T-spline fitting using PD-T-splines	79
4.1	A simple example of composite curve.(a) The purple curve represents the composite curve. The red control points in the original parent surface represents the ones that have influence on the composite curve, while the light blue ones are the others. (b) The red curve is the 2D curve in the parametric domain that defines the composite curve. And the influencing control points are colored red.	84
4.2	Composite curve reduces to a NURBS curve	86
4.3	A simple example of T-spline surface editing using general curve handle.	100
4.4	Illustration of extracting correct finite elements.(a) The 2D domain curve is a B-spline curve, where the knot points are highlighted in purple segments. In addition to the self knots, the intersection with the boundary of the Bézier patches in parent surface (the green segments) should also be considered. And the yellow dashed line indicates the treatment of extracting the Bézier patches in parent surface using the method described in Section 3.6.1. (b) When the 2D curve itself is sufficiently smooth, only the intersection should be concerned.	104
4.5	Modeling a Cylinder into a goblet	112
4.6	Feature based manipulation.	114
4.7	Produce 3D ‘C&G’ Logo on a T-spline surface	116

5.1	The proposed method	120
5.2	(a) A 2D mapping G and (b) its induced transformation G_M^*	122
5.3	G is locally bijective but G_M^* folds over.	124
5.4	G is not locally bijective, but G_M^* is foldover free.	124
5.5	The insertion of a Steiner point makes the mesh transformation foldover free.	124
5.6	The Propagation of foldover through one splitting.	125
5.7	The LEB process that starts from triangle t_0	126
5.8	Bound the 2D embedding with a virtual rectangular boundary.	133
5.9	Generation of non-intersecting warping trajectories. (a) The input mesh and constrained points. (b) The Delaunay triangulation on the constrained points. (c) and (d) The compatible triangulations of the source and target. (e) The warping trajectories. (f) Foldover-free mapping.	134
5.10	An example of constrained mapping that maps A, B and C to A^*, B and C , respectively.	140
5.11	(a) An input mesh with 8 constraints. The 4 corner vertices are constrained to remain unchanged and each of the 4 inner vertices moves to its next as indicated by the red arrows. (b) A foldover free mesh is obtained by our algorithm, which adds 13 Steiner points. (c) Using the algorithm of [139], we can obtain a locally bijective mapping G , which is depicted by the red curves in the zoomed window, but foldover occurs when the mapped edges are straightened.	143
5.12	Texturing models (a)-(c).	144
5.13	Texturing models (d)-(f).	145
5.14	Texturing models (g)-(i).	146
6.1	Input T-spline surfaces for morphing	151
6.2	Morphing without feature alignment. The transition is generated through linear interpolation of the surfaces in Figure 6.1	153

6.3	User specified feature correspondence	153
6.4	Adaptive tessellation, and the user specified features are incorporated into the tessellated meshes	157
6.5	Constrained reparameterization	159
6.6	Procedure of iterative consistent approximation	161
6.7	Consistent approximation result applied to the surfaces in Figure 6.1	167
6.8	Morphing with feature alignment. The transition is generated through linear interpolation of the surfaces in Figure 6.7.	168
6.9	Problem of linear averaging. The unwanted flatening is observed in this case	169
6.10	Physically based morphing without external force. The source and target surfaces are given in Figure 6.12, and the timestep is set to 0.1. During transition, the feature of the target surface quickly appear in the early stage.	172
6.11	Another input of T-spline surface	181
6.12	Consistent approximation result applied to the surfaces in Figure 6.1(a) and 6.11.	181
6.13	Consistent approximation result of restoring three T-spline surfaces in Fig- ure 6.1 and 6.11.	182
6.14	Physically based morphing keeps the wavy shapes, and small wiggles are eliminated.	184
6.15	Physically based morphing with force control. The initial speed is set to be 0, and the rest pose is the target state.	185
6.16	Morphing based modeling	186
6.17	Physically based morphing with rest post control.	187
6.18	Interpolate key frames using physically based morphing. The rest pose is set to the intermediate key frame.	189
6.19	Interpolate key frames using physically based morphing. The rest pose is set to the target surface.	190

7.1	Input of dynamic rendering	194
7.2	Initial constrained mapping	197
7.3	Example of foldover in the induced map from a piece-wise linear 2D map. .	198
7.4	Dynamic rendering	199
7.5	Paste a new figure	200
7.6	Another rendering example. The target frame is employed to compute the initial map.	202

Chapter 1

Introduction

1.1 Background and motivation

Generating a 3D smooth surface is a ubiquitous task in computer graphics and computer aided design. It concerns with mathematical description and object modification. A suitable mathematical representation should be efficient in modeling and rendering. T-splines [99][96] are a generalization of NURBS by permitting the existence of T-junctions. They possess truly local refinement enabling the modeler to focus on the region where details are called for. Meanwhile, they are also watertight [94]. It means that they can represent complex objects with fewer patches, which makes them more convenient for the subsequent processing and analysis. Furthermore, they are both forward and backward compatible with the current industrial standard NURBS. These three valuable capabilities make T-splines desirable for both geometric modeling and iso-geometric analysis.

Like NURBS, T-splines support interactive modification through tuning its defining parameters, i.e. moving control points, changing weights and adjusting knot configuration. However, those inherent deformation methods are essentially geometrically-based. It lacks precision between the defining parameters and the object itself, thus requiring the understanding of the underlining formulation. Alternatively, physically based approaches provide intuitive and convenient means for object definition and deformation. Within the physically

based framework for modeling, geometric primitives simulate physical objects with elastic properties imposition and mass distribution. The mechanics of motion under force defines the object deformation. That is the deformed shape is the solution to the corresponding continuum equilibrium equation. Currently, mass-spring models and Finite Element Methods(FEM) are the two dominant physically based modeling methods, which differ in how the simulated physical object is discretized. While in Mass-spring models objects are represented by a set of sampling mass points connected by springs, the objects within FEM become the assemblage of small elements, over which the equilibrium equation is approximated. Both of them achieved great success in physical simulation and animation. Especially with the development of advanced interaction machine, i.e. haptics, physically based modeling methods show their greater potential in object modeling. However, due to discrete nature, these physically based methods only gain popularity within mesh representations. As for splines, one may discretize them into meshes, onto which those physically based approaches may be applicable. Nevertheless, such indirect processing fails to make full use of the compact representation of splines themselves. As the deformed object should also be represented by splines, other approximation algorithms have to be performed to transform the meshes into splines. This lays extra heavy burden on such indirect approaches, because object approximation itself is a challenging problem in computer graphics, making the deformation much more uncontrollable.

Aiming towards developing more continuous physical representation, DNURBS were proposed by Qin et.al.[122], which made full use of the compact NURBS representation. Within the DNURBS scheme, NURBS primitives are similarly regarded as realistic physical objects with mass distribution and elastic properties imposition. Yet their defining parameters, control points and weights, rather than sample points on the objects serve as the generalized coordinates of the physical system. The user interacts with the objects through applying simulated force and local and global constraints. Accordingly, the objects deform and evolve in a physically realistic manner in response to the interaction following Lagrangian Dynamic. Due to the physical attribute, the deformation of NURBS objects

become much more intuitive and physically accurate. Compared with the previous physically based modeling approaches [8] [132], which are merely concerned about the final static equilibrium state, DNURBS also take the dynamic evolution into consideration. The dynamic process provides more physically realistic candidates for decision of the desired shape, hence accelerating the modeling process. Moreover, as dynamic evolution is crucial in a lot of applications such as computer animation and simulation, the inherent physically precise evolution makes DNURBS also suitable for their use. The practical results in [122] demonstrate its applicability in these fields.

Although the methodology in DNURBS really pulls the splines modeling techniques a step forward, it still fails to meet all the requirements for modeling purpose. To make it clear, we list some drawbacks according to the criteria that are frequently demanded in modeling:

- **Flexibility.** When modeling, details are progressively added by performing refinement. This can be achieved through knot insertion of NURBS to add more degrees of freedom in the region where details are needed. However, as NURBS lack truly local refinement ability, insertion of one extra point will be at the cost of increasing a whole row or column of points, hence significantly increasing the dimension of dynamic equation of motion. Since the numerical integration of the dynamic equation requires recalculation of the coefficient matrices at every time step, obviously the increased dimension will slow down the simulation.
- **Efficiency.** Interactive modeling within DNURBS is actually to find the most satisfactory intermediate state during the evolution governed by their dynamic equation. Hence their efficiency mainly depends on how the dynamic equation is solved. However, the current numerical integration method adopted is confronted with two serious problems. First, it only allows a small range of time steps in order to ensure accuracy. But for dynamic modeling, as the evolution is under force imposition, the speed will not be uniform. Hence the time step ought to be set adaptively in order to provide the

best visual feedback of the dynamic evolution. Apparently, this can not be satisfied by the numerical approach. As a result, the user is likely to miss lots of ideal shapes, or wait for a long time to view the equilibrium. Second, the evolution in numerical approach is unidirectional. It means that once we miss the best shape, evolution needs to be re-performed from the beginning. But for dynamic modeling, users usually have to repeatedly try and compare till finding the most satisfactory one. Therefore, confinement in one direction wastes a lot of time. To improve this, one may store every intermediate state in the memory, but since the time step is small, a lot of memory is required to store such huge data, most of which may not be even retrieved once throughout the evolution.

- Easy handling. Force indeed provides an effective way to meet such criteria, but it fails in several cases especially when curve features are involved. In current DNURBS, no tools are available to deal with curves lying on the surface, especially non-isoparametric curves, which have been proved to be rather effective in surface modification. For instance, when editing nose in a head model, the most intuitive way for the user is through editing the feature lines of the nose. However, with most of existing techniques, the user has to pull a lot of springs along those feature lines. This will cause ambiguity, because user has to make the choice of how many springs ought be attached. Besides, pulling so many springs in 3D tends to introduce noise into modeling process, making the modified shape unpredictable.

1.2 Research problems

This research is to investigate how to introduce physics into T-spline representation, and to explore its geometric processing algorithms. Specifically, we concentrate on the following subproblems:

The first subproblem is to formulate an efficient physically based model based on T-

splines. As T-splines generalize NURBS and solve most of the problems of NURBS, marrying T-splines with physics is a promising way to remedy the flexibility issue of D-NURBS. And Physics can be readily introduced into T-spline representation following the similar strategy as in D-NURBS. But replacing NURBS with T-splines can not avoid the highly non-linear dynamic equation, which is still problematic. As analyzed above, current numerical solvers cannot fully satisfy the modeling requirement. Then how to transform the model to yield an efficient solvable equation should be considered. Although T-splines possess the local refinement property, the dynamic equation for the refined surface should also be updated. Since the refinement is local, most of the generalized coordinates remain. Then it is obviously not economic to recalculate all those matrices from scratch. How to locally update those matrix entries is worth investigating, which could reduce the computational complexity.

The second subproblem is to enhance the modeling capability. As far as physically based model is concerned, the modeling tools are mainly point based. With the development of computerized sketching tools, curve handles for surface modification are becoming popular and are shown more effective and intuitive [75]. In spline domain, only a few works considered to use non-iso-curves for surface manipulation. They mainly focused on B-splines. Basically, they specify some curves on the B-spline surface as handles, sketch some spatial curves as the target curves, establish the matching of the shape handles and the target curves, and then deform the surface. As curve handles provide a much more convenient way for the user to express his manipulation intention, such curve-based methods improve the easy-handling of surface manipulation. However, when coming to T-splines, things become complicated. The computation of explicit relation of control points and weights between the composite curve and the T-spline surface is difficult and time consuming. The specification of a general spatial target curve through sketching is challenging. How to incorporate such curve handles into the physically based model also deserves to be investigated.

With efficient physically based model in hand, it's tempting to explore its processing

algorithms such as metamorphosing between two T-spline objects. Shape morphing refers to a process that automatically generate a smooth transition from a source to a target object, which can find wide applications in geometric modeling and animation. It is well known that it is difficult to measure the ‘naturalness’ of a transition, although some attempts have been made. Paper [36] let the user make the decision by providing the functionality to interactively adjust the morphing process. This is an obvious tedious task that requires the understanding of the underlining techniques. On the other hand, it’s reasonable to consider the motion following the physical laws to be the desirable one. As a merit, physically based morphing involves an inherent concept of speed. Speed control is desirable to maintain the continuity between the consecutive morphing sequences. However, extra control means reducing the system DOFs. How to introduce extra DOFs without diminishing the physical property is challenging. While physics can only be utilized to generate the vertex paths, it is well known that we should first set an inter-map between the input objects. As morphing is often employed to transform objects with similar features. Then the correspondence of those features should be preserved in the inter-map. Unlike the mesh representation, it’s the control points, weights and knot configuration that define a T-spline object. It means the initial T-spline representations have to be reparameterized to align those features, which leads to the problem of constrained mapping. Furthermore, in order to generate the transitive shape in terms of T-spline representation, we have to enforce the requirement that the reparameterized representation should have the same knot configuration. To satisfy those two requirements simultaneously makes the vertex correspondence issue for T-spline morphing also a challenging task.

In both physically based modeling and morphing processes, the outputs are dynamic evolution following physical laws. Especially for the morphing that is frequently employed in producing animation, it is desirable to implement the texture mapping for this evolution sequence. In the rendering pipeline, spline objects are usually tessellated into meshes. Thus this problem is essentially how to paste a single figure onto the dynamic sequence. In order to enhance the visual appearance, the important features should also be aligned. It is also

known the constrained mapping problem as in the previous issue. But here we have to establish a serial of such maps while satisfying the positional constraints, and speed should be placed in high priority, because it is desirable to achieve rendering along with dynamic evolution. [45] presented the state-of-art technique to address this problem. However, [45] involves a post non-linear smoothing processing, which is time consuming. So this method does not suit for our problems. An efficient and robust constrained mapping technique is demanded. As the sequential objects possess the similar features, establishing the map respectively for each frame is obviously not an economic approach. How to construct the desired map making use of the existing one is worth investigating.

1.3 Research contribution

To solve the above-mentioned problems, thorough research has been conducted and several novel techniques have been proposed and developed. Briefly, the major contributions of the thesis include:

- Physics is introduced into T-spline representation producing physically based dynamic T-splines (or PD-T-splines). In order to provide an explicit solution, we further propose efficient PD-T-splines (or E-PD-T-splines) by freezing the weights during evolution. A closed form solution is derived based on the physical perception that the vibration system can be considered as a coupling of individual vibrating modes. Such physical perception also allows us to develop modal reduction technique to further reduce computational complexity. Furthermore, we consider how to locally update the physical matrices by identifying the exactly changed entries, which avoids the duplicate computation.
- General curve handle is incorporated into PD-T-splines to enhance the easy-handling of the physically based model. The surface deformation is driven by interactively adjusting the handling curve, which is a composition between a 2D domain curve

and the T-spline surface. To facilitate curve modification, we propose the physically based dynamic composite (or PBDC) curve directly based on the original composite definition, which supports arbitrary curve types. Observing the risk of rank deficiency which could cause instability of the physically based model, we present a novel way to identify a basis for the composite blending functional space and compute the dependency. Then the regularized PDBC curve serves as a tool to interactive modify the handling curve in PD-T-splines, which is stable and efficient. Finally the surface deformation is achieved through introducing the curve force connecting the handling curve and the target curve into PD-T-splines.

- We propose a foldover free guaranteed constrained mapping technique, which is frequently encountered in the applications of T-spline morphing and dynamic rendering. We comprehensively analyze the relation between a C^2 continuous 2D mapping and its induced piecewise linear transformation, and propose a strategy to refine the mesh such that if the C^2 mapping is locally bijective, its induced transformation on the refined mesh keeps the orientation of each triangle of the refined mesh. An iterative RBF-based warping scheme is proposed and an explicit formula for the stepsize of the warping is derived, which can assure the corresponding RBF-based mapping to be locally bijective. An effective, provably foldover free algorithm for smooth mesh warping with hard constraints is presented, which consists of preprocess, construction of non-intersecting warping trajectories, iterative RBF-based mapping and foldover free induced transformation.
- We present a novel solution to the problem of T-spline morphing. The correspondence issue is resolved through consistent approximation, which fulfills the feature alignment and structural compatibility simultaneously. Physics is also introduced into morphing process to generate plausible vertex path, producing physically based morphing. To enable speed control, a novel approach to introduce extra DOFs into a physical system is proposed by incorporating external force in terms of T-spline vol-

ume, whose control points serve as the desired DOFs. Our approach dose not violate the analytical solvability of the resultant physical system.

1.4 Thesis organization

The rest of the thesis is organized as follows:

- Chapter 2 reviews the related prior works on spline representations, and their various deformation methods. We also review the constrained mapping and morphing techniques, which are required in the processing of our physical models.
- Chapter 3 first introduces the physically based dynamic T-spline. Then we derive the closed form solution for an efficient version of PD-T-spline from physical point of view. Local update of the dynamic equation is also investigated along with T-spline refinement.
- Chapter 4 first presents the theory on the relation between the dimension of a functional space and the rank of its related matrix integral. Then based on this theory, we develop the regularized physically based dynamic composite curve to interactively adjust the handling curve on T-spline surface for surface editing.
- Chapter 5 first presents the theory on the relation between a continuous 2D map and its induced piecewise linear transformation. Then a foldover free guaranteed constrained reparamterization method is proposed based on this theory.
- Chapter 6 presents a comprehensive solution to T-spline morphing. Vertex correspondence is accomplished through consistent approximation, and the vertex path is generated by the evolution of PD-T-spline.
- Chapter 7 examines the application of constrained mapping techniques in dynamic rendering.

- Chapter 8 concludes the thesis and points out potential future works.

Chapter 2

Prior Work

This chapter provides some background and reviews previous works related to NURBS-based geometric modeling, dynamic NURBS, constrained mapping and shape morphing.

2.1 NURBS

Non-uniform Rational B-Splines (or NURBS for short) are a flexible and powerful mathematical representation for curves and surfaces, which are widely used in computer graphics and computer aided design. They can represent both free form shapes and analytical curves or surfaces such as circles, spheres and cylinders in a unified form. Since 1990s, NURBS have become an industry standard in CAD/CAM applications.

A NURBS curve of degree $k - 1$ can be expressed as

$$\mathbf{C}(u) = \frac{\sum_{i=0}^n B_{i,k}(u)\omega_i \mathbf{P}_i}{\sum_{i=0}^n B_{i,k}(u)\omega_i} \quad (2.1)$$

where $\mathbf{P}_i$ represent its control points, ω_i are the weights, and $B_{i,k}(u)$ are B-spline basis functions. The control points form the control polygon. ANURBS-curve is displayed in Figure 2.1, where the white squares denote the control points.

Figure 2.1: A NURBS curve and its control polygon.

Figure 2.2: B-spline basis functions

The B-spline basis functions $B_{i,k}(u)$ are defined over a non-decreasing knot sequence $\{t_0, t_1, \dots, t_{n+k}\}$ and their equations can be written recursively by

$$B_{i,k}(u) = \frac{u - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(u) + \frac{t_{i+k} - u}{t_{i+k} - t_{i+1}} B_{i+1,k-1}(u) \quad (2.2)$$

with the initial setting

$$B_{i,1}(u) = \begin{cases} 1 & t_i \leq u < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \quad (2.3)$$

Figure 2.2 shows a plot of basis functions of degree 3. The red vertical short lines indicate the knots. The B-spline basis functions have several good properties such as local support, positivity and normalization, which result in a close relationship between a NURBS curve and its control polygon. As shown in Figure 2.1, the control polygon roughly indicates the shape of the curve it defines.

Figure 2.3: A NURBS surface and its control mesh.

Similarly, a NURBS surface is defined by

$$\mathbf{S}(u, v) = \frac{\sum_{i=0}^m \sum_{j=0}^n B_{i,k}(u) B_{j,l}(v) \omega_{i,j} \mathbf{P}_{i,j}}{\sum_{i=0}^m \sum_{j=0}^n B_{i,k}(u) B_{j,l}(v) \omega_{i,j}} \quad (2.4)$$

where the control points $\mathbf{P}_{i,j}$ form a control mesh and each weight $\omega_{i,j}$ is associated with control point $\mathbf{P}_{i,j}$. The B-spline basis functions $B_{i,k}(u)$ are defined over the knot sequence $\{u_0, u_1, \dots, u_{m+k}\}$ along the u axis, and $B_{j,l}(v)$ is defined over $\{v_0, v_1, \dots, v_{n+l}\}$ along the v axis. Figure 2.3 shows a NURBS surface and its control mesh.

2.2 T-splines

2.2.1 Definition

T-splines generalize the tensor-product NURBS surfaces by allowing the existence of partial rows or columns of control points [99, 96]. Like a NURBS surface, a T-spline surface is constructed from a control grid called T-mesh, which consists of rows and columns of

control points. Each control point is associated with a weight. The final control point in a partial row or column is a T-junction. Figure 2.4 shows a T-spline surface and its T-mesh where the T-junctions are highlighted in yellow. Compared to the control net of a NURBS surface, the structure of a T-mesh is more flexible and meanwhile more complicated.

Figure 2.4: A T-spline surface and its T-mesh.

T-mesh itself is not sufficient to define a T-spline surface. It must be provided with the knot configuration. Knot information can be expressed using knot intervals indicating the difference between two knots and assigned to each edge in the T-mesh. Figure 2.5 shows the pre-image of a part of a T-mesh in (u, v) parameter space, where the d_i and e_i denote the knot intervals along the horizontal and vertical directions, respectively. A valid knot configuration should satisfy the constraint that the sum of all knot intervals along one side of any face must equal the sum of the knot intervals on the opposing side. For example, in Figure 2.5 on face F_1 , $e_6 + e_7 = e_3$, and on face F_2 , $d_{10} + d_{11} = d_{14} + d_{15}$. In order to derive an explicit formula for a T-spline surface, we impose a local knot coordinate system on a T-mesh, and the knot coordinates are inferred from the knot intervals. We first choose a control point whose pre-image serves as the knot origin $(u, v) = (0, 0)$ for the

Figure 2.5: Pre-image of T-mesh in parametric domain

parameter domain. For the example in Figure 2.5, $\mathbf{P}_0$ is designated to be the knot origin. We then assign a u knot value to each vertical edge in the pre-image of T-mesh and a v knot value to each horizontal edge in the pre-image of T-mesh. Then each control point has a 2D coordinate. Based on our choice of knot origin, we have $u_0 = v_0 = 0$, $u_1 = d_1$, $u_2 = d_1 + d_2$, $u_3 = d_1 + d_2 + d_3$, $v_1 = e_1$, $v_2 = e_1 + e_2$, and so forth. In this way, each control point gets a pair of knot coordinates. For example in Figure 2.5, based on our choice of knot origin, the knot coordinates for $\mathbf{P}_1$ are $(u_2, v_2 + e_3)$, and for $\mathbf{P}_2$ are $(u_1 + d_7 + d_9, v_3)$.

Based on the T-mesh and its knot information, the explicit equation of a T-spline surface with $n + 1$ control points can be written as

$$\mathbf{T}(u, v) = \frac{\sum_{i=0}^n B_i(u, v) \omega_i \mathbf{P}_i}{\sum_{i=1}^n B_i(u, v) \omega_i} \quad (2.5)$$

or an equation in homogeneous form:

$$\mathcal{T}(u, v) = \sum_{i=0}^n B_i(u, v) (\omega_i \mathbf{P}_i, \omega_i) \quad (2.6)$$

where $\mathbf{P}_i = (x_i, y_i, z_i)$ are control points and ω_i are associated control weights. The $B_i(u, v)$ in (2.5) or (2.6) are called blending functions and are given by

$$B_i(u, v) = N[\mathbf{u}_i](u)N[\mathbf{v}_i](v) \quad (2.7)$$

where $N[\mathbf{u}_i](u)$ and $N[\mathbf{v}_i](v)$ are the cubic B-spline basis functions associated with the local knot vectors

$$\mathbf{u}_i = [u_{i0}, u_{i1}, u_{i2}, u_{i3}, u_{i4}] \quad (2.8)$$

and

$$\mathbf{v}_i = [v_{i0}, v_{i1}, v_{i2}, v_{i3}, v_{i4}] \quad (2.9)$$

respectively. Each of these basis functions is a piecewise polynomial given explicitly by

$$N[\mathbf{u}_i](u) = \begin{cases} \frac{(u-u_{i0})^3}{(u_{i1}-u_{i0})(u_{i3}-u_{i0})(u_{i2}-u_{i0})}, & u_{i0} < u \leq u_{i1} \\ \frac{(u-u_{i0})^2(u_{i2}-u)}{(u_{i2}-u_{i1})(u_{i3}-u_{i0})(u_{i2}-u_{i0})} + \frac{(u_{i3}-u)(u-u_{i0})(u-u_{i1})}{(u_{i2}-u_{i1})(u_{i3}-u_{i1})(u_{i3}-u_{i0})} + \frac{(u_{i4}-u)(u-u_{i1})^2}{(u_{i2}-u_{i1})(u_{i4}-u_{i1})(u_{i3}-u_{i1})}, & u_{i1} < u \leq u_{i2} \\ \frac{(u-u_{i0})(u_{i3}-u)^2}{(u_{i3}-u_{i2})(u_{i3}-u_{i1})(u_{i3}-u_{i0})} + \frac{(u_{i4}-u)(u_{i3}-u)(u-u_{i1})}{(u_{i3}-u_{i2})(u_{i4}-u_{i1})(u_{i3}-u_{i1})} + \frac{(u_{i4}-u)^2(u-u_{i2})}{(u_{i3}-u_{i2})(u_{i4}-u_{i2})(u_{i4}-u_{i1})}, & u_{i2} < u \leq u_{i3} \\ \frac{(u_{i4}-u)^3}{(u_{i4}-u_{i3})(u_{i4}-u_{i2})(u_{i4}-u_{i1})}, & u_{i3} < u \leq u_{i4} \\ 0, & u < u_{i0} \text{ OR } u > u_{i4} \end{cases} \quad (2.10)$$

The local knot vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ are inferred from the T-mesh neighborhood of $\mathbf{P}_i$. The rule whereby knot vectors $\mathbf{u}_i$ of (2.8) and $\mathbf{v}_i$ of (2.9) for the blending function $B_i(u, v)$ corresponding to $\mathbf{P}_i$ are determined is as follows. Suppose the knot coordinates of $\mathbf{P}_i$ are $(u_{i2} + \alpha, v_{i2})$. Consider a ray in parameter domain $\mathbf{R}(\alpha) = (u_{i2} + \alpha, v_{i2})(t > 0)$. Then u_{i3} and u_{i4} are defined as the u coordinates of the first two vertical line segments of constant u intersected by the ray. The other knots in $\mathbf{u}_i$ and $\mathbf{v}_i$ are found similarly. Take $\mathbf{P}_1$ and $\mathbf{P}_3$ in

Figure 2.5 for example. The knot vectors for $\mathbf{P}_1$ are $\mathbf{u}_1 = [u_1, u_1 + d_7, u_2, u_3, u_4]$ and $\mathbf{v}_1 = [v_1, v_2, v_2 + e_6, v_3, v_4]$ as visualized using blue color. For $\mathbf{P}_3$, $\mathbf{u}_3 = [u_2 + d_{11}, u_3, u_4, u_5, u_6]$ and $\mathbf{v}_3 = [v_1, v_2 + e_6, v_3, v_4, v_5]$ as visualized using purple color. Once these knot vectors are determined for each basis function, the T-spline surface is defined using (2.5) and (2.7).

2.2.2 T-spline Local Refinement

Local refinement plays an important role in progressive modeling, which can provide extra degrees of freedom. It is usually achieved by knot insertion. However, For a NURBS surface, inserting one knot will result in propagating an entire row or column of control points, which significantly introduces superfluous degrees of freedom. On the other hand, the existence of T-junction in T-splines allows local refinement. That is the insertion of one new point will result in only a small number of extra control points.

Blending Function Refinement

Basically, T-spline local refinement is directly resulted from blending function refinement. Therefore, we first introduce the B-spline basis function refinement.

If $\mathbf{u} = [u_0, u_1, u_2, u_3, u_4]$ is a knot vector and $\tilde{\mathbf{u}}$ is a knot vector with m knots with $\mathbf{u}$ as its subsequence, then the B-spline basis defined on $\mathbf{u}$, $N[u_0, u_1, u_2, u_3, u_4]$ can be expressed as a linear combination of $m - 4$ B-spline bases functions defined over the substrings of length 5 $\tilde{\mathbf{u}}$.

In the case of $m = 6$, all the possible refinement relations can be expressed as follows:

If $\mathbf{s} = [u_0, u_1, u_2, u_3, u_4]$, $N(u) = N[u_0, u_1, u_2, u_3, u_4](u)$, and $\tilde{\mathbf{u}} = [u_0, k, u_1, u_2, u_3, u_4]$ then

$$N(u) = c_0 N[u_0, k, u_1, u_2, u_3](u) + d_0 N[k, u_1, u_2, u_3, u_4](u) \quad (2.11)$$

where $c_0 = \frac{k-u_0}{u_3-u_0}$ and $d_0 = 1$.

$$\text{If } \tilde{\mathbf{u}} = [u_0, u_1, k, u_2, u_3, u_4],$$

$$N(u) = c_1 N[u_0, u_1, k, u_2, u_3](u) + d_1 N[u_1, k, u_2, u_3, u_4](u) \quad (2.12)$$

where $c_1 = \frac{k-u_0}{u_3-u_0}$ and $d_1 = \frac{u_4-k}{u_4-u_1}$.

$$\text{If } \tilde{\mathbf{u}} = [u_0, u_1, u_2, k, u_3, u_4],$$

$$N(u) = c_2 N[u_0, u_1, u_2, k, u_3](u) + d_2 N[u_1, u_2, k, u_3, u_4](u) \quad (2.13)$$

where $c_2 = \frac{k-u_0}{u_3-u_0}$ and $d_2 = \frac{u_4-k}{u_4-u_1}$.

$$\text{If } \tilde{\mathbf{u}} = [u_0, u_1, u_2, u_3, k, u_4],$$

$$N(u) = c_3 N[u_0, u_1, u_2, u_3, k](u) + d_3 N[u_1, u_2, u_3, k, u_4](u) \quad (2.14)$$

where $c_3 = 1$ and $d_3 = \frac{u_4-k}{u_4-u_1}$. If $k \leq u_0$ or $k \geq u_4$, $N(u)$ does not change.

Note that for those cases $m \geq 6$, the refinement equations can be found by repeatedly applying the above relations.

Referring to (2.7), a T-spline blending function is just the multiplication of two B-spline basis functions. Then knot insertion in either direction will split the blending function into two scaled ones that sum to it. Further insertion will result in a set of scaled blending functions that also sum to the initial one.

Local Refinement Algorithm

T-spline local refinement refers to inserting one or more control points into a T-mesh while keeping the shape of the T-spline surface. As inserting a control point is accompanied by knot insertion into the neighboring blending functions, it is also named local knot insertion.

The local refinement algorithm is composed of two phases: topology phase and the geometry phase. The topology phase determines which control points must be inserted into the mesh in addition to the one specified. For the geometric phase, after inserting the additional control points, the T-mesh becomes a new one. Since it is resulted from knot insertion into the original T-mesh, according to above discussion the old blending function can be expressed as a linear combination of the new one,

$$\mathbf{B}_i(u, v) = \sum_{j=0}^{\tilde{n}} c_j^i \tilde{\mathbf{B}}_j(u, v) \quad (2.15)$$

The T-spline surface can be represented using the new Blending functions as well as the new control points,

$$\mathcal{T}(u, v) = \sum_{j=0}^{\tilde{n}} \tilde{\mathbf{B}}_j(u, v) \tilde{\mathbf{P}}_j \quad (2.16)$$

Comparing (2.16) and (2.6), and using the relation (2.15), we have

$$M_{1,2} \mathbf{P} = \tilde{\mathbf{P}} \quad (2.17)$$

where $\mathbf{P}$ and $\tilde{\mathbf{P}}$ are the columns of old and new homogeneous control points, and the row i and column j element of matrix $M_{1,2}$ is c_i^j in (2.15). In the following, we focus on explaining the topological phase.

The main idea of the algorithm is to gradually correct the blending functions that are not compatible with the new T-mesh. During this procedure, there are 3 types of possible violations:

- **Violation 1:** A blending function is missing a knot inferred from the current T-mesh.
- **Violation 2:** A blending function has a knot that can not be inferred from the current T-mesh.
- **Violation 3:** A control point has no blending function associated with it.

The algorithm tries to resolve all those violations one by one until no violation exists.

The details consist of four steps:

- (1) Insert all desired control points into the T-mesh.
- (2) If any blending function is guilty of violation 1, perform necessary knot insertions into that blending function.
- (3) If any blending function is guilty of Violation 2, add an appropriate control point into the T-mesh.
- (4) Repeat Steps 2 and 3 until there are no more violations.

An example is shown to explain how this algorithm works. In Figure 2.6(a), a new control points $\mathbf{P}_2$ is intended to insert into the initial T-mesh. After simply inserting it into the T-mesh (Figure 2.6(b)), some violations occur. Those blending functions centered at (u_1, v_3) , (u_2, v_3) , (u_4, v_3) , and (u_5, v_3) become guilty of Violation 1. To resolve those violations, a knot at u_3 should be inserted into these blending functions using the discussion in last section. Take the blending function centered at (u_2, v_3) for example, which is $N[u_0, u_1, u_2, u_4, u_5](u)N[v_1, v_2, v_3, v_4, v_5](v)$. After inserting the knot $u = u_3$ into the U knot vector, the blending function is split into two scaled blending functions: $c_2N[u_0, u_1, u_2, u_3, u_4](u)N[v_1, v_2, v_3, v_4, v_5](v)$ (Figure 2.6(c)) and $d_2N[u_1, u_2, u_3, u_4, u_5](u)N[v_1, v_2, v_3, v_4, v_5](v)$ (Figure 2.6(d)) using the relation in last section.

The blending function $N[u_0, u_1, u_2, u_3, u_4](u)N[v_0, v_1, v_2, v_3, v_4](v)$ is compatible with current T-mesh. However, the V knot vector of the blending function $N[u_1, u_2, u_3, u_4, u_5](u)N[v_0, v_1, v_2, v_3, v_4](v)$ shown in Figure 2.6(d) is guilty of Violation 2. Because current T-mesh does not infer the knot v_3 . This can not be resolved by blending function refinement. An additional control point $\mathbf{P}_3$ is needed to insert into T-mesh. But it creates new cases of Violation 1. The U knot vector of the blending function centered at (u_2, v_2) does not include u_3 as inferred from the current T-mesh. It can be easily fixed by inserting u_3 into the blending function. As a result, no violation exists and all the blending functions become compatible with the current T-mesh, thus finishing the local refinement algorithm.

Figure 2.6: Local refinement example

2.3 Spline manipulation

2.3.1 Problem of traditional methods

Figure 2.7: Original surface

The traditional approaches to modify the T-spline surfaces are to tune its defining parameters

Figure 2.8: Move control points.

- Moving control points. Figure 2.8 shows the deformation result from the surface in

Figure 2.7 by moving the control points highlighted in yellow color. It is seen that the surface deforms in the direction where the control points move.

Figure 2.9: Adjust weights.

- Adjusting weights. In Figure 2.9, we adjust the weights associated with the yellow points to a large number based on the shape in Figure 2.8. And the surface gets close to these two points.

(a) Resultant T-spline surface

(b) Changed pre-image of T-mesh

Figure 2.10: Change the knot configuration

- Changing knot configuration. Figure 2.10 illustrates the result by changing the knot

information based on the surface in Figure 2.8. It is inferred that knot configuration can produce significant shape variation as well.

The above three examples show that the defining parameters do provide some hints for the surface modification, but modeling through directly adjusting them is not so effective in practice. In particular, they suffer the following problems [122]:

- Achieving a certain design requires modification of several control points, weights or even knots simultaneously, which is difficult and tedious. Modifying many DOFs at the same time requires expertise. Moreover, geometric design should be shape oriented rather than control points/weights oriented.
- Although control points, weights, or knots provide hints for shape modification, they still lack precision to some extent. It is difficult or even impossible for users to make the unique correct decision. For example, sometimes moving control points and changing weights may result in similar shape modification. In that case, the user may get confused in making the “best” choice.
- Typically geometric design requires the result shape satisfying both local and global criteria. For instance, the result surface should be smooth or fair (global) and meanwhile interpolates several cross-section curves (local). It is difficult for traditional NURBS-based methods to achieve both local and global constraints.

2.3.2 Direct manipulation

To remedy the above-mentioned drawbacks, a lot of efforts have been made towards developing more convenient and intuitive methods for modifying spline curves and surfaces. One of the approaches for this purpose is direct manipulation, which first specifies the desired shape’s features such as point, curve or derivative constraints and then automatically converts these features into modification of corresponding shape parameters or degrees of freedom (DOFs). In [76, 77], Piegl practised this idea to adjust the shape of NURBS curves

and surfaces, in which only one control point or weight was allowed to change. Fowler and Bartels [22] proposed an approach to adjust the shape according to user specified curve's location and derivatives. Au et.al. [5] and Sánchez-Rayes [91] proposed to achieve direct manipulation through altering control points and weights simultaneously. By constructing a displacement function, Ishida [35] developed an algorithm that enables arbitrary and direct shape modification. Based on the knot refinement and knot removal algorithms, Zheng et.al. [143] presented a novel approach for local curve manipulation. Generally, these direct manipulation methods are mainly concentrated on incorporating as many useful features as possible. When converted into NURBS DOFs, the system is most likely to be over-determined, which means the DOFs are insufficient. Then a common strategy in these methods is to derive an optimal solution in the least square sense. Hence, direct manipulation can be solved efficiently, which may even enable an explicit solution as in [76, 77]. On the other hand, however, the direct manipulation often means local modification and only a few DOFs around the features will be involved. As shown in [32], when too few DOFs serve for modification, the final results tend to be unnatural or even produce topological change. Therefore, further consideration has to be taken in order to improve the performance of these essentially constraint-based approaches.

2.4 Energy minimization models

Due to the pioneer works of Terzopoulos et.al. [121, 119, 123, 120, 124, 116] in introducing deformable models into computer graphics, many efforts were also made to marry physics with spline modeling, which produce physically based modeling techniques. Under physically based schemes, spline objects are no longer a purely geometric representation, but simulate real objects with stretching and bending stiffness imposition. Users interact with objects through synthesized force, to which the objects tend to respond in a predictable and natural way. In this way, both global and local constraints can also be incorporated for precise modification. When only static equilibrium is considered, the corresponding

physically-based approach is also known as energy minimization, which is characterized as that the modeling problem is eventually reduced to an optimization problem subjected to certain constraints. The objective functional is actually some physical energy. Among those popular objective functionals, thin-plate energy, which contains stretching and bending terms, is the most intensively used energy form due to its simplicity and good results. Celniker et al. [7] pioneered to develop an interactive modeling system based on this energy, in which the minimization problem was solved by Ritz element method [38]. Their method was applied by McNerney et al. [68] to fit a deformable model to 3D data. Also making use of thin-plate energy, Celniker and Welch [8] investigated the deformable B-splines with linear constraints. Due to lack of local refinement in B-spline surfaces, Welch and Witkin [132] extended the idea to trimmed hierarchical B-splines, where different constrained methods were considered. Hu et.al. [32] discussed manipulating NURBS surfaces through minimizing the thin-plate energy with extra normal control. Meanwhile through highlight line visualization, they also claimed that the energy minimization methods produced smoother results than direct manipulation. Liu et.al.[54, 53] presented new methods to incorporate partial derivatives of an arbitrary order into the energy minimization models. When dealing with NURBS, weights are also of great importance. Pourazady and Xu [78, 79] provided a way to guide weights variation, which accomplished the minimization in the homogeneous space. As for modeling purpose, local modification sometimes is also desired, Pusch and Samavati [83] investigated an approach that enabled local adjustment within the energy minimization approach. In addition to thin-plate functional, many also took the functionals based on purely geometric information, i.e. curvature and its variation. In the early days, such functional was often adopted for fairing spline objects [56, 72, 27, 71, 18, 28, 29, 30]. In[135], Xie and Qin applied variety of such functionals to automatically determine the knot configuration of NURBS curves and surfaces.

2.5 D-NURBS and Lagrangian dynamic models

In addition to energy minimization that concentrates on the final static equilibrium, there exists another attractive category of physically based modeling techniques. They are based on Lagrangian dynamic models, which take both the final state and the transitive evolution into consideration. Within Lagrangian dynamic models, the purely geometric object is imposed with mass distribution and damping effects as well as the material properties. Accordingly, as a physical object, it follows physical laws, i.e. Newton-Lagrangian equation.

