Geometric processing of compound T-spline surfaces

Li, Yusha

2016

Li, Y. (2016). Geometric processing of compound T-spline surfaces. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/69071>

<https://doi.org/10.32657/10356/69071>

NANYANG TECHNOLOGICAL UNIVERSITY

GEOMETRIC PROCESSING OF COMPOUND T-SPLINE
SURFACES

BY

LI YUSHA

A THESIS SUBMITTED TO THE NANYANG TECHNOLOGICAL UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

2016

Abstract

This thesis is concerned with geometric processing algorithms for compound T-spline surfaces. A compound T-spline surface model is usually formed by a collection of T-spline surfaces. It is suitable for representing complicated shapes in computer-aided geometric design and solid modeling. Geometric processing refers to the theory and algorithms for analyzing and manipulating geometric objects. While T-spline technology is becoming popular, there is need to develop a rich family of geometric processes for T-splines. This research investigates four fundamental geometric processes of compound T-spline surfaces: rational Bézier extraction from T-splines, adaptive tessellation of compound T-spline models, T-spline knot removal, and T-spline surface extension, which are relatively less explored. Our objectives are to gain deep understanding of these processes and develop effective and efficient algorithms on GPU to handle compound T-spline surface models accurately, interactively and seamlessly.

First, we present a method for extracting Bézier patches from T-spline surfaces, which is a useful process for T-spline tessellation, T-spline surface intersection, T-spline based iso-geometric analysis, etc. The difficulty of the process lies on the flexibility of T-spline topological structure and the complexity of GPU implementation. The underlying steps of our method contain correctly figuring out C^∞ zones, partitioning non-rectangular zones into Bézier patch domains and computation of Bézier representation. We design techniques to implement all these steps on GPU. As a result, our method can correctly extract a minimum number of Bézier patches from a compound T-spline surface model in real time.

Second, we propose a novel framework for tessellating compound T-spline surfaces. The underlying techniques involve boundary merging, surface decomposition, tessellation estimation and mesh generation. Except for boundary merging, all others are designed to be GPU-friendly. We improve the tessellation factor estimation for rational Bézier curves and surfaces, and design parallel strategies for curve and patch tessellation and mesh generation. As a result, our method can adaptively tessellate T-spline models into crack-free

triangular meshes in real time on GPU. The generated triangular meshes are guaranteed to approximate the T-spline models within the given tolerance.

Third, we analyze the characteristics of T-spline knot removal, which is surprisingly much more complicated than B-spline knot removal. Based on typical patterns of T-spline knot insertion and removal, we propose a T-spline multi-point removal algorithm. The algorithm first identifies removal groups in the mesh, and then removes all the points in a group simultaneously. Compared to a single-point removal, the multi-point removal is more effective. Moreover, we parallel the removal process and implement it on GPU, which greatly improves the efficiency of the algorithm in terms of run time. We also consider the approximate T-spline multi-point removal, which can be used to simplify T-spline models.

Fourth, we propose a scheme to represent T-splines with complicated boundaries and develop techniques for extending T-splines from partial boundaries, which results in T-spline models with complicated boundaries. This can alleviate the workload of handling cracks in compound models and reduce the number of surfaces used for modeling complicated shapes. We provide T-spline surface extension and trimming tools for the user to perform interactive design. The user can extend the surface to interpolate a curve or trim part of the surface. These operations keep the original surface unchanged. The complex boundary T-splines enhance the representation power of T-splines in modeling complicated shapes including those with complicated boundaries or holes.

Acknowledgements

At the very beginning, I would like to express my deep sense of gratitude to my supervisors, Associate Professor Zheng Jianmin and Associate Professor Cai Yiyu. They have offered me lots of valuable ideas and criticisms with their profound knowledge and rich research experience. They also broaden my view and knowledge to let me have an overview of my research field which is very helpful in exploring new ideas. Thanks for their patience with my research progress in the last few years in which that I have learned is more than academic knowledge but a studying attitude. Their rigorous academic attitude exerts great influence on me not only at present but also in the future, even in the whole life.

I would express my sincere appreciation to Dr. Chen Wenyu. His kindness and patience are greatly appreciated. Thank him for giving me wonderful advices on both research and programming. Without his construction of the project framework, the experiments cannot be realized easily.

Thank all the friends, Zhou Hailing, Wang Kai, Ma Yuewen, Wu Xiaoqun, Zhang Yuzhe, Zhang Wenjing, Zhang jvzheng, Wang Tianyi and visiting staffs in GameLab, Pan Jianjiang, Chen Xiaodiao, Cao Juan, Xu Huixia, Wang Aizeng, for their help and many valuable hints. They stayed with me through a unforgettable period of time.

Thank my whole family for their continuing support and encouragement. Without them, the thesis is not possible to present here. I am grateful for the encouragement and tolerance of my husband Wang Qiang. In my busy time, he took great responsibility for the whole family. My lovely baby, Wang Chuyi is always the pride and joy of my life. I am sorry for spending so less time with her in the last two years. Thank my mother for taking care of her which is not an easier job than PhD study.

Finally, I appreciate the financial support for this research provided by AcRF MOE Tier-2 Project (MOE2008-T2-1-075), and the space and equipment provided by gameLAB, School of Computer Engineering of Nanyang Technological University and Multi-Platform Game Innovation Centre.

Contents

Abstract	i
Acknowledgements	iii
Table of Contents	iv
List of Figures	viii
1 Introduction	1
1.1 Background and motivation	1
1.2 Objectives and contributions	6
1.3 Organization of the thesis	10
2 Literature Review	12
2.1 Fundamental processing of NURBS	12
2.1.1 Foundations	12
2.1.2 Knot insertion	14
2.1.3 Knot removal	16
2.2 Basic T-spline techniques	17
2.2.1 Foundations	18
2.2.2 Local refinement: control point insertion	19
2.2.3 Local simplification: control point removal	22
2.3 Compound models	25
2.3.1 Watertight models	27
2.3.2 Boundary representation of compound surfaces	30

2.3.3	Cracks filling in visualization	32
2.4	GPU rendering of geometric models	34
2.4.1	CUDA architecture	35
2.4.2	GPU performance analysis	37
2.4.3	B-rep model rendering on GPU	40
3	Extraction of Rational Bézier Patches from T-spline Surfaces on GPU	43
3.1	Introduction	43
3.2	Underlying techniques for Bézier patch extraction	45
3.2.1	C^∞ domain identification	46
3.2.2	Partitioning of irregular zones	46
3.2.3	Computation of Bézier control points	48
3.3	GPU implementation	50
3.3.1	Special encoding of T-splines for GPU	52
3.3.2	Kernel function 1: Topology construction	54
3.3.3	Kernel function 2: Bézier domains identification	55
3.3.4	Kernel function 3: Bézier patch computation	56
3.4	Experimental results and discussion	58
4	GPU-friendly Crack-free Adaptive Tessellation of B-rep Models	64
4.1	Introduction	64
4.2	Algorithm overview	67
4.3	Boundary merging	70
4.4	Surfaces decomposition	72
4.4.1	Bézier patch extraction	72
4.4.2	Bézier curve identification	73
4.4.3	Bézier curve computation	75
4.5	Tessellation factor estimation	78
4.6	Mesh generation	84

4.6.1	Transition region identification	84
4.6.2	Transition region creation	85
4.7	Results and discussion	86
4.7.1	Tessellation effect	87
4.7.2	Comparison of tessellation factor estimation	88
4.7.3	More examples	88
5	Parallel T-spline Multi-point Removal	93
5.1	Introduction	93
5.2	Complexity of T-spline knot removal	94
5.3	GPU-based multi-point removal	98
5.3.1	Local removal group identification	100
5.3.2	Removal error assessment	104
5.3.3	Global removal group assemble	105
5.3.4	Global removal group update	106
5.4	Approximate multi-point removal	107
5.5	Results and discussion	110
6	Construction of T-spline Surfaces with Complicated Boundaries	117
6.1	Introduction	117
6.2	Preliminaries	119
6.2.1	Knot modification of NURBS curves	119
6.2.2	NURBS surface extension	120
6.3	Construction of T-spline surfaces with complicated boundaries	122
6.3.1	Structure of complicated boundary T-spline	122
6.3.2	T-spline surface extension algorithm	124
6.3.3	Surface trimming	128
6.3.4	Local refinement	129

6.4	Experimental results	131
7	Summary and Future Work	137
7.1	Summary	137
7.2	Future Work	139
	References	142
	Author's Publications	153

List of Figures

1.1	A model represented by NURBS and T-splines (T-junctions are highlighted with purple.) (a) the NURBS surface with 4422 control points, (b) the T-spline surface with 1094 control points,(c) the NURBS topology, and (d) the T-mesh topology. (For all the figures of T-mesh topology in the thesis, row is along the s direction and column is along the t direction if not mentioned.)	3
1.2	(a) trimmed NURBS representation with a gap between teapot body and spout, (b) gap free teapot created by T-spline. Knot intervals are adjusted to control the continuity of the joint. [1]	4
1.3	(a) a car model composed of 412 surfaces,(b) gaps between surfaces when rendering.	5
1.4	Constructive solid geometry representation and boundary representation [2]	5
2.1	Knot insertion on a curve (a) the original curve with polar values, (b) the new curve after inserting a knot.	15
2.2	T-mesh topology and the polar label identification of control points.	18
2.3	Ambiguous connections	19
2.4	Blending function refinement. Insert knot 5 into [0,2,4,6,8].	20
2.5	Reverse blending function refinement. Remove knot 5 from [0,2,4,5,6].	24
2.6	Knot removal (a) the original curve, (b)remove P_r from the curve.	25
2.7	Models with complex topology	27
2.8	Patching Catmull-Clark meshes [3]	28

2.9	Modeling using manifold T-splines [4]	30
2.10	Modeling using PHT-splines [5]	30
2.11	Folding in sewing [6]	33
2.12	CUDA memory model	37
2.13	CUDA architecture and its execution on GPU hardware	38
2.14	CUDA occupancy (a) occupancy with varying block sizes, (b) occupancy with varying register count per thread, (c) occupancy with varying shared memory sizes per block. (The device capability is 3.0 and total shared memory is 49152 bytes. This figure is drawn using the CUDA GPU Occupancy calculator provided by NVIDIA.)	39
3.1	An example: the two-bay extension is not sufficient for identifying the Bézier patch domains. (a) A T-spline model with knot lines mapped onto the surface; (b) the results of extending each T-junction by two-bays; (c) the local regions in the parameter domain which shows that the yellow rectangle should be split by the knot lines of control point A.	45
3.2	Bézier regions extraction: (a) an input T-mesh; (b) C^∞ domains; (c) Bézier patch domains.	45
3.3	Knot line partition (a) the knot vector of a control point, (b) its knot line grid, (c)&(d) one control point's partition	47
3.4	Irregular region partitioning: (a) an irregular region, (b) partitioning without concave lines, (c) partitioning with concave lines	48
3.5	Partitioning with intersecting concave lines.	48
3.6	Irregular region partitioning: (a) correct partition pattern; (b) incorrect partition pattern	49
3.7	Bézier patch extraction from a T-spline surface. (a) input T-spline surface, (b) the Bézier patch partitioning, (c) the enlargement of one Bézier patch. .	50