Celniker and Welch in their early work [8] suggested the use of such a dynamic approach. Their method was simple, which only utilized an identity matrix as the mass and damping matrix. Hence, the dynamic process was not physically realistic. In 1994, a more systematic and sophisticated Lagrangian dynamic modeling system, D-NURBS [122], was proposed based on the NURBS representation. A D-NURBS surface is defined as

$$\mathbf{S}(u, v, t) = \frac{\sum_{i=0}^m \sum_{j=0}^n B_{i,k}(u) B_{j,l}(v) \omega_{i,j}(t) \mathbf{P}_{i,j}(t)}{\sum_{i=0}^m \sum_{j=0}^n B_{i,k}(u) B_{j,l}(v) \omega_{i,j}(t)} \quad (2.18)$$

Compared with (2.4), the control points and weights become the function of time t , which are also referred to generalized coordinates in a physical system. Note that it is the variation of those generalized coordinates that produce the dynamic evolution of the D-NURBS surface. As a physics system, those generalized coordinates have to follow the Lagrangian dynamic equation:

$$\mathbf{M}\ddot{\mathbf{P}} + \mathbf{D}\dot{\mathbf{P}} + \mathbf{K}\mathbf{P} = \mathbf{f}_p - \mathbf{I}\dot{\mathbf{P}} \quad (2.19)$$

where $\mathbf{P}$ refers to the concatenation of control points and weights:

$$\mathbf{P} = [\mathbf{P}_{0,0}^T, \omega_{0,0}, \mathbf{P}_{0,1}^T, \omega_{0,1}, \dots, \mathbf{P}_{m,n}^T, \omega_{m,n}]^T,$$

$\mathbf{M}$, $\mathbf{D}$, $\mathbf{K}$ and $\mathbf{f}_p$ represent the mass, damping, stiff matrices and generalized force for an

Lagrangian system respectively. Specifically,

$$\begin{aligned}
 \mathbf{M} &= \iint \mu \mathbf{J}^T \mathbf{J} du dv \\
 \mathbf{D} &= \iint \gamma \mathbf{J}^T \mathbf{J} du dv \\
 \mathbf{f}_p &= \iint \mathbf{J}^T \mathbf{f} du dv \\
 \mathbf{I} &= \iint \mu \mathbf{J}^T \dot{\mathbf{J}} du dv
 \end{aligned} \tag{2.20}$$

Here the matrix $\mathbf{J}$ is the Jacobian matrix of surface $\mathbf{S}$ with respect to the generalized coordinates $\mathbf{P}$. The expression of $\mathbf{J}$ is

$$\mathbf{J}(u, v, t) = [\mathbf{B}_{0,0}, \mathbf{W}_{0,0}, \mathbf{B}_{0,1}, \mathbf{W}_{0,1}, \dots, \mathbf{B}_{m,n}, \mathbf{W}_{m,n}] \tag{2.21}$$

where $\mathbf{B}_{i,j}$ is a 3 by 3 diagonal matrix with the diagonal entries

$$N_{i,j}(u, v, t) = \frac{\partial \mathbf{S}}{\partial \mathbf{P}_{i,j}} = \frac{B_{i,k}(u) B_{j,l}(v) \omega_{i,j}}{\sum_{c=0}^m \sum_{d=0}^n B_{c,k}(u) B_{d,l}(v) \omega_{c,d}}$$

and $\mathbf{W}_{i,j}$ is the partial derivative of $\mathbf{S}$ with respect to $\omega_{i,j}$,

$$\mathbf{W}_{i,j}(u, v, t) = \frac{\partial \mathbf{S}}{\partial \omega_{i,j}} = \frac{\sum_{c=0}^m \sum_{d=0}^n B_{c,k}(u) B_{d,l}(v) \omega_{c,d} (\mathbf{P}_{i,j} - \mathbf{P}_{c,d})}{\left(\sum_{c=0}^m \sum_{d=0}^n B_{c,k}(u) B_{d,l}(v) \omega_{c,d} \right)^2}$$

Referring to (2.19), we can infer that D-NURBS offers a unified way to guide the variation of both control points and weights, which is apparently superior to most of the direct manipulation and energy minimization approaches. In addition, the time t actually provides another parameter for modeling. It means that the modified shape not only is confined in the equilibrium state, but also lies in the intermediate transition. This is extremely useful when the system is over constrained, which means the number of constraints exceeds

that of DOFs. Furthermore, as D-NURBS can inherently produce dynamic evolution, it is extremely attractive for simulation and animation, especially when physics is involved.

After the establishing the dynamic equation that governs the motion of all the generalized coordinates, the performance of D-NURBS mainly relies on how efficiently its dynamic equation can be solved. Originally in [122], finite element and implicit integration methods were applied to numerically solve (2.19). First, the derivatives of the generalized coordinates $\mathbf{P}$ are discretized over time steps Δt .

$$\ddot{\mathbf{P}}^{(t+\Delta t)} = \frac{\mathbf{P}^{(t+\Delta t)} - 2\mathbf{P}^{(t)} + \mathbf{P}^{(t-\Delta t)}}{\Delta t^2}$$

$$\dot{\mathbf{P}}^{(t+\Delta t)} = \frac{\mathbf{P}^{(t+\Delta t)} - \mathbf{P}^{(t-\Delta t)}}{2\Delta t}$$

Then substituting them into (2.19), we get the discrete dynamic equation governing the motion

$$(4\mathbf{M} + 2\Delta t\mathbf{D} + 4\Delta t^2\mathbf{K}) \mathbf{P}^{(t+\Delta t)} = 4\Delta t^2\mathbf{f}_p + 8\mathbf{M}\mathbf{P}^{(t)} - (3\mathbf{M} - 2\Delta t)\mathbf{P}^{(t-\Delta t)} - \iint \mu \mathbf{J}^T \mathbf{S}^{(t-\Delta t)} \quad (2.22)$$

This equation shows that the state at time $t + \Delta t$ is integrated by the previous two states t and $t - \Delta t$. It is claimed that such a method can achieve interactive rate.

Lagrangian dynamic model is also extended to other geometric representations. Qin et.al. generalized the idea to swung NURBS [86] for cross-sectional design, triangular NURBS [85] and hierarchical NURBS [142]. While the conventional NURBS are difficult to modeling arbitrary topology shapes, subdivision surface were developed to remedy this drawback. As subdivision surfaces are generalization of parametric surfaces, efforts were also made to marry subdivision surfaces to physically based techniques. Qin et al. [84] developed the dynamic scheme for Catmull-Clark subdivision surfaces, which showed the great potential in modeling and 3D reconstruction. In this work, a recursively subdivision approach was used to evaluate the internal energy. Later, after Stam [110] [109] gave nice

local parameterizations of Catmull-Clark and loop subdivision surfaces, Qin et al. [88, 87] applied his parameterizations to calculate the internal energy efficiently and precisely. A similar idea was extended to Butterfly subdivision [62]. Yet, Butterfly subdivision doesn't have the close-form expression even for a regular control mesh. Alternatively, a patch based finite element method was presented and a mass-spring system was used to model the internal elastic energy. [61] demonstrated that such a finite element method could be generalized to any subdivision surfaces. As the development of non-uniform subdivision scheme [100], Xie and Qin [134] developed a dynamic modeling framework based on it. These dynamic extensions of subdivision surfaces have successfully found applications in 2D and 3D data reconstruction [60, 63, 64]. In addition to the parametric and semi-parametric representations, PDE is also a popular primitive in geometric modeling. Their dynamic generalization [13, 14] also inherited both the advantages of PDE and physics.

In addition to 1D curve and 2D surface cases, Lagrangian dynamic models were also applied to 3D solid modeling and simulation. Lenoir et al. [46] made use of the refinement property of NURBS to simulate the thread knots. By adding the twisting energy to the deformable 1D objects, Theetten et al. [126, 125] proposed the geometrically exactly dynamic splines, which produced very accurate results. They have been applied in virtual reality applications such as cable positioning in a car. With rapid development of 3D interface and input/output devices, haptic has become a powerful device interacting with computer. Based on haptic devices, a lot of dynamic modeling systems were established [10, 11, 67, 66, 141] to facilitate the modeling process.

2.6 Constrained parameterization

Mesh parameterization is the problem of computing a one-to-one map between two 3D triangular surfaces with the same topology. It can be found wide applications in computer graphics including texture mapping, remeshing and morphing. In the past two decades, many methods have been proposed targeting diverse parameter space and focusing on dif-

ferent geometric properties. The common goal is to find a bijective map that preserves the geometric properties of the original mesh as much as possible, e.g., area, angle and length. [21] and [105] gave a nice review in this area. According to whether the boundary serves as part of the solution, these methods can be categorized into fixed boundary[15][19][20] and free boundary [92][48][104]. Generally speaking, boundary-free approach introduces significantly less distortion than the other one.

It is worth pointing out that mesh parameterization is also an important tool in numerous real-world engineering applications. For instance, garment modeling and metal forming or forging require the rapid computation of planar patterns to form the desired 3D shapes. In these cases, reasonability and availability of design are very important, so material properties of products should be taken into consideration, which motivates the energy based flattening methods[128][145][65][49]. Within these methods, models are first segmented into nearly developable charts, and these charts are then parameterized in the plane[105]. Compared with the purely geometric methods, they are more suitable for parameterizing objects with complex and invalid topology[49]. In this thesis, we are dealing with virtual models, so only the geometric approaches will be considered.

Figure 2.11: Texture mapping and constrained texture mapping.

A more challenging problem is how to enforce the bijective mapping to meet some user specified correspondence, which is regarded as constrained parameterization. Such extra constraints are crucial in some applications including texture mapping and morphing. Take Figure 2.11 as an example, where we want to paste a 2D figure onto a 3D geometry.

Simply using an unconstrained parameterization may produce undesired results (see Figure 2.11(c)). On the other hand, after enforcing the feature correspondence between the 3D geometry and texture, i.e. the eyes, nose, and mouth, constrained parameterization produces a much better visual effect (see Figure 2.11(d)).

As planar parameterization will be eventually reduced to an optimization problem, a direct solver to the constrained parameterization could be a constrained optimization, which incorporates the specified positional constraints as extra conditions. By introducing a quadratic item into the energy functional, Lévy [47] solved this problem in a soft sense. Desbrun et al. [12] and Gingold et al. [25] applied the classical Lagrangian multiplier method to deal with such constrained problems to satisfy hard constraints. However, due to ignorance of the mesh topology when enforcing the constraints, these methods tend to produce foldovers, thus violating the bijection rule as required. A more common approach is a two-step way. Usually, an initial 2D embedding is first generated without concerns in constraints. Later, the 2D embedded triangulation is deformed to aligned user specified constraints, which is also known as reparameterization process. A desirable reparameterization of 2D mesh should be bijective, smooth, and able to satisfy hard constraints. But meeting these requirements simultaneously is an extremely challenging task. Through constrained simplification and multi-resolution reconstruction in a reverse order, Eckstein et al. [16] achieved hard constraints interpolation. Steiner points are added in order to avoid foldover. But their method is extremely complicated and not robust [42]. Kraevoy et al. [42] and Lee et al. [45] adopted a two level strategy involving a coarse mesh. The feature alignment is achieved in the coarse mesh while the remaining position of the mesh is computed using barycentric coordinates. Usually after these operations, the resultant mesh will be extremely distorted, hence requiring time-consuming post processing. Kraevoy et al. [42] failed to avoid the foldover due to neglecting the consistent ordering of the path [93]. Zöckler et al. [146] directly embedded the triangulation into the coarse mesh, and warped the coarse mesh using the warping scheme proposed in [23] to align the correspondence. But the authors admitted that their approach could produce foldover especially when the feature points were

very distinctive. As we can see, although the coarse mesh can help achieve the satisfaction of hard constraints, it fails to generate a smooth map. To remedy this, many efforts have been made to build up the transformation based on a smooth 2D map. Typically among this type of method, the hard constrained is achieved through interpolation by the smooth map, and the mesh transformation is obtained through its linear induction as in Equation 5.1. The differences among these methods lie in how they realize the hard constraints. Tang et al. [117] and Lee et al. [44] directly formulated the constrained mapping problem as scattered data interpolation, which employed radial basis functions (RBF) for this purpose. The hard constraints and smoothness are readily satisfied. However, due to ignorance of the local properties of the maps, the induced mesh transformations are likely to get foldover, which violates the bijective requirement. Since the direct interpolation fails to ensure the bijectivity, Yu et al. [139] and Seo et al. [101] adopted the progressive deformation similar as in [127] to realize the interpolation. During each step, the smooth map was carefully established to ensure one-to-one, and the original 2D mesh was iteratively transformed to satisfy the predefined constraints. Therein, [139] still utilized the RBF interpolation owing to its less distortion, while [101] generated the warping using a time-dependent vector field. On the other hand, Lui et al. [57] sought to build up a one-step bijective smooth map that could align the specified correspondence. They employed Teichmüller extremal mapping, which is bijective and quasi-conformal, to induce the transformation. Their results were rather appealing in the sense of angle preserving. All these three approaches concentrated on the bijectivity of the smooth map, and assumed that the foldover-free of the induced mesh transformation can be guaranteed by the local one-to-one of the map. However, this only holds when the original mesh is densely sampled. Generally, it is not the case, because the mapping image of vertices will be connected using a straight line rather than a curve. Figure 5.3 presents a simple example of this situation. Therefore, more investigation has to be made in order to generate a constrained mapping that simultaneously satisfies the above three requirements.

While the above methods mainly focus on the planar parameterization, the mapping

between general surfaces of the same topology also draws great attentions. Such problem is considered as cross parameterization, and frequently encountered in mesh morphing, which we will cover later. It also involves the fulfillment of matching features as mentioned above. Typical approach is to parameterize the models over a simple base domain. The difference lies in the types of domain and how the feature points are matched. Alexa [1] utilized a sphere as the base domain and established an initial map onto it using a relaxation method. To match the features, he adopted RBF-based approach to gradually deform the constrained points towards the target ones. Although foldover was carefully checked, it could not completely resolve it. In contrast, Lin et al. [50] used edge swapping to align the features of two spherical parameterizations. But edge swapping cause damage in the surface geometry. A more popular base domain is the triangular layout formed by the triangulation of the given set of matching features. Praun et al.[81] proposed a tracing method to construct the map onto such base domain, but their approach has a potential in producing swirling path [45]. Their method was extended to map objects with different genus [93]. Kraevoy et.al [41] also adopted such strategy, but the inter-map within the base domain is constructed through remeshing. Pioneered by Tarini et al. [118], polycube has gained much popularity in global parameterization. Many methods have been proposed to automatically construct the polycube structure [51] [31], and establish the less distorted map onto this domain [129] [130]. These approaches are parallel to the unconstrained planar parameterization. Xia et al. [133] presented the editable polycube mapping technique which enabled feature matching provided by user sketching. This actually realizes the utilization of polycube as the base domain for cross parameterization. Then Garcia et al. [24] extended this idea to crossly parameterize general models, and exploited its application in shape morphing.

2.7 Shape morphing

Shape morphing, or metamorphosis, refers to a process that automatically generates the transition from a source object to a target object. It has been found wide applications in computer graphics, animation and entertainment industries. Generally, shape morphing involves two steps: vertex correspondence and vertex path. The former is to find the the correspondence between points in the two shapes and the latter accounts for how the transition is performed for each corresponding point pair.

Morphing was initially proposed to blend 2D polygonal contours. In the seminal work of Sederberg et al.[98], 2D polygonal contours were treated as wires and then the mapping between the vertexes was established through minimizing the work done to bend or stretch the wire from the source shape to the target one. As to the vertex path, the corresponding vertices were simply interpolated. Yet, such a brute-force approach ignored the features of initial objects. Therefore, it often results in unwanted effects such as unnatural shrink and self-intersection, especially when the difference between the two objects is large . Later in [97], an intrinsic approach was introduced to determine the transitive path between the corresponding vertexes. The method interpolated two intrinsic parameters—the edge length and the angle between each two adjacent edges. As a result, some of the unwanted shrink could be avoided. But it doesn't solve the problems completely. Due to lack of considering the interior information, the intrinsic interpolation still produces self-intersection and distorts the polygonal area during transition sometimes. Based on this observation, Shapira and Rappaport [102] pointed out that a proper morphing should incorporate the blending of interior of the objects as well as the boundaries. They proposed the star-skeleton to represent the interior of the polygonal shapes and applied the blend to the parametric description of this skeleton. Although the self-intersection can be avoided to some extent, the transition appears ambiguous when the shapes differ largely. To further improve the results, Alexa et. al [2] developed the as-rigid-as-possible shape interpolation which took the interior parts into consideration through Delaunay triangulation. In this approach, each

pair of corresponding triangles established a transformation which was factorized into a rotation transformation and a stretching transformation. Then the optimal morphing was given by linearly interpolating the rotation parts and stretching parts respectively. To meet the connectivity constraints, the actual morphing paths were obtained by minimizing the error from the optimal morphing. In this way, the morphing results appear rigid in the sense that the local triangulations are least distorted. However, it demands high quality meshes to avoid numerical problems. By assuming the existence of consistent interior triangulation, Yan et. al [137] achieved natural morphing through interpolating the strain field, which was borrowed from mechanics. Like [2], the optimal interpolation was difficult to achieve due to constraints. Alternatively, they defined an error functional, which was minimized using the finite element method to generate the vertexes paths.

Parallel to the 2D case, the idea can be naturally generalized to 3D meshes, which is even more challenging and complicated. For the vertex correspondence, it is essentially a constrained parameterization problem, which we have gone over in last section. So in this situation, we mainly focus on the vertex path issue. In [43][37][80][41], as soon as the correspondence is established, the path is obtained using simple linear interpolation. As mentioned in the 2D case, when the deformation between source and target objects gets large, it tends to produce unsatisfactory transition. Hence, people sought to compute more sophisticated trajectories. The method in [2] can also be applied to 3D mesh, but it demands a 3D tetrahedralization, which remains an open problem. Instead, as the popular application in mesh editing, differential coordinates draw the attention. Alexa [3] set the vertex path by interpolating Laplacian coordinates, which further achieved great success in mesh deformation [4, 140]. Yet Laplacian coordinates are only translation invariant. When the object undergoes a rotation or scaling, obvious artifacts are observed. Although Sorkine et al [106] further considered the rotation and scaling factors in transformation, the method works plausibly only for small rotation. Later, Sheffer and Kraevoy [103] proposed the so-called Pyramid coordinates, which are rotation-invariant. However, it requires to solve a non-linear optimization problem. Then by defining tangential and normal components of

the surfaces, Lipman et.al [52] introduced differential coordinates of rotation invariant into mesh morphing, which only needs to solve two linear systems for each intermediate frame. Xu et.al [136] utilized a gradient field for interpolation. Instead of linear treatment, they interpolated the rotation and anisotropic scaling respectively as in [2]. The intermediate shape was reconstructed by solving Poisson equation. As a result, the method can produce pretty good results in many cases even when the two initial objects are dissimilar largely. Observing the success of as rigid as possible approach in shape modeling [34, 107], [55] generalized the approach in [2] to generate the path through interpolating the deformation gradient similar as in [112], which avoids the tetrahedralization process. All methods discussed above deal with vertex trajectories based on geometric information. Yan et al [138] generalized the strain interpolation [137] to the 3D case, which adopted physical quantities as the distortion measurement. Incorporating physics can not only produce natural and visually appealing results, but also provide special effects. Huang et al [33] formulated the trajectory as a boundary-valued problem of a physical dynamic equation, which was solved in gradient domain. Then a reconstruction method similar to that in [136] was applied to generate the intermediate poses. Consequently, it can automatically produce oscillatory motion, which usually requires extra setting to link the trajectory spline in animation [39].

While morphing algorithms thrive in the polygonal mesh domain, only a few consider the continuous parametric representations. Parametric free form objects such as NURBS and T-spline are widely used in CAD and CAM. As morphing is also a powerful modeling technique, the problem of morphing parametric surfaces is also worth investigating. Unlike discrete representation, parametric geometry possesses an inherent correspondence through existing parameterization. However such correspondence is unsatisfactory in practice for its failure in reflecting the correspondence of some important features. Sederberg and Greenwood [95] extended their physically-based approach for polygonal contours to 2D shapes bounded by piecewise Bézier curves to establish the correspondence. The piecewise Bézier curves were converted into B-spline curves and appropriate knots were also inserted into the B-spline curves so as to minimize the work required to bend and stretch one shape into the

other. But the extension of their method into the surface case is unknown. [9] and [113] seek to achieve the correspondence automatically through minimizing the similarity between the two objectives. The geometric information such as tangent field and Gauss field serve as the measurement for the similarity. They adopted dynamic implementation to obtain the final reparameterization. Their methods can not guarantee the resultant reparameterized geometry has the same representation as the input ones. However, this is prerequisite in some applications such as morphing based modeling. Ju and Goldman [36] noticed this importance, and proposed to first transform the input objects into the same representation. That is, both the degree and knot vector are unified, which ensures the transitory surface maintain the same representation as the input ones. But their method ignores the important feature correspondence. They also proposed to averaging the two input surfaces in the Grassmann space. Such treatment allows users to interactively adjust the morphing process through altering the mass distribution. But to determine an appropriate mass distribution is a tedious task. [114] proposed to generate the trajectory through interpolating the curvature. It achieved good results in metamorphosing spiral curves. However each frame requires solving an approximation problem, and such a method is difficult to be extend into the surface cases. From the analysis above, we can infer that the morphing problem in parametric geometries has not been well addressed, hence deserving further investigation.

Chapter 3

Physically based Dynamic T-spline

3.1 Introduction

This chapter introduces physics into T-spline representation. T-splines are a generalization of NURBS, and solve most of the existing drawbacks in NURBS. Especially, T-splines possess local refinement property. Hence marrying physics with T-splines can easily solve the refinement issue in D-NURBS. To further enhance the physically based model, we improve the efficiency in the following aspects.

- E-PD-T-splines are developed through freezing the weights variation. Consequently, the dynamic equation leads to a linear ordinary differential equation (ODE). Due to the sparsity of the coefficient matrices, the equation can be further decomposed into three independent ones with identical coefficients. This means the dimension of equation is reduced by two thirds compared with the initial one.
- As the equation is linearized, we derive a closed form solution to the resultant dynamic equation, which is apparently advantageous over the numerical solver. Moreover, our solver is based on the physical insight that a vibration system is a coupling of separated vibration modes. This enables us to develop a modal reduction technique that stores a merely small portion of modes for calculation to further reduce the

computational complexity.

- Although the dimension of the system for T-splines will not increase too much when a local refinement is performed, we have to update the coefficients of the dynamic equation. Thorough recalculation is obviously time-consuming and computationally intensive. As the refinement is local, we propose to locally update the dynamic T-spline system through carefully examining the change of elements, thereby at best avoiding repeated calculations.

3.2 Physically-based dynamic T-splines

We now introduce time and physics into T-splines, producing physically-based dynamic T-splines, or PD-T-splines for short.

3.2.1 Dynamic T-spline surfaces

A dynamic T-spline (or D-T-spline) surface extends the T-spline definition by explicitly incorporating time and thus is defined by

$$\mathbf{T}(u, v, t) = \frac{\sum_{i=0}^n B_i(u, v) \omega_i(t) \mathbf{P}_i(t)}{\sum_{i=0}^n B_i(u, v) \omega_i(t)} \quad (3.1)$$

which is similar to (2.5) except that the control points and weights are functions of time t . The time dependent control points $\mathbf{P}_i(t)$ and weights $\omega_i(t)$ form the generalized coordinates in a physical system. We assemble all these generalized coordinates into a $4(n + 1)$ dimensional vector:

$$\mathbf{P}(t) = [\mathbf{P}_0^T, \omega_0, \dots, \mathbf{P}_n^T, \omega_n]^T. \quad (3.2)$$

Here we assume the control points $\mathbf{P}_i$ are column vectors and the superscript T stands for transposition.

In order to emphasize the dependency on the generalized coordinates $\mathbf{P}$, we express $\mathbf{T}(u, v, t)$ as $\mathbf{T}(u, v, \mathbf{P})$. The velocity of the dynamic T-spline surface is

$$\dot{\mathbf{T}}(u, v, \mathbf{P}) = \mathbf{J}\dot{\mathbf{P}} \quad (3.3)$$

where $\dot{\mathbf{T}}(u, v, \mathbf{P}) = \frac{d\mathbf{T}(u, v, \mathbf{P})}{dt}$, $\dot{\mathbf{P}} = \frac{d\mathbf{P}}{dt}$, and $\mathbf{J}$ is the Jacobian matrix. By some calculations, we have

$$\mathbf{J}(u, v, \mathbf{P}) = [\mathbf{B}_0, \mathbf{W}_0, \mathbf{B}_1, \mathbf{W}_1, \dots, \mathbf{B}_n, \mathbf{W}_n] \quad (3.4)$$

in which $\mathbf{B}_i = \begin{bmatrix} N_i(u, v, \mathbf{P}) & 0 & 0 \\ 0 & N_i(u, v, \mathbf{P}) & 0 \\ 0 & 0 & N_i(u, v, \mathbf{P}) \end{bmatrix}$ are 3×3 diagonal matrices with the diagonal entries

$$N_i(u, v, \mathbf{P}) = \frac{\partial \mathbf{T}}{\partial \mathbf{P}_i} = \frac{B_i(u, v)\omega_i}{\sum_{c=0}^n B_c(u, v)\omega_c} \quad (3.5)$$

and the 3-component column vectors $\mathbf{W}_i$ are the partial derivative of $\mathbf{T}(u, v, t)$ with respect to ω_i :

$$\mathbf{W}_i(u, v, \mathbf{P}) = \frac{\partial \mathbf{T}}{\partial \omega_i} = \frac{\sum_{c=0}^n B_i(u, v)B_c(u, v)\omega_c(\mathbf{P}_i - \mathbf{P}_c)}{\left(\sum_{c=0}^n B_c(u, v)\omega_c\right)^2}.$$

Furthermore, $\mathbf{J}\dot{\mathbf{P}} = \sum_{i=0}^n \mathbf{B}_i\dot{\mathbf{P}}_i + \sum_{i=0}^n \mathbf{W}_i\dot{\omega}_i = \frac{\sum_{i=0}^n B_i(u, v)\omega_i(t)\mathbf{P}_i(t)}{\sum_{i=0}^n B_i(u, v)\omega_i(t)} + \sum_{i=0}^n \mathbf{W}_i\dot{\omega}_i$. Since

$$\begin{aligned} \sum_{i=0}^n \mathbf{W}_i\dot{\omega}_i &= \frac{\sum_{i=0}^n \sum_{c=0}^n B_i(u, v)B_c(u, v)\omega_i\dot{\omega}_c(\mathbf{P}_i - \mathbf{P}_c)}{\left(\sum_{c=0}^n B_c(u, v)\omega_c\right)^2} = \frac{\sum_{c=0}^n \sum_{i=0}^n B_c(u, v)B_i(u, v)\omega_c\dot{\omega}_i(\mathbf{P}_c - \mathbf{P}_i)}{\left(\sum_{c=0}^n B_c(u, v)\omega_c\right)^2} \\ &= -\frac{\sum_{i=0}^n \sum_{c=0}^n B_i(u, v)B_c(u, v)\omega_i\dot{\omega}_c(\mathbf{P}_i - \mathbf{P}_c)}{\left(\sum_{c=0}^n B_c(u, v)\omega_c\right)^2} = -\sum_{i=0}^n \mathbf{W}_i\dot{\omega}_i, \end{aligned}$$

we have $\sum_{i=0}^n \mathbf{W}_i\dot{\omega}_i = 0$. Hence the dynamic T-splines can also be represented as the

product of the Jacobian matrix and the generalized coordinate vector:

$$\mathbf{T}(u, v, \mathbf{P}) = \mathbf{J}\mathbf{P} \quad (3.6)$$

3.2.2 Physically-based motion for dynamic T-splines

While a dynamic T-spline surface of (3.1) can have arbitrary dynamic behavior, we restrict our PD-T-splines such that their motion is governed by the work-energy version of Lagrangian Dynamics [26]

$$\frac{d}{dt} \frac{\partial H}{\partial \dot{p}_i} - \frac{\partial H}{\partial p_i} + \frac{\partial F}{\partial \dot{p}_i} + \frac{\partial U}{\partial p_i} = f_i \quad (3.7)$$

where H, F, U are the kinetic energy, dissipation energy and potential energy, respectively, p_i are the generalized coordinates, and f_i represent the generalized forces that act on p_i .

For a dynamic T-spline surface of (3.1), we let $\mu(u, v)$ be the mass density distribution, and then the kinetic energy of the surface can be computed by

$$H = \frac{1}{2} \iint \mu \dot{\mathbf{T}}^T \dot{\mathbf{T}} du dv = \frac{1}{2} \dot{\mathbf{P}}^T \mathbf{M} \dot{\mathbf{P}} \quad (3.8)$$

where

$$\mathbf{M} = \iint \mu \mathbf{J}^T \mathbf{J} du dv \quad (3.9)$$

is a mass matrix. Similarly, if the damping density distribution is denoted by $\gamma(u, v)$, the dissipation energy functional F is

$$F = \frac{1}{2} \iint \gamma \dot{\mathbf{T}}^T \dot{\mathbf{T}} du dv = \frac{1}{2} \dot{\mathbf{P}}^T \mathbf{D} \dot{\mathbf{P}} \quad (3.10)$$

where

$$\mathbf{D} = \iint \gamma \mathbf{J}^T \mathbf{J} du dv \quad (3.11)$$

is the damping matrix. As to the potential energy functional U , the thin-plate under tension energy model is used to simulate the elastic property. The thin-plate under tension energy is defined by

$$U = \frac{1}{2} \iint \left(\alpha_{1,1} \frac{\partial \mathbf{T}^T}{\partial u} \frac{\partial \mathbf{T}}{\partial u} + \alpha_{2,2} \frac{\partial \mathbf{T}^T}{\partial v} \frac{\partial \mathbf{T}}{\partial v} + \beta_{1,1} \frac{\partial^2 \mathbf{T}^T}{\partial u^2} \frac{\partial^2 \mathbf{T}}{\partial u^2} + \beta_{1,2} \frac{\partial^2 \mathbf{T}^T}{\partial u \partial v} \frac{\partial^2 \mathbf{T}}{\partial u \partial v} + \beta_{2,2} \frac{\partial^2 \mathbf{T}^T}{\partial v^2} \frac{\partial^2 \mathbf{T}}{\partial v^2} \right) dudv = \frac{1}{2} \mathbf{P}^T \mathbf{K} \mathbf{P} \quad (3.12)$$

where $\alpha_{i,j}(u, v)$, $\beta_{i,j}(u, v)$ are material parameters that provide control for different simulation results ranging from rigid to elastic, and $\mathbf{K}$ is the stiffness matrix. The stiffness matrix for the dynamic T-spline surface is expressed as

$$\mathbf{K} = \iint (\alpha_{1,1} \mathbf{J}_u^T \mathbf{J}_u + \alpha_{2,2} \mathbf{J}_v^T \mathbf{J}_v + \beta_{1,1} \mathbf{J}_{uu}^T \mathbf{J}_{uu} + \beta_{1,2} \mathbf{J}_{uv}^T \mathbf{J}_{uv} + \beta_{2,2} \mathbf{J}_{vv}^T \mathbf{J}_{vv}) dudv \quad (3.13)$$

where $\mathbf{J}_u$, $\mathbf{J}_v$, $\mathbf{J}_{uu}$, $\mathbf{J}_{uv}$ and $\mathbf{J}_{vv}$ are the first order and second order partial derivatives of $\mathbf{J}$ with respect to u , v or both.

Substituting (3.8), (3.10), and (3.12) into (3.7), we finally obtain the dynamic equation that governs the motion behavior of the PD-T-splines:

$$\mathbf{M} \ddot{\mathbf{P}} + \mathbf{D} \dot{\mathbf{P}} + \mathbf{K} \mathbf{P} = \mathbf{f}_p - \mathbf{I} \dot{\mathbf{P}} \quad (3.14)$$

where

$$\mathbf{f}_p = \iint \mathbf{J}^T \mathbf{f}(u, v, t) dudv \quad (3.15)$$

is the generalized forces vector derived through the principle of virtual work done by the applied forces distribution $\mathbf{f}(u, v, t)$, and

$$\mathbf{I} = \iint \mu \mathbf{J}^T \dot{\mathbf{J}} dudv. \quad (3.16)$$

Note that the dynamic equation (3.14) appears to be the same as that in D-NURBS [122].

The difference lies in the Jacobian matrix $\mathbf{J}$. As T-splines are a generalization of NURBS, our PD-T-splines generalize D-NURBS as well. When the T-mesh is a rectangular grid, the PD-T-splines reduce to D-NURBS. In addition, when only the final static equilibrium is concerned, both $\dot{\mathbf{P}}$ and $\ddot{\mathbf{P}}$ vanish and the equation becomes

$$\mathbf{K}\mathbf{P} = \mathbf{f}_p \quad (3.17)$$

which is the equation for energy minimization. In this sense, PD-T-splines extend energy minimization models.

It is worth pointing out that the thin-plate under tension energy model (3.12) used above is based on the absolute spatial coordinates. In physics, however, the object gains internal energy when the displacement away from its rest shape occurs. Based on this consideration, we may modify the potential energy model into

$$\begin{aligned} U &= \frac{1}{2} \iint \left(\alpha_{1,1} \frac{\partial(\mathbf{T} - \mathbf{T}_0)^T}{\partial u} \frac{\partial(\mathbf{T} - \mathbf{T}_0)}{\partial u} + \alpha_{2,2} \frac{\partial(\mathbf{T} - \mathbf{T}_0)^T}{\partial v} \frac{\partial(\mathbf{T} - \mathbf{T}_0)}{\partial v} \right. \\ &\quad + \beta_{1,1} \frac{\partial^2(\mathbf{T} - \mathbf{T}_0)^T}{\partial u^2} \frac{\partial^2(\mathbf{T} - \mathbf{T}_0)}{\partial u^2} + \beta_{1,2} \frac{\partial^2(\mathbf{T} - \mathbf{T}_0)^T}{\partial u \partial v} \frac{\partial^2(\mathbf{T} - \mathbf{T}_0)}{\partial u \partial v} \\ &\quad \left. + \beta_{2,2} \frac{\partial^2(\mathbf{T} - \mathbf{T}_0)^T}{\partial v^2} \frac{\partial^2(\mathbf{T} - \mathbf{T}_0)}{\partial v^2} \right) dudv \\ &= \frac{1}{2} (\mathbf{P} - \mathbf{P}_0)^T \mathbf{K} (\mathbf{P} - \mathbf{P}_0) \end{aligned} \quad (3.18)$$

where $\mathbf{T}_0$ denotes the rest shape and $\mathbf{P}_0$ represents its corresponding generalized coordinates. Accordingly, the dynamic equation based on this new energy model becomes

$$\mathbf{M}\ddot{\mathbf{P}} + \mathbf{D}\dot{\mathbf{P}} + \mathbf{K}\mathbf{P} = \mathbf{f}_p - \mathbf{I}\dot{\mathbf{P}} + \mathbf{K}\mathbf{P}_0. \quad (3.19)$$

In the following we still proceed with (3.14), but all the processes can be completely extended to (3.19).

3.2.3 Efficient PD-T-splines

(3.14) governs the behavior of the Dynamic T-splines, which provides the candidates for users to make the final decision. For PD-T-splines, a suitable solver should satisfy two requirements. First, it should be efficient enough to produce the feedback in realtime. It means the intermediate shape ought to be visualized smoothly as time flows. Second, it should support both forward and backward integration. Because modeling with PD-T-splines will frequently invoke the forward and backward evolution for comparison. As the equation is a second-order nonlinear differential equation. Generally, no analytical solution is available. The numerical solution as illustrated in (2.22) can be adopted to integrate this differential equation. However, the recalculation of the time dependent coefficient matrices makes it difficult to integrate it in realtime, especially when the system gets large. In addition, such numerical integration is not suitable for backward integration either. Since, forward and backward evolution will be frequently invoked, numerical integration can hardly converge to the real solution due to the error accumulation.

Therefore, a full version of PD-T-splines may not be suitable for practical applications. We try to seek some simplification to enable PD-T-splines more applicable for practical use while keeping most of their advantages. Based on the analysis above, we can infer that it's the nonlinearity of the motion equation that results in the troublesome computational complexity, and the nonlinearity is caused by the change of the associated weights. Therefore, one reasonable approach to enhance PD-T-splines is to freeze the weights. In general, such treatment will not limit the applicability of PD-T-splines very much. Although weights freezing leads to a $1/4$ reduction in DOFs, T-splines have a power of local refinement which can increase the DOFs when necessary.

When the weights are frozen, D-T-splines can be expressed as

$$\mathbf{T}(u, v, t) = \frac{\sum_{i=0}^n B_i(u, v) \omega_i \mathbf{P}_i(t)}{\sum_{i=0}^n B_i(u, v) \omega_i} \quad (3.20)$$

Compared with (3.1), the weights are no longer functions of time t . The totally independent generalized coordinates change into the assembly of all the control points,

$$\mathbf{Q} = [\mathbf{P}_0^T, \mathbf{P}_1^T, \dots, \mathbf{P}_n^T]^T$$

Following Lagrangian dynamics, the motion equation for this generalized coordinates becomes:

$$\mathbf{M}_q \ddot{\mathbf{Q}} + \mathbf{D}_q \dot{\mathbf{Q}} + \mathbf{K}_q \mathbf{Q} = \mathbf{f}_q \quad (3.21)$$

where

$$\begin{aligned} \mathbf{M}_q &= \iint \mu \bar{\mathbf{J}}^T \bar{\mathbf{J}} du dv \\ \mathbf{D}_q &= \iint \gamma \bar{\mathbf{J}}^T \bar{\mathbf{J}} du dv \\ \mathbf{f}_q &= \iint \bar{\mathbf{J}}^T \mathbf{f} du dv \\ \mathbf{K}_q &= \iint \left(\alpha_{1,1} \bar{\mathbf{J}}_u^T \bar{\mathbf{J}}_u + \alpha_{2,2} \bar{\mathbf{J}}_v^T \bar{\mathbf{J}}_v + \beta_{1,1} \bar{\mathbf{J}}_{uu}^T \bar{\mathbf{J}}_{uu} + \beta_{1,2} \bar{\mathbf{J}}_{uv}^T \bar{\mathbf{J}}_{uv} + \beta_{2,2} \bar{\mathbf{J}}_{vv}^T \bar{\mathbf{J}}_{vv} \right) du dv \end{aligned} \quad (3.22)$$

Here the Jacobian matrix is

$$\bar{\mathbf{J}}(u, v, \mathbf{P}) = [\mathbf{B}_0, \mathbf{B}_1, \dots, \mathbf{B}_n] \quad (3.23)$$

We call the reduced D-T-splines in (3.20) whose motion is governed by the equation in (3.21) the Efficient PD-T-splines (or E-PD-T-splines for short).

Observing the sparse coefficient matrices in (3.21), we can further separate it into 3 independent equations. In fact, if we rearrange the generalized coordinates as

$$\begin{aligned} \mathbf{Q}_x &= [\dots, \mathbf{P}_{ix}, \dots]^T \\ \mathbf{Q}_y &= [\dots, \mathbf{P}_{iy}, \dots]^T \\ \mathbf{Q}_z &= [\dots, \mathbf{P}_{iz}, \dots]^T \end{aligned} \quad (3.24)$$

D-T-splines can also be written as

$$\mathbf{T}(u, v, t) = [\mathbf{j}\mathbf{Q}_x, \mathbf{j}\mathbf{Q}_y, \mathbf{j}\mathbf{Q}_z]^T \quad (3.25)$$

where

$$\mathbf{j}(u, v) = [\cdots, N_i(u, v), \cdots] \quad (3.26)$$

The equation (3.21) can be separated into

$$\begin{aligned} \widetilde{\mathbf{M}}_q \ddot{\mathbf{Q}}_x + \widetilde{\mathbf{D}}_q \dot{\mathbf{Q}}_x + \widetilde{\mathbf{K}}_q \mathbf{Q}_x &= \mathbf{f}_{qx} \\ \widetilde{\mathbf{M}}_q \ddot{\mathbf{Q}}_y + \widetilde{\mathbf{D}}_q \dot{\mathbf{Q}}_y + \widetilde{\mathbf{K}}_q \mathbf{Q}_y &= \mathbf{f}_{qy} \\ \widetilde{\mathbf{M}}_q \ddot{\mathbf{Q}}_z + \widetilde{\mathbf{D}}_q \dot{\mathbf{Q}}_z + \widetilde{\mathbf{K}}_q \mathbf{Q}_z &= \mathbf{f}_{qz} \end{aligned} \quad (3.27)$$

The matrices coefficients become

$$\begin{aligned} \widetilde{\mathbf{M}}_q &= \iint \mu \mathbf{j}^T \mathbf{j} du dv \\ \widetilde{\mathbf{D}}_q &= \iint \gamma \mathbf{j}^T \mathbf{j} du dv \\ (\mathbf{f}_{qx}, \mathbf{f}_{qy}, \mathbf{f}_{qz}) &= \iint \mathbf{j}^T \mathbf{f}(u, v, t)^T du dv \\ \widetilde{\mathbf{K}}_q &= \iint (\alpha_{1,1} \mathbf{j}_u^T \mathbf{j}_u + \alpha_{2,2} \mathbf{j}_v^T \mathbf{j}_v + \beta_{1,1} \mathbf{j}_{uu}^T \mathbf{j}_{uu} + \beta_{1,2} \mathbf{j}_{uv}^T \mathbf{j}_{uv} + \beta_{2,2} \mathbf{j}_{vv}^T \mathbf{j}_{vv}) du dv \end{aligned} \quad (3.28)$$

The three equations in (3.27) have the same coefficients with different homogeneous terms, and the dimension is 1/3 of that of (3.21). Obviously, solving (3.27) instead of (3.21) will be much more economic. In the following, $\mathbf{Q} = [\mathbf{Q}_x, \mathbf{Q}_y, \mathbf{Q}_z]$ will be considered as the generalized coordinates. It is noted that currently the three equations are still dependent due to the existence of homogeneous terms. But as we will see later, most of the useful forces are coordinates independent, which yield independent homogeneous terms. Hence, in practice the three equations can be solved independently.

3.3 Forces and Constraints

In addition to specify the physical quantities such as mass density and elastic properties, modelers can also interact with the object with forces and constraints within our PD-T-splines scheme by virtue of its physical nature. The simulated forces provide a much more convenient approach for users to express their design intention. For instance, if the surface point is desired to move towards a specific spatial position, it can be simply realized by attaching a spring force connecting them. Such process is straightforward and vivid just like interacting with a real-world physical object. This implies that modeling with PD-T-splines is absolutely shape-oriented rather than control points/weights oriented as in traditional T-splines, which totally frees the users from understanding the underlining formulation and struggling in the choice of which parameter to adjust. Besides, both the local and global constraints, which are really troublesome in traditional T-splines, can be imposed in PD-T-splines without much efforts. For example, modelers often demand to model a fair surface while interpolating certain curves or spatial points. Within PD-T-spline, the smoothness has been encoded into the material property. As soon as those geometric constraints such as interpolation of points, curves, and normals are imposed, the high-level modeling requirement can be easily satisfied by PD-T-splines. In the following, we will show how those useful forces and constraints can be incorporated into our PD-T-splines scheme.

3.3.1 Applied forces

In general, force can be any function of time t . Here we present some of the most frequently used types of forces.

Gravitational Force

Due to mass distribution, adding gravitational force will improve the physical reality of the motion. To incorporate it into PD-T-splines, we suffice to set the force density to

$$f_g(u, v, t) = g\mu(u, v) \begin{pmatrix} a \\ b \\ c \end{pmatrix} \quad (3.29)$$

where $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ is the force direction, $\mu(u, v)$ refers to the mass density, and g is the gravitational acceleration. Note that the force direction does not need to be fixed in one direction, and it can be modified to achieve desired effect. Such gravitational force is very effective in local protrusion. Furthermore, it can also be modified to support local editing by introducing a smooth kernel (e.g. Gaussian kernel),

$$f_g(u, v, t) = g\mu(u, v)G(u - u_0, v - v_0) \begin{pmatrix} a \\ b \\ c \end{pmatrix} \quad (3.30)$$

Pressure

Pressure is another common force in physics. Within PD-T-splines scenario, it exhibits as inflation or deflation like a balloon. To incorporate such kind of force, we only have to add the force density by

$$f_p = pN(u, v) \quad (3.31)$$

where $N(u, v)$ represents the normal of the surface.

Spring

Note that the above two kinds of force are constant, thus irrelevant to the DOFs. Next we will introduce two kinds of force that depend on the time-varying DOFs. Spring Force is closely related to the geometric direct manipulation. To introduce it, we need to add the

following force density

$$f_s(u, v, t) = k\delta(u - u_0, v - v_0)(\mathbf{p}_0 - \mathbf{T}(u, v, t)) \quad (3.32)$$

where $\mathbf{p}_0$ is the spring spatial anchor point, (u_0, v_0) is the parameter for the attached point on surface, k is the spring stiffness, and δ represents the unit delta function. When integrating the function density throughout the domain, we get

$$\mathbf{f}_s = \iint k\mathbf{j}^T\delta(u - u_0, v - v_0)(\mathbf{p}_0 - \mathbf{T}(u, v, t))^T dudv \quad (3.33)$$

Making use of the facts in (3.25) and

$$\iint \delta(u - u_0, v - v_0)f(u, v)dudv = f(u_0, v_0). \quad (3.34)$$

the spring force term becomes

$$(\mathbf{f}_{sx}, \mathbf{f}_{sy}, \mathbf{f}_{sz}) = k\mathbf{j}^T(u_0, v_0)\mathbf{p}_0^T - k\mathbf{j}^T(u_0, v_0)\mathbf{j}(u_0, v_0)(\mathbf{Q}_x, \mathbf{Q}_y, \mathbf{Q}_z) \quad (3.35)$$

We can easily infer that $\mathbf{f}_s$ is linear in the generalized coordinates.

Curve force

Recently, curve handles for surface modification are becoming popular with the development of computerized sketching tools and are shown more effective and intuitive. Especially when feature manipulation is required, curve handles are of greater advantages. This is because most surface features can be interpreted as curves on the surface. Compared with D-NURBS, PD-T-splines support much more sophisticated modeling tools based on general curve handles. We propose a new type of force named by curve force for interactive modeling. Given a spatial curve $C(s)$ and a parametric curve $D(s) = (u(s), v(s))$ in the

parametric domain, curve force is defined by

$$(\mathbf{f}_{cx}, \mathbf{f}_{cy}, \mathbf{f}_{cz}) = \int_{\mathbf{T}} k(s) \mathbf{j}(u(s), v(s))^T (\mathbf{C}(s)^T - \mathbf{j}(u(s), v(s)) (\mathbf{Q}_x, \mathbf{Q}_y, \mathbf{Q}_z)) ds \quad (3.36)$$

where $k(s)$ represents the curve stiffness. Compared with (3.33), curve force can be viewed as infinite number of springs connecting the points of the surface curve $\mathbf{T}(u(s), v(s), t)$ to the corresponding points on the spatial curve $C(s)$.

3.3.2 Constraints

Geometric constraints are also of importance to express user's design objectives such as maintain some geometric feature even applied force cannot change. Particularly, we focus on linear constraints which can be expressed as

$$\mathbf{A}\mathbf{Q} + \mathbf{B} = \mathbf{0}. \quad (3.37)$$

Point Constraints

When the surface evolves, we may want some points to be fixed. This is regarded as the point constraint. For example, if the parameter of the surface point to be fixed is (u_0, v_0) , then during the evolution the generalized coordinates $\mathbf{Q}(t)$ should satisfy

$$\mathbf{T}(u_0, v_0, 0) = (\mathbf{j}(u_0, v_0)\mathbf{Q}_x, \mathbf{j}(u_0, v_0)\mathbf{Q}_y, \mathbf{j}(u_0, v_0)\mathbf{Q}_z)^T. \quad (3.38)$$

Curve constraints

Similarly, we can also freeze certain curves on the surface. In general, the curve to be frozen is specified by a parametric curve $(u(s), v(s))$ in the surface parameter domain. Then the curve on the surface is defined by

$$\mathbf{C}(s) = \mathbf{T}(u(s), v(s), 0) = (\mathbf{j}(u(s), v(s))\mathbf{Q}_x, \mathbf{j}(u(s), v(s))\mathbf{Q}_y, \mathbf{j}(u(s), v(s))\mathbf{Q}_z)^T \quad (3.39)$$

To maintain this curve on the surface during evolution, PD-T-splines ought to satisfy

$$\mathbf{j}(u(s), v(s))\mathbf{Q}_\alpha = \mathbf{j}(u(s), v(s))\mathbf{Q}_\alpha(0), \alpha = x, y, z \quad (3.40)$$

where $\mathbf{Q}_\alpha(0)$ is the initial generalized coordinates. Obviously, (3.40) is not a linear constraint, because it involves another variable s . Instead, we use the following condition to constrain the curve,

$$\tilde{\mathbf{K}}_c(\mathbf{Q}_x, \mathbf{Q}_y, \mathbf{Q}_z) = \tilde{\mathbf{K}}_c(\mathbf{Q}_x(0), \mathbf{Q}_y(0), \mathbf{Q}_z(0)) \quad (3.41)$$

where

$$\tilde{\mathbf{K}}_c = \int \mathbf{j}(u(s), v(s))^T \mathbf{j}(u(s), v(s)) ds \quad (3.42)$$

(3.41) is obtained through multiplying both sides in (3.40) by $\mathbf{j}(u(s), v(s))^T$, and integrate them throughout the range of s . Therefore, (3.41) is a necessary condition for (3.40). In next chapter, we will verify that it is also sufficient. Apparently, (3.41) is also linear with respect to the generalized coordinates.

Normal constraint

Sometimes the normal is also required to be preserved. We may directly use normal formula:

$$\mathbf{N}_0 = \frac{\frac{\partial \mathbf{T}}{\partial u}(u_0, v_0) \times \frac{\partial \mathbf{T}}{\partial v}(u_0, v_0)}{\left\| \frac{\partial \mathbf{T}}{\partial u}(u_0, v_0) \times \frac{\partial \mathbf{T}}{\partial v}(u_0, v_0) \right\|} \quad (3.43)$$

However, this formula is nonlinear in the generalized coordinates $\mathbf{Q}(t)$ because $\mathbf{Q}(t)$ also appears in the denominator. Alternatively, we constrain the normal by imposing stronger constraints, which is to freeze the two tangent vectors:

$$\begin{aligned} \mathbf{T}_u(u_0, v_0, 0) &= (\mathbf{j}_u(u_0, v_0)\mathbf{Q}_x, \mathbf{j}_u(u_0, v_0)\mathbf{Q}_y, \mathbf{j}_u(u_0, v_0)\mathbf{Q}_z)^T \\ \mathbf{T}_v(u_0, v_0, 0) &= (\mathbf{j}_v(u_0, v_0)\mathbf{Q}_x, \mathbf{j}_v(u_0, v_0)\mathbf{Q}_y, \mathbf{j}_v(u_0, v_0)\mathbf{Q}_z)^T \end{aligned} \quad (3.44)$$

Region of Interest

In the current stage, we allow the entire DOFs to vary through time. However, in many cases local editing is preferred. As in mesh deformation method, they would first choose region of interest (ROI). Such operation can also be incorporated into PD-T-splines. It is realized by selecting the DOFs which influence the ROI while maintaining others. Such a constraint not only improves the flexibility of PD-T-splines, but also reduces the system dimension significantly. Denote the indices set of l selected control points by Φ . ROI selection can be achieved through imposing

$$\begin{aligned} \mathbf{Q}_{i,x} &= \mathbf{Q}_{i,x}(0) \\ \mathbf{Q}_{i,y} &= \mathbf{Q}_{i,y}(0) \\ \mathbf{Q}_{i,z} &= \mathbf{Q}_{i,z}(0) \end{aligned} \tag{3.45}$$

for all $i \in \Phi$.

3.4 Dynamic equation in unconstrained DOFs

The above-mentioned forces and constraints provide rich tools for users to interact with PD-T-splines. As the generalized coordinates are constrained, they will no longer follow the equation (3.27), so new equations in the un-constrained ones have to be derived. As the constraints are imposed on $\mathbf{Q}_\alpha, \alpha = x, y, z$ individually, they can be expressed as

$$\begin{aligned} \mathbf{A}\mathbf{Q}_x + \mathbf{B}_x &= 0 \\ \mathbf{A}\mathbf{Q}_y + \mathbf{B}_y &= 0 \\ \mathbf{A}\mathbf{Q}_z + \mathbf{B}_z &= 0 \end{aligned} \tag{3.46}$$

Using Gaussian elimination, the generalized coordinates can be represented as

$$\begin{aligned} \mathbf{Q}_x &= \mathbf{G}\mathbf{R}_x + \mathbf{r}_{0x} \\ \mathbf{Q}_y &= \mathbf{G}\mathbf{R}_y + \mathbf{r}_{0y} \\ \mathbf{Q}_z &= \mathbf{G}\mathbf{R}_z + \mathbf{r}_{0z} \end{aligned} \quad (3.47)$$

where the columns of $\mathbf{G}$ are the bases for $\mathbf{Ker}(\mathbf{A})$, and $\mathbf{r}_{0\alpha}, \alpha = x, y, z$ are the special solutions for the corresponding linear equations. In the case of ROI selection,

$$\begin{aligned} \mathbf{G} &= [\cdots, \mathbf{I}_i, \cdots]_{i \in \Phi} \\ \mathbf{r}_{0\alpha}[i] &= \begin{cases} \mathbf{Q}_\alpha(0)[i] & i \in \Phi \\ 0 & \text{otherwise} \end{cases} \quad \alpha = x, y, z \end{aligned} \quad (3.48)$$

where $\mathbf{I}_i$ is the i th column of the $n \times n$ identity matrix and $[i]$ denotes the i th component of the vector.

Using the new unconstrained generalized coordinates $[\mathbf{R}_x, \mathbf{R}_y, \mathbf{R}_z]$, dynamic T-splines change into

$$\mathbf{T}(u, v, t) = (\mathbf{j}(\mathbf{G}\mathbf{R}_x + \mathbf{r}_{0x}), \mathbf{j}(\mathbf{G}\mathbf{R}_y + \mathbf{r}_{0y}), \mathbf{j}(\mathbf{G}\mathbf{R}_z + \mathbf{r}_{0z})) \quad (3.49)$$

Following the Lagrangian dynamics, the motion of PD-T-splines in the new generalized coordinates is characterized by

$$\mathbf{M}_r \ddot{\mathbf{R}}_\alpha + \mathbf{D}_r \dot{\mathbf{R}}_\alpha + \mathbf{K}_r \mathbf{R}_\alpha = \mathbf{f}_{r\alpha} + \mathbf{g}_{r\alpha}, \alpha = x, y, z \quad (3.50)$$

where

$$\begin{aligned}
\mathbf{M}_r &= \mathbf{G}^T \widetilde{\mathbf{M}}_q \mathbf{G} \\
\mathbf{D}_r &= \mathbf{G}^T \widetilde{\mathbf{D}}_q \mathbf{G} \\
\mathbf{K}_r &= \mathbf{G}^T \widetilde{\mathbf{K}}_q \mathbf{G} \\
\mathbf{g}_{r\alpha} &= -\mathbf{G}^T \widetilde{\mathbf{K}}_q \mathbf{r}_{0\alpha} \\
\mathbf{f}_{r\alpha} &= \iint (\mathbf{j}\mathbf{G})^T \mathbf{f}_\alpha dudv
\end{aligned} \tag{3.51}$$

Note that when the potential energy is calculated based on displacement, the residue term changes into

$$\mathbf{g}_{r\alpha} = \mathbf{G}^T \widetilde{\mathbf{K}}_q (\mathbf{Q}_\alpha(0) - \mathbf{r}_{0\alpha}) \tag{3.52}$$

Interestingly, if we consider the ROI selection only, (3.52) can be recast into

$$\mathbf{g}_{r\alpha} = \mathbf{K}_r \widetilde{\mathbf{r}}_{0\alpha} \tag{3.53}$$

where

$$\widetilde{\mathbf{r}}_{0\alpha} = [\cdots, \mathbf{Q}_\alpha(0)[i], \cdots]_{i \in \Phi} \tag{3.54}$$

Referring to (3.51) and (3.48), the calculation of the coefficient in (3.50) only involves the blending functions associated with the selected control points. Therefore, in practise, if ROI are chosen, we advise to use displacement based energy to reduce computational complexity.

It is worth pointing out that the force terms corresponding to spring force and curve

force in terms of the new generalized coordinates become

$$\mathbf{f}_{s\alpha} = k(\mathbf{j}(u_0, v_0)\mathbf{G})^T(\mathbf{P}_{0\alpha} - \mathbf{j}(u_0, v_0)\mathbf{r}_{0\alpha}) - k(\mathbf{j}(u_0, v_0)\mathbf{G})^T(\mathbf{j}(u_0, v_0)\mathbf{G})\mathbf{R}_\alpha \quad (3.55)$$

$$\mathbf{f}_{c\alpha} = \int_{\mathbf{T}} k(s)(\mathbf{j}(u(s), v(s))\mathbf{G})^T (\mathbf{C}_\alpha(s) - \mathbf{j}(u(s), v(s))\mathbf{r}_{0\alpha}) ds \quad (3.56)$$

$$- \int_{\mathbf{T}} k(s)(\mathbf{j}(u(s), v(s))\mathbf{G})^T (\mathbf{j}(u(s), v(s))\mathbf{G}) ds \mathbf{R}_\alpha \quad (3.57)$$

Therefore, in general, we can write the collective force term as

$$\mathbf{f}_{r\alpha} = \mathbf{V}_{0\alpha} - \mathbf{K}_f \mathbf{R}_\alpha \quad (3.58)$$

where $\mathbf{K}_f$ is symmetric and semi-positive definite. If we move all the generalized coordinates to the left, (3.50) changes into

$$\mathbf{M}_r \ddot{\mathbf{R}}_\alpha + \mathbf{D}_r \dot{\mathbf{R}}_\alpha + (\mathbf{K}_r + \mathbf{K}_f) \mathbf{R}_\alpha = \mathbf{V}_{0\alpha} + \mathbf{g}_{r\alpha}, \alpha = x, y, z \quad (3.59)$$

which is the ultimate dynamic equation that describes the evolution of E-PD-T-splines.

3.5 Analytical solution

Although (3.59) contains 3 individual equations, the coefficient matrices are the same. Compared with (3.27), the dimension is further reduced owing to separation and constraints. So numerical solution such as Runge-Kutta, suffice to satisfy the efficiency requirement even when dealing with large models. But as mentioned above, numerical solution has the inherent drawbacks for PD-T-splines. By the virtue of weights freezing, (3.50) are a typical second order linear ODEs. For the past decades, intensive researches have been concentrated on this issue which aim to diagonalize such system. In general, diagonalization can speed up the numerical integration. In our case, it also means an analytical solution, because the homogeneous term in (3.59) is constant. In the following, we will show how to derive

the closed form solution for (3.59), which completely suits the requirement of PD-T-splines modeling.

The three equations have the following common form

$$\mathbf{M}\ddot{\mathbf{p}} + \mathbf{D}\dot{\mathbf{p}} + \mathbf{K}\mathbf{p} = \mathbf{f} \quad (3.60)$$

where $\mathbf{M}, \mathbf{D}, \mathbf{K}$ are symmetric and positive definite, and $\mathbf{f}$ is constant. (3.60) is the underlying equation in many engineering applications including the fields of mechanical and electrical oscillation. According to ODE theory, we suffice to solve its homogeneous form

$$\mathbf{M}\ddot{\mathbf{p}} + \mathbf{D}\dot{\mathbf{p}} + \mathbf{K}\mathbf{p} = \mathbf{0} \quad (3.61)$$

and find a special solution. It is well known that a vibration system is coupled with independent vibration modes. From the physical point of view, the most ideal way is to decompose all those coupling modes, as it is easy to obtain their vibration behavior. Decoupling means to establish a transformation in the configuration space such that simultaneously diagonalize the $\mathbf{M}, \mathbf{D}, \mathbf{K}$ into diagonal matrices $\mathbf{M}_d, \mathbf{D}_d, \mathbf{K}_d$. Thereafter, the original vibration system (3.60) is transformed into

$$\mathbf{M}_d\ddot{\mathbf{q}} + \mathbf{D}_d\dot{\mathbf{q}} + \mathbf{K}_d\mathbf{q} = \mathbf{0} \quad (3.62)$$

which describes m individual vibration modes. Each of them can be expressed as

$$\mathbf{q}_i(t) = c_i e^{\alpha_i t} \cos(\omega_i t + \phi_i) \quad (3.63)$$

3.5.1 Modal analysis

Modal analysis is a popular technique for this purpose. Making use of the symmetry and positiveness of $\mathbf{M}$ and $\mathbf{K}$, we can simultaneously diagonalize them through solving a gen-

eralized eigenvalue problem

$$\mathbf{K}\mathbf{u} = \lambda\mathbf{M}\mathbf{u} \quad (3.64)$$

The resultant eigenvalues λ_i are real and positive, and the corresponding eigenvectors $\mathbf{u}_i$ are also real orthogonal with respect to either $\mathbf{M}$ or $\mathbf{K}$. Define

$$\begin{aligned} \mathbf{U} &= [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m] \\ \mathbf{\Omega} &= \text{diag}[\lambda_1, \lambda_2, \dots, \lambda_m] \end{aligned} \quad (3.65)$$

Upon normalization, the orthogonality of the eigenvectors can be expressed as

$$\begin{aligned} \mathbf{U}^T \mathbf{M} \mathbf{U} &= \mathbf{I} \\ \mathbf{U}^T \mathbf{K} \mathbf{U} &= \mathbf{\Omega} \end{aligned} \quad (3.66)$$

Modal analysis is to perform the following transformation

$$\mathbf{p} = \mathbf{U}\mathbf{q} \quad (3.67)$$

Then (3.61) yields

$$\ddot{\mathbf{q}} + (\mathbf{U}^T \mathbf{D} \mathbf{U}) \dot{\mathbf{q}} + \mathbf{\Omega} \mathbf{q} = \mathbf{0}. \quad (3.68)$$

If $\tilde{\mathbf{D}} = \mathbf{U}^T \mathbf{D} \mathbf{U}$ is also a diagonal matrix, we successfully diagonalize the original equation, and can easily get its closed form solution. We regard the damping matrix $\mathbf{D}$ which can be diagonalized by modal analysis as proportional damping or classical damping. In 1894, Rayleigh [89] presented that if

$$\mathbf{D} = \alpha \mathbf{M} + \beta \mathbf{K} \quad (3.69)$$

the system is proportional damped, but this is merely a sufficient condition. Caughey and O'Kelly [6] gave a necessary and sufficient condition

$$\mathbf{D}\mathbf{M}^{-1}\mathbf{K} = \mathbf{K}\mathbf{M}^{-1}\mathbf{D} \quad (3.70)$$

under which it is a proportional damping. In our case, the three matrices are calculated using (3.51). As the damping ratio can be chosen arbitrary, (3.70) generally can not be satisfied. Therefore, modal analysis does not suit our case.

3.5.2 State-space approach

Another approach in dealing with general damped system is to recast (3.60) into state space,

$$\begin{pmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{pmatrix}' = \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{pmatrix} \quad (3.71)$$

Denote

$$\mathbf{A} = \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{D} \end{pmatrix} \quad (3.72)$$

The solution to (3.60) can be expressed as

$$\mathbf{q} = (\mathbf{I} \ \mathbf{0}) \mathbf{e}^{\mathbf{A}t} \quad (3.73)$$

where $\mathbf{e}^{\mathbf{A}t}$ refers to matrix exponential. [70] reviewed the common ways to evaluate matrix exponential. Modeling with PD-T-splines will repeatedly invoke evaluation of (3.73) in order to get the realtime feedback of the dynamic evolution. Therefore, the matrix decomposition approach is the most suitable for our use. It involves one-time matrix decomposition

$$\mathbf{A} = \mathbf{S}\mathbf{B}\mathbf{S}^{-1} \quad (3.74)$$

The objective of the decomposition is to make $\mathbf{e}^{\mathbf{B}t}$ easy to compute while making $\mathbf{S}$ well conditioned so that the computation is stable. For the detailed numerical issues, please refer to [70]. In general, this approach can produce stable analytical solution. However, such state-space approach does not prevail in practicing engineers for two reasons. First, the twice of the number of degrees of freedom make it computational more involved. Secondly,

the most important, it lacks physical insight, which makes it impossible for the further analysis.

3.5.3 Phase synchronization

The last two sections present two typical ways to solve (3.60). Modal analysis is advantageous in expressing the physical insights, while state-space approach gives us a complete stable solution. Our objective is to absorb both of their advantages, but avoid their drawbacks. The underlining basic idea is quite similar as phase synchronization [58, 59, 73, 74, 40], but we modify it to best suit our purpose.

Quadratic eigenvalue problem

As stated above, the vibration system consists of independent vibration modes. Postulate the vibration mode is

$$\mathbf{p} = \mathbf{v}e^{\lambda t} \quad (3.75)$$

where λ is scalar value, and $\mathbf{v}$ is an n dimensional vector. Substituting it into (3.61), we obtain a quadratic eigenvalue problem

$$(\mathbf{M}\lambda^2 + \mathbf{D}\lambda + \mathbf{K})\mathbf{v} = \mathbf{0} \quad (3.76)$$

The eigenvalue λ are the roots of the polynomial of degree $2n$

$$|\mathbf{M}\lambda^2 + \mathbf{D}\lambda + \mathbf{K}| = 0 \quad (3.77)$$

Due to the fact that the coefficient matrix is real, the complex roots and their corresponding eigenvectors appear in conjugate pairs. If the roots are all distinct, the general solution can be expressed by

$$\mathbf{p} = \sum_{i=1}^{2n} c_i \mathbf{v}_i e^{\lambda_i t} \quad (3.78)$$

Assumptions

In general, we cannot guarantee the distinction of the eigenvalues. Some of them may have multiplicity larger than one. Two possible cases should be classified. For all eigenvalues, if its algebraic multiplicity is equal to the corresponding number of independent eigenvectors, such system is regarded as semi-simple. The general solution to (3.78) is still valid. Otherwise, it is defective. For a defective system, the coefficient c_i may involve polynomial of time t , which makes things complicated. In fact, as stated in [58, 59], the probability that (3.60) becomes defective is zero. Moreover, defectiveness is very sensitive to matrix perturbation. That means if it becomes defective, a small perturbation can make it semi-simple. Hence, in the following, we assume the system is semi-simple, thus (3.78) always applies.

Solution

To obtain the solution (3.78), we are required to solve the quadratic eigenvalue problem (3.76). Many software packages provide functions to deal with it, such as the `polyeig` function in Matlab. Therefore, the solution in (3.78) is very stable and efficient. However, as it consist of complex mode which is not physical excitable, it still diminishes the physical insight as the state-space approach. Note that the eigenvalue and corresponding eigenvector appear in conjugate pairs. We can rearrange (3.78) into

$$\mathbf{p} = \sum_{i=1}^o (c_i \mathbf{v}_i e^{\lambda_i t} + \overline{c_i} \overline{\mathbf{v}_i} e^{\overline{\lambda_i} t}) + \sum_{i=2o+1}^{2n} c_i \mathbf{v}_i e^{\lambda_i t} \quad (3.79)$$

Here we use another fact that the coefficients are also conjugate owing to the real initial condition. Compared with that in [58, 59], we do not pair the real modes. In [58, 59], in order to decouple the system uniformly, the real modes are paired to represent a virtual complex vibration. Our goal is to obtain a solution with physical meaning. As the real mode already represents a decaying non-oscillatory mode, we merely need to tackle the complex

modes. Next, let's focus on a single complex pair

$$\mathbf{s}_j(t) = c_j \mathbf{v}_j e^{\lambda_j t} + \overline{c_j \mathbf{v}_j} e^{\overline{\lambda_j} t} \quad (3.80)$$

According to the analysis in [58, 59], $\mathbf{s}_j$ exactly stands for an real vibration mode. But compared with the normal mode, the system components do not pass through the equilibrium at the same time due to their different phase angles. Phase synchronization aims to eliminates the difference through a time-dependent transformation. For our application, (3.79) already presents an efficient solution, whose separate component represents a single vibration mode. To avoid complex arithmetic, denote

$$\begin{aligned} \lambda_j &= \alpha_j + i\omega_j \\ \mathbf{v}_j &= \mathbf{A}_j + i\mathbf{B}_j \\ c_j &= c_{1j} + ic_{2j} \end{aligned} \quad (3.81)$$

(3.80) can be expressed as

$$\mathbf{s}_j(t) = e^{\alpha_j t} \cos(\omega_j t) (c_{1j} \mathbf{A}_j - c_{2j} \mathbf{B}_j) - e^{\alpha_j t} \sin(\omega_j t) (c_{2j} \mathbf{A}_j + c_{1j} \mathbf{B}_j) \quad (3.82)$$

from which the natural vibration can be observed.

3.5.4 Analytical solution

In order to get the final solution to (3.60), we have to solve another special solution and determine the coefficients c_{1j}, c_{2j} . The coefficients are dependent on the special solution. Generally speaking, if the force term is an arbitrary function of time t , it is difficult to compute a special solution. Fortunately, in our situation, $\mathbf{f}$ is constant.

$$\mathbf{K}\mathbf{p} = \mathbf{f} \quad (3.83)$$

gives a simple one. Moreover, (3.83) exactly represents the equilibrium shape. When the intermediate shapes are ignored, (3.83) directly gives the final shape that satisfies the user requirement. In other words, E-PD-T-splines include energy minimization models as their subspace. After finding the special solution $\mathbf{q}_0$, the coefficients can be determined according to the initial condition $\mathbf{p}_0, \dot{\mathbf{p}}_0$. Let

$$\begin{aligned}
 \mathbf{c}_{\text{img}} &= [\cdots, c_{1j}, c_{2j} \cdots]_{j=1}^o{}^T \\
 \mathbf{c}_{\text{real}} &= [\cdots, c_j, \cdots]_{j=2o+1}^{2n}{}^T \\
 \mathbf{H}_{\text{img1}} &= [\cdots, \mathbf{A}_j, -\mathbf{B}_j, \cdots]_{j=1}^o \\
 \mathbf{H}_{\text{img2}} &= [\cdots, \alpha_j \mathbf{A}_j - \omega_j \mathbf{B}_j, -\omega_j \mathbf{A}_j - \alpha_j \mathbf{B}_j, \cdots]_{j=1}^o \\
 \mathbf{H}_{\text{real}} &= [\cdots, \mathbf{v}_j, \cdots]_{j=2o+1}^{2n} \\
 \Lambda_{\text{real}} &= \text{diag}([\cdots, \lambda_j, \cdots]_{j=2o+1}^{2n})
 \end{aligned} \tag{3.84}$$

The coefficients $\mathbf{c}_{\text{img}}, \mathbf{c}_{\text{real}}$ are obtained through

$$\begin{pmatrix} \mathbf{H}_{\text{img1}} & \mathbf{H}_{\text{real}} \\ \mathbf{H}_{\text{img2}} & \mathbf{H}_{\text{real}} \Lambda_{\text{real}} \end{pmatrix} \begin{pmatrix} \mathbf{c}_{\text{img}} \\ \mathbf{c}_{\text{real}} \end{pmatrix} = \begin{pmatrix} \mathbf{p}_0 - \mathbf{q}_0 \\ \dot{\mathbf{p}}_0 \end{pmatrix} \tag{3.85}$$

After getting the coefficients, the final solution to (3.60) can be expressed as

$$\mathbf{p} = \sum_{j=1}^{2n} \mathbf{s}_j(t) + \mathbf{q}_0 \tag{3.86}$$

3.5.5 Modal reduction

Note that when the general damping reduces to the classical damping, the eigenvectors are all real, which means $\mathbf{B}_j = \mathbf{0}$. The paired mode (3.80) becomes

$$\mathbf{s}_j(t) = \mathbf{e}^{\alpha_j t} (c_{1j} \cos(\omega_j t) - c_{2j} \sin(\omega_j t)) \mathbf{A}_j \tag{3.87}$$

It indicates that all the system components pass the equilibrium positions simultaneously. Therefore, in this case the method reduces to modal analysis. In other words, our solving strategy generalizes Modal Analysis. As one of the merits by expressing the solution in terms of separate vibration modes, we can discard the modes that are not needed. As a consequence, the computational complexity can be further reduced. In PD-T-splines, the dynamic evolution provides the candidate shapes transiting from the initial shape to the final equilibrium shape. To some extent, the final equilibrium represents the user design intention. Therefore, the intermediate shapes ought not to be too far different. Based on this consideration, the modes with high vibration frequency are not preferable, since too high frequency produce exaggerated jiggly appearance. In our application, we may choose 20% of the oscillatory modes. Furthermore, the non-oscillatory modes merely generate translation effect, thereby having little impact for the intermediate shapes. Upon these two discussions, the solution may be chosen as

$$\mathbf{p} = \sum_{j=1}^r \mathbf{s}_j(t) + \mathbf{q}_0 \quad (3.88)$$

where the coefficient are also computed using the initial condition

$$\begin{pmatrix} \mathbf{H}_{\text{img1}}[:, 1 : 2r] \\ \mathbf{H}_{\text{img2}}[:, 1 : 2r] \end{pmatrix} \mathbf{c}_{\text{img}}[1 : 2r] = \begin{pmatrix} \mathbf{p}_0 - \mathbf{q}_0 \\ \dot{\mathbf{p}}_0 \end{pmatrix} \quad (3.89)$$

which is an over-determined system. Here $\mathbf{H}_{\text{img1}}[:, 2r]$ refers to the submatrix consisting of the first $2r$ columns in $\mathbf{H}_{\text{img1}}$, and $\mathbf{c}_{\text{img}}[1 : 2r]$ represents the subvector consisting of the first $2r$ elements in $\mathbf{c}_{\text{img}}$. We compute the least square solution to (3.89). However, it's likely to generate a residue between $\mathbf{p}(0)$ and $\mathbf{p}_0$. Change of the initial shape is not desirable. To remedy this, we add a non-oscillatory mode into the solution,

$$\mathbf{p} = \sum_{j=1}^r \mathbf{s}_j(t) + (\mathbf{p}_0 - \tilde{\mathbf{p}}_0)e^{-t} + \mathbf{q}_0 \quad (3.90)$$

where $\tilde{\mathbf{p}}_0$ is the residue. Note that the additional mode only serves to maintain the initial shape. (3.90) presents the solution using modal reduction. Compared with (3.86), the initial and equilibrium states are the same. But (3.90) produces only the vibration modes of interest thereby requiring less computation, which is usually 20%.

3.6 Implementation

In order to apply E-PD-T-splines in practical applications, we ought to first calculate the coefficient matrices in (3.51). Note that constraints will be frequently changed during design process, but the required matrices in (3.28) remain. As soon as the topology of T-mesh does not change, they only have to be calculated once. In the following, we focus on how to numerically compute the matrix integrands in (3.28). We adopt the finite element method (FEM) for this purpose.

3.6.1 Extracting Bézier patches

The first task to apply FEM is to determine the type of element. Due to the existence of T-junctions in T-splines, we can not directly choose the face of the T-mesh as the finite element. Take Figure 3.1(a) as an example, where two close edges $\mathbf{P}_0\mathbf{P}_6$ and $\mathbf{P}_1\mathbf{P}_7$ indicate the multiple knots. In this case, the blue transparent face of T-mesh is non-rectangular, which does not support the following numerical integration. This is because popular approaches, such as Gaussian Quadrature usually requires rectangular parametric domain. We can split those polygons into rectangular faces. However, they may be still not suitable for finite element. Take the rectangle $\mathbf{P}_3\mathbf{P}_4\mathbf{P}_8\mathbf{P}_9$ in Figure 3.1(b) as an example. The blending function associated with $\mathbf{P}_6$ (refer to the green line segments) has only continuity of order 1 along the edge $\mathbf{Q}_0\mathbf{Q}_1$. Therefore, the surface patch would also be continuous of order 1. However, for most existing numerical methods, the integrand throughout the parametric domain should be sufficiently continuous in order to ensure precision. Therefore, simple face of T-mesh does not suit the need of finite element.

(a) Non-rectangular face

(b) Rectangular face

(c) Bézier patch

Figure 3.1: Possible element types

Alternatively, we can choose the Bézier patches as elements like the patch $P_4P_8Q_0Q_1$ shown in Figure 3.1(c). Bézier patches are always rectangular. Most importantly, the surface has continuity of order infinity within a Bézier patch. Then the remaining issue is how to extract Bézier patches from a T-spline surface. In [99], the authors proposed a way to convert T-spline surface into Bézier representation for tessellation purpose. They suggested extending all T-junctions by two bays. However, in some special cases, such operation may fail to extract the correct Bézier patches. For example, Figure 3.2(a) shows the results extending each T-junction by two bays. However, the blue transparent face in Figure 3.2(b) is not the correct Bézier patch, because the blending function associated with P_0 is only continuous of order 2 along edge Q_0Q_1 . To ensure the correctness, we may extend all the T-junction to the boundary of domain. But it will cause the correct Bézier patches will also be split into small pieces, which increases the number of elements. As the computational intensity is relatively equal in each element, it will also increase the overall computational cost.

Figure 3.2: Failure case by simply extending all the T-junction by two bays

Based on the above discussion, a suitable approach should be able to extract all the correct Bézier patches while minimizing its number. We propose a two phase strategy. First, we still follow the method in [99], because this approach will not increase the excess Bézier patches. After this, we need a postprocessing. For each resultant patch, we traverse all the control points, and check whether it involves knot within the interior of this patch.

Specifically, we traverse the U and V knot vector respectively. If there exists U knot within this patch, we use its corresponding knot line (as the violet one in Figure 3.3) to split it, and the V knot case follows suit. In this way, we can extract the Bézier patches exactly.

Figure 3.3: Splitting the false Bézier patch

3.6.2 Data structure

After determining the type of finite element, we also define an element data structure associated with it, which contains its geometric specification together with its physical property. To be specific, the element structure contains all the control points whose support of associated blending function covers the corresponding Bézier patch. Each influencing control point stores the index to the whole control point list, facilitating the final global matrix assemblage. Note that neighboring elements will share some of the control points. It means the shared ones will have multiple control points indexing to them. Besides, an elemental mass, damp, and stiff matrix are also allocated in each element as well as the quantities, i.e. mass density $\mu(u, v)$, damping density $\lambda(u, v)$, elasticity $\alpha_{i,i}, \beta_{i,j}$. These physical quantities may be represented as analytical functions or arrays of some sampled values.

3.6.3 Calculation of elementary matrices

To numerically evaluate the integral of the elementary matrices, we adopt Gaussian Quadrature for its precision with less evaluations. We shall explain the computation of elemental

stiff matrix. The computation of mass and damping matrices follows in a similar way. Assume the domain of the Bézier patch is $[u_0, u_1] \times [v_0, v_1]$. The entries can be expressed as

$$k_{i,j} = \int_{u_0}^{u_1} \int_{v_0}^{v_1} f_{i,j}(u, v) du dv \quad (3.91)$$

where

$$f_{i,j} = \alpha_{1,1} \frac{\partial N_i}{\partial u} \frac{\partial N_j}{\partial u} + \alpha_{2,2} \frac{\partial N_i}{\partial v} \frac{\partial N_j}{\partial v} + \beta_{1,1} \frac{\partial^2 N_i}{\partial u \partial u} \frac{\partial^2 N_j}{\partial u \partial u} + \beta_{1,2} \frac{\partial^2 N_i}{\partial u \partial v} \frac{\partial^2 N_j}{\partial u \partial v} + \beta_{2,2} \frac{\partial^2 N_i}{\partial v \partial v} \frac{\partial^2 N_j}{\partial v \partial v} \quad (3.92)$$

Here N_i are the associated weighted blending functions defined in (3.5).

The integral can be approximately calculated by applying Gaussian Quadrature in 1D interval twice. Gaussian Quadrature first requires calculating two sets of Gauss weights a_g, b_h and abscissas u_g, v_h . The integral (3.91) can be approximated by

$$k_{i,j} = \sum_{g=1}^{N_g} \sum_{h=1}^{N_h} a_g b_h f_{i,j}(u_g, v_h) \quad (3.93)$$

Referring to equations (3.5), (3.92), (3.93), we notice that the computation mainly lies in the evaluation of blending functions on u_g, v_h . As in a B-spline surface, many blending functions may share the same U or V . If we directly evaluate the blending function on u_g, v_h , it will cause duplicated calculation. In our implementation, we first check whether the U vector or V vector is identical with the previous calculated ones to avoid such double calculation. It's also worth pointing out that when the displacement energy functional is used and ROI is chosen, we only have to calculate the entries that are selected. This is extremely useful when dealing with large models to achieve real-time interaction.

3.6.4 Dynamic update of local refinement

As mentioned in Section 2.2.2, local refinement is one of the most important attributes in T-splines. It will be frequently performed during interactive modeling. In this section, we try to incorporate such wonderful property into our PD-T-splines scheme. Once the

T-splines are refined, it means the increase of DOFs of our physical system as well as the change of the dynamic equation. In order to proceed with PD-T-splines, the equation (3.59) needs to be updated. The straightforward approach is to reestablish all the matrix after performing local refinement algorithm from scratch. It means the refined T-spline surface will be viewed as an absolutely new one, and one needs to reconstruct the finite elements from the beginning and reassemble the global matrix from the scratch. Apparently, this approach is not wise and economic. Because refinement is local, merely a small portion of elements will be affected. If we totally destroy the original structure and rebuild them, a lot of double calculations will occur. Further more, from our experiments if the number of control points exceeds 1000, the initialization of elements could take several seconds, and hence updating E-PD-T-splines in this way can not be done in real-time.

Alternatively, we propose to locally update those coefficient matrices. Referring to the definition in (3.51), we are required to recalculate the matrix integrals in (3.28), and the transformation in (3.47). Note that (3.47) merely involves one-time gaussian elimination, which is efficient. So next we mainly focus on updating the matrices in (3.28) after refinement. Our algorithm is composed by three phases.

(1) Update element geometry. This involves two tasks. First, after inserting extra points, some of the elements will no longer be Bézier patches, we have to delete those violated ones and specify the new ones. Second, even if the Bézier patch is still valid, the associated control point list may change, which is also required to be updated. Note that new Bézier patches are resulted from splitting the original Bézier patches by new inserted knot lines. Hence, as to the first task we suffice to traverse all the new inserted control points. Then for each new control point, we traverse all the existing elements, and compare the support of the new control point with the domain of the incident Bézier patches. We use the example in Figure 3.4 to illustrate the different cases and treat them respectively.

- No intersection. As shown in Figure 3.5, the new control point $\mathbf{P}_0$ has no influence on S_1 . No operation is required for this patch.

Figure 3.4: Another local refinement example. The squares with red boundary are the new inserted points

Figure 3.5: The support of the blending function associated with P_0 has no intersection with the Bézier patch S_1

- The support of the control point covers the Bézier patch, and the knot line does not cut the patches. As shown in Figure 3.6, P_0 has influence on S_2 , but S_2 is still a Bézier patch after refinement. For this case, we merely have to append P_0 to its control point list.

Figure 3.6: the support of the blending function covers the Bézier patch, but the knot lines does not cut it.

- The knot line cut the patches. As shown in Figure 3.7, S_3 is no longer a Bézier patches. Using the approach illustrated in Figure 3.3, we can split the S_3 into several new ones according to the cutting knot lines. Meanwhile, the previous calculated quantities, such as B-spline basis evaluations and its derivatives as well as the elemental matrix should be deleted. And the control points list is delivered to the new created patches. Meanwhile P_0 is also inserted into the list.

Figure 3.7: the knot lines cut Bézier patch

After all the above processes are completed, we get the exact Bézier patches, but the associated control point list may not be correct. As shown in Figure 3.8, P_1 used to belong to the influencing control point list of S . After refinement, The support of its associated blending function no longer intersect with S . Therefore, we are still required to traverse all the elements to delete those redundant ones. Once the control point is deleted from its list, so do the corresponding quantities such as the B-spline basis evaluation and the matrix entries related to it.

Figure 3.8: The associated control point may no long influence the Bézier patch after refinement.

(2) Update elemental physical quantities. In this phase, the elemental matrices will be locally updated. If the element is a newly created one, we simply apply the approach in Section 3.6.3 to compute all the necessary quantities and matrices. Since refinement is local, most of the elements will remain the same. We choose to locally update the matrix entries that are changed rather than recalculate them from the scratch, thereby at best avoiding recalculation. To update the corresponding matrices, first we ought to carefully identify those control points, whose related entries should be recalculated. There are three types of control points that need to be treated.

- the newly inserted ones.
- the ones whose blending function change.
- the ones whose weight change.

Figure 3.9: The weight changed case

The first two situations can be easily observed. Figure 3.9 shows an example of the third case, where the associated blending function remains while the weight changes. A new control point P_0 (Figure 3.8(a)) is intended to be inserted into the T-mesh. According to the refinement algorithm in Section 2.2.2, the knot $u = u_3$ will be inserted into the blending function centered at (u_4, v_4) (P_1 in Figure 3.8(b)), producing two scaled blending functions: $c_2 N[u_1, u_2, u_3, u_4, u_5](u) N[v_1, v_2, v_4, v_5, v_6](v)$ and $d_2 N[u_2, u_3, u_4, u_5, u_6](u) N[v_1, v_2, v_4, v_5, v_6](v)$. But the V vector of the blending function $N[u_2, u_3, u_4, u_5, u_6](u) N[v_1, v_2, v_4, v_5, v_6](v)$ is still guilty of Violation 1. Therefore, knot v_3 is further required to be inserted into this blending function, splitting into another 2 scaled blending functions: $c'_2 N[u_2, u_3, u_4, u_5, u_6](u) N[v_1, v_2, v_3, v_4, v_5](v)$ and $d'_2 N[u_2, u_3, u_4, u_5, u_6](u) N[v_2, v_4, v_3, v_5, v_6](v)$. The former blending function is exactly the one associated with the control point P_2 . Therefore, as a result of the refinement process, the control point P_2 will be added by extra $c'_2 d_2 P_1$, thereby modifying its weight. For updating matrices, we simply follow Section 3.6.3: First, compute the B-spline basis function evaluation as well as the derivatives of the above 3 type control points; Then use these quantities to compute matrix entries using Gaussian quadrature.