3.8	The pre-image (row for the s direction and column for the t direction) (a) for a NURBS surface; and (b) for a T-spline surface (No.1, No.2 and No.3 are on the same column; N-point No.4 is located on the same column as No.5 and the same row as No.6).	53
3.9	T-mesh grids (a) continuous regions of a T-mesh (b) T-mesh grids	55
3.10	Global memory allocation	56
3.11	The input and output of hash function	57
3.12	Performance on CPU and GPU	59
3.13	Complex T-mesh with regions contain more than sixty grids	59
3.14	Bézier patch extraction from a chair model: (a) the control mesh; (b) Bézier patches with some Bézier control points; (c) a close view of some Bézier control points	60
3.15	Bézier patch extraction from a sun face model: (a) the control mesh (b) Bézier patches with some Bézier control points; (c) a close view of some Bézier control points	61
3.16	Bézier patch extraction from a shark model: (a) the control mesh; (b) isocurves of the original model; (c) Bézier regions; (d) a close view of one Bézier patch	62
3.17	Bézier patch extraction from a dinosaur model: (a) the control mesh; (b) Bézier patches with some control points; (c) a close view of some Bézier control points	62
3.18	Bézier patch extraction from a submarine model: (a) the control mesh; (b) Bézier patches with some Bézier control points; (c) a close view of some Bézier control points	63
3.19	Bézier patch extraction from a car model: (a) the control mesh; (b) Bézier patches with some Bézier control points; (c) a close view of some Bézier control points	63

4.1	Crack caused by different tessellation densities	65
4.2	Overview of the B-rep model rendering pipeline on GPU.	67
4.3	Algorithm overview (a) original surfaces and their boundary curves, (b) boundary merge, (c) Bézier patch extraction, (d) Bézier curve extraction, (e) transition region identification and central region tessellation, (f) the final mesh after transition region creation.	68
4.4	Boundary pre-process of adjacent boundary curves. (The colors are used to indicate individual Bézier curves) (a) split the boundary curves into a set of Bézier curves according to the original knots, (b) produce the same set of Bézier curves for adjacent boundaries with curve refinement.	70
4.5	Bézier edge identification. (a) the original T-mesh, (b) the partitioned T-mesh(the circled point is a <i>new T-point</i>), (c) edges associated with some T-points (the $s+$ direction edge is in red; the $t+$ direction edge is in green).	74
4.6	Preparation for running in T-point parallel	75
4.7	Input of the hash functions (a) a control point (the red point) with its domain, (b) all T-points it covers (highlighted in green and red).	76
4.8	Output of the hash functions	76
4.9	Error between rational curve and its approximation line segment	78
4.10	Illustration of transition region creation	86
4.11	Illustration of the transition region creation with large shrink intervals. (a) the sum of opposite shrink interval > 1 and at least one shrink interval is < 1 , (b) both opposite shrink intervals > 1	86
4.12	(a) A car model, (b) cracks between several surfaces, (c) zoom in of those cracks, (d) the normal tessellation pattern, (e) the tessellation pattern using our algorithm, (f) our rendering effect.	91
4.13	(a) A submarine model, (b) cracks between several surfaces, (c) zoom in of those cracks, (d) the normal tessellation pattern, (e) the tessellation pattern using our algorithm, (f) our rendering effect.	92

5.1	The single-point removal procedure cannot remove inserted control points introduced by refinement: (a) part of the pre-image, (b) inserting one control point introduces three control points,(c-e) knot removal following the order 2,1,0, and (f-h)knot removal following the order 0,1,2.	95
5.2	(a) control points contribute to the newly inserted point No.0, (b) control points contribute to the residual when removing point No.0	96
5.3	(a) a point is not removable, (b) the point can be removed after another point is removed, (c) the final T-mesh	96
5.4	(a) a removable point,(b) removing another point from the T-mesh,(c)this point becomes not removable	97
5.5	The single-point insertion: (a) the point insertion process and the possible range of the insertion group, (b) inserting point No.0 introduce points No.1, 2.	101
5.6	A possible insertion pattern	101
5.7	Identify a removal group: (a) a point on an s -edge; and (b) a point on a t -edge	102
5.8	A T-mesh refinement by inserting four control points (No.0 point in each figure)	102
5.9	The flexibility of the multi-point removal method. (a)(b)refined surface and its pre-image (Figure 5.8(d)), (c)(d) the surface after perturbation, (e)(f) the single-point removal results and its pre-image, (g)(h) the multi-point removal results and its pre-image	103
5.10	Two intersecting domains (dash lines are from the pre-image).	108
5.11	The T-mesh regularity	109
5.12	Single-point removal vs. multi-point removal (a) original model with 1148 control points, (b) pre-image of the original model, (c)exact single-point removal results: 1057 control points, (d) exact multi-point removal results: 1028 control points (Some removal groups are circled out compared to single-point removal).	111

5.13	Single-point removal vs. multi-point removal (a) original model with 9170 control points, (b)after exact single-point removal: 8354 control points, (c)after exact multi-point removal: 8321 control points, (d)the pre-image of the front surface on the model, (e)pre-image of the surface after exact single-point removal, (f)pre-image of the surface after exact multi-point removal.	112
5.14	Exact multi-point removal (a) original model with 1969 control point,(b) multi-removal result with 1805 control points	113
5.15	Exact multi-point removal (a)original car model with 36317 control points . (b) multi-point removal result with 33172 control points (only car body is considered, not including the wheels)	114
5.16	Approximate model simplification (a) the original model with 1148 control points, (b) the simplified model with 1008 control points (error tolerance 0.001), (c) the simplified model with 982 control points (error tolerance 0.01), (d) the simplified model with 728 control points (error tolerance 0.1).	116
5.17	The same portion of the T-mesh after knot removal (a)the T-spline surface, (b) $(\alpha, \beta)=(1, 0)$, (c) $(\alpha, \beta)=(0.5, 0.5)$, and (d) $(\alpha, \beta)=(0, 1)$	116
6.1	Designing models with complicate boundaries (a) using a set of NURBS surfaces, (b) using one complex boundary T-spline surface, (c) one possible pre-image of the T-spline surface.	118
6.2	Domain boundary loop and mesh boundary loop.	123
6.3	Extension boundary selection(a) a T-spline surface and a target curve, (b)legal iso-parameter boundaries (c)illegal selected boundaries	125

6.4	T-mesh extension with violation points in the shaded area. (a) the original T-mesh with knot vector $[-0.2, 0, 0.2, 0.22, 0.29, 0.34, 0.4, 0.6, 1.2]$ in t direction and its virtual boundaries, (b) the extended T-mesh with knot vector $[-0.2, 0, 0.2, 0.22, 0.29, 0.34, 0.4, 0.6, 0.6, 0.6]$ in t direction and its virtual boundaries, (c) the extend T-mesh with knot vector $[-0.2, 0, 0.2, 0.22, 0.29, 0.34, 0.4, 0.7, 0.7, 0.7]$ in t direction and its virtual boundaries. .	127
6.5	T-spline surface extension results of Figure 6.4 (a) the original T-spline surface corresponding to Figure 6.4(a), and a target curve, (b) the extended surface corresponding to Figure 6.4(b), (c) the extended surface corresponding to Figure 6.4(c). (d)(e)(f)corresponding index graphes of (a)(b)(c).	129
6.6	Trimming of T-spline surfaces. (a) trimming loop(red lines), (b) refine the mesh to obtain the domain boundary (red points), (c) refine to get mesh boundary (blue lines and red points), (d) remove the trimmed points	130
6.7	A T-spline surface extension example. (a) the original T-mesh and the extended T-mesh, (b) the original surface and the extended surface, (c) separate the original surface and the extended surface.	133
6.8	A T-spline surface extension. (a) the original surface and four target curves, (b) the extended surface,(c) its pre-image	134
6.9	T-spline surface extension that needs the shrink of a mesh boundary. (a) the original T-mesh, (b) surface extension along s direction, (c) shrink some boundary control points, (d) insert some necessary control points, (e) surface extension along t direction.	135
6.10	T-spline surface extension from a hole	136
6.11	Propeller model design using Complex Boundary T-splines.	136

Chapter 1

Introduction

1.1 Background and motivation

In computer aided geometric design (CAGD) and solid modeling, free-form surfaces play an important role. Spline surfaces are one of major representations for various shapes in automotive, aerospace and other industries. In particular, Non-Uniform Rational B-splines (NURBS) are defined using a set of control points which are topologically arranged in a rectangular grid as shown in Figure 1.1(c) and intuitively control the shape of the surfaces. There are many efficient and stable algorithms for NURBS. Examples include knot insertion/removal, degree elevation/reduction and stable recursive evaluation. All these nice properties make NURBS an industry standard in computer aided design and engineering (CAD/CAE).

However, NURBS has some limitations. For example, after one control point is inserted in a NURBS model, a whole row or column of control points will be inserted to maintain the grid topology [7]. Therefore, NURBS models often contain superfluous points which do not substantially contribute to the geometry. In interactive design, the designer would like to concentrate on local areas of interest at one time. In engineering computation such as iso-geometric analysis [8], the local refinement will lead to smaller linear systems to find the solution [9]. As shown in Figure 1.1(a) and Figure 1.1(b) two control nets define

the same surface, whereas the NURBS control grid has 3328 more points than the T-spline surface. Moreover, the domain of NURBS surfaces is constrained in rectangular. As a result, it is inconvenient to represent complicated shapes using a single NURBS surface.

The research of new geometry schemes leveraging the power of NURBS but with more flexibility is highly of interest. T-splines [10, 11] are a free-form geometric shape technology that solves many limitations inherent in NURBS. Some advantages that T-splines have over NURBS include local refinement, local coarsening and providing a solution to the gap problem. The domain of the T-splines surface is also restricted to a rectangular grid formed by two global knot vectors. However, not every intersection is associated with a control point. Partial row and partial column are legitimate in a T-mesh, which gives rise to T-junctions, L-junctions or even I-junctions. A single knot can be inserted into or removed from the mesh while preserving the surface shape. T-splines are forward and backward compatible with NURBS. In Figure 1.1(b), a T-mesh is obtained by simplifying a NURBS surface in Figure 1.1(a). Purple points indicate T-junctions or L-junctions. Compared to the NURBS surface, the T-spline surface requires much fewer number of control points. In general, two patches of mis-matched parameterizations trimmed NURBS patches cannot exactly join together theoretically. T-splines provide a feasible solution to the gap problem [1]. A gap between the teapot body and spout is shown in Figure 1.2(a). The resulting seamless teapot is shown in Figure 1.2(b). It is reported that using T-splines instead of NURBS can reduce modeling time from couple of days to several hours. In this thesis, our research focuses on T-splines and the developed algorithms can be applied to NURBS surface as well.

In practice, a single surface may be incapable of constructing complex models. It is more prevalent to represent the models by multiple surfaces. In recent years, the design is becoming increasingly complex and elaborated. For example, a typical automobile consists of about 3,000 parts, a fighter jet over 30,000, and a modern nuclear submarine over 1,000,000 parts [9]. When operations are applied to compound models, each surface element is processed independently, during which continuity and smoothness should be main-

Figure 1.1: A model represented by NURBS and T-splines (T-junctions are highlighted with purple.) (a) the NURBS surface with 4422 control points, (b) the T-spline surface with 1094 control points, (c) the NURBS topology, and (d) the T-mesh topology. (For all the figures of T-mesh topology in the thesis, row is along the s direction and column is along the t direction if not mentioned.)

tained. It is difficult to maintain smoothness at the seams of the patchwork [12, 3], let alone as the model is animated [13]. Even an initial model is defined exactly seamlessly in geometry, different tessellation densities may also lead to gaps between surfaces [14, 15]. Gaps may cause serious problems to interoperability between CAD, CAM, CAE. In manufacture, filling these gaps requires several days and billions of dollars, which decelerates the product development cycles. It will also cause problems in analyzing physical properties, such as stress and strain, volume, heat transfer. Therefore, for the sake of aesthetic or numerical requirements, a seamless and smooth model is always pursued in industrial design or physical simulation.