(3) Update global matrices. The final global matrices is updated accordingly by reassembling all the entries corresponding to the above 3 types of control points.

3.7 Applications

This section presents several applications of PD-T-splines related to solid rounding and optimal surface fitting. E-PD-T-splines instead of the full version of PD-T-splines are adopted for their efficiency.

3.7.1 Solid rounding

Solid rounding is a common operation in CAM, which is usually achieved through enforcing geometric continuity on the fillet between two or more patches. Due to physical nature, the dynamic evolution of PD-T-splines is actually a process of minimizing the internal deformation energy subject to geometric constraints. Therefore, PD-T-splines provide a natural solution to the solid rounding problem.

Figure 3.10 demonstrates the rounding of a trihedral corner of a cube, which is represented using a PD-T-spline surface with 169 control points. The sharp edges and corners are produced using multiple control points. Here we propose a different approach to realize the rounding compared with that in [122]. Rather than imposing the positional and normal constraints along the boundary, we introduce the use of curve force. That is, we first specify the desirable boundary curves as shown in Figure 3.10(b), which serve as the target curves for the curve forces. With these forces imposition, PD-T-splines evolve in a smooth and natural way. Figure 3.10(c) records the evolution generated using our analytical solution in Section 3.5. And the intermediate shape during evolution is chosen as the desirable one (as shown in Figure 3.10(d)). This example also demonstrates the advantage of dynamic modeling over the traditional energy minimization techniques, which focus on the final equilibrium static state only. Obviously, in this example if the minimization techniques are adopted, users will struggle in adjusting the parameters and constraints, because the equilibrium is over smoothed (see Figure 3.10(b)). In contrast, within Efficient PD-T-splines, we can invoke arbitrary forward and backward evolution for comparison till the ideal shape is found. We also implement the method in [122] to achieve the rounding which is shown

Figure 3.10: Rounding a trihedral vertex

in Figure 3.11. As suggested, we also freeze the boundary control points that are far from the sharp edges. But as seen in Figure 3.11(b), it is rather difficult to achieve the boundary continuity. By contrast, curve force provides us more intuitive and convenient way to do this job. As for specification of the desired target curves, we will investigate it in the next chapter.

Figure 3.11: Rounding method in [122]

Figure 3.12 presents the rounding of a bevel joint. The joint is represented by a PD-T-splines with 121 control points. The initial configuration is shown in Figure 3.12(a). And the final desired rounded surface (Figure 3.12(d)) is also chosen from the dynamic evolution process in Figure 3.12(c).

3.7.2 Adaptive optimal T-spline fitting

PD-T-splines are also applicable to fit scattered data, which is a useful technique in reverse engineering. PD-T-splines can achieve an optimal solution in terms of minimizing the deformation energy. In our case, the input is a triangular mesh as seen in Figure 3.13(a). The interpolation of data is interpreted as a Hookean spring that pulls the surface point of PD-T-splines to the scattered data. In this application, we ignore the intermediate dynamic process. When the PD-T-splines reaches equilibrium, its generalized coordinates satisfy

$$(\tilde{\mathbf{K}}_q + \tilde{\mathbf{K}}_{sq})[\mathbf{Q}_x, \mathbf{Q}_y, \mathbf{Q}_z] = \mathbf{V}_{sq} \quad (3.94)$$

Figure 3.12: Rounding a bevel joint.

Figure 3.13: Illustration of adaptive T-spline fitting using PD-T-splines

where $\tilde{\mathbf{K}}_q$ is defined in (3.22), and

$$\begin{aligned}\tilde{\mathbf{K}}_{sq} &= \sum_{i=0}^l \mathbf{j}(u_i, v_i)^T \mathbf{j}(u_i, v_i) \\ \mathbf{V}_{sq} &= \sum_{i=0}^l \mathbf{j}(u_i, v_i)^T \mathbf{p}_i^T\end{aligned}\tag{3.95}$$

Here $\mathbf{p}_i$ represent the scattered data points, (u_i, v_i) are the PD-T-splines parametric coordinates associated with the data points, and $\mathbf{j}$ refers to the Jacobian matrix of E-PD-T-splines defined as (3.26). Generally, it is difficult to determine the optimal parameters for (u_i, v_i) . Similar to [131], we employ the mesh parameterization in [20] to obtain their associated parameters on PD-T-splines. Note that as the coefficient matrix of (3.94) is positive definite, we employ the pre-conditioned conjugate gradient method to obtain an iterative solution [82, 111]. However, such method is dependent on the initial PD-T-splines. If too few DOFs are involved, it's likely to obtain an unsatisfactory result as shown in Figure 3.13(b). It's impossible to determine the best configuration of PD-T-splines beforehand. In Section 3.6.4, we have investigated local refinement of PD-T-splines. So we proceed our T-spline fitting in an iterative manner. For each step, after obtaining the incident optimal PD-T-splines, we check the error, which is defined as the largest distance between the scattered points and the surface. If it does not pass the test, we perform the local refinement at the region where the test has been failed. Then PD-T-splines is updated through the method in Section 3.6.4. The process stops when all the scattered points pass the check. Figure 3.13 illustrates the iterative process to fit a triangular mesh. It is worth pointing out that our T-spline fitting method is equivalent to [131]. The difference lies in the the update of the matrix $\tilde{\mathbf{K}}_q$. In our case, we choose to locally update the entries only when necessary, which thus is more efficient and less computationally involved.

Chapter 4

Physically based Composite Curve for T-spline Editing

4.1 Introduction

With the introduction of computerized sketching tools, curve handles for surface modification are shown to be more effective and intuitive for their ease in expressing users modeling intention. In spline domain, the use of arbitrary curves in modification mainly focuses on constraining the curve on the surface to be fixed during the deformation. There are only a few works considering to use the curves as the deformation tools. Those approaches usually comprise three steps: (1) sketch a curve handle on the surface, (2) generate a target curve in space, and (3) match the two curves and deform the surface. [69, 144] proposed to generate the target curve through sketching. The matching is achieved through converting the composite curve into B-spline representation, and the surface is deformed to interpolate the target curve in least square sense. The restriction of B-spline domain curve and the requirement of explicit conversion prevent them from being popularized. As we know that some useful types of curves such as circle can not be expressed using a B-spline representation exactly. Moreover, for NURBS or T-spline surface, the explicit conversion is also complicated.

In this chapter, we introduce the use of general curve handle into PD-T-splines in order to enhance its modeling tools. The general curve handle is the composition of a 2D domain curve and the T-spline surface. We support arbitrary type of domain curve, thereby providing more flexibility for users to express their modeling purpose. Our approach also follows the above three-step strategy. However, rather than sketching, we propose to interactively deform the general curve handle to obtain the desired target curve. The surface deformation is realized by introducing the new curve force into PD-T-splines as presented in last chapter. As a merit, the complicated explicit conversion is easily avoided, and matching is automatically established through the identical parameterization. It means that the two drawbacks of the above-mentioned methods can be remedied. In order to provide a natural and intuitive approach to interactively deform the general curve handle, we develop a new physically based model directly based on the composite representation, producing physically based dynamic composite curve (or PBDC curve). Through analysing the dependency of the composite blending functions, we further regularize it to obtain a more stable and efficient physically based model.

The novelty of this chapter lies in the following aspects:

(1) We propose a novel approach to compute the dependency of a set of composite blending functions based on constant matrix operation. It is applicable to arbitrary type of domain curves.

(2) We propose a new physically based dynamic model based on the composite representation. We directly use the DOFs in its parent surface as the generalized coordinates, thus avoiding the difficult explicit composition conversion.

(3) Observing the risk of rank deficiency, we further develop a method to regularize the proposed physical model based on the analysis of the composite blending functions. The resultant dynamic equation is always regular and analytically solvable.

(4) This new dynamic generalization of composite curves is introduced into PD-T-splines to achieve curve based editing. As the PBDC curve supports an arbitrary type of the 2D domain curve, our curve based surface editing approach is more flexible than the

previous ones.

(5) We seamlessly incorporate the local refinement of T-splines into PBDC curves through locally updating the matrix coefficients. By performing the same local refinement, we can maintain the topological consistency between the parent surface of the composite curve and the PD-T-splines. As a result, we avoid the updating of the force term, making the curve-based editing more efficient.

4.2 Theory

We begin with some definitions and notations that will be intensively retrieved in the following.

4.2.1 Notations

Consider an ordered set $\mathfrak{F}$,

$$\mathfrak{F} = \{F_i | i = 0, \dots, k\} \quad (4.1)$$

whose cardinality is denoted by $|\mathfrak{F}| = k + 1$. Throughout this chapter, index is counted from 0. Given an index set Φ , which is a subset of

$$\mathbf{N}_k = \{0, 1, \dots, k\} \quad (4.2)$$

$\mathfrak{F}[\Phi]$ means a subset of $\mathfrak{F}$ defined by

$$\mathfrak{F}[\Phi] = \{F_i | i \in \Phi\} \quad (4.3)$$

This notation is also extended into matrix case. If $\mathbf{H}$ is an $m \times n$ matrix, and $\Xi \subseteq \mathbf{N}_{m-1}$ and $\Phi \subseteq \mathbf{N}_{n-1}$ are two indices sets, then $\mathbf{H}[\Xi, \Phi]$ represents a sub-matrix of $\mathbf{H}$ consisting of all the entries in $\mathbf{H}$ with the row indices in Ξ and column indices in Φ . If $\Xi = \mathbf{N}_{m-1}$,

then we can replace Ξ with $\cdot$. So does Φ . This means

$$\mathbf{H} = \mathbf{H}[:, :] \quad (4.4)$$

Note when the matrix becomes a vector, without ambiguity the dimension with only one index is usually omitted.

Referring to the definition in (4.1), the ordered set is very similar to a vector. If we assemble its elements according to the indices order, it forms a vector, which is usually denoted by

$$\mathbf{J}_{\mathfrak{F}} = (F_0, F_1, \dots, F_k) \quad (4.5)$$

The subscription is used to indicate the relation with the set $\mathfrak{F}$. Then according to the definition above, we have

$$\mathbf{J}_{\mathfrak{F}[\Phi]} = \mathbf{J}_{\mathfrak{F}}[\Phi] \quad (4.6)$$

4.2.2 Composite curve

Figure 4.1: A simple example of composite curve. (a) The purple curve represents the composite curve. The red control points in the original parent surface represents the ones that have influence on the composite curve, while the light blue ones are the others. (b) The red curve is the 2D curve in the parametric domain that defines the composite curve. And the influencing control points are colored red.

Given a 2D curve $D(s) = (u(s), v(s))$ in the parametric domain as shown in Figure 4.1(b). Composite curve is produced through mapping $D(s)$ onto the T-spline surface, which can be expressed as

$$\mathbf{S}_c(s) = \mathbf{T}(D(s)) = \frac{\sum_{i=0}^n B_i(u(s), v(s)) \omega_i \mathbf{P}_i}{\sum_{i=1}^n B_i(u(s), v(s)) \omega_i} \quad (4.7)$$

As shown in Figures 4.1(a) and 4.1(b), not all the control points in the parent surface have influence in defining the composite curve. If we denote the indices of those influencing control points as Ξ , the composite curve equation can be written as

$$\mathbf{S}_c(s) = \frac{\sum_{i \in \Xi} B_i(u(s), v(s)) \omega_i \mathbf{P}_i}{\sum_{i \in \Xi} B_i(u(s), v(s)) \omega_i}. \quad (4.8)$$

or in the homogeneous form,

$$\mathbf{S}_c(s) = \sum_{i \in \Xi} B_i(u(s), v(s)) (\omega_i \mathbf{P}_i^T, \omega_i). \quad (4.9)$$

From the definition, we can infer that as soon as $D(s)$ is specified, the geometry of composite curve is determined by the incident control points and weights in the parent surface. We call $B_i(u(s), v(s))$ the composite blending functions, which play a similar role as B-spline basis functions in NURBS curve definition. As far as we know, the property of the composite blending functions is seldom investigated. In fact, they are far different from B-spline basis functions. They even can not form a basis in general.

Consider a simple example as illustrated in Figure 4.2. The 2D domain curve $D(s)$ is an iso-curve

$$(u(s), v(s)) = (s, v'), s \in [u_1, u_5]$$

Then in this case, the corresponding composite curve is essentially a NURBS curve of

Figure 4.2: Composite curve reduces to a NURBS curve

degree 3, which can also be written as

$$\mathbf{S}_c(s) = \sum_{i=0}^6 B_{i,3}(s)(\omega_c \mathbf{Q}_{ic}^T, \omega_c), s \in [u_1, u_5] \quad (4.10)$$

where $B_{i,3}(s)$ are the cubic B-spline basis functions defining over the knot vector $[\cdots, u_i, \cdots]_{i=-2}^8$.

Denote the set of composite blending functions associated with the influencing control points in its parent surface by

$$\mathfrak{N} = \{B_i(u(s), v(s)) | i \in \Xi\} \quad (4.11)$$

and that of the B-spline bases by

$$\mathfrak{B} = \{B_{i,3}(s) | i = 0, \cdots, 6\} \quad (4.12)$$

Because the two curves in (4.8) and (4.10) coincide for the entire T-spline class defined over the pre-image in Figure 4.1(b), we have

$$\text{Dim}(\text{span}\{\mathfrak{N}\}) = \text{Dim}(\text{span}\{\mathfrak{B}\}) = 7. \quad (4.13)$$

On the other hand, $|\mathfrak{N}| = |\Xi| = 31$, based on which we can assert that the blending functions $\mathfrak{N}$ are linearly dependent.

Refer to the four control points highlighted in Figure 4.2(b). The associated local vectors in the u direction of the four points are all $[u_1, u_2, u_3, u_4, u_5]$. Accordingly, the associated four blending functions can be uniformly expressed as

$$B_{i_k}(s) = c_k B_{3,3}(s), k = 0, 1, 2, 3 \quad (4.14)$$

which are obviously linearly dependent.

The dependency of blending functions directly results in the redundancy of degree of freedoms (DOFs) in the definition, which will bring inconvenience in some applications. First of all, a straightforward drawback is the lose of uniqueness of the representation in (4.8). Furthermore, when the composite curve is used in multiple points interpolation as the application in sketch based modeling, the linear system is likely to be singular. Thing is getting worse, when physically based model is developed based on this representation. The final discrete dynamic equation is always singular, which causes trouble in both efficiency and stability issues.

4.2.3 Problem statement

Therefore, in order to apply a composite curve stably, we need to filter out the redundancy in these DOFs. We propose to find a basis $\mathfrak{b} = \{b_i(s) | i = 0, \dots, k\}$ for the composite blending functional space $\text{span}\{\mathfrak{N}\}$, and represent the composite curve in terms of the new basis functions, which can be characterized as

Problem 4.1. *Find a basis $\mathfrak{b}$ for $\text{span}\{\mathfrak{N}\}$, and compute the transformation such that*

$$\mathbf{J}_{\mathfrak{N}} = \mathbf{J}_{\mathfrak{b}} \mathbf{R} \quad (4.15)$$

As soon as the basis and transformation are obtained, the composite curve using the

basis can be expressed as

$$\mathbf{S}_c(s) = \mathbf{J}_b \mathbf{R} \mathbf{P}_\omega \quad (4.16)$$

where

$$\mathbf{P}_\omega = [\dots, [\frac{\omega_i \mathbf{P}_i}{\omega_i}], \dots]_{i \in \Xi}^T \quad (4.17)$$

If we denote

$$\mathbf{Q}_\omega = [\dots, [\frac{\omega_{b,i} \mathbf{Q}_i}{\omega_{b,i}}], \dots]_{i=0,\dots,k}^T = \mathbf{R} \mathbf{P}_\omega$$

the composite curve can be recast into

$$\mathbf{S}_c(s) = \mathbf{J}_b \mathbf{Q}_\omega = \sum_{i=0}^k \mathbf{b}_i(\omega_{b,i} \mathbf{Q}_i^T, \omega_{b,i}) \quad (4.18)$$

where $\mathbf{Q}_\omega$ exactly represents the degrees of freedom.

The remaining issue is how to find such bases for the composite functional space. As we know that when the 2D domain curve $D(s)$ is a B-spline curve, the composite curve is essentially a NURBS curve of high degree. Then a brute-force approach is to convert the composite curve into NURBS form as in [69], where

$$\mathbf{J}_{\mathfrak{N}} = \mathbf{J}_{\mathfrak{B}} \mathbf{R} \quad (4.19)$$

$\mathfrak{B}$ represents the set of resultant B-spline basis functions $\mathfrak{B} = \{B_{i,l}(s) | i = 0, \dots, m\}$. It is worth pointing out that the B-spline bases $\mathfrak{B}$ generally cannot serve as the bases for $\text{span}\{\mathfrak{N}\}$. By analyzing the transformation matrix $\mathbf{R}$, we can easily identify the dependency of the original composite blending functions as well as the rank of $\mathfrak{N}$. Then the linear independent sub set whose cardinality is $\text{rank}(\mathfrak{N})$ can serve as the basis for the blending functional space. However, the computation of the conversion into NURBS representation is rather complicated. More importantly, it has a prerequisite that $D(s)$ must be a B-spline curve. Even when the 2D domain curve is NURBS, such conversion is impossible not to mention an arbitrary curve.

4.2.4 Theorems

Instead of searching tedious conversion, we propose a novel approach based on some interesting observation. One of the most attractive feature of our approach is that it can apply to any arbitrary domain curve, which is obviously advantageous over the previous one.

Let's consider a more general functional set,

$$\mathfrak{F} = \{F_i(s) \mid F_i(s) \geq 0, F_i(s) \neq 0, s \in [a, b], F_i(s) \text{ is } C_0 \text{ continuous}, i = 0, \dots, m\} \quad (4.20)$$

Obviously, the composite blending functions $B_i(u(s), v(s))$ belong to this type of set. From the above analysis, we can actually find a basis among the generator $\mathfrak{F}$ instead of the entire spanned space. Then the corresponding problem is

Problem 4.2. *Find a indices set Φ , such that $\mathfrak{F}[\Phi]$ forms a basis for $\text{span}\{\mathfrak{F}\}$, and compute the dependency with respective to this basis,*

$$\mathbf{J}_{\mathfrak{F}} = \mathbf{J}_{\mathfrak{F}}[\Phi]\mathbf{R} \quad (4.21)$$

Our approach is based on the observation of the close relation between the functional set $\mathfrak{F}$ and a matrix

$$\mathbf{H}_{\mathfrak{F}} = \int_a^b \mathbf{J}_{\mathfrak{F}}^T \mathbf{J}_{\mathfrak{F}} ds \quad (4.22)$$

with the entries

$$h_{i,j} = \int_a^b F_i(s) F_j(s) ds \quad (4.23)$$

From this definition, we can quickly infer some properties of $\mathbf{H}_{\mathfrak{F}}$.

Proposition 4.1. *$\mathbf{H}_{\mathfrak{F}}$ is symmetric and semi-positive definite.*

Proposition 4.2. *When $\mathfrak{F}$ is linearly dependent, $\mathbf{H}_{\mathfrak{F}}$ is singular.*

More importantly, the rank of $\mathbf{H}_{\mathfrak{F}}$ exactly reflects that of $\mathfrak{F}$.

Theorem 4.1. $\text{rank}(\mathfrak{F}) = \text{rank}(\mathbf{H}_{\mathfrak{F}})$

We prove this theorem through its equivalent assertion. According to Proposition 4.1, Theorem 4.1 is equivalent to

Theorem 4.2. *Suppose $\mathfrak{F}[\Phi]$ is a subsect of $\mathfrak{F}$ with indices Φ , then $\mathfrak{F}[\Phi]$ is linearly independent if and only if $\mathbf{H}_{\mathfrak{F}}[\Phi, \Phi]$ is symmetric and positive definite (SPD).*

Proof. The necessary condition is a direct inference of Proposition 4.2.

To prove the sufficient condition, we assume that $\mathbf{H}_{\mathfrak{F}}[\Phi, \Phi]$ is not SPD when $\mathfrak{F}[\Phi]$ is linearly independent. Then there exists a vector $X \neq \mathbf{0}$ such that

$$X^T \mathbf{H}_{\mathfrak{F}}[\Phi, \Phi] X = 0 \quad (4.24)$$

By definition,

$$\mathbf{H}_{\mathfrak{F}}[\Phi, \Phi] = \int_a^b \mathbf{J}_{\mathfrak{F}}[\Phi]^T \mathbf{J}_{\mathfrak{F}}[\Phi] ds \quad (4.25)$$

Then we have

$$\int_a^b (\mathbf{J}_{\mathfrak{F}}[\Phi] X)^2 ds = 0 \quad (4.26)$$

As $\mathbf{J}_{\mathfrak{F}}[\Phi] X$ is continuous and non-negative, the following relation holds

$$\mathbf{J}_{\mathfrak{F}}[\Phi] X \equiv 0$$

which is contradicted with the condition that $\mathfrak{F}[\Phi]$ is linearly independent. $\square$

As soon as we find the basis indices Φ , the dependency of others with respect to the basis also coincides.

Theorem 4.3. *If Φ is the indices such that the columns of $\mathbf{H}_{\mathfrak{F}}[:, \Phi]$ is linearly independent*

and $|\Phi| = \text{rank}(\mathbf{H}_{\mathfrak{F}})$, and $\mathbf{H}_{\mathfrak{F}}$ with respect to the independent columns are connected by

$$\mathbf{H}_{\mathfrak{F}} = \mathbf{H}_{\mathfrak{F}}[:, \Phi] \mathbf{R} \quad (4.27)$$

then the transformation of the blending functions can be represented by

$$\mathbf{J}_{\mathfrak{F}} = \mathbf{J}_{\mathfrak{F}}[\Phi] \mathbf{R} \quad (4.28)$$

Using Theorem 4.3, we can show that the condition (3.41) is also sufficient in constraining a general curve on the surface.

Corollary 4.1. *Given a PD-T-spline surface $\mathbf{T}(u, v, t) = (\mathbf{jQ}_x, \mathbf{jQ}_y, \mathbf{jQ}_z)^T$ and a 2D domain curve $(u(s), v(s))$, $\mathbf{T}(u, v, t)$ interpolates the composite curve $\mathbf{T}(u(s), v(s), 0)$ if and only if*

$$\tilde{\mathbf{K}}_c \mathbf{Q}_\alpha = \tilde{\mathbf{K}}_c \mathbf{Q}_\alpha(0), \alpha = x, y, z \quad (4.29)$$

where $\tilde{\mathbf{K}}_c$ is defined in (3.42), and $\mathbf{Q}_\alpha(0)$ refers to the initial generalized coordinates.

Proof. The necessary part is trivial. We only focus on the sufficient side.

Assume (4.29) holds for arbitrary time t and $\text{Rank}(\tilde{\mathbf{K}}_c) = k$. Then we can find k columns in $\tilde{\mathbf{K}}_c$ that are linear independent, and denote their indices as Φ . The dependency can be written as

$$\tilde{\mathbf{K}}_c = \tilde{\mathbf{K}}_c[:, \Phi] \mathbf{R} \quad (4.30)$$

where $\mathbf{R}$ is the transformation matrix. Substituting (4.30) into (4.29) yields

$$\tilde{\mathbf{K}}_c[:, \Phi] \mathbf{R} \mathbf{Q}_\alpha = \tilde{\mathbf{K}}_c[:, \Phi] \mathbf{R} \mathbf{Q}_\alpha(0) \quad (4.31)$$

Since $\tilde{\mathbf{K}}_c[:, \Phi]$ is of full rank, we obtain

$$\mathbf{R}\mathbf{Q}_\alpha = \mathbf{R}\mathbf{Q}_\alpha(0), \alpha = x, y, z \quad (4.32)$$

On the other hand, referring to the definition in (3.42), $\tilde{\mathbf{K}}_c$ can be viewed as the construction matrix for the functional set

$$\mathfrak{N} = \{N_i(u(s), v(s)) \mid i \in \mathbf{N}_n\} \quad (4.33)$$

where $N_i(u, v)$ is defined in (3.5). Then according to Theorem 4.3, we can obtain

$$\mathbf{J}_\mathfrak{N} = \mathbf{J}_\mathfrak{N}[\Phi]\mathbf{R} \quad (4.34)$$

Note that $\mathbf{j}(u(s), v(s)) = \mathbf{J}_\mathfrak{N}$. Then the composite curve can be expressed as

$$\mathbf{T}(u(s), v(s), t) = (\mathbf{J}_\mathfrak{N}[\Phi]\mathbf{R}\mathbf{Q}_x, \mathbf{J}_\mathfrak{N}[\Phi]\mathbf{R}\mathbf{Q}_y, \mathbf{J}_\mathfrak{N}[\Phi]\mathbf{R}\mathbf{Q}_z)^T \quad (4.35)$$

Making use of the relation in (4.32),

$$\begin{aligned} \mathbf{T}(u(s), v(s), t) &= (\mathbf{J}_\mathfrak{N}[\Phi]\mathbf{R}\mathbf{Q}_x(0), \mathbf{J}_\mathfrak{N}[\Phi]\mathbf{R}\mathbf{Q}_y(0), \mathbf{J}_\mathfrak{N}[\Phi]\mathbf{R}\mathbf{Q}_z(0))^T \\ &= (\mathbf{J}_\mathfrak{N}\mathbf{Q}_x(0), \mathbf{J}_\mathfrak{N}\mathbf{Q}_y(0), \mathbf{J}_\mathfrak{N}\mathbf{Q}_z(0))^T \\ &= \mathbf{T}(u(s), v(s), 0) \end{aligned} \quad (4.36)$$

which indicates that $\mathbf{T}(u, v, t)$ interpolates $\mathbf{T}(u(s), v(s), 0)$ at arbitrary time t . Therefore, (3.41) is a necessary condition for constraining the composite curve on the surface. $\square$

4.2.5 Solution

Theorem 4.2 and 4.3 imply that the dependency of the matrix $\mathbf{H}_{\mathfrak{F}}$ directly reflects that of the corresponding functional set $\mathfrak{F}$. Hence, we also call the $\mathbf{H}_{\mathfrak{F}}$ the construction matrix

for the functional set $\mathfrak{F}$. Those observations provide us a new approach to deal with the dependency of functions. If we can solve the following problem

Problem 4.3. *If $\text{rank}(\mathbf{H}_{\mathfrak{F}}) = k$, find the indices Φ such that $|\Phi| = k$, and $\text{rank}(\mathbf{H}_{\mathfrak{F}}[:, \Phi]) = k$. Furthermore, establish the transformation $\mathbf{R}$ such that*

$$\mathbf{H}_{\mathfrak{F}} = \mathbf{H}_{\mathfrak{F}}[:, \Phi] \mathbf{R} \quad (4.37)$$

according to Theorems 4.2 and 4.3, Φ and $\mathbf{R}$ are exactly the solution to Problem 4.2. It means that we successfully transform the problem of dealing with functional space into constant matrix processing. As abundant robust matrix tools exist, Problem 4.3 can be easily solved.

In order to obtain the linearly independent columns, we adopt QR decomposition with column pivoting, obtaining

$$\mathbf{H}_{\mathfrak{F}} = Q \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} P \quad (4.38)$$

where R_{11} is an upper triangular matrix with full rank of $k + 1$, and P refers to the permutation matrix. Then the basis indices Φ can be obtained through

$$\Phi = \{P(i) | i = 0, \dots, k\} \quad (4.39)$$

where P is the permutation map. The basis columns are expressed as

$$\mathbf{H}_{\mathfrak{F}}[:, \Phi] = Q \begin{bmatrix} R_{11} \\ 0 \end{bmatrix} P_1 \quad (4.40)$$

Here P_1 is the corresponding permutation matrix of dimension $k + 1$. Moreover, the dependency of other columns with respect to this basis can be easily computed as well using this

decomposition. Observing that R_{11} is triangular, the linear system

$$R_{11}X = R_{12} \quad (4.41)$$

can be efficiently solved through back substitution. Substituting (4.41) into the original decomposition (4.38) yields

$$\mathbf{H}_{\mathfrak{F}} = Q \begin{bmatrix} R_{11} \\ 0 \end{bmatrix} P_1 P_1^T [I \ X] P \quad (4.42)$$

where I is identity matrix of dimension $k + 1$. Compared with (4.40), we have

$$\mathbf{H}_{\mathfrak{F}} = \mathbf{H}_{\mathfrak{F}}[:, \Phi] P_1^T [I \ X] P$$

Accordingly, the transformation matrix $\mathbf{R}$ can be determined through

$$\mathbf{R} = P_1^T [I \ X] P$$

As we can see, this solution involves only one QR decomposition (4.38) and the back substitution to solve the linear system (4.41), which is obviously robust and efficient.

The composite blending functions with an arbitrary 2D domain curve satisfy the conditions (4.20). Therefore, our approach is not confined by the type of the 2D domain curve. But it requires an extra computation of the corresponding construction matrix $\mathbf{H}_{\mathfrak{F}}$, which is a matrix integral. As we will see later, such a construction matrix is usually compulsory in some applications. In this sense, our approach does not increase the computation cost.

4.3 Regularized physically based dynamic composite curve

After introducing the main theory, we are in a position to introduce our T-spline surface modification method using general curve handle. Different from that in [69, 144], we

choose to obtain the target curve through deforming the general curve handle. To improve the intuition of the curve deformation, we first develop a physically based model directly based on the composite representation.

4.3.1 Problem of PBDC curve

Given a composite curve defined in (4.8), we first introduce time into its representation, obtaining the dynamic composite curve

$$\mathbf{S}_c(s) = \frac{\sum_{i \in \Xi} B_i(u(s), v(s)) \omega_i \mathbf{P}_i(t)}{\sum_{i \in \Xi} B_i(u(s), v(s)) \omega_i}. \quad (4.43)$$

Here we merely allow the control points to move for two reasons. First, it can streamline the illustration of the main idea. Second, it is much more suitable for practical use for the same reason explained in last chapter. In this case, if we recast the dynamic composite into

$$\mathbf{S}_c(s) = \sum_{i \in \Xi} \frac{B_i(u(s), v(s)) \omega_i}{\sum_{j \in \Xi} B_j(u(s), v(s)) \omega_j} \mathbf{P}_i(t), \quad (4.44)$$

then

$$N_i(s) = \frac{B_i(u(s), v(s)) \omega_i}{\sum_{j \in \Xi} B_j(u(s), v(s)) \omega_j} \quad (4.45)$$

are viewed as the composite blending functions. Furthermore, if we concatenate the time variant parameters into

$$\mathbf{P}_c(t) = [\cdots, \mathbf{P}_i, \cdots]_{i \in \Xi}^T,$$

the dynamic composite curve can also be represented using matrix multiplication

$$\mathbf{S}_c(s, t)^T = \mathbf{J}_{\mathfrak{F}} \mathbf{P}_c$$

Here the $\mathfrak{F}$ represents the set of all the composite blending functions $N_i(s)$. Referring to (4.44), $\mathbf{J}_{\mathfrak{F}}$ can also be regarded as the Jacobian matrix of $S_c(s, t)$ with respect to $\mathbf{P}_c(t)$.

When the mass distribution and elastic properties are imposed on the dynamic curve to simulate a physical elastic object, its motion satisfies

$$\mathbf{M}\ddot{\mathbf{P}}_c + \mathbf{D}\dot{\mathbf{P}}_c + \mathbf{K}\mathbf{P}_c = \mathbf{f}_c \quad (4.46)$$

where the coefficient matrices are

$$\begin{aligned} \mathbf{M} &= \int \mu(s) \mathbf{J}_{\mathfrak{F}}^T \mathbf{J}_{\mathfrak{F}} ds \\ \mathbf{D} &= \int \gamma(s) \mathbf{J}_{\mathfrak{F}}^T \mathbf{J}_{\mathfrak{F}} ds \\ \mathbf{f}_c &= \int \mathbf{J}_{\mathfrak{F}}^T \mathbf{f}_d(s) ds \\ \mathbf{K} &= \int \left(\alpha(s) \frac{d\mathbf{J}_{\mathfrak{F}}^T}{ds} \frac{d\mathbf{J}_{\mathfrak{F}}}{ds} + \beta(s) \frac{d^2\mathbf{J}_{\mathfrak{F}}^T}{ds^2} \frac{d^2\mathbf{J}_{\mathfrak{F}}}{ds^2} \right) ds \end{aligned} \quad (4.47)$$

In (4.47), the coefficients $\mu(\cdot)$, $\gamma(\cdot)$, $\alpha(\cdot)$ and $\beta(\cdot)$ represent mass, damping, stretching and bending distribution density respectively. $\mathbf{f}_c$ refers to the applied force density. The dynamic composite curve which follows (4.46) is called physically based dynamic composite curve (PBDC curve).

Referring to (4.47), the matrices coefficients in (4.46) are all constant and symmetric. So it appears that the nice analytical solver described in Section 3.5 can be adopted to efficiently solve it. However, this is not the case when the blending functions $\mathfrak{F}$ get linearly dependent. Referring to Section 3.5, the solution requires solving a quadratic eigenvalue problem (QEP),

$$(\mathbf{M}\lambda^2 + \mathbf{D}\lambda + \mathbf{K})\mathbf{v} = \mathbf{0} \quad (4.48)$$

Take the situation in Figure 4.2 as an example. According to the relation in (4.14), the

corresponding new blending functions are similarly related by

$$N_{i_k} = c_k/c_0 N_{i_0}, k = 1, 2, 3 \quad (4.49)$$

Then referring to the definitions (4.47), we have

$$\begin{aligned} \mathbf{M}[:, i_k] &= c_k/c_0 \mathbf{M}[:, i_0] \\ \mathbf{D}[:, i_k] &= c_k/c_0 \mathbf{D}[:, i_0] \\ \mathbf{K}[:, i_k] &= c_k/c_0 \mathbf{K}[:, i_0] \end{aligned} \quad (4.50)$$

from which we can infer

$$|\mathbf{M}\lambda^2 + \mathbf{D}\lambda + \mathbf{K}| \equiv 0 \quad (4.51)$$

It means that the QEP is non-regular, and the analytical solution is no longer applicable. More than that, the iterative numerical method does not apply as well, because the resultant discrete dynamic equation will also become singular in this case.

4.3.2 Regularized PBDC curve

Based on the above discussion, PBDC curve based on the composite representation is quite risky to use, which significantly suppresses its applicability. But its idea to build the dynamic version of the composite curve directly based on its parent surface representation is quite interesting and smart. If we can resolve the existing rank deficiency problem and maintain its flexibility in representing arbitrary curve on surface, PBDC curve can be powerful. The theory in last section provides us with the exact solution to this issue.

First, we need to built up the construction matrix for the functional set $\mathfrak{F}$. Comparing matrices defined in (4.22) and (4.47), we can find that they are almost the same expect the coefficient. It is not difficult to infer that the column dependency of these two matrices are identical. So to obtain the basis indices Φ and their transformation, we suffice to analyze $\mathbf{M}$ instead of $\mathbf{H}_{\mathfrak{F}}$ using the approach as described in Section 4.2.5. After getting the indices

and transformation, the original dynamic composite curve can be transformed into

$$\mathbf{S}_c(s, t) = \mathbf{J}_{\mathfrak{F}}[\Phi] \mathbf{Q}_c(t) \quad (4.52)$$

where the time dependent DOFs

$$\mathbf{Q}_c(t) = \mathbf{R} \mathbf{P}_c(t) \quad (4.53)$$

Here we can see that the redundant DOFs are synthesized through this transformation. Then based on this representation, the dynamic equation becomes

$$\widehat{\mathbf{M}} \ddot{\mathbf{Q}}_c + \widehat{\mathbf{D}} \dot{\mathbf{Q}}_c + \widehat{\mathbf{K}} \mathbf{Q}_c = \widehat{\mathbf{f}}_c \quad (4.54)$$

with the coefficients

$$\begin{aligned} \widehat{\mathbf{M}} &= \mathbf{M}[\Phi, \Phi] \\ \widehat{\mathbf{D}} &= \mathbf{D}[\Phi, \Phi] \\ \widehat{\mathbf{K}} &= \mathbf{K}[\Phi, \Phi] \\ \widehat{\mathbf{f}}_c &= \mathbf{f}_c[\Phi, :] \end{aligned} \quad (4.55)$$

Referring to Section 4.2.4, the indices Φ are chosen such that $\mathfrak{F}[\Phi]$ is linearly independent. According to Theorem 4.2, $\widehat{\mathbf{M}}$ is always of full rank. It follows that the corresponding quadratic eigenvalue problem is regular, which means the analytical solution in Section 3.5 is always applicable to (4.54). PBDC curve based on this representation is also named as regularized physically based dynamic curve (or R-PBDC curve for short). From the definition in (4.55), the coefficient matrices in (4.54) are actually the submatrices of that in (4.46). It means R-PBDC curve also involves less computation.

Similar as PD-T-splines, constraints and forces are two type of important modeling tools in R-PBDC curve. Linear constraints such as the clamping of points or normals can

be easily incorporated, which generally can be expressed as

$$\mathbf{A}\mathbf{Q}_c + \mathbf{b} = \mathbf{0} \quad (4.56)$$

Instead of using Gaussian elimination to express the R-PBDC curve in the new generalized coordinates, we adopt the penalty method for its simplicity by adding an extra term

$$\frac{1}{2}\sigma(\mathbf{A}\mathbf{Q}_c + \mathbf{b})^T(\mathbf{A}\mathbf{Q}_c + \mathbf{b}) \quad (4.57)$$

into the internal energy functional. Accordingly, the dynamic equation changes into

$$\widehat{\mathbf{M}}\ddot{\mathbf{Q}}_c + \widehat{\mathbf{D}}\dot{\mathbf{Q}}_c + (\widehat{\mathbf{K}} + \sigma\mathbf{A}^T\mathbf{A})\mathbf{Q}_c = \widehat{\mathbf{f}}_c + \sigma\mathbf{A}^T\mathbf{b} \quad (4.58)$$

As a merit of this method, we can avoid the recalculation of the coefficient matrices when the constraints are changed.

Besides, simulated force is the main sculpting tools for interactive modification. Particularly, the spring force is the most important. To implement a spring force that connects a point $\mathbf{S}_c(s_0)$ on the curve to a spatial point $\mathbf{d}_0$, we simply add

$$\int \widehat{\mathbf{J}}_{\mathfrak{F}}[\Phi]^T k(\mathbf{d}_0 - \mathbf{S}_c(s, t))^T \delta(s - s_0) ds$$

into the force term of (4.54), where k is the spring stiffness coefficient and δ is the unit delta function.

4.4 Curve-based T-spline manipulation

Now we are ready to introduce our T-spline surface editing method based on general curve handle. Our approach involves three components: handling curve creation, target curve generation, and surface modification. R-PBDC curve is mainly used to generate the arbi-

Figure 4.3: A simple example of T-spline surface editing using general curve handle.

trary 3D target curve. In the following, we use the simple example as shown in Figure 4.3 to illustrate the main procedures.

4.4.1 Handling curve creation

Given an under deforming T-spline surface $\mathbf{T}(u, v)$, which is a plane in this case. Adding a handling curve is usually achieved by specifying a 2D domain curve $D(s)$ in the parametric domain (the black circle in Figure 4.3(a)). [69] proposed to obtain $D(s)$ making use of sketching on the T-spline surface, where B-spline curve is used to fit the resultant preimage of this sketch in the parametric domain. Then in their case, $D(s)$ is a B-spline curve. As R-PBDC curve allow arbitrary type of domain curve, we also implement some useful analytical quadratic curves, and straight line in any direction. Like in Figure 4.3(a), $D(s)$ is a circle with parameterization

$$\begin{aligned} u(s) &= u_0 + r \cos(s) \\ v(s) &= v_0 + r \sin(s) \end{aligned} \tag{4.59}$$

It is well known that circle can not be exactly represented by a B-spline curve. Hence the method in [69] cannot apply in this example. After specifying the 2D domain curve, the handling curve is represented as the composition of the T-spline surface and the curve $D(s)$.

4.4.2 Target curve generation

In order to deform the T-spline surface, another 3D target curve in space is required, which indicates how the handling curve should look like. Here we generate this target curve through directly modifying the handling curve obtained in the preceding subsection. The handling curve is regarded as R-PBDC curve, where the composite curve is transformed into basis representation (4.52). With the forces and constraints described in section 4.3.2, the dynamic revolution resulted from the analytical solution to the dynamic equation produces intuitive and physically realistic variation. In this example, four spring forces and

four constraints are imposed. Finally, the evolution stops at the equilibrium(as shown in Figure 4.3(b)). Note that R-PBDC curve evolves follows the physical laws, and is not necessary to interpolate all the points, which is the essential difference from direct manipulation [32]. Meanwhile, by virtue of the basis transformation and the analytical solver, the evolution can be both forward and backward, which facilitates the user interaction and decision making. It is worth pointing out that the advanced 3D sketching tools can also be incorporated into R-PBDC scheme. [69] propose to sketch out the 3D target curve, and to use B-spline curve to approximate the sketches. However, such treatment has a potential drawback that the handling curve perhaps cannot go through the target curve exactly, because the spatial target curve may not belong to the space of the composite curve. Instead, we can regard the original sketches as multiple points, and project them onto the R-PBDC curve. By connecting spring forces between these points and projections, R-PBDC curve will evolve towards the sketches. Essentially, we use composite curve to approximate the sketches.