There are different schemes [2] for representing compound models. Two popular ones

Figure 1.2: (a) trimmed NURBS representation with a gap between teapot body and spout, (b) gap free teapot created by T-spline. Knot intervals are adjusted to control the continuity of the joint. [1]

are Constructive Solid Geometry (CSG) and Boundary Representation (B-rep) (Figure 1.4). CSG represents objects as trees of primitives. Each primitive is defined by a set of surfaces. The CSG representation associates with three basic Boolean operations: add, subtract and intersect. All other operations are taken as a combination of these Boolean operations. B-rep has more flexibility. It is represented by a set of topology related surfaces. Vertices, edges and faces are the three basic attributes of each element, while loops, face groups and shells indicate the topology information among surfaces. The topological information of the surfaces can be retrieved more easily from boundary representations.

Great efforts have been devoted to handle gaps between faces on B-rep models. Some of existing algorithms cannot achieve interactive performances during the manipulation [16, 17, 18, 6, 19]. Recently, a host of works has enjoyed a great success by utilizing the power of Graphics Processing Unit (GPU). For example, the gap-free rendering of NURBS or T-spline surfaces can be done using either tessellation [20, 15] or ray-casting [21, 22, 23]. For ray-casting, its performance is limited by numerical computations. For tessellation, most

Figure 1.3: (a) a car model composed of 412 surfaces,(b) gaps between surfaces when rendering.

Figure 1.4: Constructive solid geometry representation and boundary representation [2]

of previous works decompose the surface into a set of rational Bézier patches and then tessellate them simultaneously. For NURBS surfaces the Bézier extraction is quite straightforward, which can be done by following a regular pattern. However, for T-spline surface, the extraction is much more complicated. Previously, the domains of Bézier patches are obtained by all the T-mesh edges and the extended T-junctions by two bays. Unfortunately, this does not accurately represent all the Bézier patches on the surface. Moreover, special attention should be paid to the cracks between Bézier patches. Many prior works fill the gaps in the screen space [15] or draw some extra geometries along the gap [20] to achieve

interactive performances. This is sufficient for rendering purposes but not for numerical analysis or physical simulation. Another problem is that, because of the approximation nature of tessellation, even if the model is stitched to be watertight, it is hard to guarantee the approximation is within the given tolerance, especially along boundaries.

In summary, we can see that there are still a lot of problems for T-splines and especially compound T-spline models, which need further exploration.

1.2 Objectives and contributions

Our research focuses on geometric processing of compound T-spline surface models. A compound T-spline surface model is usually formed by a collection of topologically unrelated T-spline surfaces specified in a domain of interest. It is more suitable for representing complicated shapes which are often seen in practice. Geometric processing refers to the theory and algorithms for analyzing and manipulating geometric objects. We will investigate four geometric processes of compound T-spline surfaces: rational Bézier extraction from T-splines, adaptive tessellation of compound T-spline models, T-spline knot removal, and T-spline surface extension. These are very fundamental processes and can benefit many T-spline applications. We aim to gain deep understanding of these geometric processes and develop effective and efficient algorithms for them. Specific objectives are to make these algorithms handle compound T-spline models **accurately**, **interactively** and **seamlessly**.

We conduct research by first examining existing techniques for these geometric processes on compound T-spline surfaces, identifying technical challenges and problems and then developing new solutions. Due to the nature of compound T-spline surface models, we pay attention to the seams of the adjacent surfaces in the compound surface models. Moreover, as an emerging technology, GPU provides powerful computing capability. In our research, we are devoted to making our geometric processing parallel and GPU-friendly to enhance the performance if possible. The descriptions of the four geometric processes in our research and our contributions are elaborated below.

Extraction of Rational Bézier Patches from T-spline Surfaces on GPU: The first geometric process to be addressed is to convert T-spline surfaces into a set of rational Bézier patches, which we call Bézier patch extraction. Since NURBS and T-spline surfaces are composed of rational Bézier patches and rational Bézier patches have the same topological structure, this conversion is very useful in practice. For example, it is used for parallel processing [14, 24, 15, 25], narrowing down the domain of computation [26, 27] and facilitating numerical analysis [28]. Compared to NURBS, Bézier extraction from T-spline surfaces is much more complicated.

The complexity of the problem lies in the determination of Bézier patch domains. Previous methods extend each T-junction by two bays and consider the regions bounded by the edges of the extended T-mesh to be the Bézier patch domains. We observe that this is not correct sometimes. Moreover, the partition of the regions by knot lines may result in non-rectangular zones, which need to be further partitioned. There are many ways to partition a non-rectangular region into a set of rectangles. How to optimize the partitioning is a nontrivial task. Due to the flexibility of T-mesh structure and complexity of the determination process, it is difficult to implement the Bézier extraction on GPU. Previous methods perform the extraction on CPU, which requires quite high time cost [20, 15].

We present an algorithm to extract a minimal number of Bézier patches from T-spline surfaces on GPU. The algorithm consists of three technical components: C^∞ domain construction, Bézier domain identification, and Bézier patch computation. The C^∞ domain construction runs in parallel in control points. The Bézier domains are determined by partitioning non-rectangular C^∞ domains into rectangular grids, which is designed to perform in parallel in grids. The Bézier representation is computed in parallel in Bézier domains. The algorithm is designed to ensure that it already works correctly and the number of the extracted Bézier patches is minimized. Meanwhile the algorithm makes use of GPU to achieve real-time extraction. Moreover, we also design a compact GPU-suitable T-spline representation which decreases the time consumed in the CPU-GPU data transfer. This representation is also useful for other geometric processes.

GPU-friendly Crack-free Adaptive Tessellation of Compound T-spline Models:

The second geometric process to be addressed is to tessellate T-spline surface models into a set of triangles forming a seamless triangular mesh. This is useful for rendering T-spline models. Moreover, many game engines do not accept spline models and polygonal meshes as a common representation in game applications. The tessellation provides a tool to generate game objects from existing spline models.

Tessellating compound spline models may produce cracks. One reason of the generation of gaps is due to different sampling rates for two adjacent surfaces with different parameterizations. To handle the potential cracks between surfaces, one common approach is to over sample the shared boundaries [25, 14] or cells [29] and let the adjacent surfaces share the same sampling rate. For rendering purpose, some methods are proposed to fill the gaps in the screen spaces [15] or just to create additional geometry along the boundaries [30, 20].

We propose a CUDA based parallel algorithm for generating seamless meshes from compound T-spline surfaces. In our algorithm, there is no need to over sample the shared boundaries. Both shared boundaries and interior Bézier patches are taken as independent primitives and their tessellation factors are computed according to their own geometric variation. To deal with the gaps between adjacent surfaces, we first make the boundary curves be consistent in both parameters and geometries, so that they can be processed in a way similar to handle interior patches. We improve the tessellation factor estimation so that fewer sample points will be needed under the same error tolerance, which benefits the computational cost. Our method can assure that the tessellated mesh approximates the input model within the given tolerance and the process can run in an interactive rate.

Parallel T-spline Multi-point Removal: The third geometric process to be addressed is T-spline knot removal. In NURBS, the knot insertion and the knot removal are two fundamental algorithms that can be used as tools for understanding and analyzing B-splines and also as practical tools for manipulating B-spline curves and surfaces [31]. Particularly, the knot removal is suitable for data compressions. This is applicable to T-splines. Compared to NURBS knot removal, the knot removal for T-spline is much more challenging.

We carefully examine the complexity and typical patterns of T-spline knot removal. We observe that a group of points may be inserted after the insertion of one point. Thus we propose the T-spline multi-point removal strategy. The algorithm first identifies removal groups in the mesh, and then removes all the points in a group simultaneously. This approach has two advantages: it can generally remove more control points than a single-point removal algorithm and the process can be completed in fewer iterations. We also implement the algorithm in a parallel structure on GPU. Each thread is first assigned to estimate the residual for all single points and small removal groups. After eliminating all the removable candidates, each thread deals with one control point to resolve the blending function violations. The whole removal process terminates until no more control points can be removed.

Construction of T-spline Surfaces with Complicated Boundaries: The fourth geometric process to be addressed is T-spline surface extension, which allows the user to create new models from existing T-spline models by the extension process and provides a simple way to create compound T-spline surface models with complicated boundaries or holes. The process also makes use of the flexibility of T-spline representation and T-spline surfaces with complicated boundaries provide a single representation for complicated shapes.

We propose a scheme to represent T-splines with complicated boundaries and develop an approach which allows the user to perform extension and trimming. The extension will keep the original surface unchanged and create a new part of the surface which smoothly connects the target curve specified by the user to a curve (usually a part of the boundary) on the surface. The trimming can remove parts of the surface to create a T-spline surface with holes. The splitting is used to convert an interior curve on the surface to one used as the boundary for extension. The combination of these manipulations provides a simple tool to create seamless compound T-spline surfaces. Compared to NURBS, T-spline surface extension is more flexible and can be performed from a partial boundary of the surface. Hence, in some cases, new knot insertion is necessary to preserve the shape and parametrization of the original surface. Similarly, in the T-spline surface trimming, the refinement is required along the trimming curve to maintain the remaining surface shape.

After the extension, some of the blending functions may not match the new T-mesh. If such mismatching is caused by the boundary parameter updating, new control points should be derived and added into the mesh.

In the first three works we used GPU for fast processing. It is not easy to design parallel algorithms for T-splines who often have complex mesh topology. Our basis idea is to use the basic element of T-mesh control points as the parallel unit. That is what we did in most stages of the GPU implementations. Then the performance can be significantly improved compared to CPU implementation which usually processes all the control points/Bezier patches sequentially. In all the three works, the CPU algorithms take orders of magnitude longer than our GPU algorithms. On CPU, it always takes several or even tens of minutes to complete the whole process for a surface with thousands of control points. On GPU, only several seconds are needed.

1.3 Organization of the thesis

The remainder of the thesis is organized as follows:

- Chapter 2 briefly describes some background knowledge to make the thesis self-contained and reviews previous work that is related to the research described in the thesis.
- Chapter 3 considers the problem of rational Bézier extraction from T-spline surfaces and presents a GPU-based extraction algorithm.
- Chapter 4 proposes a GPU-friendly framework for crack-free, adaptive tessellation of compound T-spline models.
- Chapter 5 studies the T-spline knot removal problem and presents a parallel T-spline multi-point removal algorithm.
- Chapter 6 considers how to extend T-spline surfaces from the partial boundaries and

develops efficient techniques for T-spline extension which results in T-splines with complicated boundaries.

- Chapter 7 summarizes the research work that has been described in the thesis and also outlines a few related topics for future study.

Chapter 2

Literature Review

This chapter gives some background knowledge to make the thesis be self-contained and reviews some geometry modeling technologies related to our research.

2.1 Fundamental processing of NURBS

2.1.1 Foundations

In computer aided design, NURBS is an industry standard. It defines shapes by means of control points. Its expression is piecewise polynomial or piecewise rational function which gives a nice description of the shapes. Compared to polygon meshes and subdivision surfaces, NURBS [7, 32] has advantages in representing compact high-order surfaces.