4.4.3 Surface modification

After obtaining the desired target curve, we use it together with the handling curve for surface modification. We achieve this through the use of PD-T-splines. The under-deforming T-spline is regarded as Dynamic T-spline surface. Based on the practical consideration, the sub-class E-PD-T-splines are adopted, which also support analytical solver. Since the target curve is obtained by deforming the handling curve, the parametric correspondence is natural. Denote the target curve by $\mathbf{S}_0(s)$. We can define a curve force using these two curves

$$\mathbf{f}_c = \int_{\mathbf{D}} k(s) \mathbf{j}(u(s), v(s))^T (\mathbf{S}_0(s)^T - \mathbf{T}(u(s), v(s), t)^T) ds \quad (4.60)$$

where the $\mathbf{j}(u, v)$ represents the Jacobian matrix of E-PD-T-splines(referring to (3.26)) and $k(s)$ is the stiffness density along the curve. As shown in Figure 4.3(d), such curve force actually simulates an infinite number of springs connecting points of the handling curve to the corresponding points on the target curve, which pulls the PD-T-spline surface toward the target curve. Finally, the surface deformed and interpolates the target curve, producing a saddle shape(as shown in Figure 4.3(e)). Note that although R-PBDC curve is developed based on the T-spline surface representation, the state of PD-T-splines is irrelative to the parent surface of R-PBDC curve. This can be obviously seen in Figures 4.3(c) and 4.3(e). Actually, the parent surface for R-PBDC curve is just a copy of the original T-spline surface at the moment of handling curve creation. After that the position of the control points in the parent surface is uniquely determined by the state of R-PBDC curve.

4.5 Implementation

As described in Section 4.3.2, the dynamic evolution of R-PBDC curve not only produces vivid animation in response to simulated force and constraints, but also provides candidate optimal shapes. Therefore, solving the dynamic equation (4.58) plays a crucial role in R-PBDC curve, which is a linear ordinary differential equation. Moreover, owing to the transformation into basis representation, it is always regular. Therefore, the analytical solver described in Section 3.5 can be readily applied, which is generally more stable and efficient than numerical methods.

Before applying the analytical solver, we need to evaluate the matrix integrals in (4.55) to build up the dynamic equation. As seen in the definition, those matrix integrals are actually the sub-matrices of those in (4.47). We suffice to calculate the matrices in (4.47) and determine the basis indices Φ . As mentioned above, the basis indices Φ are determined by analyzing the mass matrix $\mathbf{M}$. The exact quantity that needs to be calculated is those matrix integrals in (4.47). As an arbitrary 2D domain curve is allowed, we can not evaluate them explicitly in general. It is well known that the rank is very sensitive to the matrix perturba-

tion. So in order to obtain the correct basis indices, the accuracy of the computation should be in high priority. We adopt the similar finite element method and Gaussian quadrature as in Section 3.6 for its precision with less computation intensity. As required in Gaussian quadrature, the integrand must be sufficiently continuous in order to ensure the precision. It means that the finite element should be chosen in which the composite blending functions have high order of continuity. So first, we need to determine the suitable type of finite element.

4.5.1 Finite element

Figure 4.4: Illustration of extracting correct finite elements.(a) The 2D domain curve is a B-spline curve, where the knot points are highlighted in purple segments. In addition to the self knots, the intersection with the boundary of the Bézier patches in parent surface (the green segments) should also be considered. And the yellow dashed line indicates the treatment of extracting the Bézier patches in parent surface using the method described in Section 3.6.1. (b) When the 2D curve itself is sufficiently smooth, only the intersection should be concerned.

We will use cubic B-spline as the example to illustrate our approach. A brute-force approach is to choose the curve arc defined by consecutive knots in the parametric domain of the 2D domain curve $D(s)$ as the type of finite element (as purple knots shown in Figure 4.4(a)). However, this generally can not satisfy the continuity requirement. Although $D(s)$ has continuity of infinity order within each element, the composite blending functions $N_i(s)$

have at most C^2 continuity at the intersections between $D(s)$ and boundaries of Bézier patches in its parent T-spline surface. Thus, in order to ensure higher order continuity, the intersections also need to be taken into account (the green segments in Figure 4.4(a)), which requires the evaluation of the intersections between $D(s)$ and the horizontal and vertical line segments.

We propose a simple approach to deal with this issue. First, $D(s)$ is approximated by poly-lines using adaptive tessellation. Then the desired intersection with $D(s)$ is replaced by that with poly-lines. Note that the intersections must lie on the knot lines. Therefore, instead of transversing all the incident Bézier patches in the parent, we check the intersection with the knot lines directly. As the knot line is either horizontal or vertical, the intersection is very simple. Take the knot line $x = u_0$ for instance. Suppose the incident segment is p_0p_1 with corresponding parameters s_0, s_1 . Then by checking $u_0 \in [p_0.u, p_1.u]$ we can quickly tell whether it has intersection with the knot-line. If yes, then the intersection points can be computed using

$$p = \frac{p_1.u - u_0}{p_1.u - p_0.u}p_0 + \frac{u_0 - p_0.u}{p_1.u - p_0.u}p_1$$

and the corresponding parameter is

$$s = \frac{p_1.u - u_0}{p_1.u - p_0.u}s_0 + \frac{u_0 - p_0.u}{p_1.u - p_0.u}s_1$$

The intersection with the knot line in the v direction follows similarly. Suppose the number of tessellated line segments is nt , and those of the knot lines are nu and nv in the u and v directions respectively. Then the complexity of this method is $nt \times (nu + nv)$. This method will contain excess elements, because intersection may not belong to any edge of any Bézier patch. We have to filter out those superfluous elements. We transverse all the incident Bézier patches in the parent T-spline surface to delete those intersections that lie inside a Bézier patch. Finally, we sort those parameters along with the original knots to form a new parameter vector. The curve arcs defined by the consecutive parameters in the

new vector are designated as our finite elements.

When the 2D domain curve itself is sufficiently continuous, we only have to care about its intersections with the knot lines. Then the algorithm above still applies as soon as the tessellation is obtained. Figure 4.4(b) shows the example when the 2D domain curve is a simple circle with parameterization,

$$(u(s), v(s)) = (u_0 + r_0 \cos(s), v_0 + r_0 \sin(s)), s \in [0, 2\pi]. \quad (4.61)$$

4.5.2 Data structure for R-PBDC finite elements

After determining the type of element, we define an elemental data structure associated with each of them, which contains the geometric specification of the R-PBDC element along with its physical properties. A complete R-PBDC curve is then implemented as a data structure consisting of an ordered array of elements with additional information.

Referring to the definition of R-PBDC curve, its element is determined by the corresponding curve segment in the 2D domain curve and the belonging Bézier patch in the parent surface. Hence, when the parent surface is copied from the instant PD-T-spline, the Bézier structure should be inherited, which contains the influencing control points and the indices to the whole control point list. So the pointers of the belonging Bézier patch and the 2D domain curve suffice to determine the geometry of the element. Note that the indices in the belonging Bézier patch is directed to the whole control point list in the parent surface. As for R-PBDC curve, merely a small portion of them is taken into the consideration. To be specific, only the control points with indices in $\Xi[\Phi]$ are DOFs. So when assembling the elements into the global matrix, we are required to establish another map according to this index set.

Furthermore, we also allocate each element with an elemental mass, damping and stiff matrix, as well as the physical quantities required to compute these matrices. These quantities including the mass, damping, and elasticity distribution functions, which can be rep-

resented as analytic functions or as discrete arrays of sample values.

4.5.3 Calculation of element matrices

We employ Gaussian Quadrature to numerically evaluate the integral of the elemental matrices. Take the computation of elemental stiff matrix for example, and the mass and damping matrices follow similarly. Assume the domain of the element is $[s_0, s_1]$. The entries of the elemental stiffness matrix can be expressed as

$$k_{i,j} = \int_{s_0}^{s_1} f_{i,j}(s) ds \quad (4.62)$$

where

$$f_{i,j}(s) = \alpha(s)N'_i(s)N'_j(s) + \beta(s)N''_i(s)N''_j(s) \quad (4.63)$$

where $N_i(s)$ are the composite blending functions defined in (4.45).

Then the integral can be approximately calculated through applying Gaussian Quadrature. Gaussian Quadrature first requires calculating a set of Gauss weights a_g and abscissas u_g . The integral (4.62) can be approximated by

$$k_{i,j} = \sum_{g=1}^{N_g} a_g f_{i,j}(u_g) \quad (4.64)$$

In generally, Gaussian quadrature can exactly evaluate the integral with N weights and abscissas when the integrands are polynomial of degree $2N$. In our case, as the parent surface is allowed to be rational, the integrand is likely to be rational polynomial. Then in order to ensure the precession, a relatively large number of N_g is required. In our implementation, we choose N_g between 7 and 10, which are demonstrated to be stable.

Due to our basis transformation, not all the influencing control points will serve as the DOFs for R-PBDC curve. Then referring to the coefficient matrix in (4.55), we can further economize the computation of elemental matrices by skipping the irrelevant entries. But for

the elemental mass matrix, this can not be saved. Because the entire mass matrix will serve as the construction matrix to determine the basis indices as well as the transformation.

4.5.4 Local update of T-spline refinement

One of the most attractive feature of PD-T-splines are local update of the dynamic equation when T-spline local refinement is called. This avoids most of the repeated calculation. When PD-T-splines are refined, it is not necessary to update the representation of the target curve accordingly referring to definition of curve force (4.60).

In practice, local refinement is called in order to introduce more details. Then generally, the feature line is also required to be complex, which means more degree of freedom is also required for the composite curve. Therefore, there is a need to maintain the topological consistency between the parent surface and PD-T-splines. When parent surface is refined simultaneously, the generalized coordinates of RPBDC-curve change as well as the dynamic equation. We have to recalculate the coefficient matrix as well as the construction matrix. If we totally start from the scratch, a lot of double calculation will occur. Inspired by the approach in Section 3.6.4, we propose to locally update these matrices in (4.47) by carefully checking the entries, which really change.

To keep the topological consistency as PD-T-splines, the same refinement operation is applied onto the parent surface of the R-PBDC curve. The associated Bézier structure updates accordingly as described in Section 3.6.4. Due to the intersection between the 2D domain curve and the new knot lines, some of the elements of R-PBDC curve will be violated. So first, we should correct these elements. Note that these intersections only appear in the new generated Bézier patches. We suffice to focus on those new generated ones. We traverse all those new generated Bézier patches, and use the edge belonging to the new inserted knot line to intersect the 2D domain curve segments in the associated elements. Compared with the method illustrated in Section 4.5.1, such a method can avoid traversing the whole Bézier patches in the parent surface to delete the excess ones. The

parameters at the new intersections are used to locate the elements that are subdivided. For example, suppose the new parameter is s_0 , which satisfies $s_i < s_0 < s_{i+1}$, where s_i are the incremental parameters in the original parameter vector. We can infer that the i th element is subdivided, which produces two new elements with the parametric domain (s_i, s_0) and (s_0, s_{i+1}) . In this stage, the geometric specification including the pointer to the Bézier patch as well as that to the domain curve is inherited. Then s_0 is included into the parameter vector. After all the new parameters are inserted into the parameter vector, the parametric domain of the elements are corrected. As the Bézier patches in the parent surface have changed, the pointers to belonging patches are also required to be update accordingly. We do not need to traversing all the patches to re-determine the belonging patches. We suffice to update those elements whose belonging patches are subdivided. In this way, we finish the updating of the geometric specification of RPBDC elements.

The update of the elemental matrices are divided into two cases. For those new generated elements, we simply recalculate them from the scratch. But when the parametric domain is the same, most of the entries will remains. So, we can locally update them by identifying the exact changed ones. Note that in this case, it is the alternation of the associated control point list of its belonging Bézier patch cause the change of the matrix entries. Similarly to that in PD-T-splines, there are four types of changes in associated control points:

- the new inserted ones.
- the ones whose blending functions changes.
- the ones whose weights changes.
- the deleted ones

For the former three types, we suffice to recalculate the corresponding entries in the matrices using the method described in the preceding subsection. For the fourth type, we ought to delete the relevant entries. As T-spline refinement is local, merely a small portion of

Bézier patches will be split. Also only a few of the associated control points will change. Therefore, this updating method can avoid most of the double recalculation, thus enhancing the efficiency of curve-based editing, especially when involving a large number of control points in a T-spline surface.

Keeping the consistency of topology with PD-T-spline produces an unexpected merit, as the topology is maintained, referring to the definition of curve force term in (4.60)

$$\mathbf{J}(u(s), v(s)) = \mathbf{J}_{\mathfrak{M}} \quad (4.65)$$

where $\mathfrak{M}$ are all the functions defined in (4.45),

$$\mathfrak{M} = \{N_i(s) | i = 0, \dots, n\} \quad (4.66)$$

According to the definition of the indices Ξ , we have

$$\mathfrak{F} = \mathfrak{M}[\Xi] \quad (4.67)$$

and

$$N_i(s) \equiv 0, i \notin \Xi \quad (4.68)$$

On the other hand, the target curve $\mathbf{S}_0(s)$ is represented in basis form

$$\mathbf{S}_0(s) = \mathbf{J}_{\mathfrak{F}}[\Phi] \mathbf{Q}_0 \quad (4.69)$$

Let $\mathbf{P}_0$ be a $n \times 3$ vector such that

$$\begin{aligned} \mathbf{P}_0[\Xi[\Phi], :] &= \mathbf{Q}_0 \\ \mathbf{P}_0[\overline{\Xi[\Phi]}, :] &= \mathbf{0} \end{aligned} \quad (4.70)$$

where $\overline{\Xi[\Phi]}$ refers to the row indices other than $\Xi[\Phi]$. $\mathbf{S}_0(s)$ can be recast into

$$\mathbf{S}_0(s) = \mathbf{J}_{\mathfrak{M}} \mathbf{P}_0 \quad (4.71)$$

Then the force term in (4.58) can be rearranged into

$$\mathbf{f}_c = \mathbf{H} (\mathbf{P}_0 - \mathbf{P}) \quad (4.72)$$

where

$$\mathbf{H} = \int_a^b k(s) \mathbf{J}_{\mathfrak{M}}^T \mathbf{J}_{\mathfrak{M}} ds \quad (4.73)$$

Note that

$$\begin{aligned} \mathbf{H}[\Xi, \Xi] &= \int_a^b k(s) \mathbf{J}_{\mathfrak{F}}^T \mathbf{J}_{\mathfrak{F}} ds \\ \mathbf{H}[i, j] &= 0, i \notin \Xi \text{ or } j \notin \Xi \end{aligned} \quad (4.74)$$

Compared with $\mathbf{M}$ in (4.47), $\mathbf{H}$ can be simultaneously computed with $\mathbf{M}$. Especially, when the coefficient $k(s)$ and $\mu(s)$ are chosen proportional, the non-zero entries

$$\mathbf{H}[\Xi, \Xi] = k/\mu \mathbf{M} \quad (4.75)$$

Hence by maintaining the topology, the force term for PD-T-Splines is updated automatically.

4.6 Experimental results

In this section, we present some applications of the proposed curve based T-spline surface editing.

Figure 4.5: Modeling a Cylinder into a goblet

4.6.1 Modeling a goblet from a cylinder

The iso-curve as shown in the example in Figure 4.2 is a simple case of composite curves with the horizontal or vertical line segments as the 2D domain curves. Certainly, we can treat this handling iso-curve as DNURBS curve to obtain target curve through interaction with DNURBS. But as a composite curve, it will be transformed into basis representation with the same number of DOFs as its underlying NURBS representation. When physics is introduced producing a R-PBDC curve, it is equivalent to the corresponding D-NURBS curve, whose computation is also comparable. In this sense, the R-PBDC curve provides a unified curve-based approach for surface editing.

Figure 4.5 shows an example of modeling an asymmetric goblet from a cylinder making use of iso-curves. The cylinder in Figure 4.5(a) is a bicubic NURBS surface, which is a special case of T-spline surface. Figure 4.5(c) displays the modeling steps of creating the goblet. In the first step, four handling curves are created and scaled to a desired dimension. Note that in this step, we merely apply simple geometric operation to obtain the desired target curves. By imposing the four curve forces, we obtain a conventional goblet when PD-T-splines reach equilibrium. Next, the upper handling curve is rotated to form its target curve. The purple handling curve is treated as R-PBDC curve. We impose two spring forces and two point constraints to adjust the shape of circle into an ellipse. The other two handling curves are kept to serve as the curve constraints for PD-T-splines. After this step, the upper edge becomes askew and the body of the goblet gets asymmetric. In the final step, we want to make the goblet bend to one side a little bit. To do this, we add a handling curve in the other direction to generate a different contour while the lower part of the curve handle is imposed to maintain the base. Finally, a goblet with asymmetric contour is created. Note that as the goblet should maintain closed and smooth in one direction, we need to impose linear constraints such as positional and tangent continuity at the two ends of this direction onto PD-T-splines.

4.6.2 Feature based surface manipulation

Figure 4.6: Feature based manipulation.

When features are involved, the surface modeling intention can be easily expressed by the desired shape of features. In this case, the point-based tools become awkward. Because these features are often interpreted as curves on a surface, many springs or point clamping have to be imposed in order to adjust the features, which will also introduce uncontrollable error. On the other hand, composite curves have the inherent advantages to express those

features. The R-PBDC curve provides a more intuitive and accurate approach to obtain the desired feature shapes. Therefore, the proposed curve-based surface editing technique is extremely suitable for feature-based manipulation.

Figure 4.6 presents an example of modifying the shape of a head model through adjusting its feature lines. The bridge of the nose and lips shows the obvious features in this model, and modifying them can significantly change the appearance of the head. Through the standard procedures described in Section 4.4, we can easily modify these features, and in turn change the shape of head. In order to change the bridge of nose, first we create a handling curve along this feature. Here we create this handling curve by directly specifying a B-spline curve on the parametric domain, which eventually has 6 control points with uniform knot vector. Then we treat this handling curve as the R-PBDC curve, and impose two springs and two point constraints to obtain the desired shape of the nose bridge. As the spring can be pulled to arbitrary spatial position, theoretically we can get abundant complex feature line. Here we would like to create a hawk nose. As soon as the desired target curve is determined, the corresponding curve force is then imposed onto PD-T-splines to realize the feature line manipulation, where the desired hawk nose is obtained. In the second stage, we use the similar procedure to modify the appearance of the lips, where five springs and four point clamping are imposed into R-PBDC curve to get the desired lip shape. finally, a quite wired clown mouth is generated in our model (as shown in Figure 4.6(c)).

4.6.3 Creating feature on a Surface

In last example, we show how to use our curve-based approach to modified the features of a model. More than that, we can also create features from the scratch combining with the operation of T-spline local refinement. Figure 4.7 shows how to generate the *C&G* logo on a T-spline surface. Figure 4.7(a) shows the initial T-spline surface and its pre-image with 1416 control points. In the first step, we introduce 5 handling curves which form the characters *C&G* on the surface. We aim to produce the 3D logo through pulling the

Figure 4.7: Produce 3D 'C&G' Logo on a T-spline surface

region around each handling curve in its normal direction. As we see in Figure 4.7(b), the handling curves have already formed the desired shapes. We suffice to translate them along the surface normals to generate their target curves as shown in Figure 4.7(c). But here we cannot simply apply these curve forces into PD-T-splines, because there are too few DOFs around the regions of handling curves as seen in Figure 4.7(b). There will be no difference between the regions of the handling curve and their neighborhood. Then the 3D effect of the logo can not be highlighted. Instead, we introduce sufficient number of DOFs around the curves through T-spline refinement. Then using the approaches described in Section 3.6.4 and Section 4.5.4, PD-T-splines and R-PBDC curve will be updated locally, which can be done in realtime in our implementation. Currently the refinement is performed manually. The rule of the refinement is to bound the influencing region of the incident control points along the handling curves. When incorporating these curve forces into the refined PD-T-splines, we add extra linear constraints to freeze the non-incident control points. As a result, the 3D logo is generated, which is smooth, and visually pleasing. The final number of control points is 3797.

Chapter 5

Foldover Free Reparameterization and Texture Mapping

5.1 Introduction

After obtaining the formulation of the efficient physically based dynamic T-spline, it's desirable to further explore its related geometric processing, such as morphing and rendering, which can make our physically based model more complete and more attractive for practical use. But in both situations, we are confronted with the problem of finding a planar parameterization that aligns the feature points with specific positions. It is also regarded as the constrained texture mapping problem. So in this chapter, we first develop an efficient constrained texture mapping technique for the following processings.

Mathematically, the problem of constrained texture mapping can be stated: given a 3D triangular mesh $\mathcal{U}$ and a set of constrained point pairs $\{(U_i, P_i)\}$, $i = 1, 2, \dots, h$, where U_i are vertices of mesh $\mathcal{U}$ and $P_i \in \mathbb{R}^2$ are points in the image domain, we want to find a bijective piece-wise linear mapping $F : \mathcal{U} \rightarrow \mathbb{R}^2$ satisfying $F(U_i) = P_i$. In this chapter we restrict our discussion to the case of a mesh topologically equivalent to a rectangular area. For a mesh model of arbitrary topology, extra processes such as cutting and maintaining of continuity may be needed. Considering that many robust, efficient and mature parameter-

ization algorithms have been developed, we propose a two-phase approach similar to the one of [42], which begins with an unconstrained planar parameterization f of the mesh and then performs a constrained mesh transformation G^* that moves the constrained vertices to the required positions. The constrained texture mapping is achieved by the composition: $F = G^* \circ f$, as illustrated in Figure 5.1. Particularly, in the first phase we use ABF++ [104] to embed the 3D mesh into a 2D domain. ABF++ is a conformal parameterization, which is bijective and boundary free. This paper thus mainly focuses on the second phase: finding a foldover free transformation G^* that maps $f(U_i)$ to P_i .

Figure 5.1: The proposed method

Similar to [45, 42, 139], we propose to construct the constrained 2D mesh transformation by an iterative warping process. To realize each warping step, we first construct locally bijective smooth mapping using radial basis function (RBF)-based interpolation to achieve alignment of constrained vertices, and then induce the piece-wise affine transformation us-

ing the RBF function with barycentric coordinates within each mesh triangle. While some previous work [139, 101] implicitly assumes that the foldover free property of the induced transformation can be inherited from the local bijectivity of the smooth mapping, this is unfortunately not true. We reveal the relation between the smooth mapping and its induced transformation and propose a refinement strategy based on the longest edge bisection to guarantee that the induced transformation is locally foldover free if the smooth mapping is locally bijective. To make the RBF-based mapping locally bijective, we propose to find non-intersecting trajectories to guide the warping and derive a bound for the warping step-size. Furthermore, to make the warping process more effective, we derive the truly foldover free condition for the induced transformation. Based on this condition and the bound for the local bijective smooth mapping, we determine whether we perform local mesh refinement or warping and the actual warping stepsize as well. Integrating all these technical components provides a constrained mesh warping algorithm that is effective, *provably* foldover free, able to handle a large number of constraints, and able to output a visually pleasing result without extra smoothing optimization. These merits are not simultaneously achieved by existing algorithms.

The main contributions of the Chapter include:

- We comprehensively analyze the relation between a C^2 continuous 2D mapping and its induced piece-wise linear transformation, and propose a strategy to refine the mesh such that if the C^2 mapping is locally bijective, its induced transformation on the refined mesh keeps the orientation of each triangle of the refined mesh.
- An iterative RBF-based warping scheme is proposed and an explicit formula for the stepsize of the warping is derived, which can assure the corresponding RBF-based mapping to be locally bijective.
- An effective, provably foldover free algorithm for smooth mesh warping with hard constraints is presented, which consists of preprocess, construction of non-intersecting

warping trajectories, iterative RBF-based mapping and foldover free induced transformation.

5.2 Fundamental Theory

Let $M = \{\mathcal{V}, \mathcal{K}\}$ be a 2D triangular mesh, where $\mathcal{V} = \{V_i \mid V_i \in \mathbb{R}^2, i = 1, 2, \dots\}$ is the list of vertices V_i and $\mathcal{K} = \{\triangle_{ijk} = \triangle V_i V_j V_k \mid V_i, V_j, V_k \in \mathcal{V}, i \neq j, j \neq k, k \neq i\}$ is the list of triangles encoding how the vertices are connected, and $\Omega \subset \mathbb{R}^2$ be a bounded domain containing all the vertices. In this paper we assume that mesh M is conforming, which means that for any two intersecting triangles in $\mathcal{K}$, their intersection is either a vertex or a shared edge. A conforming mesh $\tilde{M} = \{\tilde{\mathcal{V}}, \tilde{\mathcal{K}}\}$ is called a refinement of M if $\mathcal{V} \subset \tilde{\mathcal{V}}$ and for any $\tilde{\triangle} \in \tilde{\mathcal{K}}$ there exists a unique $\triangle \in \mathcal{K}$ such that $\tilde{\triangle}$ is within $\triangle$.

Figure 5.2: (a) A 2D mapping G and (b) its induced transformation G_M^*

Given a C^2 continuous mapping $G : \Omega \rightarrow \mathbb{R}^2$, it maps M to a triangular mesh with curved edges (see Figure 5.2(a)). When the Jacobian of G does not vanish, G is *locally* one-to-one or bijective.

From G , we can induce a piece-wise linear mapping $G_M^* : M \rightarrow \mathbb{R}^2$ by which the vertices V_i of M are mapped to $G(V_i)$ and any other point V lying in triangle $\triangle V_i V_j V_k$ is mapped to $G_M^*(V)$, a linear combination of $G(V_i)$, $G(V_j)$ and $G(V_k)$:

$$G_M^*(V) = \lambda_i G(V_i) + \lambda_j G(V_j) + \lambda_k G(V_k) \quad (5.1)$$

with barycentric coordinates $(\lambda_i, \lambda_j, \lambda_k)$ of V with respect to $\triangle V_i V_j V_k$. G_M^* can be viewed as a triangular mesh transformation induced by G since it maps the edges of M to line segments (see Figure 5.2(b)). G_M^* is called *foldover free* if each triangle $\triangle G(V_i)G(V_j)G(V_k)$ keeps the orientation of triangle $\triangle V_i V_j V_k$.

5.2.1 Basic observations

While G and G_M^* are closely related, G_M^* relies only on triangle set $\mathcal{K}$ and the set $\{G(V_i) \mid V_i \in \mathcal{V}\}$. Some previous work implicitly assumes that the local bijectivity of G guarantees G_M^* to be foldover free. For example, [139] used the positive Jacobian of G as the condition for G_M^* to be foldover free to estimate the step bound. [101] used the local bijectivity of the vector field based warping for triangular mesh transformation. However, this is unfortunately not true. In fact, we have the following observations.

Observation 1: *G is locally bijective, but G_M^* may fold over.* In Figure 5.3, G maps vertex A_i to vertex B_i and the red curves represent the image of edges of the original triangle. It can be seen that G is locally bijective, but when the transformed vertices are connected to form the induced transformation, foldover occurs.

Observation 2: *G is not locally bijective, but G_M^* could be foldover-free.* This is because the transformed vertices are connected using straight line segments rather than curves, as

Figure 5.3: G is locally bijective but G_M^* folds over.

illustrated in Figure 5.4

Figure 5.4: G is not locally bijective, but G_M^* is foldover free.

Observation 3: When G is bijective but G_M^* is folding over, if we refine M to $\widetilde{M}$, $G_{\widetilde{M}}^*$ might be foldover free. For triangle $\triangle A_1A_2A_3$ in Figure 5.3, we refine it by inserting a point C to edge A_1A_3 (see Figure 5.5). While G is the same over $\triangle A_1A_2A_3$ and its refinement $\triangle \widetilde{A_1A_2A_3}$, in general $G_{\triangle \widetilde{A_1A_2A_3}}^* \neq G_{\triangle A_1A_2A_3}^*$. It can be easily checked that $G_{\triangle \widetilde{A_1A_2A_3}}^*$ in Figure 5.5 is foldover free.

Figure 5.5: The insertion of a Steiner point makes the mesh transformation foldover free.

Observation 4: *Foldover may propagate through subdivision.* When we split one triangle, an adjacent triangle should be subdivided simultaneously to conform the mesh. The new generated triangles may fold over even if their original parent triangle is foldover free. Figure 5.6 shows such a situation. Although triangle $\triangle B_1 B_3 B_4$ has the same orientation as triangle $\triangle A_1 A_3 A_4$, after splitting the orientation of $\triangle B_1 C^* B_4$ is different from that of $\triangle A_1 C A_4$ where $C^* = G(C)$.

Figure 5.6: The Propagation of foldover through one splitting.

5.2.2 Longest edge bisection

Observation 3 in Section 5.2.1 motivates us to raise a question: *Given a C^2 mapping $G : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^2$ that is locally bijective and a triangular mesh M contained in Ω , does there always exist a conforming refinement $\widetilde{M}$ of M such that $G_{\widetilde{M}}^*$ is foldover free?* The answer is not trivial due to the propagation of foldover, as shown in Observation 4. In the following we present a constructive procedure to give a positive answer.

Our refinement is based on the *longest edge bisection* (LEB). Figure 5.7 illustrates the process. Suppose t_0 is a triangle that folds over. The refinement proceeds in the following steps:

Step 1 Initialization: Let $i = 0$.

Step 2 Bisection:

- For triangle t_i , find the longest edge e_i and split t_i by connecting the middle point of e_i to its opposite vertex.
- Find the neighboring triangle t_{i+1} that shares e_i with t_i , and let $i \leftarrow i + 1$.
- Repeat Step 2 until the neighboring triangle t_i is t_{i-2} or nil.

In this way, we obtain an ordered sequence of triangles $\{t_0, t_1, \dots, t_m\}$ such that triangle t_{j+1} and triangle t_j share the longest edge of t_j and T_m 's adjacent triangle containing the longest edge of t_m is either t_{m-1} or empty. We call such a sequence the *longest edge propagation sequence* (LEPS) and denote it by $\text{LEPS}(t_0)$.

Step 3 Conforming: connect the middle point of the longest edge of t_{i-1} with that of t_i for $i = 1, \dots, m - 1$. The midpoints are the Steiner points and are added into the vertex list of the refined triangular mesh.

Figure 5.7: The LEB process that starts from triangle t_0 .

Now the algorithm for constructing a refinement $\widetilde{M}$ of M such that the induced transformation on $\widetilde{M}$ is foldover-free is straightforward. Referring to Algorithm 5.1, we start with $M^0 = M$ and $n = 0$, and apply LEB-based refinement to each folding-over triangle in the folding-over triangle list τ^n , which generates a refined triangular mesh M^{n+1} . We repeat this process until there is no folding-over triangle.

It is worth pointing out that there exist other strategies for refining triangles. We choose the LEB-based refinement because it has several nice properties and the algorithm is proven to stop after a finite number of iterations of refinement.

Algorithm 5.1 LEB-based Refinement**Given:**

- 1: $n \leftarrow 0, M^n \leftarrow M$.
- 2: Find the folding-over triangle list τ^n in M^n .

Goal:

- 3: **while** $\tau^n.size() > 0$ **do**
- 4: **while** $\tau^n.size() > 0$ **do**
- 5: $t_0 = \tau.front()$;
- 6: Apply LEB to t_0
- 7: Update $\tau^n \leftarrow \tau^n - \tau^n \cap \text{LEPS}(t_0)$.
- 8: **end while**
- 9: $M^{n+1} \leftarrow M^n, n \leftarrow n + 1$.
- 10: Find folding-over triangle list τ^n in M^n .

11: **end while****Output:** M^n .

Property 1. The propagation in the LEB process is guaranteed to stop in a finite number of bisections.

In fact, for any triangle t that folds over, its LEPS will not form a loop and thus consists of a finite number of triangles. That is, its elements t_i and t_j are different for any $i \neq j$. This is because the length of the longest edge of t_i is increasing with i .

Property 2. The angles of the refined triangles are greater than a positive constant.

Let M^k be the refined mesh of M obtained in Algorithm 5.1. Denote the smallest angles of triangles in M and triangles in M^k by θ_0 and θ_0^k , respectively. It can be proven that $\theta_0^k \geq \frac{\theta_0}{2}$.

Property 3. The area of a sub-triangle is not greater than half of the area of its parent triangle.

What remains now is to prove that Algorithm 5.1 will terminate after a finite number of iterations. With Properties 1 and 2, we can easily obtain

Lemma 5.1. *If Algorithm 5.1 works in an infinite loop, there exists a triangle sequence*

$\{\Delta A_n B_n C_n\}_{n=0}^\infty$, where $\Delta A_n B_n C_n \in \tau^n$, such that

$$\lim_{n \rightarrow \infty} \max\{\|A_n B_n\|, \|B_n C_n\|, \|A_n C_n\|\} = 0. \quad (5.2)$$

Lemma 5.2. Let $G = (G_1, G_2)^T : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^2$ be a C^2 continuous mapping. If the Jacobian of G satisfies $|\nabla G| > 0$ in Ω where $\nabla G = \begin{pmatrix} \frac{\partial G_1}{\partial x} & \frac{\partial G_1}{\partial y} \\ \frac{\partial G_2}{\partial x} & \frac{\partial G_2}{\partial y} \end{pmatrix}$, then for any $\epsilon > 0$, there exists an $\eta > 0$ such that for any triangle $\Delta ABC \subset \Omega$ satisfying

$$\begin{cases} \max(\sin \angle A, \sin \angle B, \sin \angle C) > \epsilon \\ \max(\|AB\|, \|BC\|, \|AC\|) < \eta \end{cases} \quad (5.3)$$

the transformed triangle $\Delta G(A)G(B)G(C)$ keeps the orientation of triangle ΔABC .

Proof: For any $\epsilon > 0$, we consider triangle ΔABC that satisfies $\max(\sin \angle A, \sin \angle B, \sin \angle C) > \epsilon$. Without loss of generality, we assume that vertices A , B and C are ordered counter-clockwise and $\sin \angle A = \max(\sin \angle A, \sin \angle B, \sin \angle C)$. We introduce operator $\otimes$ for 2D vectors $V = (a, b)$ and $W = (c, d)$ such that $V \otimes W = ad - bc$. Then

$$(B - A) \otimes (C - A) = \|AB\| \|AC\| \sin \angle A > \|AB\| \|AC\| \epsilon > 0. \quad (5.4)$$

For any $V \in \Delta ABC$, we have

$$G_i(V) = G_i(A) + \left(\frac{\partial G_i(A)}{\partial x}, \frac{\partial G_i(A)}{\partial y} \right) (V - A) + O_{i,A,V} \quad (5.5)$$

where

$$O_{i,A,V} = (V - A)^T \begin{bmatrix} \frac{\partial^2 G_i(V_i^*)}{\partial x^2} & \frac{\partial^2 G_i(V_i^*)}{\partial x \partial y} \\ \frac{\partial^2 G_i(V_i^*)}{\partial x \partial y} & \frac{\partial^2 G_i(V_i^*)}{\partial y^2} \end{bmatrix} (V - A)$$

with a point V_i^* on the line connecting V and A . Let $\mathbf{O}_{A,V} = (O_{1,A,V}, O_{2,A,V})^T$. Since Ω is bounded, G is C^2 continuous and $|\nabla G| > 0$, there exist positive numbers μ, ν and L such

that $\|\mathbf{O}_{A,V}\| \leq \mu\|V - A\|^2$ and $L < \|\nabla G\|_F < \nu$ for $V \in \Omega$ where $\|\nabla G\|_F$ represents the Frobenius norm of the matrix G .

Substituting $V = B$ and $V = C$ into Eq.(5.4) gives

$$\begin{aligned} G(B) &= G(A) + \nabla G(B - A) + \mathbf{O}_{A,B}, \\ G(C) &= G(A) + \nabla G(C - A) + \mathbf{O}_{A,C}. \end{aligned}$$

After calculation, we have

$$(G(B) - G(A)) \otimes (G(C) - G(A)) = |\nabla G|(B - A) \otimes (C - A) + \psi$$

with $\psi = \nabla G(B - A) \otimes \mathbf{O}_{A,C} - \nabla G(C - A) \otimes \mathbf{O}_{A,B} + \mathbf{O}_{A,B} \otimes \mathbf{O}_{A,C}$. We choose $\eta = \min\left(1, \frac{L\epsilon}{2\mu\nu + \mu^2}\right)$. If $\max(\|AB\|, \|BC\|, \|AC\|) < \eta$, then

$$\begin{aligned} \|\psi\| &\leq \nu\mu\|B - A\|\|C - A\|^2 + \nu\mu\|C - A\|\|B - A\|^2 \\ &\quad + \mu^2\|A - B\|^2\|A - C\|^2 < \|AB\|\|AC\|(2\mu\nu + \mu^2)\eta \\ &\leq \|AB\|\|AC\|L\epsilon \leq |\nabla G|(B - A) \otimes (C - A). \end{aligned}$$

Therefore $(G(B) - G(A)) \otimes (G(C) - G(A)) > 0$. That is, the transformed triangle $\triangle G(A)G(B)G(C)$ keeps the orientation of triangle $\triangle ABC$. ■

The geometric meaning of Lemma 5.2 is that a C^2 continuous mapping G with positive Jacobian can ensure a triangle transformed by the induced mapping of G to maintain its orientation as long as it is sufficiently small.

Theorem 5.1. *If $G : \Omega \rightarrow \mathbb{R}^2$ is a C^2 continuous mapping with $|\nabla G| > 0$ in Ω , Algorithm 5.1 will terminate after a finite number of iterations.*

Proof: If Algorithm 5.1 cannot terminate in a finite number of iterations, according to Lemma 5.1, we can find a folding-over triangle sequence $\{\triangle A_i B_i C_i\}_{i=1}^\infty$ that satisfies

$$\lim_{i \rightarrow \infty} \max(\|A_i B_i\|, \|B_i C_i\|, \|A_i C_i\|) = 0.$$

Property 2 ensures that

$$\max(\sin \angle A_n, \sin \angle B_n, \sin \angle C_n) \geq \sin \left(\frac{\theta_0}{2} \right)$$

where θ_0 is the smallest angle of the triangles in M . We let $\epsilon = \sin \left(\frac{\theta_0}{2} \right)$. By Lemma 5.2, there exists an η such that when the diameter of the triangle is less than η , its transformed counterpart keeps the orientation, which contradicts the fact that $\triangle A_n B_n C_n$ are folding-over triangles. ■

5.3 Algorithm for 2D Mesh Transformation with Positional Constraints

The observations and theory in preceding section suggest an approach to constructing a valid 2D mesh transformation, via some C^2 continuous mapping together with the LEB-based refinement if necessary. This section presents such an algorithm.

Similar to some previous work such as [42], the input to our algorithm is composed of a 2D triangular mesh M , a set of constrained mesh vertices $V_c = \{V_{c,i}\}$, and a set of matching points $P_c = \{P_{c,i}\}$ corresponding to V_c . The output is a valid transformed mesh of M or its refinement, in which all $V_{c,i}$ are replaced by $P_{c,i}$. The algorithm contains four components listed below. More details of the components are described in the following subsections.

(1) Preprocess: We first perform a similarity transformation to roughly align the constrained point pairs in order to reduce unnecessary warping in subsequent steps. Then we embed the transformed 2D mesh in a bounding rectangle and the region between the mesh boundary and the rectangle is triangulated. This generates a new mesh that expands the original one. We fix the vertices on the rectangular boundary and add them to V_c and P_c .

(2) Warping trajectories: For constrained vertices V_c and their matching points P_c , a set

of immediate points $C^j = \{C_i^j\}, j = 0, 1, \dots, m$ with $C^0 = V_c$ and $C^m = P_c$ is generated. Each C_i^j is connected to C_i^{j+1} by a straight line. All these line segments form piecewise linear trajectories, along which the constrained vertices V_c move to P_c during the warping (or iterative mapping). These trajectories do not intersect each other.

(3) RBF-based mapping: The mapping from C^j to C^{j+1} is achieved by iteratively applying the RBF-based interpolation technique. Rather than interpolating absolute positions, we interpolate the displacements of the constrained vertices. A safe stepsize that ensures the mapping derived from the RBF-based interpolation to be locally bijective on the mesh is presented.

(4) Foldover free transformation: Mesh transformation is induced from the RBF-based mapping. A stepsize for the mesh transformation to be foldover free is derived. If the stepsize is sufficient to make the transformation reach the target or the stepsize is greater than or equal to the safe stepsize of the RBF-based mapping, the transformation is constructed. Otherwise, to avoid using too small stepsize, we use the LEB-based refinement (i.e., Algorithm 5.1) to refine the mesh. Then we repeat this step by re-inducing the mesh transformation.

5.3.1 Preprocess

A similarity transformation is composed of translation, rotation and uniform scaling, thus preserving angles, and can be represented by

$$u = ax - by + c, \quad v = bx + ay + d$$

where (x, y) and (u, v) represents the coordinates of points before and after the transformation, respectively, and coefficients a, b, c and d to be determined. Suppose that $V_{c,i}$ and $P_{c,i}$ have coordinates (x_i, y_i) and (u_i, v_i) for $i = 1, 2, \dots, h$. We find the similarity transforma-

tion by minimizing the following objective function:

$$\sum_{i=1}^h ((ax_i - by_i + c - u_i)^2 + (bx_i + ay_i + d - v_i)^2)$$

which gives

$$\begin{aligned} a &= \frac{h(\sum_{i=1}^h u_i x_i + \sum_{i=1}^h v_i y_i) - \sum_{i=1}^h x_i \sum_{i=1}^h u_i - \sum_{i=1}^h y_i \sum_{i=1}^h v_i}{h \sum_{i=1}^h (x_i^2 + y_i^2) - (\sum_{i=1}^h x_i)^2 - (\sum_{i=1}^h y_i)^2} \\ b &= \frac{h(\sum_{i=1}^h v_i x_i - \sum_{i=1}^h u_i y_i) + \sum_{i=1}^h y_i \sum_{i=1}^h u_i - \sum_{i=1}^h x_i \sum_{i=1}^h v_i}{h \sum_{i=1}^h (x_i^2 + y_i^2) - (\sum_{i=1}^h x_i)^2 - (\sum_{i=1}^h y_i)^2} \\ c &= \frac{\sum_{i=1}^h u_i - (\sum_{i=1}^h x_i)a + (\sum_{i=1}^h y_i)b}{h} \\ d &= \frac{\sum_{i=1}^h v_i - (\sum_{i=1}^h y_i)a - (\sum_{i=1}^h x_i)b}{h}. \end{aligned}$$

The transformation is applied to mesh M .

In order to reduce the distortion, the boundary of M is allowed to move freely. However, we need a fixed boundary surrounding the entire parameter domain to prevent the boundary vertices from walking into another triangle. Thus we add a rectangular virtual boundary to the mesh M , and the region between the boundary of the M and the virtual one is triangulated using constrained Delaunay triangulation. This generates an expanded mesh, as shown in Figure 5.8. The expanded mesh is used as M in the subsequent processes and the vertices on the virtual boundary are added into the constrained vertex set. To simplify notations, in the following we still use V_c and P_c to represent the constrained mesh vertices and the matching points after the preprocess without causing ambiguity.

5.3.2 Construction of non-intersecting warping trajectories

Given constrained vertices $V_c = \{V_{c,i}\}$ and their matching points $P_c = \{P_{c,i}\}$, directly connecting each corresponding point pair often causes intersection. We aim to find non-intersecting trajectories for V_c to move to P_c . Specifically, for each vertex $V_{c,i}$, a trajectory is a polyline $C_i(s)$ with $m + 1$ nodes $C_i^j, j = 0, 1, \dots, m$ where $C_i^0 = V_{c,i}$ and $C_i^m = P_{c,i}$.

Figure 5.8: Bound the 2D embedding with a virtual rectangular boundary.

$\mathbf{C}_i(s)$ is parameterized by

$$\mathbf{C}_i(s) = (1 - ms + j)\mathbf{C}_i^j + (ms - j)\mathbf{C}_i^{j+1}, s \in [\frac{j}{m}, \frac{j+1}{m}].$$

Non-intersection of $\mathbf{C}_i(s)$ means that for any $i \neq k$ and any $s \in [0, 1]$, $\mathbf{C}_i(s) \neq \mathbf{C}_k(s)$.

Computing non-intersecting trajectories of points from the source to the target has been well studied in morphing [115], which requires the input to be two compatible triangulations. Two triangulations are called compatible if their face lattices are isomorphic. In general, determining whether two sets of points are compatible is NP-hard [90, 108].

Here we propose to compute compatible triangulations of two point sets using the warping scheme of [45]. First, we perform Delaunay triangulation on the constrained vertices V_c , yielding a triangular mesh S . Then we let $T = S$. T serves as the input mesh for the warping scheme of [45]. Iteratively, the warping scheme aligns T with the matching positions P_c . If we need to refine T during the iteration, we apply the same refinement to S , too. In this way, the warping scheme preserves the compatibility and the final T that interpolates P_c has the same connectivity as S . Since the input in this approach consists of only the constrained vertices, the process of computing compatible triangulations is usually

Figure 5.9: Generation of non-intersecting warping trajectories. (a) The input mesh and constrained points. (b) The Delaunay triangulation on the constrained points. (c) and (d) The compatible triangulations of the source and target. (e) The warping trajectories. (f) Foldover-free mapping.

very fast. Figure 5.9 shows a simple example of generating compatible triangulations.

After compatible triangulations S and T are obtained, the morphing algorithm in [115] is used to generate the non-intersecting trajectories of the constrained vertices. Then $C_i(s)$ can be obtained by appropriately sampling the trajectories.

5.3.3 Iterative RBF-based mapping

Once the construction of warping trajectories is complete, we obtain $\{C_i^0\} \rightarrow \{C_i^1\} \rightarrow \dots \rightarrow \{C_i^m\}$. Now we consider the mapping over the span $\{C_i^j\} \rightarrow \{C_i^{j+1}\}$. The mapping over the other spans is similarly constructed.

The mapping is an interpolation function that maps C_i^j to C_i^{j+1} . We use radial basis functions for this purpose. Radial basis functions are popular for scattered data interpolation. They do not require that the data points lie on regular grids. Notice that there may not exist a single locally bijective RBF-based interpolation. Our idea is to interpolate the displacements $C_i^{j+1} - C_i^j$ by several steps and to assure that the interpolation at each step is locally bijective.

Let $C_i^{j,l}$ be the point obtained from C_i^j after l steps, with $C_i^{j,0} = C_i^j$. Also denote by $M^{j,l}$ the 2D triangular mesh at the step where the constrained vertices become $\{C_i^{j,l}\}$. Let $f_i^j = C_i^{j+1} - C_i^j$, $i = 1, 2, \dots, h$, be the displacements of the span $\{C_i^j\} \rightarrow \{C_i^{j+1}\}$. We want to find a smooth interpolating function

$$D^{j,l}(V) = \sum_{i=1}^h \gamma_i \phi(|V - C_i^{j,l}|) + a_1x + a_2y + a_3 \quad (5.6)$$

satisfying

$$D^{j,l}(C_i^{j,l}) = \delta^{j,l} f_i^j \quad (5.7)$$

where $V = (x, y)^T$, $\phi(r) = r^2 \log(r)$, and $\delta^{j,l}$ is a stepsize controlling the displacement. Here $a_1x + a_2y + a_3$ is linear in x and y accounting for linear transformation.

The constraints (5.7) give h equations. To determine the coefficients γ_i , the following

orthogonality conditions are introduced:

$$\sum_{i=1}^h \gamma_i = \sum_{i=1}^h \gamma_i x_i = \sum_{i=1}^h \gamma_i y_i = 0. \quad (5.8)$$

Hence we have a linear system of size $(h + 3) \otimes (h + 3)$:

$$K^{j,l} \begin{pmatrix} \mathbf{\Gamma} \\ \mathbf{A} \end{pmatrix} = \begin{pmatrix} \delta^{j,l} \mathbf{f}^j \\ \mathbf{0} \end{pmatrix} \quad (5.9)$$

where

$$K^{j,l} = \begin{bmatrix} \Phi_{11} & \Phi_{12} & \cdots & \Phi_{1h} & x_1 & y_1 & 1 \\ \Phi_{21} & \Phi_{22} & \cdots & \Phi_{2h} & x_2 & y_2 & 1 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ \Phi_{h1} & \Phi_{h2} & \cdots & \Phi_{hh} & x_h & y_h & 1 \\ x_1 & x_2 & \cdots & x_h & 0 & 0 & 0 \\ y_1 & y_2 & \cdots & y_h & 0 & 0 & 0 \\ 1 & 1 & \cdots & 1 & 0 & 0 & 0 \end{bmatrix} \quad (5.10)$$

with $\Phi_{i,k} = \phi(\|C_i^{j,l} - C_k^{j,l}\|)$, $\mathbf{\Gamma} = (\gamma_1, \dots, \gamma_h)^T$, $\mathbf{A} = (a_1, a_2, a_3)^T$, and $\mathbf{f}^j = (f_1^j, \dots, f_h^j)^T$.

It has been known that the linear system (5.10) is guaranteed to be invertible if the locations of the data points to be interpolated do not lie on a line and do not intersect. This can be easily done if we move the constrained vertices along the warping trajectories generated in Section 5.3.2. Then the solution to the linear system is:

$$\begin{pmatrix} \mathbf{\Gamma} \\ \mathbf{A} \end{pmatrix} = (K^{j,l})^{-1} \begin{pmatrix} \mathbf{f}^j \\ \mathbf{0} \end{pmatrix} \delta^{j,l}. \quad (5.11)$$

After obtaining the RBF's coefficients, we define a mapping $g^{j,l} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ for this

step:

$$g^{j,l}(V) = V + D^{j,l}(V) = V + \mathbf{M}^{j,l}(V) (K^{j,l})^{-1} \begin{pmatrix} \mathbf{f}^j \\ \mathbf{0} \end{pmatrix} \delta^{j,l} \quad (5.12)$$

where

$$\mathbf{M}^{j,l}(V) = \left(\phi(\|V - C_1^{j,l}\|^2), \dots, \phi(\|V - C_h^{j,l}\|^2), x, y, 1 \right). \quad (5.13)$$

Let K^x and K^y represent two $(h+3)$ -dimensional column vectors consisting of the x and y components of $(K^{j,l})^{-1} \begin{pmatrix} \mathbf{f}^j \\ \mathbf{0} \end{pmatrix}$, respectively. The Jacobian of $g^{j,l}$ can be written:

$$\begin{aligned} |\nabla g^{j,l}| &= \begin{vmatrix} 1 + \frac{\partial \mathbf{M}^{j,l}(V)}{\partial x} K^x \delta^{j,l} & \frac{\partial \mathbf{M}^{j,l}(V)}{\partial y} K^x \delta^{j,l} \\ \frac{\partial \mathbf{M}^{j,l}(V)}{\partial x} K^y \delta^{j,l} & 1 + \frac{\partial \mathbf{M}^{j,l}(V)}{\partial y} K^y \delta^{j,l} \end{vmatrix} \\ &= \alpha (\delta^{j,l})^2 + \beta \delta^{j,l} + 1 \end{aligned}$$

where

$$\begin{aligned} \alpha &= \left(\frac{\partial \mathbf{M}^{j,l}(V)}{\partial x} K^x \right) \left(\frac{\partial \mathbf{M}^{j,l}(V)}{\partial y} K^y \right) \\ &\quad - \left(\frac{\partial \mathbf{M}^{j,l}(V)}{\partial y} K^x \right) \left(\frac{\partial \mathbf{M}^{j,l}(V)}{\partial x} K^y \right), \\ \beta &= \left(\frac{\partial \mathbf{M}^{j,l}(V)}{\partial x} K^x \right) + \left(\frac{\partial \mathbf{M}^{j,l}(V)}{\partial y} K^y \right). \end{aligned}$$

Apparently, $|\nabla g^{j,l}|$ is a quadratic function in $\delta^{j,l}$. When $\delta^{j,l} = 0$, $|\nabla g^{j,l}| = 1$ is positive, which implies that if $\delta^{j,l}$ is sufficiently small, the Jacobian of $g^{j,l}$ is positive and is thus locally bijective. Next we show how to find an appropriate stepsize $\delta^{j,l}$ according to three situations:

- $\alpha = 0$: If $\beta \geq 0$, then for any $\delta^{j,l}$, $|\nabla g^{j,l}|$ is always positive. If $\beta < 0$, $|\nabla g^{j,l}|$ has one positive root $\delta = -1/\beta$. When $\delta^{j,l} < \delta$, $|\nabla g^{j,l}| > 0$.
- $\alpha < 0$: $|\nabla g^{j,l}|$ has one positive root $\delta = \frac{-\beta - \sqrt{\beta^2 - 4\alpha}}{2\alpha}$. When $0 < \delta^{j,l} < \delta$, $|\nabla g^{j,l}|$ is positive.
- $\alpha > 0$: If $\beta \geq 0$ or $\beta^2 - 4\alpha < 0$, then for any $\delta^{j,l}$, $|\nabla g^{j,l}|$ is positive. Otherwise, $|\nabla g^{j,l}|$

has two positive roots. The smaller one is $\delta = \frac{-\beta - \sqrt{\beta^2 - 4\alpha}}{2\alpha}$ and for any $\delta^{j,l} \in (0, \delta)$, $|\nabla g^{j,l}| > 0$.

If we set

$$\delta = \frac{2}{|-\beta + \sqrt{|\beta^2 - 4\alpha|}|}, \quad (5.14)$$

it can be verified by some calculations that any $\delta^{j,l} \in (0, \delta)$ satisfies the conditions that assures $|\nabla g^{j,l}| > 0$ in all above three situations. That is, the δ in Eq.(5.14) is a bound sufficient for $\delta^{j,l}$ to guarantee positivity of $|\nabla g^{j,l}|$. Note that $|-\beta + \sqrt{|\beta^2 - 4\alpha|}|$ is a continuous function in $V \in \Omega$. Hence it can achieve its maximum value $\max_{V \in \Omega} |-\beta + \sqrt{|\beta^2 - 4\alpha|}|$. Therefore if we define

$$\delta_{RBF} = \min \left\{ \rho \frac{2}{\max_{V \in \Omega} |-\beta + \sqrt{|\beta^2 - 4\alpha|}|}, 1 \right\} \quad (5.15)$$

with a positive $\rho < 1$ (in this paper we set $\rho = 0.9$), δ_{RBF} is a positive number such that for any $\delta^{j,l} \in (0, \delta_{RBF}]$, $|\nabla g^{j,l}| > 0$ holds. Thus in the warping process we let $\delta^{j,l} = \delta_{RBF}$. When $\sum_l^{\kappa} \delta^{j,l} \geq 1$ for a positive integer κ , the interpolation process successfully reaches C^{j+1} . A locally bijective RBF-based mapping from C^j to C^{j+1} can be obtained by composing each $g^{j,l}$: $g^j = g^{j,1} \circ g^{j,2} \circ \dots \circ g^{j,\kappa}$.

5.3.4 Foldover-free induced transformation

While we can construct a locally bijective RBF-mapping g^j as in Section 5.3.3 and then construct the induced transformation from it together with an LEB-based refinement described in Section 5.2.2, the stepsize δ_{RBF} may be too conservative, which makes the process require a large number of iterations. On the other hand, Observation 2 in Section 5.2.1 suggests that we determine the stepsize $\delta^{j,l}$ directly using the condition for the induced transformation $(g^{j,l})_{M^{j,l}}^*$ to be foldover free, with which the RBF-based mapping $g^{j,l}$ may even not be locally bijective.

Consider a triangle $\triangle A_1 A_2 A_3 \subset M^{j,l}$. It is mapped by the RBF-based mapping $g^{j,l}$ to $\triangle B_1 B_2 B_3$ with $B_i = g^{j,l}(A_i) = A_i + \mathbf{M}^{j,l}(A_i) (K^{j,l})^{-1} \begin{pmatrix} \mathbf{f}^j \\ \mathbf{0} \end{pmatrix} \delta^{j,l}$. The condition for $\triangle B_1 B_2 B_3$ and $\triangle A_1 A_2 A_3$ to have the same orientation is

$$((A_2 - A_1) \otimes (A_3 - A_1)) \cdot ((B_2 - B_1) \otimes (B_3 - B_1)) > 0$$

which gives

$$F(\delta^{j,l}) \triangleq a(\delta^{j,l})^2 + b\delta^{j,l} + 1 > 0$$

where

$$\begin{aligned} n_1 &= \frac{1}{(A_2 - A_1) \otimes (A_3 - A_1)} \\ a &= n_1 \left((\mathbf{M}^{j,l}(A_2) - \mathbf{M}^{j,l}(A_1)) (K^{j,l})^{-1} \begin{pmatrix} \mathbf{f}^j \\ \mathbf{0} \end{pmatrix} \right) \\ &\quad \otimes \left((\mathbf{M}^{j,l}(A_3) - \mathbf{M}^{j,l}(A_1)) (K^{j,l})^{-1} \begin{pmatrix} \mathbf{f}^j \\ \mathbf{0} \end{pmatrix} \right) \\ b &= n_1 (A_2 - A_1) \otimes (\mathbf{M}^{j,l}(A_3) - \mathbf{M}^{j,l}(A_1)) (K^{j,l})^{-1} \begin{pmatrix} \mathbf{f}^j \\ \mathbf{0} \end{pmatrix} \\ &\quad - n_1 (A_3 - A_1) \otimes (\mathbf{M}^{j,l}(A_2) - \mathbf{M}^{j,l}(A_1)) (K^{j,l})^{-1} \begin{pmatrix} \mathbf{f}^j \\ \mathbf{0} \end{pmatrix} \end{aligned}$$

Now we look for a $\delta \in (0, 1]$ such that for any $\delta^{j,l} \leq \delta$, $F(\delta^{j,l}) > 0$. Following the approach in Section 5.3.3, we can find the following δ that satisfies the requirement:

$$\delta = \begin{cases} \min \left\{ \rho \frac{2}{-b + \sqrt{b^2 - 4a}}, 1 \right\}, & \text{if } a < 0 \text{ or } (b < 0, b^2 \geq 4a) \\ 1, & \text{otherwise} \end{cases} \quad (5.16)$$

with a positive $\rho < 1$. We find such a δ for each triangle in mesh $M^{j,l}$ to assure no foldover and then choose the smallest one.

Once we obtain the stepsize δ , the algorithm proceeds in three different ways:

- If δ is greater than or equal to $1 - \sum_{k=1}^{l-1} \delta^{j,k}$ which is sufficient for the process to reach the target C^{j+1} , we simply let $\delta^{j,l} = 1 - \sum_{k=1}^{l-1} \delta^{j,k}$, construct the RBF-based mapping $g^{j,l}$ accordingly and then the induced transformation $(g^{j,l})^*_{M^{j,l}}$.
- Else if $\delta \geq \delta_{RBF}$, we let $\delta^{j,l} = \delta$, and construct the RBF-based mapping $g^{j,l}$ and then the induced transformation $(g^{j,l})^*_{M^{j,l}}$.
- Otherwise, $\delta < \delta_{RBF}$. If we use this δ for computing the RBF-based mapping without adding Steiner points, in the subsequent warping δ may become smaller and smaller and the process cannot reach the target. Figure 5.10 shows such an example, where A cannot cross the line BC without refinement. To overcome this problem, we let $\delta^{j,l} = \delta_{RBF}$ and construct a locally bijective RBF-based mapping $g^{j,l}$. Then Algorithm 5.1 (LEB-based refinement) is applied to refine $M^{j,l}$ to construct a foldover free induced transformation.

Figure 5.10: An example of constrained mapping that maps A, B and C to A^*, B and C , respectively.

Summarizing all the above steps, we arrive at a complete algorithm which is outlined in Algorithm 5.2. Moreover, Algorithm 5.2 can be proven to always work.

Theorem 5.2. *Algorithm 5.2 terminates in a finite number of steps and outputs a valid 2D mesh transformation that satisfies given positional constraints.*

Proof: We only need to prove that the proposed warping process can reach the target in a finite number of iterations. In fact, it can be seen that each stepsize $\delta^{j,l}$ in the warping

Algorithm 5.2 Constrained 2D Mesh Transformation

Given:

- 1: Preprocess.
- 2: Compute compatible triangulation of the user specified constrained vertices V_c and the matching points P_c .
- 3: Construct non-intersecting warping trajectories $\{C_i(s)\}$.
- 4: $j \leftarrow 0$

Goal:

- 5: **while** $j \neq m$ **do**
- 6: Set $remaining_ratio = 1$
- 7: **while** $remaining_ratio > 0$ **do**
- 8: Compute δ_{RBF} at $M^{j,l}$ using (5.15).
- 9: Compute the stepsize δ using (5.16).
- 10: **if** $\delta \geq remaining_ratio$ **then**
- 11: Set $\delta = remaining_ratio$.
- 12: Set $remaining_ratio = 0$.
- 13: Compute induced mesh transformation.
- 14: Break.
- 15: **end if**
- 16: **if** $\delta \geq \delta_{RBF}$ **then**
- 17: Compute induced mesh transformation.
- 18: **else**
- 19: Set $\delta = \delta_{RBF}$
- 20: Apply Algorithm 5.1 to the mesh.
- 21: **end if**
- 22: $remaining_ratio \leftarrow remaining_ratio - \delta$.
- 23: **end while**
- 24: $j \leftarrow j + 1$.
- 25: **end while**

Output: M .

process except for the last step reaching the target is always greater than or equal to δ_{RBF} .

If we define

$$\delta_{RBF}^* = \min \left\{ \rho \frac{2}{\max_{V \in \Omega, s \in [0,1]} | -\beta + \sqrt{|\beta^2 - 4\alpha|} |}, 1 \right\},$$

δ_{RBF}^* is a positive constant and $\delta_{RBF} \geq \delta_{RBF}^*$ for all δ_{RBF} . Thus the warping from C^j to C^{j+1} needs at most $\lceil \frac{1}{\delta_{RBF}^*} \rceil$ steps. This completes the proof. ■

Figure 5.11 is an example of constrained 2D transformations. The input is given in Figure 5.11(a), which includes a 2D triangular mesh and eight constraints highlighted in green. The four corner vertices are mapped to themselves and the four inner vertices are mapped to their respective next counterparts, as indicated by red arrows. Using our algorithm, a foldover free transformation is automatically constructed with 13 Steiner points being added and the transformed mesh is shown in Figure 5.11(b). As a comparison, we run the algorithm of [139] on this input. The constructed constrained mapping G is locally bijective, but the induced transformation is not foldover free, which is depicted in Figure 5.11(c).

5.4 Experimental Results

This section provides several examples to demonstrate the proposed algorithm. These examples have varying complexity. The number of faces in these triangular mesh models ranges from 2K to 21K and the number of constrained vertices ranges from 21 to 83. Note that among only a few algorithms that can guarantee hard constraints, [45] is a relatively recent one, which is able to handle complicated constraints while adding only a small number of Steiner points, and output visually pleasing results. Hence we also provide the experimental results obtained by [45] for comparison.

Figures 5.12-5.14 show the visual results. In the figures, the first column gives the indices of the models. The second column shows the input texture on the top and the 3D mesh model at the bottom. The constrained points are highlighted in green. The third col-

Figure 5.11: (a) An input mesh with 8 constraints. The 4 corner vertices are constrained to remain unchanged and each of the 4 inner vertices moves to its next as indicated by the red arrows. (b) A foldover free mesh is obtained by our algorithm, which adds 13 Steiner points. (c) Using the algorithm of [139], we can obtain a locally bijective mapping G , which is depicted by the red curves in the zoomed window, but foldover occurs when the mapped edges are straightened.

umn and fourth column display the models textured by a checker image by [45] and our algorithm, respectively. The use of the checker image well depicts the difference of the results created by our method and [45]. It can be seen that our algorithm usually produces a smoother mapping than [45] though a non-linear post-optimization has been performed in [45]. For example, the zoomed view of the cow model (i.e., model (g)) clearly shows the difference. In fact, [45] produces visually apparent distortions in the areas around the constrained points. The last column shows the alignment of the textures with the constrained embedding of the meshes at the top and the constrained textured 3D meshes at the bottom.

Table 5.1 shows some statistics of these texture mapping examples. The fifth column reports the running time of constructing the constrained mapping on an Intel Pentium 4, 3.6GHz PC with 1G RAM. The algorithm is implemented using C++. The sixth and eighth

Figure 5.12: Texturing models (a)-(c).

	Inputs	Results of [1]	Our results	
(d)
(e)
(f)

Figure 5.13: Texturing models (d)-(f).

Figure 5.14: Texturing models (g)-(i).

columns show the numbers of the added Steiner vertices using our LEB-based algorithm and [45], respectively. It can be found that except for model (f), the number of Steiner vertices added in our method is comparable to that of [45]. Model (f) is special because it contains two cusps in its two horns and has a low resolution. When the resolution increases, the number of Steiner vertices decreases as demonstrated in model (g). It is also worth pointing out that in [42] and [45] a postprocess that removes those unnecessary Steiner vertices is performed and the current implementation of our algorithm does not perform such a postprocess. In future we will incorporate the postprocess into our algorithm, by which the number of Steiner vertices is expected to be reduced significantly as in [42] and [45]. In addition, while we propose to use LEB-based refinement in this paper, there are other possibilities to refine the mesh. For example, we can split a triangle by bisecting its largest angle, which we call the largest angle bisection (LAB). We have tested our algorithm by replacing the LEB-based refinement by the LAB-based refinement, which usually results in fewer Steiner points. However, whether the LAB-based refinement always works is not clear yet, which warrants further investigation.

Table 5.1: Statistics of the texture mapping examples in Figures 5.12-5.14

Models	#Vertices	#Triangles	#Constrained vertices	Time (Sec)	#Steiner (LEB)	#Steiner (LAB)	#Steiner ([45])
model (a)	1770	3526	24	0.656	16	7	7
model (b)	1808	3602	25	0.063	0	0	10
model (c)	10017	20008	54	0.766	0	0	34
model (d)	1772	3450	21	0.469	16	4	21
model (e)	1657	3300	27	0.078	0	0	2
model (f)	1697	3384	32	11.593	1382	167	64
model (g)	10736	21404	32	3.922	54	4	8
model (h)	5184	10354	83	6.501	35	17	38
model (i)	4149	8284	71	1.719	6	2	25

The experimental results are also evaluated quantitatively using the stretches defined in [92]. Therein, the L_2 norm measures the overall stretch of the parameterization and the L_∞ norm represents the least stretch. A good parameterization is supposed to have small stretches. Table 5.2 presents the quantitative results of all these texture mapping examples. It can be seen that the stretches of our method are smaller than or comparable to the stretches of [45].

Table 5.2: Stretches of the texture mapping examples in Figures 5.12-5.14

Models	Proposed method		Method of [45]	
	L_2	L_∞	L_2	L_∞
model (a)	4.06561	78.958	2.79953	162.037
model (b)	1.20559	2.1982	1.20486	3.41913
model (c)	1.58255	22.452	373.289	83748.3
model (d)	3.10319	38.664	470.781	31045.9
model (e)	1.13447	4.11239	1.10433	4.2356
model (f)	564.94	23146.9	1160.47	66151.9
model (g)	1713.13	207825	1884.34	372774
model (h)	2.06733	104.374	85.2849	14365
model (i)	2.95445	79.2665	1285.42	262252

5.5 Conclusions

We have carefully analyzed the relation between a C^2 continuous 2D mapping and its induced piece-wise linear transformation and proposed a refinement strategy based on the longest edge bisection, which guarantees a transformation induced from a locally bijective C^2 mapping, by adding a few Steiner vertices if necessary, to keep the orientation of each triangle of a 2D triangular mesh. Based on this, we present an efficient and theoretically robust texture mapping algorithm for triangular mesh models in the presence of hard constraints. The mapping is a composition of an unconstrained planar embedding and a series of constrained mesh transformations. The constrained mesh transformations are realized by a non-intersecting warping for constrained vertices, RBF-based interpolation and LEB-based refinement, which are proven to be foldover free. The condition for a mesh transformation to be foldover free and the condition for the RBF-based warp to be locally bijective are derived to determine the displacement step and the threshold for performing mesh refinement, which ensures the efficiency and validity of the algorithm. The use of RBF-based interpolation makes the mesh be smoothly deformed to align the user specified positional constraints exactly, without the need of performing a smoothing post-process. The experiments with several examples of varying complexity demonstrate that the proposed algorithm can effectively handle hard constraints and produce visually pleasing texture mapping results.

Chapter 6

Physically based T-spline Morphing with Speed Control

6.1 Introduction

Morphing, also known as shape metamorphosis, is a technique used to automatically generate the smooth transformation from a source into a target object. It has wide applications in computer graphics, geometric design, animation, and entertaining industry. It has been well understood that morphing technically involves two subproblems: vertex correspondence and trajectory.

While solutions to these problems mainly fall in the domain of polygonal meshes, here we concentrate on the objects represented by T-spline surfaces. Very little work has addressed morphing between smooth parametric surfaces. One may think this problem can be simply solved through tessellating the inputs into polygonal meshes, since so many powerful morphing techniques exist for mesh surfaces. However, in some applications, such as geometric modeling, the intermediate shape is demanded to be expressed in parametric representation. Then such a brute force approach is not suitable.

Paper [36] discussed morphing between two NURBS surface. As T-splines are both forward and backward compatible with NURBS, the method in [36] can be readily applied

to T-spline surfaces. However, this approach has several drawbacks. First, the conversion into NURBS will increase a lot of degrees of freedom (DOF), thereby increasing computational complexity. Second, the method of [36] mainly focused on the vertex trajectory. For the correspondence issue, it simply applied knot insertion and degree elevation to ensure the structural compatibility of the control mesh. Hence, feature alignment is ignored. As a result, the apparent features disappear during the transition as shown in Figure 6.2. Third, the method requires the user input of mass distribution to compute the vertex paths. Then a tedious trial and error approach has to be employed in order to determine a satisfactory transition. Moreover, as the method mainly focuses on morphing between two frames, it can not guarantee the smoothness between consecutive morphing sequences.

In this chapter, we present a comprehensive solution to T-spline morphing in both stages. For the correspondence issue, in addition to guarantee the structural compatibility, we also take the feature alignment into consideration. Feature alignment and structural compatibility of the control mesh are fulfilled simultaneously through consistent approximation, which aims to restore several surfaces within one T-spline class. In order to prevent the unwanted small wiggles and flattening that appear frequently in linear averaging, we develop a physically based morphing approach to compute the trajectory path. Furthermore, T-spline volume force is incorporated into the physical system providing sufficient extra DOFs, which enables additional control over the morphing process including space and time constraints. Consequently, we are able to automatically generate a series of natural and physically-plausible in-between shapes. Our major contributions are outlined as follows:

- To the best of our knowledge, we are the first to compute an inter-map between two parametric surfaces while preserving the user specified feature correspondence.
- We propose a new technique called consistent approximation. It could represent different surfaces in the same T-spline class, which is a crucial task in many applications including surface matching and example based processing.

- We develop a physically based morphing approach directly based on the T-spline representation, thus avoiding an additional T-spline conversion.
- We introduce a novel approach to add extra DOFs into a physical system while maintaining the analytical solvability. This significantly enhance the controllability over the physical system.

6.2 Correspondence establishment

Figure 6.1: Input T-spline surfaces for morphing

Given two T-spline surfaces, $\mathbf{T}_0(u, v)$ and $\mathbf{T}_1(h, l)$, morphing is the process of gradual and continuous transformation from $\mathbf{T}_0$ into $\mathbf{T}_1$. We refer $\mathbf{T}_0(u, v)$ as the source surface and $\mathbf{T}_1(h, l)$ as the target surface.

6.2.1 Problem statement

Correspondence refers to a bijective inter-map between the source and target objects. A general approach to establish such a map is to parameterize both input objects into a common base domain. Compared with discrete polygonal meshes, an inherent parametrization has already existed in T-spline representation by definition, whose parametric domain is a 2D rectangular region. It means that we can easily establish a correspondence through map in the parametric domain,

$$(u, v) = (h, l) \quad (6.1)$$

Here we assume that the parametric domain for both surfaces are the same. This can always be satisfied through a parameter linear transformation. By this setting, it means the point $\mathbf{T}_0(u_0, v_0)$ on the source surface will be mapped to $\mathbf{T}_1(u_0, v_0)$ on the target surface. However, such inherent correspondence is not acceptable in general. The models which need to be metamorphosed often possess similar features. As shown in Figure 6.1 where the two input surfaces are both human faces, then the eyes must be mapped to the eyes, the nose to the ears, and so on. Otherwise, those crucial features will be blurred during transition (as shown in Figure 6.2). It means the desired inter-map should align these features. It is typically achieved through enforcing the correspondence for a small set of feature points. Suppose the corresponding feature points on the source and target objects are denoted as $\mathbf{x}_i$ and $\mathbf{y}_i$. Feature alignment requires

$$\mathbf{T}_0^{-1}(\mathbf{x}_i) = \mathbf{T}_1^{-1}(\mathbf{y}_i) \quad (6.2)$$

The superscript $^{-1}$ represents the inverse map. Then how to identify those common features is an important issue. It seems impossible to automatically find such common features, because they are mostly defined in a semantic rather than a geometric way [41]. Therefore, in our implementation, we let the users to identify these features and provide information about their location and correspondence as shown in Figure 6.3. (6.2) implies that the

Figure 6.2: Morphing without feature alignment. The transition is generated through linear interpolation of the surfaces in Figure 6.1

Figure 6.3: User specified feature correspondence

original T-spline representations have to be reparameterized to satisfy the feature alignment requirements. That is to find two 2D bijective maps $\mathbf{R}_0, \mathbf{R}_1$,

$$\begin{aligned}(u, v) &= \mathbf{R}_0(r, s) \\ (h, l) &= \mathbf{R}_1(r, s)\end{aligned}\tag{6.3}$$

such that

$$\begin{aligned}\mathbf{R}_0^{-1}(\mathbf{T}_0^{-1}(\mathbf{x}_i)) &= \frac{\mathbf{T}_0^{-1}(\mathbf{x}_i) + \mathbf{T}_1^{-1}(\mathbf{y}_i)}{2} \\ \mathbf{R}_1^{-1}(\mathbf{T}_1^{-1}(\mathbf{y}_i)) &= \frac{\mathbf{T}_0^{-1}(\mathbf{x}_i) + \mathbf{T}_1^{-1}(\mathbf{y}_i)}{2}\end{aligned}\tag{6.4}$$

Substituting them into the original surface representations yields

$$\begin{aligned}\mathbf{T}'_0(r, s) &= \mathbf{T}_0(\mathbf{R}_0(r, s)) \\ \mathbf{T}'_1(r, s) &= \mathbf{T}_1(\mathbf{R}_1(r, s))\end{aligned}\tag{6.5}$$

Then the map through

$$\mathbf{T}'_0(r_0, s_0) \hookrightarrow \mathbf{T}'_1(r_0, s_0)\tag{6.6}$$

establishes the correspondence where the prescribed features are aligned. Note that we choose to reparameterize the two surfaces simultaneously in order to maintain the equal position of the source and target objects.

In polygonal meshes, after such an inter-map is established, there is a further need to remesh the input objects to obtain a compatible triangulation, which still maintains the prescribed feature correspondence. Here, as the reparameterization covers the entire parametric region, there is no connectivity notion in parametric surfaces, not to mention the compatible triangulation. So we are ready to proceed to the next stage. While the geometry of the objects in mesh representation is determined by the position of the surface points, it's the control mesh as well as its structure that defines the surface shape. Note that remesh operation involved in mesh morphing aims to ensure the transitory object to be represented in the same mesh representation. Therefore, similarly when the input surfaces are in T-spline representation, it demands the output transitory surfaces also to keep such representation,

especially in the application of morphing based modeling. Then it's necessary to transform the reparameterized surfaces $\mathbf{T}'_0(r, s)$ and $\mathbf{T}'_1(r, s)$ back into T-spline representations. Unfortunately, as the reparameterization $\mathbf{R}_0, \mathbf{R}_1$ can not be represented as a simple closed form function, the exact conversion into T-spline representation is generally impossible. More than that, even if such conversion exists, we have to further make the resultant representations in the same T-spline class, which means the preimage of their control meshes should be identical. Otherwise, we are required to compute an additional smooth transition for the connectivity of the control mesh. Due to the flexibility of T-spline control mesh, such transition is also challenging. We refer those control T-mesh with identical preimage as compatible structure. We can see that the issue of compatible triangulation in mesh is replaced with compatible structure here. A brute force approach is to transform the two T-spline surfaces into NURBS representation, and to apply the knot insertion as suggested in [36]. Apparently, such method will produce a large number of extra control points, which is not generally acceptable.

6.2.2 Consistent approximation

In this section, we propose a novel approach to handle the correspondence issue for T-spline morphing. From the analysis above, it is hardly able to achieve the feature alignment and the structural compatibility simultaneously. Then instead of computing an explicit T-spline conversion, we switch to reconstruct the two surfaces with compatible T-mesh structure.

Problem 6.1. *Find two T-spline surfaces $\tilde{\mathbf{T}}_0(r, s)$ and $\tilde{\mathbf{T}}_1(r, s)$ in the same T-spline class such that*

$$\begin{aligned} \|\tilde{\mathbf{T}}_0 - \mathbf{T}_0\|_H &< \epsilon_0^0 \\ \|\tilde{\mathbf{T}}_1 - \mathbf{T}_1\|_H &< \epsilon_0^1 \end{aligned} \tag{6.7}$$

Here $\|\cdot\|_H$ represents the Hausdorff norm, which is irrelevant to the parameterization.

Meanwhile, the following constraints holds

$$\begin{aligned}\|\tilde{\mathbf{T}}_0(r_i, s_i) - \mathbf{x}_i\| &< \epsilon_1^0 \\ \|\tilde{\mathbf{T}}_1(r_i, s_i) - \mathbf{y}_i\| &< \epsilon_1^1\end{aligned}\tag{6.8}$$

(6.8) aims to enforce the restored surfaces to fit the specified features at the same parameters, thus fulfilling the feature alignment.

We can see that the correspondence problem is transformed to a constrained pair approximation problem. It is inferred that it's the reparameterization that causes the difficulty for the exact conversion. But for approximation, this is no longer an obstacle. In fact, it usually requires a better parameterization to obtain a finer result. As a result, parameterization becomes a standard step in the surface approximation procedure. It means through approximation we can achieve the feature alignment as well as the T-spline conversion simultaneously, which exactly solves the correspondence issue for T-spline morphing.

Tessellation

Different from the conventional surface approximation, here we aim to restore the parametric surface from an existing surface with new T-mesh structure. (6.7) serves as the objective. But computing Hausdorff distance for two parametric surfaces is an extremely challenging task. Alternatively, we first seek two conforming triangular mesh M_0 and M_1 such that

$$\begin{aligned}\|M_0 - \mathbf{T}_0\|_H &< \frac{\epsilon_0^0}{2} \\ \|M_1 - \mathbf{T}_1\|_H &< \frac{\epsilon_0^1}{2}\end{aligned}\tag{6.9}$$

This can be easily obtained through any surface tessellation techniques. In our implementation, we adopt the adaptive tessellation method [17] for its compactness as shown in Figure 6.4. Then Problem 6.1 is equivalent to

Problem 6.2. Find two T-spline surfaces $\tilde{\mathbf{T}}_0(r, s)$ and $\tilde{\mathbf{T}}_1(r, s)$ with compatible structure

Figure 6.4: Adaptive tessellation, and the user specified features are incorporated into the tessellated meshes

such that

$$\begin{aligned} \|M_0 - \tilde{\mathbf{T}}_0\|_H &< \frac{\epsilon_0^0}{2} \\ \|M_1 - \tilde{\mathbf{T}}_1\|_H &< \frac{\epsilon_0^1}{2} \end{aligned} \quad (6.10)$$

while satisfying feature correspondence

$$\begin{aligned} \|\tilde{\mathbf{T}}_0(r_i, s_i) - \mathbf{x}_i\| &< \epsilon_1^0 \\ \|\tilde{\mathbf{T}}_1(r_i, s_i) - \mathbf{y}_i\| &< \epsilon_1^1 \end{aligned} \quad (6.11)$$

which is exactly a scatter data approximation problem.

Remark 1. Different from the most previous application of surface approximation, we use two surfaces in one T-spline class to fit two triangular mesh. We refer such technique as **consistent approximation**. In fact, we can use surfaces in one T-spline class to approximate any number of surfaces. Figure 6.13 shows an example of approximating three T-spline surfaces with the compatible structure. The underlining idea is the same.

Remark 2. Here triangular mesh tessellation is applied to approximate the input surfaces. As we know, many powerful algorithms exist to restore the parametric surface from point cloud. But it is well known that the quality of the approximation results is closely related to the parameterization of the discrete points. If point cloud is tessellated, the parameterization of these points will be dependent on that of the original T-spline surface. Especially, in the

following steps, reparameterization will be required to align the features. A good initial parameterization is in high priority. On the other hand, the connectivity information in the triangular mesh enables us to produce a fine 2D embedding, which is irrelevant to the original parameterization.

Constrained map

In most approximation algorithms, although (6.10) still serves as the objective, they use an discrete form, because computing the Hausdorff norm in (6.10) is still tedious and difficult. Similarly, we employ the discrete Hausdorff norm

$$\|\tilde{\mathbf{T}}_j - M_j\|_{\tilde{H}} = \max_i \|\tilde{\mathbf{T}}_j(r_i^j, s_i^j) - V_i^j\| \quad (6.12)$$

to measure the surface difference, where $V_{i,j}$ are the mesh vertices of M_j , and (r_i^j, s_i^j) are the parameter assigned to V_i^j for $j = 0, 1$. Note that when those parameters are well chosen, (6.10) is well approximated by (6.12). As an additional merit, if we incorporate the user specified feature points into the triangular mesh during tessellation (as shown in Figure 6.4), assign them with the identical parameters (r_i, s_i) , and choose $\epsilon_1^j = \frac{\epsilon_0^j}{2}$, the second objective (6.11) is also integrated into (6.12). Then Problem 6.2 is further transformed into

Problem 6.3. Find two T-spline surfaces $\tilde{\mathbf{T}}_0(r, s)$ and $\tilde{\mathbf{T}}_1(r, s)$ with compatible structure such that

$$\begin{aligned} \|M_0 - \tilde{\mathbf{T}}_0\|_{\tilde{H}} &< \epsilon_1^0 \\ \|M_1 - \tilde{\mathbf{T}}_1\|_{\tilde{H}} &< \epsilon_1^1 \end{aligned} \quad (6.13)$$

In the first place, we need to compute two good parameterizations U_0 and U_1 for M_0 and M_1 respectively. Meanwhile, the parameters for the features $\mathbf{x}_i$ and $\mathbf{y}_i$ should be identical. These are two typical constrained mesh parameterization problems. As seen in last chapter, our proposed method can produce less distorted parameterization with hard constraints. Therefore, it can be readily applied here. The algorithm begins with computing an initial embedding through ABF++. However, due to the restriction of T-spline parametric domain,

the initial parameterization should be mapped to a rectangular domain. As mentioned above, the definition of T-splines gives an initial parameterization. But it generally is not acceptable due to its failure in reflecting the surface property. Choosing a bad initial embedding dooms to be highly distorted after warping. As a result, the quality of the final restored surfaces will be tremendously reduced. So alternatively, we adopt Floater's mean value coordinate [20] for its less distortion(as shown in Figures 6.5(a) and 6.5(b)). Then the identical parameters

(a) Initial embedding of source surface

(b) Initial embedding of target surface

(c) Deformed embedding of source surface

(d) Deformed embedding of target surface

Figure 6.5: Constrained reparameterization

for those specified features are set by

$$(r_i, s_i) = \frac{U_0^0(\mathbf{x}_i) + U_0^1(\mathbf{y}_i)}{2} \quad (6.14)$$

which serve as the target positions in the following warping. Here U_0^j refer to these two initial embeddings. Subsequently, RBF-based warping technique is applied to iteratively move the constrained points $U_0^0(\mathbf{x}_i)$ and $U_0^1(\mathbf{y}_i)$ to their corresponding target places (as shown in Figure 6.5(c) and 6.5(d)). By foldover-free checking and refinement, the final mapping is guaranteed to be foldover-free and satisfy hard constraints. Note that during warping the boundary should also be incorporated as the constrained points as well in order to maintain the rectangular domain. Figures 6.5(c) and 6.5(d) show the resultant reparameterizations after warping.