Let us start with non-rational B-spline curves. Given degree k , $n + 1$ control points forming a control polygon, and a knot vector $T = \{s_0, \dots, s_{n+k+1}\}$, where $s_0 \leq s_1 \leq \dots \leq s_{n+k+1}$ are the knots. A B-spline curve is a piecewise polynomial curve defined within $[s_k, s_{n+1}]$. The equation of the B-spline curve of degree k is

$$C(s) = \sum_{i=0}^n P_i N_i^k(s) \quad (2.1)$$

where $N_i^k(s)$ is the B-spline basis function. The basis functions are piecewise polynomials.

The B-spline basis functions of degree k are non-zero only over $k + 1$ intervals of the knot vector. They are defined recursively:

$$N_i^0(s) = \begin{cases} 1, s \in [s_i, s_{i+1}) \\ 0, otherwise \end{cases} \quad (2.2)$$

$$N_i^k(s) = \frac{s - s_i}{s_{i+k} - s_i} N_i^{k-1}(s) + \frac{s_{i+k+1} - s}{s_{i+k+1} - s_{i+1}} N_{i+1}^{k-1}(s) \quad (2.3)$$

The basis function enables B-spline curves to have excellent locality. Specifically, changing one control point affects at most $k + 1$ knot segments. Meanwhile, B-spline curves possess many nice properties such as the affine invariance, convex hull and variation diminishing. Moreover, B-spline curves are C^{k-h} continuous at a knot with multiplicity h .

NURBS is a generation of B-spline curves and surfaces by associating a weight to each control point. It inherits all advantages of B-splines while extending the liberty of modeling. It provides precise representation for lines, planes, free-form curves/surfaces and quadratic curves/surfaces such as circles and spheres. There exist many fast and numerically stable algorithms to handle NURBS objects such as de Boor algorithms, degree elevation/reduction algorithms, knot insertion and knot removal algorithms. A NURBS curve is defined as

$$C(s) = \frac{\sum_{i=0}^n P_i w_i N_i^k(s)}{\sum_{i=0}^n w_i N_i^k(s)} \quad (2.4)$$

or using the homogenous representation

$$C(s) = \sum_{i=0}^n (P_i w_i, w_i) N_i^k(s) \quad (2.5)$$

A NURBS surface can be expressed as,

$$S(s, t) = \frac{\sum_{i=0}^m \sum_{j=0}^n P_{ij} w_{ij} N_i^k(s) N_j^l(t)}{\sum_{i=0}^m \sum_{j=0}^n w_{ij} N_i^k(s) N_j^l(t)} \quad (2.6)$$

or using the homogenous representation,

$$Q(s, t) = \sum_{i=0}^m \sum_{j=0}^n (P_{ij} w_{ij}, w_{ij}) N_i^k(s) N_j^l(t) \quad (2.7)$$

where $N_i^k(s)$, $N_j^l(t)$ are the basis functions associated with the point P_{ij} . w_{ij} can be used to further control the shape.

2.1.2 Knot insertion

In some applications, the knot insertion is desirable. For example, it can be used to increase the degrees of freedom for users to do local modifications or do curve/surface subdivision.

Let $C(s) = \sum_{i=0}^n P_i N_i^k(s)$ be a NURBS curve defined over the knot vector $T = \{s_0, \dots, s_{n+k+1}\}$, where P_i are control points in homogenous representation. We insert a knot s between the two knots s_x and s_{x+1} . Then the new corresponding curve defined on knot vector $T = \{s_0, \dots, s_x, s, s_{x+1}, \dots, s_{n+k+1}\}$ can be represented as

$$\bar{C}(s) = \sum_{i=0}^{n+1} P'_i \bar{N}_i^k(s) \quad (2.8)$$

Our goal is to compute the new positions of related control points to guarantee that $C(s)$ and $\bar{C}(s)$ define the same curve. This refers to the process of knot insertion, introducing new knots without changing the shape of the curve, both geometrically and parametrically.

There are a few efficient and stable knot insertion methods to compute the new positions of related control points such as Oslo algorithm [33] and Boehm's knot insertion algorithm [34]. Similar results can be derived using polar form [35]. Polar form is a labeling

scheme for the control points of B-splines, developed by Dr. L Ramshaw. Its underlying theory is based on symmetric polynomials and a technique called blossoming. In polar form, control points are labeled by polar values. On a cubic B-spline curve, each control point is labeled by 3 adjacent knots (as shown in Figure 2.1). It is defined on the knot vector $[s_{-2}, s_{-1}, s_0, s_1, s_2, s_3, s_4, s_5, s_6]$. After inserting a knot s between s_1 and s_2 , only the polar values of P_1 and P_2 are changed. Besides, a new point is added. Therefore it is needed to replace two old control points with three new ones.

Figure 2.1: Knot insertion on a curve (a) the original curve with polar values, (b) the new curve after inserting a knot.

Applying the symmetric and affine combination properties of the polar values, we can easily deduce the positions of the new control points as:

$$P'_1 = \frac{s_2 - s}{s_2 - s_{-1}} P_0 + \frac{s - s_{-1}}{s_2 - s_{-1}} P_1 \quad (2.9)$$

$$P_{new} = \frac{s_3 - s}{s_3 - s_0} P_1 + \frac{s - s_0}{s_3 - s_0} P_2 \quad (2.10)$$

$$P'_2 = \frac{s_4 - s}{s_4 - s_1} P_2 + \frac{s - s_1}{s_4 - s_1} P_3 \quad (2.11)$$

For a NURBS surface, inserting a knot will cause a whole row or column of control points to be added to keep the grid topology of the NURBS surface.

2.1.3 Knot removal

Knot removal is the reverse process of knot insertion. Let $C(s) = \sum_{i=0}^n P_i N_i^k(s)$ be defined on the knot vector $T = \{s_0, \dots, s_{n+k+1}\}$, s_r is an interior knot of multiplicity k . A new curve is obtained by removing s_r x ($1 \leq x \leq k$) times:

$$\bar{C}(s) = \sum_{i=0}^{n-x} P'_i \bar{N}_i^k(s) \quad (2.12)$$

If it is possible that $\bar{C}(s)$ and $C(s)$ geometrically and parametrically represent the same curve, we say that s_r is x times removable. Two processes are involved in knot removal. Firstly, we should determine whether a knot is removable and how many times it can be removed. Then, the removable knot from the knot list is removed and the control points should be updated [7]. For a NURBS surface, since all the rows or columns should have the same number of control points, the removal of a single control point is not allowed. Removing a knot is feasible only if the whole row/column at this knot can be removed.

While knot insertion is always possible, in general knot removal procedure cannot be carried out without changing the shape of the spline curve. The only exception occurs if a knot has been inserted before [36]. Another method [37] to describe the knot removal is that a knot of a B-spline curve is removable if the order of continuity at the knot is higher than the order it should have according to the multiplicity. All the above criteria are rigorous, which necessitates the use of approximation methods [36, 38, 37, 39].

The approximation strategy developed by Lyche and Morken [38] removes more than one knot together. There are three parts in this knot removal process: Rank, Remove and Approximation. The algorithm firstly assigns each knot a meaningful weight which indicates its significance in representing the curve. It is defined as the distance (measured by some norm) between the original curve and the curve obtained by removing this knot s_r

$$w_r = \text{dist}(f_T, g_{T \setminus s_r}) \quad (2.13)$$

where T is the knot vector $T = \{s_0, \dots, s_d\}$, s_r is the removed knot, f_T is the original curve and $g_{T \setminus s_r}$ is the new curve without s_r . The rank is taken as a reference of the removing order. Then a set of knots with small weights are selected as the candidates for removal. Removing all candidates together may make the error exceed the given tolerance. The algorithm ranks all candidates according to their weights and heuristically decides a subset of the candidates for removal using a binary search. The last step is approximation. After the knots removal, the resulting curve needs to be perturbed to approximate the original curve better. Suppose the knot vector τ is a subset of T , the approximation curve g to original curve f from space $S_{k,\tau}$ to $S_{k,T}$ is given as the unique solution of the following least square problem:

$$\min_{g \in S_{k,\tau}} \|f - g\|_{l^2, T} \quad (2.14)$$

2.2 Basic T-spline techniques

T-splines [10, 11] is a generation of NURBS, overcoming many deficiencies of NURBS. It excels in supporting the local refinement, surface merging [1], surface fitting [40, 41] and surface simplification [42]. Both NURBS and T-splines are based on control grids established by two knot vectors in the two parameter directions. On the surface each control point associates with a tensor-product B-spline blending function. All the control points of the NURBS surfaces traverse the entire control grid, whereas in T-splines, only partial control grids are associated with control points. Therefore, for a NURBS surface, the control points in a column have the same blending function $N_i(s)$, whereas in T-splines the control points have no regular relationship with each other. Different from grid based splines, this property gives rise to the notion of point-based splines.

2.2.1 Foundations

The equation of a T-spline surfaces is given by,

$$P(s, t) = \frac{\sum_{i=0}^n P_i B_i(s, t)}{\sum_{i=0}^n B_i(s, t)} \quad (2.15)$$

where P_i are control points and $B_i(s, t)$ are the corresponding blending functions:

$$B_i(s, t) = N_i^3(s) N_i^3(t) \quad (2.16)$$

Unlike NURBS, the knots corresponding to $N_i(s)$ and $N_i(t)$ cannot be inferred from the two global knot vectors. They depend on the local topology of the T-mesh.

Figure 2.2: T-mesh topology and the polar label identification of control points.

A T-mesh consists of rectangles. It allows T-junctions and L-junctions. One edge of a face may contain several segments as shown in Figure 2.2. The left edge of face F contains 2 intervals. Each edge in the T-mesh is labeled with a knot interval. To regularize a T-mesh, there are two constraints. First, the sum of knot intervals on opposing edges of a face must be equal. For face F in Figure 2.2, the sum of the knot intervals of its left two segments should be equal to the interval of its right edge. Second, in case of multiple knots, ambiguity may occur on identifying polar label. Take Figure 2.3 as an example. The three

interior rows of the T-mesh are with the same parameter in the vertical direction. Which point P is connected to determines the polar label of P , being $[0,1,2,3,5]$ or $[0,1,2,3,6]$.

Figure 2.3: Ambiguous connections

To identify the polar labels (knot vectors) of a control point, we use the notions introduced in [43]. For a control point P_i located at knots (s_i, t_i) , define an sK-point to be any $(i, j) \in \mathbb{Z}^2$ that contains a control point or a vertical edge. Similarly, a tK-point refers to the one containing a control point or a horizontal edge. Both sK-point and tK point are used to form polar labels, which we call *polar point*. Draw a line across the control points along the two parameter directions. Take the left nearest two sK points and the right nearest two sK points to form the s knot vector. Take the top nearest two tK points and the bottom two tK points to form the t knot vector. If there are insufficient sK or tK points, at most two virtual boundaries are supplemented around the T-mesh as shown in Figure 2.2. The virtual knots are important to the peripheral points.

T-splines support many valuable operations. The two most frequently used ones are control point insertion and control point removal which are described below.

2.2.2 Local refinement: control point insertion

Compared to NURBS, T-splines can be locally refined. Inserting (or removing) a control point into (or from) a NURBS surface causes insertion (or removal) of control points along a whole row or column. In contrast, inserting (or removing) one control point into (or from) a T-spline surface in general causes insertion (or removal) of one or a few control points. This is the reason that we sometimes call such operations *T-spline (control) point insertion or removal* to emphasize the local behavior. The T-spline point insertion and removal can

be achieved by two basic transformations: blending function refinement [11] and reverse blending function transformation [42].