Consistent approximation

As soon as the parameters for both triangular mesh M_0 and M_1 are obtained, we are now in a position to restore the surfaces through approximation. Generally, it is difficult to determine a good compatible structure in prior. Therefore, we proceed our approximation in an iterative and adaptive manner similar as that in [131]. Our consistent approximation technique comprises 4 steps:

(1) Setup the initial T-mesh configuration

Our goal is to approximate the two surfaces with the compatible structure. That means the approximated surfaces should belong to the same T-spline class. Then this step accounts for choosing the initial T-spline class to be enlarged. We employ the simple bi-cubic B-spline surface for this purpose, whose structure is illustrated in Figure 6.6(a). And the parameter domain is set to be $[0, 1] \times [0, 1]$. As the weights of B-spline surface are 1, the subsequent refined T-spline surfaces will always be standard or semi-standard. It means the objective

Figure 6.6: Procedure of iterative consistent approximation

surfaces can be written as

$$\tilde{\mathbf{T}}_j(u, v) = \sum_{i=0}^n B_i(u, v) \omega_i \mathbf{P}_i^j \quad (6.15)$$

(2) Find the optimal surfaces in current T-spline class

This step aims to find two surfaces within current T-spline class that are closest to the triangular meshes in the sense of the discrete norms in (6.12). It means we have to minimize $\|\tilde{\mathbf{T}}_j - M_j\|_{\tilde{H}}$. Referring to the definitions in (6.12) and (6.15), it is a non-linear problem with respect to the control points. Hence in practice as in [131], the least squares norm

$$\|\tilde{\mathbf{T}}_j - M_j\|_L = \sqrt{\sum_{i=0}^{l^j} \|\tilde{\mathbf{T}}_j(r_i^j, s_i^j) - V_i^j\|^2} \quad (6.16)$$

is employed to yield the optimal surfaces. Note that those two discrete norms are actually equivalent in the sense that

$$\|\cdot\|_{\tilde{H}} \leq \|\cdot\|_L \leq \sqrt{l^j} \|\cdot\|_{\tilde{H}} \quad (6.17)$$

where l^j refers the vertex number of M_j . (6.17) implies that as soon as $\tilde{\mathbf{T}}_j$ achieve optimal least square approximation, they are also close to the optimal surfaces in the sense of discrete Hausdorff norm. It also validates the replacement of the least square norm to achieve the goal in (6.13). Moreover, in order to ensure the smoothness of the restored surfaces, usually a fairness functional is also incorporated into the objective functional. Thus the final objective functional to be minimized becomes

$$\|\tilde{\mathbf{T}}_j - M_j\|_L^2 + J_{\text{fair}}(\tilde{\mathbf{T}}_j) \quad (6.18)$$

As observed in Chapter 3, thin-plate functional (3.12) puts up a good performance in producing smooth surfaces. Hence here we also choose it as the fairness functional. Then to solve this optimization problem, we suffice to differentiate the objective functional (6.18)

with respect to the control points, and let the partial derivative be zero, which yields

$$(\tilde{\mathbf{K}}_q + \tilde{\mathbf{K}}_1^j) \tilde{\mathbf{P}}^j = \mathbf{V}^j \quad (6.19)$$

where $\tilde{\mathbf{K}}_q$ is the same matrix as defined in (3.22), and

$$\begin{aligned} \tilde{\mathbf{K}}_1^j &= \sum_{i=0}^{l^j} \mathbf{j}(r_i^j, s_i^j)^T \mathbf{j}(r_i^j, s_i^j) \\ \tilde{\mathbf{P}}^j &= [\dots, \mathbf{P}_i^j, \dots]^T \\ \mathbf{V}^j &= \sum_{i=0}^{l^j} \mathbf{j}(r_i^j, s_i^j)^T \mathbf{P}_i^{jT} \end{aligned} \quad (6.20)$$

The superscript j implies that we need to solve two independent linear systems with different coefficients. But we do not add a superscript for $\tilde{\mathbf{K}}_q$, because we enforce $\mathbf{T}_j$ to maintain in the same T-spline class. Accordingly, we only need to calculate the matrix integral once to build up the two linear systems. Let's compare the two linear systems (3.94) and (6.19). We can easily infer that they are exactly the same except that the T-spline surfaces become semi-standard here. This demonstrates that the application of PD-T-spline in surface fitting is actually consistent with traditional methods. Conversely, we also give a physical explanation for this step. Then the same preconditioned conjugate gradient method can be employed to efficiently solve the two systems in (6.19). Compared with the application in Section 3.7.2, here we aim to approximate two surfaces with a single T-spline structure.

(3) Check the quality of the resultant surfaces

After obtaining the optimal surfaces with current structure, we check whether they are satisfactory. If they are not, we identify where to perform the T-spline local refinement in the following step. To achieve the goal in (6.13), we suffice to verify whether

$$\|\tilde{\mathbf{T}}_j(r_i^j, s_i^j) - V_i^j\| < \epsilon_1^j \quad (6.21)$$

holds for all the sampled points in both surfaces. (6.21) actually places a uniform global er-

ror for our restoration. As illustrated in [131], such strategy is not so effective in preserving the features in the triangular mesh. Because for the feature area, usually a small threshold is required to restore these features, while for other areas, a moderate one is sufficient. Therefore, we employ

$$\|\tilde{\mathbf{T}}_j(r_i^j, s_i^j) - V_i^j\|_{\tilde{H}} < k_i^j \epsilon_1^j \quad (6.22)$$

where

$$k_i^j = \max \left(\frac{h_{\max}^j - h_i^j}{h_{\max}^j - h_{\min}^j}, \eta \right) \quad (6.23)$$

and $h_i^j = \log(\bar{h}_i^j + 1)$ is logarithm of the mean curvature $\bar{h}_i^j$ at the vertex V_i^j . Here η is a small value used to filter the too big curvatures. Compared with (6.21), (6.22) places more emphasis on the area with high curvature. As mean curvature is a good measurement for features, it aims to well preserve the features. Note that the criteria in (6.22) does not contradict with the goal in (6.13), because $k_i^j < 1$. As long as (6.22) holds for all the vertices, (6.13) is automatically satisfied. Different from the situation in [131], the triangular meshes M_j to be fit are tessellated from the smooth parametric surfaces $\mathbf{T}_j$. Hence we can calculate the mean curvatures in (6.23) exactly using the formula

$$\bar{h} = \frac{LG - 2MF + NE}{2(EG - F^2)} \quad (6.24)$$

Here E, F, G are the corresponding coefficients of the first fundamental form, and L, M, N are those of the second fundamental form. When all the points pass the test in (6.22), we achieve the two desired optimal surfaces, and the iteration stops. Otherwise, those failed regions are identified, which will be further processed. Specifically, we find those rectangular faces in the preimage of current T-mesh structure containing those parameters of the failed points, and regard them as the offending regions. Denote the set of those faces by F^j . The superscript indicates that we perform this operation on both surfaces separately.

(4) T-spline structure refinement

As we initiate our fitting process from a simple structure, it's likely to fail the test in (6.22)

for lack of sufficient DOFs. T-splines provide the truly local refinement property, which enables us to increase the DOFs only around the offending regions. We can adopt the similar strategy in [131] by splitting the faces in F^j to insert extra DOFs in those regions. In steps (2) and (3), the operations are performed on both surfaces respectively. But in current step we can no longer deal with them separately. Because the structural compatibility of $\tilde{\mathbf{T}}_0$ and $\tilde{\mathbf{T}}_1$ is demanded, and F^0 and F^1 are likely to be different. If the refinement is performed respectively, $\mathbf{T}_0$ and $\mathbf{T}_1$ will have different structure. In order to maintain the structural compatibility, we propose to refine the two surfaces simultaneously by splitting the faces in the union set $F^0 \cup F^1$. That is why we call this technique consistent approximation. Refinement also means the different linear systems in (6.19). As the refinement is local, to build up the systems from scratch is obviously not economic. Especially, $\tilde{\mathbf{K}}_q$ is a matrix integral, which involves intensive computation. As mentioned above, the (6.19) is exactly the same as that in (3.94). We can also employ the local update method in Section 4.5.4 to at best avoid these duplicated calculation. Meanwhile, as we have maintained the structural compatibility, this update only has to be applied once.

Result

The entire algorithm to solve the correspondence issue for T-spline morphing is outlined in Algorithm 6.1. Figure 6.6 illustrates the procedure of the iterative consistent approximation, and Figure 6.7 shows the T-mesh structure of the final restored surfaces. In this example, we set the error $\epsilon_1^j = \frac{\epsilon_0^j}{2} = 0.5\%d^j$, where d^j refers the dimension of the bounding box of the triangular mesh M_j . For the application in approximation, the physical parameters $\alpha_{i,i}$ and $\beta_{i,j}$ in (3.22) should be sufficiently small. Otherwise, the iteration will not stop. In our implementation, we set $\alpha_{i,i} = 0$, and $\beta_{1,1} = \beta_{2,2} = 0.5\beta_{1,2} = 5 \times 10^{-6}$. Table 6.1 records the number of control points and the discrete Hausdorff distance in the intermediate iteration. We can infer that the restoration precision is progressively improved along with local refinement. The restored surfaces (as shown in Figure 6.6(d)) preserve the features of the input surfaces (Figure 6.1) well. The morphing sequence between these restored

Algorithm 6.1 Establish correspondence for T-spline morphing

Given: Two T-spline surfaces $\mathbf{T}_0, \mathbf{T}_1$, corresponding feature points $\mathbf{x}_i, \mathbf{y}_i$, error tolerance ϵ_i^j , physical parameters $\alpha_{i,i}, \beta_{i,j}$.

Goal: Find two T-spline surface $\tilde{\mathbf{T}}_0, \tilde{\mathbf{T}}_1$ in the same T-spline class, such that $\tilde{\mathbf{T}}_0^{-1}(\mathbf{x}_i) = \tilde{\mathbf{T}}_1^{-1}(\mathbf{y}_i)$, and $\|\tilde{\mathbf{T}}_j - \mathbf{T}_j\|_{\tilde{H}} < \epsilon_0^j$.

- 1: Apply adaptive tessellation to obtain two triangular meshes M_0, M_1 , such that $\mathbf{x}_i \in M_0$ and $\mathbf{y}_i \in M_1$.
- 2: Compute the constrained parameterization U_0 and U_1 such that $U_0(\mathbf{x}_i) = U_1(\mathbf{y}_i)$
- 3: Compute the mean curvatures at the tessellated points using the formula in (6.24).
- 4: Initialize the initial T-mesh structure.
- 5: **loop**
- 6: Compute the optimal approximations in current T-spline class.
- 7: Check the errors using (6.22) in both surfaces.
- 8: **if** (6.22) does not hold for all the points **then**
- 9: Identify the offending regions F^j .
- 10: Refine the T-mesh at the union regions $F^0 \cup F^1$.
- 11: Locally update the coefficient matrix in (6.19) using the method in Section 3.6.4.
- 12: **else**
- 13: Exit from the loop.
- 14: **end if**
- 15: **end loop**
- 16: **return** Two approximated T-spline surfaces $\tilde{\mathbf{T}}_0, \tilde{\mathbf{T}}_1$

surfaces $\tilde{\mathbf{T}}_0$ and $\tilde{\mathbf{T}}_1$ is illustrated in Figure 6.8. As the features are aligned, important features are also well preserved during transition, which is obviously advantageous over that in Figure 6.2.

Table 6.1: The statistics for the T-spline surfaces in Figure 6.6

	Initial	Inter 1	Inter 2	Final
#control points	36	114	1151	1520
$\max_j(\ \tilde{\mathbf{T}}_j - M_j\ _{\tilde{H}})$	6.51%	1.50%	0.51%	0.27%

6.3 Vertex path

As soon as the correspondence is established with compatible structure, we can proceed to compute the transitive surface $\mathbf{T}(u, v, t)$. It can also be formulated as finding a transitive

Figure 6.7: Consistent approximation result applied to the surfaces in Figure 6.1

T-spline surface $\mathbf{T}(u, v, t)$ such that

$$\begin{cases} \mathbf{T}(u, v, 0) = \tilde{\mathbf{T}}_0(u, v) \\ \mathbf{T}(u, v, 1) = \tilde{\mathbf{T}}_1(u, v) \end{cases} \quad (6.25)$$

6.3.1 Linear Averaging

A brute-force approach to achieve this is to linearly average the two surfaces. As the metamorphosed surfaces are made compatible, such linear averaging can be expressed as

$$\mathbf{T}(u, v, t) = \sum_i^m B_i(u, v) ((1-t)\omega_{i,0}\mathbf{P}_i^0 + t\omega_{i,1}\mathbf{P}_i^1) \quad (6.26)$$

As shown in Figure 6.8, when the input surfaces exhibit similar feature distribution, such approach can produce satisfactory transition. However, as the shape difference gets larger, linear interpolation tends to produce unsatisfactory results. As shown in Figure 6.9, the input surfaces are two wavy surfaces. Through linear averaging, the transitive shape generates unwanted flatten and wiggles. The reason for this may be that linear interpolation merely take the absolute Euclidean position of the surface into consideration while ignoring the local neighborhood information.

Figure 6.8: Morphing with feature alignment. The transition is generated through linear interpolation of the surfaces in Figure 6.7.

Figure 6.9: Problem of linear averaging. The unwanted flatening is observed in this case

6.3.2 Physically based morphing

On the other hand, as we see in the previous chapters, physically based dynamic T-splines naturally produces smooth evolution from the initial state to the final equilibrium governed by the physical law. Due to the physical nature, the transition is physically realistic and visually pleasing, which is exactly the objective of a morphing process. Hence, incorporating PD-T-splines into the T-spline morphing is a promising approach to generate an ideal transition. Referring to the application in modeling process, as the dynamic evolution plays an vital role in providing the candidates final shape for decision. Therefore, the initial speed is set to zero in order to produce a gradual deformation. But in the morphing process, if the object is regarded as PD-T-splines, it will be forced to evolve into the shape of $\tilde{\mathbf{T}}_1(u, v)$ at a specific time T . Without the loss of generality, T will be set to 1 in the following. Then together with the PD-T-splines formulation, the morphing problem can be expressed as

$$\begin{cases} \tilde{\mathbf{M}}_q \ddot{\mathbf{Q}} + (\alpha \tilde{\mathbf{M}}_q + \beta \tilde{\mathbf{K}}_q) \dot{\mathbf{Q}} + \tilde{\mathbf{K}}_q \mathbf{Q} = \mathbf{f}_q \\ \mathbf{Q}(0) = \tilde{\mathbf{P}}^0 \\ \mathbf{Q}(1) = \tilde{\mathbf{P}}^1 \end{cases} \quad (6.27)$$

Here Rayleigh damping (3.69) is adopted for its simplicity, and $\tilde{\mathbf{P}}^j$ refer to the collection of the control points defined as (6.20).

It is inferred from the definition of thin-plate energy (3.12) that the potential energy functional is actually a weighted combination of first order and second order differential quantities. Hence the evolution of the physically-based model makes full use of instant geometric information, which is exactly the point that linear interpolation (6.26) lacks. Therefore, physically-based morphing is a promising approach to overcoming the drawbacks of linear interpolation. Here, there are two items in (6.27) that need to be highlighted, whose choices have significant impact on the morphing process. The first is the force term $\mathbf{f}_q$. In modeling process, force serves as the main sculpting tools for interactively adjusting the shape during evolution. But in the morphing process, usually the external influence should

be minimized, therefore we naturally set it to be zero. The other item is the rest pose $\mathbf{Q}_0$. Different rest poses mean different system energies. It is not difficult to imagine that we can achieve different transitions by setting different rest poses. In other words, it also provides us with extra control to obtain different morphing effects as we will see in the following.

As the external force is ignored, the initial speed will be the only impetus that drives the PD-T-spline surface to evolve into $\tilde{\mathbf{T}}_1$ at time 1. Especially, when the rest pose is set close to the initial surface $\tilde{\mathbf{T}}_0$, the initial speed will be required extremely large. In this case, it tends to produce strong vibrations early in the interpolation, which may even destroy the surface geometry. On the other hand, if the rest pose is set as the final state, we find an interesting effect. As shown in Figure 6.10, the features in the final state quickly appear on the initial face. This can be very useful in the modeling application based on morphing. Such an application is quite similar to the detail transfer in polygonal meshes. The reason for such a phenomenon can be expressed as follows. As described in Chapter 3, a vibration system can be viewed as a coupling of several individual vibration modes. The feature area corresponds to the modes of high energy. Through modal analysis, we can decouple these modes obtaining,

$$\ddot{x}_i + (\alpha + \beta\lambda_i)\dot{x}_i + \lambda_i(x_i - x_{i,1}) = 0 \quad (6.28)$$

Note that the force term is ignored here, and x_i represent the modal coordinates, and λ_i are the corresponding stiffness. Let's focus on these modes of high energy, which means large λ_i . Then the boundary conditions

$$\begin{aligned} x_i(0) &= x_{i,0} \\ x_i(1) &= x_{i,1} \end{aligned} \quad (6.29)$$

imply the solution

$$x_i(t) = c_{i,0} \exp^{\phi_{i,0}t} + c_{i,1} \exp^{\phi_{i,1}t} + x_{i,1} \quad (6.30)$$

Figure 6.10: Physically based morphing without external force. The source and target surfaces are given in Figure 6.12, and the timestep is set to 0.1. During transition, the feature of the target surface quickly appear in the early stage.

where $\phi_{i,0}, \phi_{i,1}$ are two solutions to the quadratic equation

$$x^2 + (\alpha + \beta\lambda_i)x + \lambda_i x = 0 \quad (6.31)$$

which can be expressed as

$$\begin{aligned} \phi_{i,0} &= \frac{-(\alpha + \beta\lambda_i) - \sqrt{(\alpha + \beta\lambda_i)^2 - 4\lambda_i}}{2} \\ \phi_{i,1} &= \frac{-2}{(\frac{\alpha}{\lambda_i} + \beta) + \sqrt{(\frac{\alpha}{\lambda_i} + \beta)^2 - \frac{4}{\lambda_i}}} \end{aligned} \quad (6.32)$$

As λ_i is large, we have

$$\begin{aligned} \phi_{i,0} &\ll -100 \\ \phi_{i,0} &\sim -\frac{1}{\beta} \end{aligned} \quad (6.33)$$

Then coefficient can be determined through

$$\begin{aligned} c_{i,0} &= \frac{(x_{i,0} - x_{i,1}) \exp^{\phi_{i,1}}}{\exp^{\phi_{i,1}} - \exp^{\phi_{i,0}}} \\ c_{i,1} &= \frac{(x_{i,0} - x_{i,1}) \exp^{\phi_{i,0}}}{\exp^{\phi_{i,0}} - \exp^{\phi_{i,1}}} \end{aligned} \quad (6.34)$$

Making use of the estimation in (6.33), these coefficients have the following estimation

$$\begin{aligned} c_{i,0} &\sim x_{i,0} - x_{i,1} \\ c_{i,1} &\sim 0 \end{aligned} \quad (6.35)$$

Then lets estimate the position at the time instant 0.1. Substituting $t = 0.1$ into (6.30), we have

$$x_i(0.1) = c_{i,0} \exp^{0.1\phi_{i,0}} + c_{i,1} \exp^{0.1\phi_{i,1}} + x_{i,1} \quad (6.36)$$

Also using the estimation in (6.33), we have

$$\begin{aligned} \exp^{0.1\phi_{i,0}} &\sim 0 \\ \exp^{0.1\phi_{i,1}} &< 1 \end{aligned} \quad (6.37)$$

Finally, we obtain

$$x_i(0.1) \sim x_{i,1} \quad (6.38)$$

which means that the mode will quickly get close to the rest pose in the early interpolation. That explains why the features of the target object quickly appear on the source object.

6.3.3 Extra force control

Although by setting the rest pose as the target shape, the transition is smooth and can produce interesting transition, it is not desirable for the application of animation, because the features appear too instant. In the application of animation, usually an gradual and even transition is required. The reason for this instant feature transition still lies in the too large initial speed. So to remedy this, we ought to control the initial speed. In fact, speed control is crucial in animation involving many key-frames. Morphing techniques are usually adopted to generate animation between two consecutive key frames. In order to ensure the overall smoothness, there is a need to maintain the speed continuity between the consecutive morphing process. That is the initial speed should be equal to the final speed of the previous morphing.

When extra constraints are added into the PD-T-splines system, such as

$$\dot{\mathbf{Q}}(0) = \mathbf{0} \quad (6.39)$$

(6.27) become an over-determined system, which is likely to have no solution. In [33], they proposed to perturb the frequency and dissipation to minimize the initial speed. Such approach will definitely weaken the physical meaning of this method. Meanwhile, it only reduces the initial speed, but cannot constrain it to a specific state. Therefore, we have to seek other approach to accomplish this goal.

By investigating the system (6.27) together with the extra constraints in (6.39), we can see the reason for the nonexistence of the solution is due to the lack of DOF. As we see in

the modeling application of PD-T-splines, force is the main impetus to deform the surface. It means force can provides abundant DOFs. So by reincorporating the force into our morphing process, we can solve the shortage of DOF. Meanwhile, we still follow the criterion that the extra influence should be minimized. That is

$$\iiint \mathbf{f}(u, v, t)^T \mathbf{f}(u, v, t) du dv dt \quad (6.40)$$

is minimized, where $\mathbf{f}$ is the unknown function. On the other hand, the three constraints in (6.27) and (6.39) should be satisfied. This turns out to be an optimization problem within an infinite dimensional space, which is a very challenging task.

Instead, we narrow the external force space and search for the solution within a finite dimensional space. Since we are working on parametric surfaces, we choose to represent the force in terms of T-spline volume. Let

$$\mathbf{f}(u, v, t) = \sum_{i=0}^n \sum_{j=0}^m \omega_i B_i(u, v) B_{j,k}(t) \mathbf{f}_{i,j} \quad (6.41)$$

where $\mathbf{f}_{i,j}$ are the coefficients, $B_i(u, v)$ are T-spline blending functions defined in (2.7), and $B_{j,k}(t)$ are B-spline bases defined over knot sequence $\{-k/(m+1-k), -(k-1)/(m+1-k), \dots, 0, 1/(m+1-k), \dots, 1, (m+2-k)/(m+1-k), \dots, (m+1)/(m+1-k)\}$. Here we also add a weight ω_i in order to maintain the consistency with the PD-T-splines. As we see later, such treatment can reduce computational cost. Then $\mathbf{f}_{i,j}$ serve as the extra DOFs for (6.27). Apparently the dimension of this force space is finite.

We now examine the solution to the physically-based model with the above external force. Through Modal transformation, the equation in (6.27) is transformed into

$$\ddot{\mathbf{q}} + (\alpha \mathbf{I} + \beta \boldsymbol{\Omega}) \dot{\mathbf{q}} + \boldsymbol{\Omega}(\mathbf{q} - \mathbf{Q}_0) = \mathbf{U}^T \mathbf{f}_q \quad (6.42)$$

where $\mathbf{U}$ and $\mathbf{\Omega}$ are the two matrices resulted from modal analysis defined in (3.65), and

$$\mathbf{q} = \mathbf{U}^{-1}\mathbf{Q} \quad (6.43)$$

the generalized force term can be expressed as

$$\mathbf{U}^T \mathbf{f}_q = \mathbf{B}\mathbf{N}(t)\bar{\mathbf{F}} \quad (6.44)$$

where

$$\begin{aligned} \mathbf{B} &= \mathbf{U}^T \iint \mathbf{j}^T \mathbf{j} du dv, \\ \mathbf{N}(t) &= \begin{pmatrix} I(t) & 0 & \cdots & 0 \\ 0 & I(t) & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & I(t) \end{pmatrix}, \\ \bar{\mathbf{F}} &= \begin{pmatrix} \mathbf{f}_{0,0}^T \\ \mathbf{f}_{0,1}^T \\ \vdots \\ \mathbf{f}_{n,m}^T \end{pmatrix}, \end{aligned}$$

$\mathbf{j}$ refers to the Jacobian matrix of E-PD-T-splines defined as (3.26), and $I(t)$ is a functional vector:

$$I(t) = \begin{bmatrix} B_{0,k}(t) & \cdots & B_{m,k}(t) \end{bmatrix}$$

Then the entry of the generalized force, $\mathbf{f}_{i,j}$ in (6.44), can be expressed as

$$f_{i,j} = [\dots, a_l^i(t), \dots] \bar{\mathbf{F}}[:, j] \quad (6.45)$$

$$a_l^i(t) = b_{i,g(l)} B_{g(l),k}(t) \quad (6.46)$$

where $b_{i,j}$ are the entries of $\mathbf{B}$, $B_{j,k}(t)$ are the same as in (6.41), $\bar{\mathbf{F}}[:,j]$ refers to the j th column of matrix $\bar{\mathbf{F}}$ defined as in Section 4.2.1 and the index numbers $g(l)$ are

$$g(l) = l \% (m + 1) \quad (6.47)$$

(6.42) involve three independent ODEs, which can be expressed as

$$\ddot{\mathbf{q}}[:,j] + (\alpha \mathbf{I} + \beta \mathbf{\Omega}) \dot{\mathbf{q}}[:,j] + \mathbf{\Omega}(\mathbf{q} - \mathbf{Q}_0)[:,j] = \mathbf{U}^T \mathbf{f}_q[:,j], j = 1, 2, 3 \quad (6.48)$$

Note that the coefficient in (6.48) has already been diagonalized through Modal analysis.

Denote the roots of the quadratic equation

$$x^2 + (\alpha + \beta \lambda_i)x + \lambda_i x = 0 \quad (6.49)$$

as $x_{i,1}, x_{i,2}$, the general solution to (6.48) can be expressed as

- If $x_{i,1} \neq x_{i,2}$ are real roots,

$$\begin{aligned} \mathbf{q}[i,j] = & c_{i,1}e^{x_{i,1}t} + c_{i,2}e^{x_{i,2}t} + \frac{1}{x_{i,1} - x_{i,2}} \times \left\{ \left(\int e^{-x_{i,1}s} ds - e^{x_{i,2}t} \int e^{-x_{i,2}s} ds \right) \lambda_i \mathbf{Q}_0[i,j] \right. \\ & \left. + \left[\dots, b_{i,g(l)} \left(e^{x_{i,1}t} \int e^{-x_{i,1}s} B_{g(l),k}(s) ds - e^{x_{i,2}t} \int e^{-x_{i,2}s} B_{g(l),k}(s) ds \right), \dots \right] \bar{\mathbf{F}}[:,j] \right\} \end{aligned} \quad (6.50)$$

- If $x_{i,1} = x_{i,2} = r_i$,

$$\begin{aligned} \mathbf{q}[i,j] = & (c_{i,1}^j + c_{i,2}^j t) e^{r_i t} + e^{r_i t} \times \left\{ t \left(\int e^{-r_i s} ds - \int s e^{-r_i s} ds \right) \lambda_i \mathbf{Q}_0[i,j] \right. \\ & \left. + \left[\dots, b_{i,g(l)} \left(t \int e^{-r_i s} B_{g(l),k}(s) ds - \int s e^{-r_i s} B_{g(l),k}(s) ds \right), \dots \right] \bar{\mathbf{F}}[:,j] \right\} \end{aligned} \quad (6.51)$$

- If $x_{i,1} = a_i + b_i i$ and $x_{i,2} = a_i - b_i i$ are two conjugate complex roots,

$$\begin{aligned} \mathbf{q}[i, j] = & e^{a_i t} \left(c_{i,1}^j \cos b_i t + c_{i,2}^j \sin b_i t \right) + \frac{e^{a_i t}}{b_i} \times \\ & \left\{ \left(\sin b_i t \int \cos b_i s ds - \cos b_i t \int e^{-a_i s} \sin b_i s ds \right) \lambda_i \mathbf{Q}_0[i, j] + \right. \\ & \left. \left[\dots, b_{i,g(l)} \left(\sin b_i t \int \cos b_i s B_{g(l),k}(s) ds - \cos b_i t \int e^{-a_i s} \sin b_i s B_{g(l),k}(s) ds \right), \dots \right] \bar{\mathbf{F}}[:, j] \right\} \end{aligned} \quad (6.52)$$

We can see that the unknowns $c_{i,1}^j, c_{i,2}^j$ and $\bar{\mathbf{F}}$ appear in the linear form, which will bring a lot of convenience in the following computation.

In the above solution, some terms are actually repeated frequently. So we can rearrange the solution to speed up the calculation. The generalized force term in (6.44) can also be expressed as

$$\mathbf{U}^T \mathbf{f}_q[:, j] = \mathbf{B} \mathbf{F}_j' \mathbf{N}'(t)$$

where $\mathbf{B}$ is the same as above and

$$\begin{aligned} \mathbf{F}_j' &= \begin{pmatrix} \vdots \\ \mathbf{f}_{i,0}[j] \quad \mathbf{f}_{i,1}[j] \quad \dots \quad \mathbf{f}_{i,m}[j] \\ \vdots \end{pmatrix}, \\ \mathbf{N}'(t) &= \begin{pmatrix} B_{0,k}(t) \\ B_{1,k}(t) \\ \vdots \\ B_{m,k}(t) \end{pmatrix}. \end{aligned}$$

Then the entry of the generalized force becomes

$$f_{i,j} = (\mathbf{B} \mathbf{F}_j')[i, :] \mathbf{N}'(t). \quad (6.53)$$

Making use of this form, the solution can also be expressed as

- If $x_{i,1} \neq x_{i,2}$ are real roots,

$$\begin{aligned} \mathbf{q}[i, j] &= c_{i,1}^j e^{x_1 t} + c_{i,2}^j e^{x_2 t} + \frac{1}{x_{i,1} - x_{i,2}} \times (\mathbf{BF}'_j)[i, :] \mathbf{N}'_r(t) \\ &+ \frac{1}{x_1 - x_2} \times \left(\int e^{-x_1 s} ds - e^{x_2 t} \int e^{-x_2 s} ds \right) \lambda_i \mathbf{Q}_0[i, j] \end{aligned} \quad (6.54)$$

- If $x_{i,1} = x_{i,2} = r_i$,

$$\begin{aligned} \mathbf{q}[i, j] &= (c_{i,1}^j + c_{i,2}^j t) e^{r_i t} + e^{r_i t} \times (\mathbf{BF}'_j)[:, i] \mathbf{N}'_e(t) \\ &+ e^{r_i t} \times \left(t \int e^{-r_i s} ds - \int s e^{-r_i s} ds \right) \lambda_i \mathbf{Q}_0[i, j] \end{aligned} \quad (6.55)$$

- If $x_{i,1} = a_i + b_i i$ and $x_{i,2} = a_i - b_i i$ are two conjugate complex roots,

$$\begin{aligned} \mathbf{q}[i, j] &= e^{a_i t} (c_{i,1}^j \cos b_i t + c_{i,2}^j \sin b_i t) + \frac{e^{a_i t}}{b_i} \times (\mathbf{BF}'_j)[i, :] \mathbf{N}'_c(t) \\ &+ \frac{e^{a_i t}}{b_i} \left(\sin b_i t \int \cos b_i s ds - \cos b_i t \int e^{-a_i s} \sin b_i s ds \right) \lambda_i \mathbf{Q}_0[i, j] \end{aligned} \quad (6.56)$$

where

$$\begin{aligned} \mathbf{N}'_r(t) &= \begin{pmatrix} \vdots \\ e^{x_1 t} \int e^{-x_1 s} B_{j,k}(s) ds - e^{x_2 t} \int e^{-x_2 s} B_{j,k}(s) ds \\ \vdots \end{pmatrix} \\ \mathbf{N}'_e(t) &= \begin{pmatrix} \vdots \\ t \int e^{-r s} B_{j,k}(s) ds - \int s e^{-r s} B_{j,k}(s) ds \\ \vdots \end{pmatrix} \\ \mathbf{N}'_c(t) &= \begin{pmatrix} \vdots \\ \sin b t \int \cos b s B_{j,k}(s) ds - \cos b t \int e^{-a s} \sin b s B_{j,k}(s) ds \\ \vdots \end{pmatrix} \end{aligned}$$

Compared with (6.50), (6.51) and (6.52), the new expression only involves $m + 1$ integral evaluations. Once the instant modal coordinates $\mathbf{q}$ are found, the transition shape is

generated through the inverse transformation of (6.43).

In order to generate the transitive surface, we ought to determine those unknowns first. The constraints in (6.27) and (6.39) imply that those unknowns $c_{i,1}^j, c_{i,2}^j$ and $\bar{\mathbf{F}}$ should satisfy

$$\bar{\mathbf{A}}\bar{\mathbf{X}} + \bar{\mathbf{B}}\bar{\mathbf{F}} = \mathbf{Y}_0 \quad (6.57)$$

where

$$\begin{aligned} \bar{\mathbf{X}}[2i, j] &= c_{i,1}^j \\ \bar{\mathbf{X}}[2i+1, j] &= c_{i,2}^j \end{aligned} \quad (6.58)$$

and $\mathbf{Y}_0$ is a $(n+1) \times 3$ matrix. Note that the number of equations in (6.57) is $9(n+1)$, while the total number of unknowns is $6(n+1) + 3(m+1)(n+1)$. Therefore the linear system is under-determined. For morphing, we choose the optimal solution by solving the following three independent minimization problem:

$$\begin{aligned} \text{Minimize} \quad & \iiint \mathbf{f}(u, v, t)^T \mathbf{f}(u, v, t) du dv dt = \sum_{j=0}^2 \bar{\mathbf{F}}[:, j]^T \tilde{\mathbf{H}} \bar{\mathbf{F}}[:, j] \\ \text{subject to} \quad & \bar{\mathbf{A}}\bar{\mathbf{X}} + \bar{\mathbf{B}}\bar{\mathbf{F}} = \mathbf{Y}_0 \end{aligned} \quad (6.59)$$

where

$$\begin{aligned} \tilde{\mathbf{H}} &= \int \int \tilde{\mathbf{J}}^T \tilde{\mathbf{J}} du dt, \\ \tilde{\mathbf{J}} &= \begin{bmatrix} \tilde{N}_{0,0}(u, t) & \tilde{N}_{0,1}(u, t) & \cdots & \tilde{N}_{n,m}(u, t) \end{bmatrix}, \end{aligned}$$

and $\tilde{N}_{i,j}(u, t) = \omega_i B_i(u, v) B_{j,k}(t)$. This is a typical constrained optimization problem, which can be solved using the Lagrangian multiplier method. Specifically, (6.59) can be transformed into an equivalent linear system

$$\begin{pmatrix} \tilde{\mathbf{H}} & \mathbf{0} & \bar{\mathbf{B}} \\ \mathbf{0} & \mathbf{0} & \bar{\mathbf{A}} \\ \bar{\mathbf{B}} & \bar{\mathbf{A}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \bar{\mathbf{F}} \\ \bar{\mathbf{X}} \\ \boldsymbol{\Lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{Y}_0 \end{pmatrix} \quad (6.60)$$

The coefficient matrix is symmetric and sparse. The linear system can be solved efficiently.

6.4 Experimental Results

In this section, we will show some morphing results using our consistent approximation technique and physically based approach.

6.4.1 More consistent approximation results

Figure 6.11: Another input of T-spline surface

Figure 6.12: Consistent approximation result applied to the surfaces in Figure 6.1(a) and 6.11.

Figure 6.12 and 6.13 show another two results of our consistent approximation method. Therein, Figure 6.13 illustrates an example of approximating more than two surfaces using

Figure 6.13: Consistent approximation result of restoring three T-spline surfaces in Figure 6.1 and 6.11.

Table 6.2: The statistics for the input T-spline surfaces in Figures 6.1 and 6.11

	Fig. 6.1(a)	Fig. 6.1(b)	Fig. 6.11
#control points	1597	1238	1712

Table 6.3: The statistics for the consistent approximation results

	Fig. 6.7	Fig. 6.12	Fig. 6.13
#control points	1520	1714	1585
$\max_j (\ \mathbf{T}_j - M_j\ _{\tilde{H}})$	0.27%	0.28%	0.35%

one T-spline class, which is quite useful in parametric surface matching. Table 6.3 shows the statistics of the results including the number of control points and approximation precisions. The input errors ϵ_i^j , physical parameters $\alpha_{i,i}$ and $\beta_{i,j}$ are all set the same as the example in Section 6.2.2. It is obvious that the resultant number of control points is closely related to ϵ_i^j . According to our experiments, 0.5% of the scale of the input surfaces suffices to ensure acceptable restoration. The number of control points is also comparable with that of the input surfaces (as listed in Table 6.2). Here we can explain a little bit about this phenomenon. According to the nature of T-spline surface, dense control points are distributed around the region of features. As the main features coincide in the parametric domain, the overall feature of the surface will distribute similarly throughout the parametric domain. So the restoration will not produce excessive number of control points.

6.4.2 Eliminating small wiggles

Figure 6.14 demonstrates the result of our physically based morphing in metamorphosing the same objects as in Figure 6.9. Due to ignoring the local geometric information, linear interpolation tends to produce unwanted small wiggles as in Figure 6.9(d). On the other hand, with the imposition of elasticity property, the deformation of the surface will be influenced by local change. Hence physically based morphing tends to remedy the problem of linear interpolation. As shown in 6.14(d), the unwanted small wiggles are successfully eliminated, and the wavy shape of the input objects is maintained throughout the transition. In this case, we set the rest pose $\mathbf{Q}_0 = \mathbf{0}$, and also introduce the extra constraint in (6.39).

Figure 6.14: Physically based morphing keeps the wavy shapes, and small wiggles are eliminated.

6.4.3 Speed control

Figure 6.15: Physically based morphing with force control. The initial speed is set to be 0, and the rest pose is the target state.

Figure 6.15 illustrates the effect of imposing speed control. The input surfaces are the same as that in Figure 6.10, which involve 1712 control points. In Figure 6.10, due to the large initial speed, the feature of the target surface quickly appears in the early transition. This is generally not so desirable in the application of animation, where a gradual transition is preferable. Hence, we need slow down the deformation in the early stage. This can be achieved through imposing the extra constraint in (6.39). As seen in Figure 6.15, the deformation becomes much more even throughout the transition. Note that, the constraint in (6.39) is actually crucial in physically based morphing. In last example, if we discard such

a constraint, the over vibration in early stage will destroy the geometry. This phenomenon is conformed with that in [33], which also strived to reduce the initial speed.

6.4.4 Morphing based modeling

Figure 6.16: Morphing based modeling

Although the instant appearance of the target feature is not preferable in the application of animation, as we see in Figure 6.10, it indeed produces some interesting results during transition, which is useful for the modeling purpose. Figure 6.16 shows two new surfaces modeled from the inputs in Figure 6.1(a) and 6.11. Both of them are obtained at $t = 0.001$ in the physically based morphing sequence, where external force is ignored and the rest pose is set to the target object. In case of Figure 6.16(b), we set the elastic parameters to be zero in the right half of the surface. Then in Figure 6.16(a), the features of the target appear in the face of the source surface, i.e. the hair, eye, and nose. In contrast, the right part of the surface in Figure 6.16(b) maintains the appearance of the input face.

6.4.5 Different rest poses

Different rest poses mean different system energies as well as different transitions. The options of the rest pose provide us more flexibility to control the morphing process. Note that rest pose is the state with least deformation energy. The evolution of PD-T-splines is

Figure 6.17: Physically based morphing with rest post control.

actually a process of minimizing the deformation energy. By setting the desirable shape as the rest pose, it will appear during the transition. As shown in Figure 6.17, the shape of the rest pose appears in the transition, and finally evolves into the target surface through the impetus of the external force. It's worth pointing out that this is not a key-framing technique, because the rest pose is not exactly interpolated during the transition. For the acquisition of the desirable rest pose, we can employ the proposed modeling techniques in Chapter 3 to interactively deform the existing source or target surface. We can also use arbitrary T-spline surface, and employ the pair approximation technique described in Section 6.2.2 to guarantee the structural compatibility with the input surfaces. The rest pose in Figure 6.17 is attained using the consistent approximation as in Figure 6.13.

6.4.6 Insert key frames

Morphing is usually employed to generate transition between two consecutive key frames. There is a need to enforce the speed continuity between the consecutive morphing sequence to ensure the smoothness of the entire procedure. That is

$$\dot{\mathbf{Q}}^{i+1}(0) = \dot{\mathbf{Q}}^i(1) \quad (6.61)$$

where $\mathbf{Q}^i$ represents the generalized coordinates in the i th morphing process. This constraints can be easily imposed in our physically based morphing approach. We simply need to incorporate such constraints into the linear equation in (6.57) when computing the additional force in the following sequence. Or we can enforce the final speed to be $\mathbf{0}$ in every morphing process. However, either approach requires to solve the minimization problem in every morphing sequence, which is quite time-consuming. In our physically based morphing approach, we can introduce arbitrary extra DOFs. This enables us to combine all the morphing sequences into a single morphing problem.

Denote the key-frames by $\left\{ \tilde{\mathbf{T}}^i(u, v) \right\}_{i=1}^{l-1}$. The whole morphing procedure can be for-

Figure 6.18: Interpolate key frames using physically based morphing. The rest pose is set to the intermediate key frame.

(a) Source surface

(b) Key frame

(c) Target surface

(d) Morphing, time step=0.1

Figure 6.19: Interpolate key frames using physically based morphing. The rest pose is set to the target surface.

mulated as a problem of finding a transitive $\mathbf{T}(u, v, t)$ such that

$$\mathbf{T}(u, v, \frac{i}{l}) = \tilde{\mathbf{T}}^i(u, v) \quad (6.62)$$

In our physically based model, it means the equation in (6.27) should satisfy

$$\mathbf{Q}(\frac{i}{l}) = \mathbf{Q}_0^i \quad (6.63)$$

where $\mathbf{Q}_0^i$ refer to the assemblage of the control points in $\tilde{\mathbf{T}}^i(u, v)$. This problem can be readily solved through incorporating these constraints into (6.57). Note that the extra DOFs in (6.41) should be larger than the number of constraints in (6.63). Otherwise, the system will become over-determined. So it is suggested that when choosing the number of extra DOFs, we should set $m > l - 2$. Then by solving the minimization problem in (6.59), we can achieve the morphing sequences that interpolate all the input key frames. Moreover, the entire sequences are always C^2 continuous, which guarantees a smooth process. Figure 6.18 and 6.19 show the morphing sequence interpolating one extra key frame with different rest poses. In Figure 6.18, the intermediate key frame is set as the rest pose, while Figure 6.19 chooses the target surface for this purpose. We can see that either produces interesting results. So the user is still allowed to achieve abundant effect through adjusting the rest poses. Compared with Figure 6.17, Figure 6.18 achieves better results, and it exactly interpolates the intermediate shape. As more constraints are introduced into (6.59), the dimension of resultant equation (6.60) is also larger. But since the entire morphing process only involves one time solving, it is still acceptable.

Chapter 7

Dynamic T-spline Rendering

7.1 Introduction

Physically based dynamic T-splines can produce a physically-plausible dynamic evolution following Lagrangian dynamics. We have also exploited its application in animation through morphing. In practice of animation, these generated frames need to be further polished to enhance the visual appearance. For instance, when the morphing objects are faces, we need to assign the color for the skin, hair and eyes. A typical way to achieve this is to employ texture mapping. In chapter 5, we have investigated the problem of constrained reparamterization, and witnessed its ability in improving the visual appearance of the surface geometry. While the previous application focuses on wrapping one image onto only one geometry, here we want to paste all these frames with a single figure. Especially, we seek to implement such texture mapping along with the dynamic evolution in real-time. We refer this technique as dynamic rendering.

7.2 Problem statement

As shown in Figure 7.1, the inputs include a morphing sequence $\mathbf{T}_t(u, v)$ and a texture source image I_s . Like in chapter 5, in order to enhance the visual appearance, two sets of

(a) Morphing sequence

(b) Feature correspondence for initial frame

Figure 7.1: Input of dynamic rendering

user specified corresponding feature pairs $\{\mathbf{x}_i^t, \mathbf{v}_i\}$ is required, where $\mathbf{x}_i^t$ are the important feature points on the intermediate frame $\mathbf{T}_t(u, v)$, and $\mathbf{v}_i$ are the corresponding features in the texture figure I_s . T-spline rendering is achieved through tessellation. So first, we apply adaptive tessellation technique as in [17] to obtain the sequence approximation $\mathbf{M}_t = \{\mathcal{V}^t, \mathcal{K}^t\}$, where $\mathcal{V}^t$ refers to the vertex set and $\mathcal{K}^t$ represents the triangle list which encodes the connectivity. Note that adaptive tessellation yields different connectivity for different time instant t . We adopt the similar strategy in last chapter where the features on original T-spline surface appear in the final approximated meshes, which means $\mathbf{x}_i^t \in \mathcal{V}^t$. Then the problem of dynamic rendering can be characterized as

Problem 7.1. *Compute the time dependent parameterization U_t for $\mathbf{M}_t$ such that*

$$U_t(\mathbf{x}_i^t) = \mathbf{v}_i \quad (7.1)$$

which appears as a serial of constrained parameterization problem.

It seems that we are inevitable to deal with each frame off-line, because the feature correspondence is specified manually by users as described in chapter 6. We have to pause the dynamic evolution, and choose the feature points for the rendering. This is apparently awkward and deviates from our objective which is to achieve better visual appearance in realtime.

7.3 Our solution

Recalling that in the stage of vertex correspondence in T-spline morphing, we let the user specify the important features and provide the correspondence. As the morphing objects possess similar features, these obvious features are preserved during transition. This observation suggests us a way to avoid the tedious feature specification during transition. We let the features that need to be aligned coincide with those in the morphing process, and specify the feature points in the initial frame. Then feature points in the following sequence are considered as

$$\mathbf{x}_i^t = \mathbf{T}_t(\mathbf{u}_i^0) \quad (7.2)$$

where

$$\mathbf{u}_i^0 = \mathbf{T}_0^{-1}(\mathbf{x}_i^0) \quad (7.3)$$

Because in the morphing process, those features are aligned beforehand, such assumption is reasonable. With such strategy, we easily circumvent the tedious feature specification throughout the sequence.

Once feature correspondence is specified, the remaining issue is to compute the constrained map for each frame. Obviously, a straightforward approach is to apply the method in Chapter 5 to each frame. However, as shown in Table 5.1, it may takes several seconds to obtain a desirable constrained mapping in some extreme cases. Then such an approach is not suitable for a real-time rendering. Note that in static texture mapping, the foldover-free property is crucial, because details could be magnified. But in the application of dynamic

rendering, the objective is different. We aim to improve the visual appearance of dynamic evolution. So speed is the first priority. It should be efficient enough to achieve the constrained texture mapping in the rendering stage of dynamic evolution. Meanwhile, satisfying the specified feature correspondence is also important. Because failure in aligning the features will be clearly reflected in the rendering. Bijectivity is left as the third consideration. As users will place more attentions on the shape change rather the texture, the foldover-free requirement can be relaxed to some extent. It means that we can trade the bijectivity for the speed and hard constraints, though we still try to keep the final map as bijective as possible.

7.3.1 Initial constrained mapping

We still follow the two-phase approach as in chapter 5. That begins with an initial unconstrained parameterization, and is followed by a 2D mesh transformation for each frame. In the process of correspondence establishment in T-spline morphing, we have already computed a relatively less distorted initial parameterization. To ensure the speed in priority, we simply choose the parameterization in the T-spline definition as the initial embedding, which can be expressed as

$$\mathbf{m}_t = \{\mathcal{A}^t, \mathcal{K}^t\} \quad (7.4)$$

where

$$\mathcal{A}^t = \mathbf{T}_t^{-1}(\mathcal{V}^t) \quad (7.5)$$

and $\mathcal{V}^t$ refers to the vertex set in $\mathbf{M}_t$. Then we suffice to focus on finding the desirable 2D mesh transformation.

We start with computing the constrained 2D transformation for the initial frame, obtaining

$$U_0 : [0, 1] \times [0, 1] \hookrightarrow \mathbb{R}^2 \quad (7.6)$$

which satisfies

$$U_0(\mathbf{u}_i^0) = \mathbf{v}_i \quad (7.7)$$

Note that as $\bigcup_{\{i,j,k\} \in \mathcal{K}} \Delta_{i,j,k} = [0, 1] \times [0, 1]$, U_0 can be considered as 2D map, which is also bijective. Differing from its application in T-spline morphing, here we do not need to freeze the boundary, because for texture mapping the type of domain is free.

Figure 7.2: Initial constrained mapping

7.3.2 Induced time-dependent map

After computing the constrained map for the initial frame, we achieve the desirable texture mapping for it. For the rest frames, rather than recompute a map respectively, we choose to make full use of this existing one. Lets investigate how U_0 can help to establish a map for U_t . As mentioned above, the features in $\mathbf{S}_t$ is defined by (7.2). It means the desired 2D map U_t should satisfy

$$U_t(\mathbf{u}_i^0) = \mathbf{v}_i \quad (7.8)$$

Comparing with that in (7.7), if we let

$$U_t = U_{0\mathbf{M}_t}^* \quad (7.9)$$

the requirement is automatically satisfied. (7.9) implies that U_t is a induced map from U_0 as defined in (5.1). According to the definition, we suffice to calculate $U_0(\mathcal{A}^t)$. Suppose the vertex $\mathbf{A}_i^t$ is inside the triangle $\Delta_{i_1, i_2, i_3} \in \mathcal{K}^0$ with the barycentric coordinates (α, β, γ)

$$\mathbf{A}_i^t = \alpha \mathbf{A}_{i_1}^0 + \beta \mathbf{A}_{i_2}^0 + \gamma \mathbf{A}_{i_3}^0 \quad (7.10)$$

Then image of U_0 can be represented as

$$\mathbf{B}_i^t = \alpha \mathbf{B}_{i_1}^0 + \beta \mathbf{B}_{i_2}^0 + \gamma \mathbf{B}_{i_3}^0 \quad (7.11)$$

where

$$\mathbf{B}_i^t = U_0(\mathbf{A}_i^t) \quad (7.12)$$

Figure 7.3: Example of foldover in the induced map from a piece-wise linear 2D map.

This method only requires the calculation of the barycentric coordinates of $\mathcal{A}^t$ with respect to the comprising triangle in $\mathbf{m}_0$. Compared with the method in Chapter 5, it is obviously faster and actually it can be done in realtime. Meanwhile, referring to (7.8), positional constraints are also satisfied. Therefore, the proposed method succeeds in fulfilling

the aforementioned two requirements for dynamic rendering. It is also worth pointing out that such a method cannot guarantee to be foldover free. Figure 7.3 shows such an example, where the foldover occurs. However, in our implementation, such foldover is quite rare and not perceivable because in practice as the tessellation gets denser, being foldover free is getting close to being locally bijective.

7.4 Results

In this section, we show some examples of the proposed method applied to the morphing sequence obtained in the last chapter.

Figure 7.4: Dynamic rendering

Figure 7.4 and 7.5 show the rendering results of pasting two different figures onto the same morphing sequence in Figure 6.8. All of them are implemented in realtime along with

(a) Feature correspondence

(b) Dynamic rendering

Figure 7.5: Paste a new figure

the dynamic evolution in the physically based morphing. Note that new figure means the new target position in the constrained map. Compared with Figure 6.8, the visual appearance is significantly enhanced using our proposed method. In our method, we choose the frame with source object to compute the initial constrained map. Actually, any frame in this sequence can serve for this purpose. The method still works. Figure 7.6 shows the result of pasting the photo of a handsome man onto the sequence in Figure 6.15, and the feature correspondence is shown in Figure 7.6(b). In this example, we use the target frame to compute the initial map.

(a) Morphing sequence

(b) Feature correspondence for target frame

(c) Morphing sequence

Figure 7.6: Another rendering example. The target frame is employed to compute the initial map.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Our research aims to marry physics with T-spline representation, enabling physical methodology to be applied in purely geometric processing. We have developed an efficient physically based model directly based on T-spline representation, and explored its geometric processing algorithms including morphing and dynamic rendering. Many novel techniques have been proposed, which successfully accomplish the goal of this research.

First, we have developed physically based dynamic T-splines following Lagrangian dynamics. The analytical solution not only produces an efficient and stable evolution, but also facilitates the users to make the choice of satisfactory results in the evolution sequence, which is cumbersome in numerical solution. We have also investigated the local refinement in PD-T-splines by locally updating the physical matrices. As a result, users are capable to interactively model detailed shapes within PD-T-splines scheme.

Second, we have proposed the use of general curve handle in T-spline surface editing. The surface deformation is driven by interactively adjusting the general curve on the surface based on R-PBDC scheme. As we transform the composite representation into a basis form, the modeling process is robust and efficient. As curve handle supports arbitrary type of curve domain, it provides a convenient and flexible way for users to express their modeling

intention. It is promising to be integrated with advanced sketching devices.

Third, we have also studied the problem of constrained mapping, which is encountered both in morphing and rendering. We have revealed the relationship between a continuous 2D map and its induced piece-wise linear transformation. We have also proposed an iterative mesh warping that is provably foldover free and satisfies hard constraints. As each step of warping is driven by RBF interpolation, this approach generates smooth transformation, thus avoiding post smoothing process. The applications in both morphing and texture mapping have demonstrated its effectiveness and efficiency.

Fourth, we have studied the problem of T-spline morphing and presented a new solution. Consistent approximation has been proposed to deal with the vertex correspondence issue. It can restore the input surfaces with compatible structures, which align the user specified features at the same time. Physically based morphing technique has also been developed to generate the vertex trajectory. We have also proposed a novel approach to introduce more DOFs into the physical system, which enables more controls over the morphing sequence. As a result, the physically based morphing produces more plausible and physically realistic transition than linear interpolation. Meanwhile, the extra DOFs provide more flexibility to generate rich morphing effects.

Fifth, we have investigated the problem of pasting a texture onto a dynamic T-spline sequence. Our method is efficient and can achieve realtime rendering along with dynamic evolution of PD-T-spline. And the experimental examples have shown the capability of the proposed method in enhancing the visual appearance of the sequence.

8.2 Future work

The research has also brought some topics for further studies:

- We have developed a systematic physically based modeling paradigm to deal with T-spline representation. Currently, we mainly focus on rectangle objects. In order to make our system complete and more powerful, we intend to generalize PD-T-splines

onto more general topology, and to integrate Boolean operations with PD-T-splines. So we have to deal with objects with multiple T-spline patches. It means when PD-T-splines are applied to each single patch, the interaction between the neighboring patches are also required to be taken care. Especially, when these neighboring patches are represented by trimmed surfaces, how to maintain the continuity along their sharing boundaries is rather challenging. Obviously, the results of this thesis provide a good base for this general situation. For example, we may employ the method in [94] to glue these patches together to form a single patch. Then our method can be readily applied.

- In Chapter 5, we have investigated the problem of constrained mapping for surfaces topologically equivalent to an open disk. Currently, mesh parameterization is focusing on cross parameterization between surfaces of general topology. As our method is only applied to planar parameterization, it can not be easily generalized to deal with their case. But the polycube maps [118] provide us a promising way, and they can map the general face onto a polycube domain. Our approach is applicable to each face of the polycube. As long as we can tackle the transition issue between adjacent faces, our method can readily be generalized to parameterize the general surface with constraints.
- Although T-splines are the state of the art free-form representation, they are still awkward in representing objects with arbitrary topology. On the other hand, subdivision surfaces have been proposed to remedy this drawback. It's tempting to extend our ideas to subdivision surfaces. Although several attempts [84] have been proposed, their solver suffers the similar problem as in DNURBS [122]. So it's worth investigating whether our closed form solution is still applicable in this situation, and how to introduce the curve handle into subdivision surface deformation also warrants further studying.
- Our curve-based surface editing has shown its capability and power in T-spline sur-

face editing. In the future, we will develop a modeling system making use of the computerized sketching tools. As these tools are advantageous in obtaining the spatial points, the target curve will be obtained through interpolating those sketched discrete points. In this way, the modeling system is entirely curve driven, which could significantly enhance the modeling efficiency.

- Introducing physics into morphing is really an interesting approach to generate the transition. However, because of the current internal energy functional employed, the transition can only guarantee smoothness, but fail to preserve the geometric properties. In the future, we will consider to incorporate some energy functionals based on intrinsic geometric information, e.g. to make the deformation quasi-conformal. But such energy functional is usually highly non-linear, then an efficient solver to generate the transition will be crucial.

References

- [1] M. Alexa, “Merging polyhedral shapes with scattered features,” in *Proceedings of International Conference on Shape Modeling and Applications 1999*, Mar 1999, pp. 202–210, 278.
- [2] M. Alexa, D. Cohen-Or, and D. Levin, “As-rigid-as-possible shape interpolation,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 157–164.
- [3] M. Alexa, “Local control for mesh morphing,” in *Shape Modeling and Applications, SMI 2001 International Conference on*. IEEE, 2001, pp. 209–215.
- [4] —, “Differential coordinates for local mesh morphing and deformation,” *The Visual Computer*, vol. 19, no. 2, pp. 105–114, 2003.
- [5] C. K. Au and M. M.-F. Yuen, “Unified approach to NURBS curve shape modification,” *Computer-Aided Design*, vol. 27, no. 2, pp. 85–93, 1995.
- [6] T. Caughe and M. Okelly, “Classical normal modes indamped linear dynamic systems,” *ASME J. Appl. Mech.*, pp. 583–588, 1965.
- [7] G. Celniker and D. Gossard, “Deformable curve and surface finite-elements for free-form shape design,” in *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1991, pp. 257–266.

-
- [8] G. Celniker and W. Welch, “Linear constraints for deformable non-uniform B-spline surfaces,” in *I3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*. New York, NY, USA: ACM, 1992, pp. 165–170.
- [9] S. Cohen, G. Elber, and R. Bar-Yehuda, “Matching of freeform curves,” *Computer-Aided Design*, vol. 29, no. 5, pp. 369–378, 1997.
- [10] F. Dachille, IX, H. Qin, A. Kaufman, and J. El-Sana, “Haptic sculpting of dynamic surfaces,” in *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*. New York, NY, USA: ACM, 1999, pp. 103–110.
- [11] F. I. Dachille, H. Qin, and A. Kaufman, “A novel haptics-based interface and sculpting system for physics-based geometric design,” *Computer-Aided Design*, vol. 33, no. 5, pp. 403 – 420, 2001.
- [12] M. Desbrun, M. Meyer, and P. Alliez, “Intrinsic parameterizations of surface meshes,” *Computer Graphics Forum*, vol. 21, no. 3, pp. 209–218, 2002.
- [13] H. Du and H. Qin, “Dynamic PDE surfaces with flexible and general geometric constraints,” in *PG '00: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA: IEEE Computer Society, 2000, p. 213.
- [14] —, “Dynamic PDE-based surface design using geometric and physical constraints,” *Graphical Models*, vol. 67, no. 1, pp. 43–71, 2005.
- [15] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, “Multiresolution analysis of arbitrary meshes,” in *Proceedings of SIGGRAPH'95*. New York, NY, USA: ACM, 1995, pp. 173–182.
- [16] I. Eckstein, V. Surazhsky, and C. Gotsman, “Texture mapping with hard constraints,” *Computer Graphics Forum*, vol. 20, no. 3, pp. 95–104, 2001.

- [17] F. Espino, M. Bóo, M. Amor, and J. Bruguera, “Adaptive tessellation of NURBS surfaces,” in *Journal of WSCG*. Citeseer, 2003.
- [18] G. Farin, G. Rein, N. Sapidis, and A. J. Worsey, “Fairing cubic B-spline curves,” *Computer Aided Geometric Design*, vol. 4, no. 1, pp. 91–103, 1987.
- [19] M. S. Floater, “Parametrization and smooth approximation of surface triangulations,” *Computer Aided Geometric Design*, vol. 14, no. 3, pp. 231 – 250, 1997.
- [20] —, “Mean value coordinates,” *Computer Aided Geometric Design*, vol. 20, no. 1, pp. 19 – 27, 2003.
- [21] M. S. Floater and K. Hormann, “Surface parameterization: a tutorial and survey,” in *Advances in Multiresolution for Geometric Modelling*, ser. Mathematics and Visualization, N. A. e. a. Dodgson, Ed. Springer Berlin Heidelberg, 2005, pp. 157–186.
- [22] B. Fowler and R. Bartels, “Constraint-based curve manipulation,” *IEEE Comput. Graph. Appl.*, vol. 13, pp. 43–49, September 1993.
- [23] K. Fujimura and M. Makarov, “Foldover-free image warping,” *Graphical Models and Image Processing*, vol. 60, no. 2, pp. 100 – 111, 1998.
- [24] I. Garcia, J. Xia, Y. He, S. Xin, and G. Patow, “Interactive applications for sketch-based editable polycube map,” *IEEE transactions on visualization and computer graphics*, vol. 19, no. 7, p. 1158, 2013.
- [25] Y. I. Gingold, P. L. Davidson, J. Y. Han, and D. Zorin, “A direct texture placement and editing interface,” in *Proceedings of the 19th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM, 2006, pp. 23–32.
- [26] B. Gossick, *Hamilton’s Principle and Physical systems*. New York and London: Academic Press, 1967.

-
- [27] G. Greiner, “Variational design and fairing of spline surfaces,” in *Computer Graphics Forum*, vol. 13, no. 3. Wiley Online Library, 1994, pp. 143–154.
 - [28] H. Hagen and P. Santarelli, “Variational design of smooth B-spline surfaces,” *Topics in geometric modeling*, pp. 85–94, 1992.
 - [29] H. Hagen and G. Schulze, “Automatic smoothing with geometric surface patches,” *Computer Aided Geometric Design*, vol. 4, no. 3, pp. 231–235, 1987.
 - [30] S. Hahmann, “Shape improvement of surfaces,” in *Geometric Modelling*. Springer, 1998, pp. 135–152.
 - [31] Y. He, H. Wang, C.-W. Fu, and H. Qin, “A divide-and-conquer approach for automatic polycube map construction,” *Computers & Graphics*, vol. 33, no. 3, pp. 369–380, 2009.
 - [32] S. Hu, Y. Li, T. Ju, and X. Zhu, “Modifying the shape of NURBS surfaces with geometric constraints,” *Computer-Aided Design*, vol. 33, no. 12, pp. 903 – 912, 2001.
 - [33] J. Huang, Y. Tong, K. Zhou, H. Bao, and M. Desbrun, “Interactive shape interpolation through controllable dynamic deformation,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 7, pp. 983–992, 2011.
 - [34] T. Igarashi, T. Moscovich, and J. F. Hughes, “As-rigid-as-possible shape manipulation,” in *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3. ACM, 2005, pp. 1134–1141.
 - [35] J. Ishida, “The general B-spline interpolation method and its application to the modification of curves and surfaces,” *Computer-Aided Design*, vol. 29, no. 11, pp. 779 – 790, 1997.
 - [36] T. Ju and R. Goldman, “Morphing of rational b-spline curves and surfaces using mass distributions,” in *Proceedings of Eurographics, short papers*, 2003.

- [37] T. Kanai, H. Suzuki, and F. Kimura, “Metamorphosis of arbitrary triangular meshes,” *IEEE Comput. Graph. Appl.*, vol. 20, no. 2, pp. 62–75, 2000.
- [38] H. Kardestuncer and D. Norrie, Eds., *Finite element handbook*. New York, NY, USA: McGraw-Hill, Inc., 1987.
- [39] M. Kass and J. Anderson, “Animating oscillatory motion with overlap: wiggly splines,” in *ACM SIGGRAPH 2008 papers*. Los Angeles, California: ACM, 2008, pp. 1–8.
- [40] D. Kawano, M. Morzfeld, and F. Ma, “The decoupling of defective linear dynamical systems in free motion,” *Journal of Sound and Vibration*, vol. 330, no. 21, pp. 5165–5183, 2011.
- [41] V. Kraevoy and A. Sheffer, “Cross-parameterization and compatible remeshing of 3D models,” *ACM Trans. Graph.*, vol. 23, no. 3, pp. 861–869, 2004.
- [42] V. Kraevoy, A. Sheffer, and C. Gotsman, “Matchmaker: constructing constrained texture maps,” *ACM Trans. Graph.*, vol. 22, no. 3, pp. 326–333, Jul. 2003.
- [43] A. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin, “Maps: multiresolution adaptive parameterization of surfaces,” in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 1998, pp. 95–104.
- [44] T.-Y. Lee and P.-H. Huang, “Fast and intuitive metamorphosis of 3d polyhedral models using smcc mesh merging scheme,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 1, pp. 85 – 98, Jan.-March 2003.
- [45] T.-Y. Lee, S.-W. Yen, and I.-C. Yeh, “Texture mapping with hard constraints using warping scheme,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 2, pp. 382 –395, March-April 2008.

- [46] J. Lenoir, L. Grisoni, P. Meseure, Y. Rémion, and C. Chaillou, “Smooth constraints for spline variational modeling,” in *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. New York, NY, USA: ACM, 2004, pp. 58–64.
- [47] B. Lévy, “Constrained texture mapping for polygonal meshes,” in *Proceedings of SIGGRAPH'01*. New York, NY, USA: ACM, 2001, pp. 417–424.
- [48] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, “Least squares conformal maps for automatic texture atlas generation,” in *Proceedings of SIGGRAPH'02*. New York, NY, USA: ACM, 2002, pp. 362–371.
- [49] B. Li, X. Zhang, P. Zhou, and P. Hu, “Mesh parameterization based on one-step inverse forming,” *Computer-Aided Design*, vol. 42, no. 7, pp. 633–640, 2010.
- [50] C.-H. Lin, T.-Y. Lee, H.-K. Chu, and C.-Y. Yao, “Progressive mesh metamorphosis,” *Computer Animation and Virtual Worlds*, vol. 16, no. 3-4, pp. 487–498, 2005.
- [51] J. Lin, X. Jin, Z. Fan, and C. C. Wang, “Automatic polycube-maps,” in *Advances in Geometric Modeling and Processing*. Springer, 2008, pp. 3–16.
- [52] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or, “Linear rotation-invariant coordinates for meshes,” in *ACM SIGGRAPH 2005 Papers*. Los Angeles, California: ACM, 2005, pp. 479–487.
- [53] X. Liu and F. Feng, “Energy-based shape modification of NURBS surfaces,” in *Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization - Volume 01*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 607–609.
- [54] X.-Y. Liu, “Geometric features modification of NURBS curves via energy optimization,” in *Proceedings of the 2009 First International Workshop on Education Tech-*

- nology and Computer Science - Volume 01*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 929–932.
- [55] Y.-S. Liu, H.-B. Yan, and R. R. Martin, “As-rigid-as-possible surface morphing,” *Journal of Computer Science and Technology*, vol. 26, no. 3, pp. 548–557, 2011.
- [56] N. Lott and D. Pullin, “Method for fairing B-spline surfaces,” *Comput. Aided Des.*, vol. 20, no. 10, pp. 597–604, 1988.
- [57] L. Lui, K. Lam, S. Yau, and X. Gu, “Teichmüller extremal mapping and its applications to landmark matching registration,” *arXiv preprint arXiv:1211.2569*.
- [58] F. Ma, A. Imam, and M. Morzfeld, “The decoupling of damped linear systems in oscillatory free vibration,” *Journal of sound and vibration*, vol. 324, no. 1, pp. 408–428, 2009.
- [59] F. Ma, M. Morzfeld, and A. Imam, “The decoupling of damped linear systems in free or forced vibration,” *Journal of Sound and Vibration*, vol. 329, no. 15, pp. 3182–3202, 2010.
- [60] C. Mandal, H. Qin, and B. Vemuri, “Dynamic smooth subdivision surfaces for data visualization,” in *VIS '97: Proceedings of the 8th conference on Visualization '97*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1997, pp. 371–ff.
- [61] —, “A novel FEM-based dynamic framework for subdivision surfaces,” in *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*. New York, NY, USA: ACM, 1999, pp. 191–202.
- [62] —, “Dynamic modeling of Butterfly subdivision surfaces,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 3, pp. 265–287, 2000.
- [63] C. Mandal, B. Vemuri, and H. Qin, “A new dynamic FEM-Based subdivision surface model for shape recovery and tracking in medical images,” in *MICCAI '98*:

- Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention*. London, UK: Springer-Verlag, 1998, pp. 753–760.
- [64] ———, “Shape recovery using dynamic subdivision surfaces,” in *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*. Washington, DC, USA: IEEE Computer Society, 1998, p. 805.
- [65] J. McCartney, B. Hinds, and K. Chong, “Pattern flattening for orthotropic materials,” *Computer-Aided Design*, vol. 37, no. 6, pp. 631–644, 2005.
- [66] K. McDonnell and H. Qin, “FEM-based subdivision solids for dynamic and haptic interaction,” in *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*. New York, NY, USA: ACM, 2001, pp. 312–313.
- [67] K. McDonnell, H. Qin, and R. Wlodarczyk, “Virtual clay: a real-time sculpting system with haptic toolkits,” in *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*. New York, NY, USA: ACM, 2001, pp. 179–190.
- [68] T. McInerney, “Finite element techniques for fitting deformable models to 3D data,” Master’s thesis, Department of Computer Science, University of Toronto, Toronto, ON, Canada, 1992.
- [69] P. Michalik, D. H. Kim, and B. D. Bruderlin, “Sketch-and constraint-based design of b-spline surfaces,” in *Proceedings of the seventh ACM symposium on Solid modeling and applications*. ACM, 2002, pp. 297–304.
- [70] C. Moler and C. Van Loan, “Nineteen dubious ways to compute the exponential of a matrix,” *SIAM review*, vol. 20, no. 4, pp. 801–836, 1978.
- [71] H. P. Moreton and C. H. Séquin, *Functional optimization for fair surface design*. ACM, 1992, vol. 26, no. 2.

-
- [72] H. Moreton and C. Séquin, “Functional optimization for fair surface design,” in *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1992, pp. 167–176.
- [73] M. Morzfeld and F. Ma, “The decoupling of damped linear systems in configuration and state spaces,” *Journal of Sound and Vibration*, vol. 330, no. 2, pp. 155–161, 2011.
- [74] M. Morzfeld, F. Ma, and B. Parlett, “The transformation of second-order linear systems into independent equations,” *SIAM Journal on Applied Mathematics*, vol. 71, no. 4, pp. 1026–1043, 2011.
- [75] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, “Fibermesh: designing freeform surfaces with 3D curves,” *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, p. 41, 2007.
- [76] L. Piegl, “Modifying the shape of rational B-splines. part 1: curves,” *Computer-Aided Design*, vol. 21, no. 8, pp. 509 – 518, 1989.
- [77] —, “Modifying the shape of rational B-splines. part 2: surfaces,” *Computer-Aided Design*, vol. 21, no. 9, pp. 538 – 546, 1989.
- [78] M. Pourazady and X. Xu, “Direct manipulations of B-spline and NURBS curves,” *Advances in Engineering Software*, vol. 31, no. 2, pp. 107 – 118, 2000.
- [79] —, “Direct manipulations of NURBS surfaces subjected to geometric constraints,” *Computers & Graphics*, vol. 30, no. 4, pp. 598 – 609, 2006.
- [80] E. Praun, W. Sweldens, and P. Schröder, “Consistent mesh parameterizations,” in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 179–184.
- [81] —, “Consistent mesh parameterizations,” in *Proceedings of SIGGRAPH'01*. New York, NY, USA: ACM, 2001, pp. 179–184.

-
- [82] W. H. Press, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.
 - [83] R. Pusch and F. Samavati, “Local constraint-based general surface deformation,” *Shape Modeling and Applications, International Conference on*, vol. 0, pp. 256–260, 2010.
 - [84] H. Qin, C. Mandal, and B. Vemuri, “Dynamic Catmull-Clark subdivision surfaces,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 3, pp. 215–229, 1998.
 - [85] H. Qin and D. Terzopoulos, “Triangular NURBS and their dynamic generalizations,” *Computer Aided Geometric Design*, vol. 14, no. 4, pp. 325 – 347, 1997.
 - [86] —, “Dynamic NURBS swung surfaces for physics-based shape design,” *Computer-Aided Design*, vol. 27, no. 2, pp. 111–127, 1995.
 - [87] K. Qin, Z. Chang, H. Wang, and D. Li, “Physics-based loop surface modeling,” *J. Comput. Sci. Technol.*, vol. 17, no. 6, pp. 851–858, 2002.
 - [88] K. Qin, H. Wang, D. Li, R. Kikinis, and M. Halle, “Physics-based subdivision surface modeling for medical imaging and simulation,” in *MIAR ’01: Proceedings of the International Workshop on Medical Imaging and Augmented Reality (MIAR ’01)*. Washington, DC, USA: IEEE Computer Society, 2001, p. 117.
 - [89] B. J. W. S. Rayleigh, *The theory of sound*. Macmillan, 1896, vol. 2.
 - [90] A. Saalfeld, “Joint triangulations and triangulation maps,” in *Proceedings of the 3rd Annual Symposium on Computational Geometry*. New York, NY, USA: ACM, 1987, pp. 195–204.
 - [91] J. Sánchez-Reyes, “A simple technique for NURBS shape modification,” *IEEE Comput. Graph. Appl.*, vol. 17, pp. 52–59, January 1997.

-
- [92] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe, “Texture mapping progressive meshes,” in *Proceedings of SIGGRAPH’01*. New York, NY, USA: ACM, 2001, pp. 409–416.
- [93] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe, “Inter-surface mapping,” in *Proceedings of SIGGRAPH’04*. New York, NY, USA: ACM, 2004, pp. 870–877.
- [94] T. W. Sederberg, G. T. Finnigan, X. Li, H. Lin, and H. Ipson, “Watertight trimmed NURBS,” in *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3. ACM, 2008, p. 79.
- [95] T. W. Sederberg and E. Greenwood, “Shape blending of 2-D piecewise curves,” *Mathematical Methods in CAGD*, vol. 3, no. 1, p. 3, 1995.
- [96] T. Sederberg, D. Cardon, G. Finnigan, N. North, J. Zheng, and T. Lyche, “T-spline simplification and local refinement,” *ACM Trans. Graph.*, vol. 23, no. 3, pp. 276–283, 2004.
- [97] T. Sederberg, P. Gao, G. Wang, and H. Mu, “2-D shape blending: an intrinsic solution to the vertex path problem,” in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. Anaheim, CA: ACM, 1993, pp. 15–18.
- [98] T. Sederberg and E. Greenwood, “A physically based approach to 2-D shape blending,” *SIGGRAPH Comput. Graph.*, vol. 26, no. 2, pp. 25–34, 1992.
- [99] T. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, “T-splines and T-NURCCs,” *ACM Trans. Graph.*, vol. 22, no. 3, pp. 477–484, 2003.
- [100] T. Sederberg, J. Zheng, D. Sewell, and M. Sabin, “Non-uniform recursive subdivision surfaces,” in *SIGGRAPH ’98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1998, pp. 387–394.

- [101] H. Seo and F. Cordier, “Constrained texture mapping using image warping,” *Computer Graphics Forum*, vol. 29, no. 1, pp. 160–174, 2010.
- [102] M. Shapira and A. Rappoport, “Shape blending using the star-skeleton representation,” *IEEE Comput. Graph. Appl.*, vol. 15, no. 2, pp. 44–50, 1995.
- [103] A. Sheffer and V. Kraevoy, “Pyramid coordinates for morphing and deformation,” in *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*. IEEE Computer Society, 2004, pp. 68–75.
- [104] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov, “Abf++: fast and robust angle based flattening,” *ACM Trans. Graph.*, vol. 24, no. 2, pp. 311–330, Apr. 2005.
- [105] A. Sheffer, E. Praun, and K. Rose, “Mesh parameterization methods and their applications,” *Found. Trends. Comput. Graph. Vis.*, vol. 2, no. 2, pp. 105–171, Jan. 2006.
- [106] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel, “Laplacian surface editing,” in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Nice, France: ACM, 2004, pp. 175–184.
- [107] O. Sorkine and M. Alexa, “As-rigid-as-possible surface modeling,” in *Symposium on Geometry processing*. Citeseer, 2007, pp. 109–116.
- [108] D. L. Souvaine and R. Wenger, “Constructing piecewise linear homeomorphisms,” 1994, DIMACS Technical Report 94-52.
- [109] J. Stam, “Evaluation of Loop subdivision surfaces,” in *SIGGRAPH98 CDROM Proceedings*, 1998.
- [110] —, “Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values,” in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 1998, pp. 395–404.

-
- [111] G. Strang and K. Aarikka, *Introduction to applied mathematics*. Wellesley-Cambridge Press Wellesley, MA, 1986, vol. 16.
 - [112] R. W. Sumner, M. Zwicker, C. Gotsman, and J. Popović, “Mesh-based inverse kinematics,” in *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3. ACM, 2005, pp. 488–495.
 - [113] T. Surazhsky and G. Elber, “Matching free-form surfaces,” *Computers & Graphics*, vol. 25, no. 1, pp. 3–12, 2001.
 - [114] —, “Metamorphosis of planar parametric curves via curvature interpolation,” *International Journal of Shape Modeling*, vol. 8, no. 02, pp. 201–216, 2002.
 - [115] V. Surazhsky and C. Gotsman, “Controllable morphing of compatible planar triangulations,” *ACM Trans. Graph.*, vol. 20, no. 4, pp. 203–231, Oct. 2001.
 - [116] R. Szeliski and D. Terzopoulos, “From splines to fractals,” *SIGGRAPH Comput. Graph.*, vol. 23, no. 3, pp. 51–60, 1989.
 - [117] Y. Tang, J. Wang, H. Bao, and Q. Peng, “RBF-based constrained texture mapping,” *Computers & Graphics*, vol. 27, no. 3, pp. 415 – 422, 2003.
 - [118] M. Tarini, K. Hormann, P. Cignoni, and C. Montani, “Polycube-maps,” in *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3. ACM, 2004, pp. 853–860.
 - [119] D. Terzopoulos and K. Fleischer, “Modeling inelastic deformation: viscoelasticity, plasticity, fracture,” in *SIGGRAPH ’88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1988, pp. 269–278.
 - [120] D. Terzopoulos and D. Metaxas, “Dynamic 3D models with local and global deformations: Deformable superquadrics,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 7, pp. 703–714, 1991.

- [121] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, “Elastically deformable models,” *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 205–214, 1987.
- [122] D. Terzopoulos and H. Qin, “Dynamic NURBS with geometric constraints for interactive sculpting,” *ACM Trans. Graph.*, vol. 13, no. 2, pp. 103–136, 1994.
- [123] D. Terzopoulos and A. Witkin, “Physically based models with rigid and deformable components,” *IEEE Comput. Graph. Appl.*, vol. 8, no. 6, pp. 41–51, 1988.
- [124] D. Terzopoulos, A. Witkin, and M. Kass, “Constraints on deformable models:recovering 3D shape and nonrigid motion,” *Artificial Intelligence*, vol. 36, no. 1, pp. 91 – 123, 1988.
- [125] A. Theetten, L. Grisoni, C. Andriot, and B. Barsky, “Geometrically exact dynamic splines,” *Comput. Aided Des.*, vol. 40, no. 1, pp. 35–48, 2008.
- [126] A. Theetten, L. Grisoni, C. Duriez, and X. Merlhiot, “Quasi-dynamic splines,” in *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*. New York, NY, USA: ACM, 2007, pp. 409–414.
- [127] C. C. Wang, K.-C. Hui, and K.-M. Tong, “Volume parameterization for design automation of customized free-form products,” *Automation Science and Engineering, IEEE Transactions on*, vol. 4, no. 1, pp. 11–21, 2007.
- [128] C. C. Wang, S. S. Smith, and M. M. Yuen, “Surface flattening based on energy model,” *Computer-Aided Design*, vol. 34, no. 11, pp. 823–833, 2002.
- [129] H. Wang, Y. He, X. Li, X. Gu, and H. Qin, “Polycube splines,” *Computer-Aided Design*, vol. 40, no. 6, pp. 721–733, 2008.
- [130] H. Wang, M. Jin, Y. He, X. Gu, and H. Qin, “User-controllable polycube map for manifold spline construction,” in *Proceedings of the 2008 ACM symposium on Solid and physical modeling*. ACM, 2008, pp. 397–404.

-
- [131] Y. Wang and J. Zheng, “Curvature-guided adaptive T-spline surface fitting,” *Computer-Aided Design*, vol. 45, no. 8, pp. 1095 – 1107, 2013.
 - [132] W. Welch and A. Witkin, “Variational surface modeling,” in *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1992, pp. 157–166.
 - [133] J. Xia, I. Garcia, Y. He, S.-Q. Xin, and G. Patow, “Editable polycube map for gpu-based subdivision surfaces,” in *Symposium on interactive 3D graphics and games*. ACM, 2011, pp. 151–158.
 - [134] H. Xie and H. Qin, “A physics-based framework for subdivision surface design with automatic rules control,” in *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA: IEEE Computer Society, 2002, p. 304.
 - [135] H. Xie and H. Qing, “A novel optimization approach to the effective computation of NURBS knots,” *International Journal of Shape Modeling*, 2001.
 - [136] D. Xu, H. Zhang, Q. Wang, and H. Bao, “Poisson shape interpolation,” in *Proceedings of the 2005 ACM symposium on Solid and physical modeling*. Cambridge, Massachusetts: ACM, 2005, pp. 267–274.
 - [137] H. Yan, S. Hu, and R. Martin, “Morphing based on strain field interpolation: Research articles,” *Comput. Animat. Virtual Worlds*, vol. 15, no. 3-4, pp. 443–452, 2004.
 - [138] —, “3D morphing using strain field interpolation,” *J. Comput. Sci. Technol.*, vol. 22, no. 1, pp. 147–155, 2007.
 - [139] H. Yu, T.-Y. Lee, I.-C. Yeh, X. Yang, W. Li, and J. Zhang, “An RBF-based reparameterization method for constrained texture mapping,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 7, pp. 1115 –1124, july 2012.

-
- [140] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H. Shum, “Mesh editing with Poisson-based gradient field manipulation,” in *ACM SIGGRAPH 2004*. Los Angeles, California: ACM, 2004, pp. 644–651.
- [141] G. Zhan and G. Ian, “Haptic sculpting of multi-resolution B-spline surfaces with shaped tools,” *Computer-Aided Design*, vol. 38, no. 6, pp. 661 – 676, 2006.
- [142] M. Zhang and H. Qin, “Hierarchical D-NURBS surfaces and their physics-based sculpting,” in *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*. Washington, DC, USA: IEEE Computer Society, 2001, p. 257.
- [143] J. M. Zheng, K. W. Chan, and I. Gibson, “A new approach for direct manipulation of free-form curve,” *Computer Graphics Forum*, vol. 17, no. 3, pp. 327–334, 1998.
- [144] W. Zhiguo, Z. Laishui, and W. Xiaoping, “Direct manipulation of B-spline surfaces,” *Chinese Journal of Mechanical Engineering*, vol. 18, no. 1, 2005.
- [145] Y. Zhong and B. Xu, “A physically based method for triangulated surface flattening,” *Computer-Aided Design*, vol. 38, no. 10, pp. 1062–1073, 2006.
- [146] M. Zöckler, D. Stalling, and H.-C. Hege, “Fast and intuitive generation of geometric shape transitions,” *The Visual Computer*, vol. 16, pp. 241–253, 2000.

Author's Publications

1. Wenjing Zhang, **Yuewen Ma**, and Jianmin Zheng, "Inversion free tetrahedral mesh warping driven by boundary surface deformation," In proceeding of *CAD/Graphics*, Hongkong, 2013.
2. **Yuewen Ma**, and Jianmin Zheng, "Physically-based NURBS Surface Editing With Curves," *Computer Aided Design and Applications*, Vol. 9, No. 3, pp 361-374, 2012.
3. Kai Wang, Jianmin Zheng, Hock-Soon Seah, and **Yuewen Ma**, "Triangular Mesh Deformation via Edge-based Graph," *Computer Aided Design and Applications* (Best Paper Award from 2011 International Conference on Computer-Aided Design and Exhibition), Vol. 9, No. 3, pp 345-359, 2012.
4. **Yuewen Ma**, and Jianmin Zheng, "Physics-Based B-spline Morphing," In proceeding of 9th *NicoGraph International*, Singapore, June 18-19, 2010.
5. **Yuewen Ma**, Jianmin Zheng and Jian Xie, "Foldover-Free Mesh Warping for Constrained Texture Mapping," submitted to *IEEE Transactions on Visualization and Computer Graphics*, 2013.
6. **Yuewen Ma**, and Jianmin Zheng, "Physically-based Dynamic T-splines and Local refinement," to be submitted, 2013.
7. **Yuewen Ma**, and Jianmin Zheng, "T-spline Surface Editing Using General Curve Handle," to be submitted, 2013.

8. **Yuewen Ma**, and Jianmin Zheng, “Physically Based T-spline Morphing with Force Control,” to be submitted, 2013.