In the T-spline control point insertion, the blending function refinement is one of the atomic operations. It is originally invented for the B-spline knot insertion. The general idea is splitting an existing blending function into the linear combination of two new blending functions which is exactly equal to the original. The idea can be expressed as

$$B_i(s) = c_1 B_{i1}(s) + c_2 B_{i2}(s) \quad (2.17)$$

Figure 2.4: Blending function refinement. Insert knot 5 into [0,2,4,6,8].

From Figure 2.4 we can see that one of the scaled blending functions is centered at its original position; the other's center is slightly different but all of them contain the new knot. The values of the coefficients depend on the knot intervals of the original blending function and the inserted knot. If $\mathbf{s}=[s_0, s_1, s_2, s_3, s_4]$ is the initial polar label of a point located at knot s_2 . It is split only if new parameters are introduced into the four knot intervals. Given two coefficients $c = (k - s_0)/(s_3 - s_0)$, $d = (s_4 - k)/(s_4 - s_1)$, all the situations for a cubic B-spline blending function refinement are listed below. Insert parameter in the first

interval $[s_0, k, s_1, s_2, s_3, s_4]$

$$N_{[s]}(s) = cN_{[s_0, k, s_1, s_2, s_3]}(s) + N_{[k, s_1, s_2, s_3, s_4]}(s) \quad (2.18)$$

Insert parameter in the second interval $[s_0, s_1, k, s_2, s_3, s_4]$

$$N_{[s]}(s) = cN_{[s_0, s_1, k, s_2, s_3]}(s) + dN_{[s_1, k, s_2, s_3, s_4]}(s) \quad (2.19)$$

Insert parameter in the third interval $[s_0, s_1, s_2, k, s_3, s_4]$

$$N_{[s]}(s) = cN_{[s_0, s_1, s_2, k, s_3]}(s) + dN_{[s_1, s_2, k, s_3, s_4]}(s) \quad (2.20)$$

Insert parameter in the fourth interval $[s_0, s_1, s_2, s_3, k, s_4]$

$$N_{[s]}(s) = N_{[s_0, s_1, s_2, s_3, k]}(s) + dN_{[s_1, s_2, s_3, k, s_4]}(s) \quad (2.21)$$

The control point insertion introduces inconsistency between the new T-mesh and the existing blending functions. For curves, the refinement is performed in one parameter direction. For surfaces, we take the tensor product blending function of each control point as a whole. Refinement process is still performed in one direction at a time. The blending function in the other direction does not change. Although the NURBS knot insertion is performed on a two-dimensional parameter domain, all the refinement processes can simply be resolved in one parameter direction. This is attributed to the rigid grid topology of NURBS surfaces. The control point insertion for T-spline is different. Starting from one discrepant blending function, resolve the violation of one parameter direction, the violation may occur in the other direction. After several decompositions, one tensor product blending function may decompose into the sum of several scaled tensor product blending functions. It is important to note that some blending functions cannot match the mesh and further refinement will not work definitely. In this situation, one or more additional control point insertions are needed.

In implementation, some details should be treated with care. Following is a detailed flow of the T-spline local refinement algorithm.

This algorithm is guaranteed to terminate, because in the process of resolving violations, both the blending function refinements and the control points insertion do not introduce new knot into the mesh. They are performed along the existing control grid. The worst case is extending the T-mesh to a NURBS grid so all the blending functions and control points are matched.

2.2.3 Local simplification: control point removal

Like the control point insertion, the essential of control point removal is also to resolve the violations between the new T-mesh and the existing blending functions after eliminating a control point from the mesh, while maintaining the shape of the surface. A reserve blending function transformation process was presented by Wang and Zheng [42]. It handles the inconsistent situation where a blending function contains an extra knot which is not in the mesh. In their method, the idea can be expressed as

$$B_{i1}(s) = \frac{1}{c_1}B_i(s) - \frac{c_2}{c_1}B_{i2}(s) \quad (2.22)$$

We can see that the decomposition comes to two blending functions. One has the same center as original but misses the removed knot; the other one is centered at the missed knot but associated with a negative coefficient, which we call the negative term residual.

Take the polar label of one control point $\mathbf{s} = [s_0, s_1, s_2, s_3, s_4]$ for example. According to which knot is removed from the polar label, the reverse blending function transformation falls into four situations.

Remove s_0 from $\mathbf{s}$

$$N_{[\mathbf{s}]}(s) = N_{[s_{-1}, s_1, s_2, s_3, s_4]}(s) + \frac{s_{-1} - s_0}{s_3 - s_{-1}}N_{[s_1, s_0, s_1, s_2, s_3]}(s) \quad (2.23)$$

Algorithm 1 T-spline local refinement

Input: A T-spline mesh with control points P_i , weights w_i and knots list S and T. A single control point with knot value (s_{in}, t_{in}) will be inserted into the mesh.

Output: Returns the T-mesh after the control point being inserted. Some new edges will be added if necessary. The output T-mesh is equivalent to the input one.

- 1: Compute the polar labels for all the control points on the T-mesh before inserting (s_{in}, t_{in}) .
- 2: $P \leftarrow \{P_i\}$
- 3: $B \leftarrow \{(w_i, B_i)\}$
- 4: Find the position of (s_{in}, t_{in}) in the knot list
- 5: **if** s_{in} or t_{in} is not in the knot list **and** in the range of T-mesh **then**
- 6: Add new knot to the knots list
- 7: Change the type of the relative control points
- 8: **else if** s_{in}, t_{in} are all in the knot list and at (s_{in}, t_{in}) there is no existing control point **then**
- 9: Check along s-, s+, t-, t+ direction
- 10: **if** there is no new added edge or the new added edge doesn't pass other knots **then**
- 11: Only check the four directions and save the B_{old} to a base stack while $\{B_{new}\} \neq \{B_{old}\}$
- 12: **else if** the new added edge passes other knots **then**
- 13: Check all the control points and save the $\{B_{old}\}$ to a base stack while $\{B_{new}\} \neq \{B_{old}\}$
- 14: **end if**
- 15: **end if**
- 16: **for** all $\{(w_i, B_i)\} \subset B_s$ (base stack) **do**
- 17: **if** B_j matches current mesh **then**
- 18: Add the coefficients of B_j to the corresponding control point
- 19: **else if** B_j^{new} is more refined than B_j^{old} **then**
- 20: Add a knot to the old blending function using refinement: $w_j B_j = w_j(c_1 \tilde{B}_j + c_2 \tilde{B}_k)$
- 21: $B_s \leftarrow B_s - (w_j, B_j)$
- 22: $B_s \leftarrow B_s \cup (w_j \cdot c_2, \tilde{B}_k)$
- 23: $B_s \leftarrow B_s \cup (w_j \cdot c_1, \tilde{B}_j)$
- 24: **else if** B_j has a knot that is not in the current mesh **then**
- 25: Insert a vertex at B_j 's knot value into the T-mesh
- 26: **end if**
- 27: **end for**
- 28: **for** all P_j **do**
- 29: $P_j \leftarrow \sum_i w_j^i P_i$
- 30: $w_j \leftarrow \sum_i w_j^i$
- 31: **end for**

Figure 2.5: Reverse blending function refinement. Remove knot 5 from $[0,2,4,5,6]$.

Remove s_1 from $\mathbf{s}$

$$N_{[\mathbf{s}]}(s) = \frac{s_4 - s_1}{s_4 - s_0} N_{[s_{-1}, s_0, s_2, s_3, s_4]}(s) + \frac{(s_{-1} - s_1)(s_4 - s_1)}{(s_3 - s_{-1})(s_4 - s_0)} N_{[s_{-1}, s_0, s_1, s_2, s_3]}(s) \quad (2.24)$$

Remove s_3 from $\mathbf{s}$

$$N_{[\mathbf{s}]}(s) = \frac{s_4 - s_0}{s_3 - s_0} N_{[s_0, s_1, s_2, s_4, s_5]}(s) + \frac{(s_5 - s_3)(s_4 - s_0)}{(s_1 - s_5)(s_3 - s_0)} N_{[s_1, s_2, s_3, s_4, s_5]}(s) \quad (2.25)$$

Remove s_4 from $\mathbf{s}$

$$N_{[\mathbf{s}]}(s) = N_{[s_0, s_1, s_2, s_3, s_5]}(s) + \frac{s_5 - s_4}{s_1 - s_5} N_{[s_1, s_2, s_3, s_4, s_5]}(s) \quad (2.26)$$

An example illustrates how the control point removal process is performed on a curve (See Figure 2.6). If we remove the control point P_r from the curve, the control points whose blending function does not consistent with the new curve include: P_0, P_1, P_2, P_3 . We decompose their blending functions according to each particular case until they exactly match with the new curve. The original curve can be expressed as $C(s) = \sum_{i=0}^n P_i B_i(s, t)$; the new curve equation is $\bar{C}(s) = \sum_{i \neq r} P'_i B'_i(s, t) + P'_r B_r(s, t)$. If $C(s) = \bar{C}(s)$, the residual

Figure 2.6: Knot removal (a) the original curve, (b) remove P_r from the curve.

should be zero.

The control point removal process on a T-mesh can be described as follows. Given a T-mesh, first determine whether a knot is removable using the above criteria. If removable, remove it and its relevant edges from the mesh. Then compute the new control points to maintain the shape of the surface. The control point removal on a T-mesh may encounter more complicated situations than NURBS since we need to handle violations in two directions. Algorithm 2 elaborates T-spline local simplification process. Algorithm 3 shows the detail of the *Recheck* function in the control point removal process.

When removing edges from the T-mesh, for a control point we only remove the edge in the t direction if it is a valence-four point. This is why we check whether the blending functions in the base stack match the mesh in step 18. On the other hand, the control point insertion may change the blending function of the residual, so the following steps are necessary.

2.3 Compound models

To model real-world objects (especially bounded by closed surfaces) using NURBS surfaces or T-spline surfaces, a set of surfaces are often needed. Restricted to the planar parametric domain, it is impossible to model a complex surface with a single NURBS or T-spline surface. For this reason the topic of watertight surfaces has received a lot of

Algorithm 2 T-spline local simplification

Input: A T-spline mesh with control points P_i , weights w_i and knots list S and T. A single control point with knots value (s_{rmv}, t_{rmv}) will be removed from the mesh.

Output: Returns the T-mesh after the control point being removed. Some edges will be removed sometimes. The output T-mesh is equivalent to the input one.

- 1: Compute the polar labels for all the control points on the T-mesh before removing (s_{rmv}, t_{rmv}) .
- 2: $P \leftarrow \{P_i\}$
- 3: $B \leftarrow \{(w_i, B_i)\}$
- 4: Find the position of (s_{rmv}, t_{rmv}) in the knot list
- 5: **if** there is no control point at $(s_{rmv}$ or $t_{rmv})$ **then**
- 6: return
- 7: **else if** s_{rmv}, t_{rmv} are all in the knots list and at (s_{rmv}, t_{rmv}) there is an existing control point **then**
- 8: Remove Connections
- 9: **if** there is no edge removed or the removed edges do not pass other knots **then**
- 10: Only check the four directions and save the B_{old} to a base stack while $\{B_{new}\} \neq \{B_{old}\}$
- 11: **else if** the removed edges pass other knots **then**
- 12: Check all the control points and save the $\{B_{old}\}$ to a base stack while $\{B_{new}\} \neq \{B_{old}\}$
- 13: **end if**
- 14: **end if**
- 15: remove s_{rmv} or t_{rmv} from the knot list if they are not used anymore
- 16: **for** all $\{(w_j, B_j)\} \subset B_s$ (base stack) **do**
- 17: $Recheck^*$
- 18: **if** B_j contains but not centered at (s_{rmv}, t_{rmv}) **and** B_j does not match with the mesh **then**
- 19: do reverse transform: $w_j B_j = w_j(a_1 \tilde{B}_j + a_2 \tilde{B}_k)$
- 20: $B_s \leftarrow B_s - (w_j, B_j)$
- 21: $B_s \leftarrow B_s \cup (w_j \cdot a_2, \tilde{B}_k)$
- 22: $B_s \leftarrow B_s \cup (w_j \cdot a_1, \tilde{B}_j)$
- 23: **else if** B_j^{new} is more refined than B_j^{old} **then**
- 24: Add a knot to the old blending function using refinement: $w_j B_j = w_j(c_1 \tilde{B}_j + c_2 \tilde{B}_k)$
- 25: $B_s \leftarrow B_s - (w_j, B_j)$
- 26: $B_s \leftarrow B_s \cup (w_j \cdot c_2, \tilde{B}_k)$
- 27: $B_s \leftarrow B_s \cup (w_j \cdot c_1, \tilde{B}_j)$
- 28: **else if** B_j has a knot that is not in the current mesh **then**
- 29: Insert a vertex at B_j 's knot value into the T-mesh
- 30: **end if**
- 31: **end for**
- 32: **for** all P_j **do**
- 33: $P_j \leftarrow \sum_i w_j^i P_i$
- 34: $w_j \leftarrow \sum_i w_j^i$
- 35: **end for**

Algorithm 3 Recheck

```

1: if  $B_j$  matches the mesh then
2:   Add the coefficients of  $B_j$  to the corresponding control point
3: else if  $B_j$  is centered at  $(s_{rmv}, t_{rmv})$  then
4:   if  $B_j$  is the same with the residual base then
5:     Add to Residual
6:   else if The residual base is finer than  $B_j$  then
7:     Blending function refinement to  $B_j$ 
8:   else if  $B_j$  is finer than the residual base then
9:     Take  $B_j$  as the residual base
10:    Push the previous residual base to base stack
11:   end if
12: end if

```

Figure 2.7: Models with complex topology

attention recently.

The traditional method for creating watertight models is to smoothly piece many patches together. Enforcing a certain order of continuity between adjacent surfaces is not an easy problem. In the following section, some watertight model generation methods are reviewed.

2.3.1 Watertight models

Subdivision is a powerful paradigm for the generation of surfaces of arbitrary topology. It can produce a visually smooth surface estimated by the initial polygon mesh. Catmull-Clark subdivision is a generalization of the recursive bi-cubic B-spline subdivision algorithm [44], making it especially suitable for being approximated by B-spline surfaces.

One approach [3, 45, 46] is to produce smooth surfaces from Catmull-Clark subdivision surfaces. One previous work used Gregory patches [46] to approximate the Catmull-Clark

subdivision surfaces. In general, replacing the quads on the initial subdivision mesh can guarantee the smoothness except for edges containing extraordinary points. A set of Gregory patches are created, which are connected with user specified continuity across the boundaries. Gregory patches are a modified form of traditional tensor product and triangular polynomial patches, which were introduced by Gregory [47]. Gregory patches provide more degrees of freedom to control the corner derivatives, but fewer degree of freedom to approximate the CC surfaces. Peters [3] described an algorithm that converts Catmull-Clark surfaces into a set of NURBS surfaces. This method creates one bi-cubic NURBS surface for each face of a quad mesh. The whole algorithm includes three parts. Firstly, find all the initial quads on the subdivision mesh; then insert knots into the mesh to let the surface of each quad interpolate the boundaries; finally perturb the corner points to obtain the corner smoothing. The final surfaces are C^2 continuity almost everywhere except for the extraordinary points where only C^1 continuity is achieved. Loop [45] also provided a method to approximate the Catmull-Clark surface using bi-cubic Bézier patches. Their results are smooth everywhere except along edges containing extraordinary vertex where they are only C^0 . Additional tangent patches are also introduced to control the normal field around extraordinary points, thereby generating visually smooth surfaces. Though all above methods generate watertight NURBS or Bézier surfaces, great effort is spent on maintaining the continuities. However, none of them consider the local refinement of the surface.

Figure 2.8: Patching Catmull-Clark meshes [3]

Other approaches directly define watertight surfaces with arbitrary topology. To handle the unavoidable gaps between trimmed boundaries, Sederberg et al. [1] proposed a method using T-splines and non-uniform NURBS surfaces. The trimmed NURBS surfaces

are firstly converted to untrimmed T-Spline surfaces. When merging the two T-meshes, consistent knot intervals between patches are not necessary based on the definition of the non-uniform NURBS surface. The output is a single watertight T-spline model. However, they only focus on how to stitch two intersecting surfaces. Handling three or more intersecting surfaces is not explored. Song et al. [48] proposed a method for topologically consistent representation of trimmed free-form surfaces. The two trimming curves are on parametric domain. They perturbed the two surfaces so that the mapping of the two trimming curves on the surface are exactly the same. However, along the trimmed boundaries, it is only C^0 (approximated C^1) continuity. C^1 and C^2 are only possible if using relatively high degree of surfaces. Gu et al. [49] proposed manifold splines based on triangular B-splines and followed by the manifold T-splines [4]. They aimed to directly define continuous surfaces over arbitrary manifold surfaces. A manifold can be treated as a set of charts in R^2 . Each chart has its own parametrization. When the charts are glued to form a manifold, affine transformation functions are defined between different charts. However, there are singularities which cannot be covered by any chart. The manifold T-splines allow local refinement. Since T-splines are parametric affine invariant, the manifold theory can be applied to generalize the manifold domain automatically. In the construction algorithm, they generate a global parametrization for an existing polygonal model and then fit a coarse T-mesh according to the parameterization. After assigning knots and computing the initial control points of the coarse T-mesh, they locally refine the T-mesh where the fitting error is bigger than the user-specified tolerance.

Another potential tool to model local refinable surfaces with arbitrary topology is polynomial splines over hierarchical T-meshes [5]. Li, Deng and Chen constructed a single PHT-spline patch to approximate genus-zero meshes. In the algorithm, they first partition the meshes into several parts, fitting each part with a PHT-spline surface, and then stitching them into a single patch while maintaining a certain degree of continuity. However, it is needed to adjust some points to achieve the required continuity. They discussed how to stitch three patches together as well as how to handle the extraordinary point, which were

Figure 2.9: Modeling using manifold T-splines [4]

not figured out in watertight T-splines [1]. The future research includes how to construct polynomial spline surfaces over hierarchical T-splines to fit triangular meshes of arbitrary topology.

Figure 2.10: Modeling using PHT-splines [5]

2.3.2 Boundary representation of compound surfaces

In the solid modeling, the models contain not only the geometry information but also the topology information. There are two popular representations for solid modeling: Constructive Solid Geometry (CSG) and Boundary Representation (B-rep). In our work, all the models use the Boundary Representation. The boundary representation can be considered as an extension to the winged-edge data structure [50]. The winged-edge data structure was

initially used for representing polygonal meshes. A polygonal model is defined by a set of vertices, edges and faces. In the winged-edge data structure, all the attributes are stored in a well-organized order to record the topology information of the model. In particular, each edge has two vertices and belongs to exactly two faces. Each face consists of a list of edges. The edges of a face are consistently oriented. It provides three tables: edge table, face table and vertex table. In the edge table, the information includes the start and end vertex indices, its two adjacent faces, and four additional edges. The two adjacent faces are labeled as right and left when traveling from the starting vertex to the ending vertex. The four adjacent edges are also denoted by the next clockwise edge, the previous clockwise edge, next counterclockwise edge and the previous counterclockwise edge. In the vertex table, its coordinates and one of its incident edges are stored. The face table keeps one of the surrounding edges of each face. To handle the holes in models, Braid [51] extended the winged-edge data structure to represent solid with holes. In his method, a face is represented using several loops. The loop can be parametric curves. They are oriented in the direction following the right-handed rule, the normal vectors point to the exterior of the solid. This is used to distinguish faces and holes.

There are two types of information in a B-rep: the topological information and the geometric information [2]. Topological information provides the relationships among vertices, edges and faces similar to those in the winged-edge data structure. In addition to connectivity, the topological information also includes the orientations of the edges. Geometric information usually includes the definitions of the edges and faces. A B-rep model is bounded by a set of well-organized faces, each of which is a piece of some kind surfaces (e.g. B-spline surface or T-spline surface). Faces may share vertices and edges that are curve segments (e.g. B-spline curves or Bézier curves). Therefore, B-rep is an extension of the wire frame by adding the face and edge information. B-reps have several variants in different system designs.

Since all edges and faces are explicitly represented in a B-rep, a wire-frame picture of a B-rep model can be quickly drawn. It is also easy and quick to ascertain topological

relationships of the surfaces, such as which vertices are connected to an edge, which edges are attached to a face, and so on [52]. We make use of this property in our work.

2.3.3 Cracks filling in visualization

To render a spline surface, in the current rendering pipeline, we have to convert it to some primitives such as triangles or quads, which the pipeline can process. Tessellation is such an operation which maps a regular grid in the parameter domain onto the surface. Then the surface is only evaluated at these grid points and rendered as polygons. The approximation error is computed as the maximal distance between the original surface and the approximation polygons. Even watertight spline models, in visualization, it may still generate cracks in rendering because of the different tessellation densities in different surfaces. For some applications, this lack of smoothness may be acceptable. However, for the smooth visualization, we need a continuous normal field over the entire surfaces. Especially when applying displacement maps, those artifacts will be enlarged. Various methods are proposed to fill in the cracks generated in rendering.

Some works fill in the cracks only in the screen space by drawing some extra geometries. However, the crack filling needs to be re-calculated according to the view change. Besides, the generated models are not really crack-free but only visually seamless.

Early works rely on the surface topology dependencies to connected adjacent polygon mesh together [16, 17, 18, 6, 19]. Barequet and Kumar [16] proposed a method to repair polyhedral CAD models that have cracks, degeneracies, duplications, holes or overlaps in the B-Rep. Their algorithm generates manifolds with consistent normal vectors. It closes small cracks and fills in larger gaps with polygons. The basic idea is to find the most possible edge-to-edge correspondence and stitch them together. However, they did not compute the approximation error accurately. Users need to inspect the faults on the surface and supervise the repair process. In Chhugani and Kumar’s method [17], to prevent cracks, they set a common set of sample points for adjacent boundary curves. How-

ever, they only consider Bézier patches and Bézier curves with consistent parameterization. Spline surfaces with different parameterizations were not considered. The work [18] concentrates on the trimmed NURBS. They tessellated all the trimmed NURBS surfaces and the trimmed boundaries to polygons or polylines, and then merge them. The tessellated trimmed NURBS surfaces are merged sequentially one by one, which is slow and not suitable for parallelization. Kahlesz [6] presented an approach to alleviate the dependence on the topology information by storing the boundary list of all patches. Their sewing along boundaries also guarantees an error tolerance if the boundaries are close enough. However, with their method, the polygon folding may occur as shown in Figure 2.11. The method used by stöger [19] heavily relies on the topology information and therefore it can hardly be done on the fly during rendering. For this kind of methods, the design and implementation of really fast algorithms is a hard task.

Figure 2.11: Folding in sewing [6]

For regularly arranged surfaces, the tessellation factors of shared boundaries are often set to be the large one. In this way, though the mesh seems to be watertight, there are duplicated sampling points on the shared boundaries. Additionally, even given the same sampling parameters, the positions and normals are computed from different control point sets. Different numerical roundings may accumulate to cause output vertices to project to different pixels. Slightly difference of the normal vectors along a shared boundary may cause crack problems when applying displacement maps to the model [24].

Recent works abandon the traditional maintaining connectivity strategy. The cracks are

filled trickly in the screen space. Fat borders [30, 20] are one of such methods, in which the gaps generated by independent tessellation are filled by shaded fat borders. In particular, at each boundary polyline, a small surfaces perpendicular to the current view direction is generated. The small surface generated between the two adjacent boundaries guarantees to fill the cracks. However, artifacts may occur when adjacent surfaces are with different colors or they form sharp corners. Another method dynamically adjusts the thickness of triangle strips with respect to the curvature of matching trimming curves [53]. Most recent work starts to integrate both tessellation and ray tracing. Tessellation is done on the main part of the surfaces while the ray tracing is used to fill gaps between surfaces [15].

2.4 GPU rendering of geometric models

In recent years, much progress has been made on the graphics industry's widespread shift to Graphics Processing Unit (GPU) computing, with which many independent tasks can be carried out simultaneously. Incorporating with the GPU computing, a variety of efficient and novel algorithms have emerged and been applied to graphics display, including geometry processing, rendering, physical simulation and so on [54].

In the history of GPU development, there are mainly three stages in which the developers have more and more control to the GPU resources. Firstly, it is the Open Graphics Library (OpenGL). It is an abstract API for drawing 2D and 3D graphics and typically used to interact with the hardware to obtain the hardware acceleration. At the very beginning, it is popularized by the company Silicon Graphics as the programming interface to their graphics hardware. Later, NVIDIA released the first GPU that implements Microsoft's DirectX 8.0 standard. This standard requires that the hardware contains both programmable vertex and programmable pixel shading stages [55]. The developers began to have some control over the exact computations that would be performed on the GPUs. By that time, the general approach to access GPU is very limited because the APIs such as OpenGL and DirectX are the only interfaces to GPU. Anyone who wants to perform general-purpose computa-

tions would have to convert his tasks appear as they are standard rendering [56, 57, 58]. In those applications, all the input, output and intermediate results must be converted and stored in the vertex buffer, frame buffer or texture buffer. Accessing arbitrary memory in the pipeline is impossible. A few years later, NVIDIA unveiled the GeForce 8 series built with NVIDIA's CUDA architecture. CUDA stands for Compute Unified Device Architecture and is a general purpose parallel computing platform and programming model that leverages the parallel computation engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU [59]. This stands out as a milestone in GPU technology. From then on, the developers are given more flexibility to harness the GPU power for general-purpose computations. There is no need to trickily convert the computing into vertex and fragment shaders.

With the use of OpenGL and DirectX, the users can only access the GPU using shading languages. NVIDIA added partial C, C++ and Fortran as the developing languages, making it easy for developers to explore the general-purpose computing on GPUs by utilizing the GPU resources.

2.4.1 CUDA architecture

Unlike previous GPU computing that must fit the computing to vertex and fragment shaders, CUDA offers a Single Instruction, Multiple Data (SIMD) architecture. At any given clock cycle, each processor executes the same instruction, but operates on different data [60]. The instructions are sequential codes loaded from CPU and they can be any general-purpose computations. They are called *kernel functions*.

In CUDA, each parallel unit is handled by a thread. To well organize the threads and improve the performance, CUDA batches all the threads in a hierarchy structure. After launching the kernel functions, all threads in a grid are invoked and run the same instructions. Inside each grid, threads are grouped into a set of blocks. Within each block there is a shared memory, a cache on chip that can be fast accessed by all threads in this block.

The threads in one block do not certainly terminate simultaneously. They may encounter divergency or with different workloads. In case the consequent computation needs the all results of the previous step, it is necessary to set a synchronization point. Some threads will be idle to wait all the threads to finish. In addition, 32 threads from a block are bundled into warps which are implicitly synchronized.

Furthermore, the threads on the GPU are allowed to access the memory and caches freely. Hence the GPU can be used very sufficiently and efficiently. There are several kinds of memories on chip as shown in the following memory model (Figure 2.12). The memories link CPU and GPU are the global memory, constant memory and texture memory. Both the CPU program input data transferring and the GPU program output data retrieved are through those memories. They are visible to all threads. The global memory has a long access latency (maybe hundreds of clock cycles) and limited access bandwidth. Both the constant memory and the texture memory have their own cache spaces but the size is quite limited. The constant memory holds some constant variables provided as input values to kernel functions. The texture memory is extremely efficient to handle 2D array data. However, both memories can be very slow in case of spilling, all the variables will be wrote to the global memory. Inside each block, there is a shared memory for read and write by all threads in this block. Its access is extremely fast and highly parallel. It's mainly used to hold the portion of global data that are heavily used in the execution of a kernel. Threads can cooperate by sharing the results of their work in the shared memory. The shared memory in each block is limited, it must be used efficiently. Registers hold frequently accessed variables that are private to each thread. The scalars or small arrays defined in the kernel function may be stored in registers by the compiler automatically. The capability 2.0 GPUs only support 63 registers per thread. If this is exceeded, the register values will be spilled to the local memory of each thread. Actually, local memory is not really a separate memory, they are stored in global memory, but cached in the L1 memory. Therefore, we should use each kind of memory appropriately to make the program be efficient.

On the aspect of hardware, one GPU handles several grids. Each GPU is organized as a

Figure 2.12: CUDA memory model

collection of multiprocessors, with each multiprocessor being responsible for handling one or more blocks in a grid. A block is never divided across multiple multiprocessors. Each multiprocessor is further divided into a number of cores, with each core handling one or more threads in a block. This is illustrated in Figure 2.13.

My works are all carried on a machine equipped with NVIDIA Quadro K5000 GPU and Intel Xeon CPU with 2.4GHz clock speed.

2.4.2 GPU performance analysis

With hundreds of arithmetic units on the GPU, often the bottleneck is not the arithmetic throughput of the chip but the memory bandwidth. There are so many ALUs on graphics processors that sometimes we just cannot keep the input coming to them fast enough to sustain such high rates of computation. Therefore, it is worth investigating how to reduce

Figure 2.13: CUDA architecture and its execution on GPU hardware

the amount of memory traffic required for a given problem. Some of the efforts could be compressing the data and decompressing in the device memory. For the input data passed to the device, it is important to make a trade-off between transfer time reduction caused by the data compression and the time cost of the uncompress.

In the devices, there are many factors affecting the performance, such as CUDA occupancy, memory coalescing, shared memory bank conflict, instruction optimization and so on. Here we only elaborated the CUDA occupancy problem. Other optimizations will not be explained in detail.

The maximum number of threads "in flight" at the same time has an upper limit. Theoretically, this number is determined by the device technical specifications. In particular, it is the number of blocks that can be launched per grid and the number of threads that can be assigned in each block. However, other factors will limit the number further. Firstly, the

resources allocated to each block are limited. Utilizing too many resources per thread may limit the GPU occupancy. Therefore, in each kernel function, it is better not to define too many variables or arrays, which may lead to overuse of registers. Secondly, shared memory is with fixed size to each multiprocessor. It is divided into many parts. Each thread block within a multiprocessor accesses its own part of shared memory. If each block occupies too much, it will reduce the number of active blocks. Thirdly, in the device, since each core have limited number of active clocks, a small block size will restrict the total number of threads "in flight". Figure 2.14 shows the occupancy variation with the change of registers used per thread, block sizes and the shared memory used per block. Those three factors should be considered comprehensively.

Figure 2.14: CUDA occupancy (a) occupancy with varying block sizes, (b) occupancy with varying register count per thread, (c) occupancy with varying shared memory sizes per block. (The device capability is 3.0 and total shared memory is 49152 bytes. This figure is drawn using the CUDA GPU Occupancy calculator provided by NVIDIA.)

2.4.3 B-rep model rendering on GPU

Many researchers have developed a lot of novel, reliable methods to render B-rep models on CPU, but their solutions have not been put into practice for a very simple reason that the computation is expensive and prohibitively time-consuming. The previous CPU methods can be summarized into three categories: full-model micropolygonization, ray-casting and tessellation. Based on these methods, in recent years, a number of publications shift the interest to GPU for high-quality real-time rendering.

Reyes (micropolygonization) [61] is an architecture proposed by Lucasfilm and used for fast high-quality rendering of complex models. The stages of Reyes include bounding/splitting, dicing, shading, sampling, compositing and filtering. The main idea is that, they diced the whole model into micropolygons which are flat-shaded subpixel-sized quadrilaterals on the screen. The dicing density is estimated under the criterion that each micropolygon is approximately half a pixel in the screen space. The high-order spline surfaces [25, 62] and Catmull-Clark surfaces all can use this architecture for accurate rendering. However, the traditional Reyes implementations like Pixar's RenderMan are still primarily non-interactive [63]. The increasing computation capability and programmability of the modern GPU motivates its use for a real-time Reyes implementation. However, all parts of the Reyes pipeline do not map well to such a parallel architecture. Many subsequent works re-designed the pipelines to fit programmable GPU better. NVIDIA's Gelato rendering system is a GPU-accelerated REYES system. However, only the hidden surface removal stage of the pipeline is accelerated on the GPU [64]. Patney and Owens [65] implemented the bounding/splitting and dicing stage on GPU. The stage recursively divides the model using a depth-first algorithm which involves a complex and irregular data structure management. Their results indicate that real-time Reyes split and dicing can indeed be obtained on GPUs. Patney [66] also discussed the challenges on implementing the Reyes pipeline on GPU. Most of these issues are resolved in Zhou Kun's work [67]. Zhou Kun et al. designed RenderAnts, the first system that enables interactive Reyes rendering on

GPUs. All stages of the basic Reyes pipeline, including bounding/splitting, dicing, shading, sampling, composting and filtering, are executed on GPUs using carefully designed data-parallel algorithms. One major limitation of micropolygonization method is the data size generated. Since the GPU memory size is limited, this is always the bottleneck of the micropolygonization. In the work [67], the GPU program crashes when there are too many sample points.

The ray tracing approach is another approach to render parametric surfaces. In the ray tracing process, several stages are involved: ray generation, ray tracing and ray shading. The color of each pixel is computed by tracing rays from the camera and finding out which surfaces are intersected by each ray. After obtaining the ray-surface intersections, then ray casting methods render the surfaces directly in terms of their polynomial representations, instead of a set of approximating triangles. The advantages are the high quality of the rendering results and the reduction in memory usage compared to both the micropolygonization and the tessellation. Since there are too many surfaces in the scene to test, some acceleration structures are adopted to narrow down the scope of testing. Previously, the computation complexity of ray tracing parametric surface [68, 69, 70, 71] make it rarely be accelerated using GPU. Recently, several GPU-based ray casting algorithm [21, 22, 23] have designed some new structures and demonstrated that the entire ray tracing computation can be implemented on GPU.

Purcell et al. [21] firstly mapped the whole process of ray tracing to GPU parallel computation. In their algorithm, the main stages of the ray tracing algorithm, such as traversal, intersection, and shading, are redesigned as small "kernels". Those kernels operate on a stream of pixels and textures, where each pixel corresponds to exactly one ray. The different kernels can then be implemented using pixels shaders [72]. In [73], the authors presented a GPU-based NURBS ray casting implementation. Convex hulls are used as tight bounding volumes for NURBS patches for early ray termination. Iterative Bézier clipping is proposed as root finding algorithm for NURBS trimming and rendering. Ray casting of trimmed parametric surfaces presents further difficulties since the polynomials involved may be of

higher degree, requiring numerical root extraction and the trimming of the surfaces requires a point classification algorithm. Guthe et al. [20] generate trimming texture to determine whether a point is trimmed or not. Schollmeyer et al [74] split the trim curve into a set of monotonic segments. Then the points is classified as trimmed or not by counting the number of ray-curve intersections. This method does not perform any actual intersection test, thereby accelerating the trimming process of NURBS surfaces on GPU.

Both micropolygonization and ray tracing methods render the models very accurately and avoid cracks between surfaces naturally. However, it is hard to achieve real-time performance due to the intensive computation. This makes tessellation the best option for real-time rendering of parametric surfaces. Besides, currently the game engines such as Unreal and Unity do not support NURBS surfaces. When designers use NURBS for the initial modeling, the NURBS surfaces need to be tessellated for display. For large scale NURBS models, the tessellation process could be very slow. To produce a real-time crack-free rendering, surfaces are always divided into several parts and tessellated in parallel [20, 14, 15, 29, 75]. For NURBS and T-spline surfaces, the common approach is to convert them into a set of Bézier patches. However, all previous works do the Bézier conversion on CPU. In [29], they proofed that the Bézier extraction cannot be implemented in real time since the iterative knot insertion process. This is not accurate. After conversion, some of the works directly apply the graphics pipeline developed for current GPU [25, 75]. This pipeline consists of several shader stages [24]: *vertex shader*, *hull shader*, *tessellator*, *domain shader*, *geometry shader* and *pixel shader*. The *hull shader* is to determine the tessellation factor of each patch. It compares the tessellation factors of adjacent boundary edges. Then set the common tessellation factor to be the larger one to prevent cracks. The *tessellator* is to generate all the (u, v) point for evaluation; the *domain shader* do the evaluation. Then the *geometry shader* generates triangles for final rendering.

Chapter 3

Extraction of Rational Bézier Patches from T-spline Surfaces on GPU

3.1 Introduction

T-spline surfaces consist of (rational) Bézier patches. Since Bézier patches have the same connectivity structure and simple mathematical expression, it is beneficial to decompose T-spline surfaces into a set of Bézier patches for many applications such as surface-surface intersection, isogeometric analysis and adaptive tessellation. For example, when computing the intersection of two surfaces, one common technique is the subdivision method which recursively subdivides the surfaces into a set of smaller patches until they are sufficiently flat to perform simpler computations and Bézier patches have nice properties and algorithms for such processes [76, 77]. In isogeometric analysis using NURBS surfaces or T-spline surfaces, Bézier patches have been shown to be a good candidate for the finite element representation [78, 28]. In surface rendering, Bézier patches facilitate adaptive tessellation of the surfaces [79]. Moreover, after the surfaces are decomposed into a set of Bézier patches, each Bézier patch can be processed individually, which provides much convenience for developing parallel algorithms.

The extraction of Bézier patches for NURBS surfaces is relatively straightforward due to

the regular structure of NURBS. However extracting Bézier patches from T-spline surfaces is not simple. Some previous work simply extends all the T-junctions by two-bays for degree 3 surfaces to form the domains of Bézier patches [80, 78] and then calculates the Bézier control points by iterative refinement. This may cause incorrect results because in some cases extending T-junctions only by two-bays is not enough. Figure 3.1 is such an example where the yellow shaded region actually corresponds to two Bézier patches. This is because one horizontal knot line of the control point A crosses the yellow region, which means that the surface is not C^∞ continuous within this region. By two-bay extension, the partition line stops at the boundary of the yellow region and the yellow region is then wrongly taken as a single Bézier patch domain. One brute force approach to overcoming this problem is to simply extend the T-junctions all the ways in the T-mesh. However, this will increase the number of Bézier patches significantly.

Another issue in Bézier extraction from T-spline surfaces is that due to the topology complexity of the T-mesh, irregular regions for C^∞ may exist, as shown in the central region of Figure 3.2(b). How to partition the irregular regions into a minimal set of rectangles is not trivial.

This chapter presents a solution to the problem of correctly extracting rational Bézier patches from T-spline surfaces such that the number of the resulting rational Bézier patches is as small as possible. We also aim at a parallel extraction algorithm that can be implemented on GPU. Previous work basically performs the extraction process in CPU and passes the results to GPU for subsequent processes [20, 15]. It was even argued that the Bézier extraction could not be directly done on GPU due to the three limitations in the extraction process: the convoluted iterative refinement which is used to obtain Bézier patches, the intensive memory used, and the high time cost on Bézier re-conversion when the model is modified [29]. In our approach, the convoluted iterative refinement is replaced by a linear extraction operator. When the model is modified, we only update the modified regions. However, the memory required to store the Bézier control points is indeed inevitable. We design a data lossless and compact representation for T-spline surfaces, which greatly

reduces the time consumption of data transfer from CPU to GPU and also facilitates subsequent computations on GPU.

Figure 3.1: An example: the two-bay extension is not sufficient for identifying the Bézier patch domains. (a) A T-spline model with knot lines mapped onto the surface; (b) the results of extending each T-junction by two-bays; (c) the local regions in the parameter domain which shows that the yellow rectangle should be split by the knot lines of control point A.

Figure 3.2: Bézier regions extraction: (a) an input T-mesh; (b) C^∞ domains; (c) Bézier patch domains.

3.2 Underlying techniques for Bézier patch extraction

The input to our Bézier patch extraction problem is a set of T-spline surfaces forming the compound T-spline model. The output is a set of rational Bézier patches. Our extraction algorithm is composed of three stages carried out on GPU. The first stage is to analyze T-mesh structure to identify C^∞ domains, which might not be rectangular. The second stage

is to construct Bézier patch regions which are rectangular regions. The third stage is to generate the Bézier representation for each Bézier patch. The details of these stages are described below. The GPU implementation is given in the next section.

3.2.1 C^∞ domain identification

For a T-spline surface, each control point is associated with a blending function whose domain is partitioned by all its knot lines. Across the knot lines, the blending function and hence the surface are in general only C^2 continuous. Figure 3.3(b) shows some knot lines. Since a Bézier patch is C^∞ continuous in its domain. Hence a key to determine the Bézier patch domains is to identify those regions within which the surface is C^∞ continuous.

To partition the whole surface domain into a set of C^∞ continuous regions, we first derive the two knot vectors and corresponding knot lines for all control points according to the T-mesh topology [80]. Then the parameter domain of the whole T-spline surface is divided by all these knot lines, which results in a set of C^∞ continuous regions as shown in Figure 3.2(b).

3.2.2 Partitioning of irregular zones

Note that the regions obtained from the first stage corresponds to C^∞ continuity of the surface, but they are not necessarily rectangles. Therefore, for a non-rectangular region which we call an irregular region, we need to partition it into a set of rectangles. To minimize the size of the output data set, it is better to partition each irregular region into a minimum number of rectangles. According to Ritu Chadha et al [81], an irregular region containing n vertices and h holes has a minimum partition of size

$$P = n/2 + h - C - 1 \quad (3.1)$$

Figure 3.3: Knot line partition (a) the knot vector of a control point, (b) its knot line grid, (c)&(d) one control point's partition

where C is the number of concave lines. A partition line is said to be concave if it joins two concave vertices with respect to the irregular region.

We adapt Ritu Chadha et al's algorithm for our purpose. In our problem, there is no hole. Consider an irregular zone bounded by a set of vertices on the contour, we first extract the edges on the contour according to the coordinates of the vertices. Based on the edges, the contour of the figure is re-ordered in a counter-clockwise direction. After that, the concave and convex vertices on the contour are identified. For each concave vertex, we create a partition line. This splits an irregular corner into two regular parts. If there are two overlapping partition lines, they will be merged into one *concave line*. A partition example is given in Figure 3.4. The blue lines are the contour of the figure and the red dash lines are the partition lines. If we draw a line from each concave vertex, the rectilinear figure is partitioned into three rectangles (Figure 3.4(b)). However, if we replace the two lines with a concave line, the number of rectangles is reduced by one (Figure 3.4(c)). Therefore

the minimal partition function is related to the number of vertices and concave lines. To minimize the number of rectangles, we should maximize the use of the concave lines. This is achieved by a graph algorithm introduced in [82]. It is worth mentioning that the concave lines intersecting with each other should not be counted. This is because after partitioned by one concave line, the other line is not concave anymore, as shown in Figure 3.5. A partitioning example is given in Figure 3.6, where the left pattern is what we want and the right one is not.

Figure 3.4: Irregular region partitioning: (a) an irregular region, (b) partitioning without concave lines, (c) partitioning with concave lines

Figure 3.5: Partitioning with intersecting concave lines.

3.2.3 Computation of Bézier control points

Once the rectangular domain of a Bézier patch within the T-mesh is identified, we now compute the Bézier representation from T-splines, which involves B-spline knot insertion.

Let us begin with a cubic B-spline curve with knot vector $\{\cdots, s_0, s_1, s_2, s_3, s_4, \cdots\}$. To derive a Bézier curve segment for a specified domain $[t_0, t_1]$ from the B-spline curve, we need to insert the knots t_0 and t_1 into the knot vector until they are triple knots. Then

Figure 3.6: Irregular region partitioning: (a) correct partition pattern; (b) incorrect partition pattern

four corresponding control points of the refined B-spline curves are just the Bézier control points. The refinement can be done step-by-step. Each time when a knot is inserted, a new control point will be introduced and two neighboring control points may be updated. The new control points are linear combination of the old control points. In particular, we can derive a 4 matrix $\mathbf{C}$ such that

$$\begin{bmatrix} Pb_0 \\ Pb_1 \\ Pb_2 \\ Pb_3 \end{bmatrix} = \mathbf{C} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \quad (3.2)$$

where Pb_0, Pb_1, Pb_2, Pb_3 are the four Bézier control points, and P_0, P_1, P_2, P_3 are the initial B-spline control points that have contribution to the Bézier curve. The matrix $\mathbf{C}$ serves as a conversion operator that solely depends on the knot vector of the B-spline curve and can be easily derived using polar form of B-splines. Using matrix $\mathbf{C}$ to extract Bézier representation can reduce the low-performance branch predictions on GPU.

For NURBS, the knot insertion is needed on both parameter directions. Hence the above operation has to be conducted on control points on both rows and columns.

Different from NURBS that has a global tensor product structure, T-spline surfaces have flexible T-mesh structure. To compute the control points of a Bézier patch from a T-spline surface, we treat each control point of the T-spline surface as a B-spline surface which has

only one non-zero control point and process each of such B-spline surfaces. Specifically, we first identify all the control points that contribute to this region. Then for each control point, 16 Bézier patches within its domain can be obtained by the conversion operator. After extracting the Bézier patches from all the control points, the Bézier patches corresponding to the same domain are summed up, which gives the final Bézier representation for that domain. Figure 3.7 illustrates the extraction of one Bézier patch from a T-spline surface.

Figure 3.7: Bézier patch extraction from a T-spline surface. (a) input T-spline surface, (b) the Bézier patch partitioning, (c) the enlargement of one Bézier patch.

3.3 GPU implementation

To perform Bézier extraction from T-spline surfaces on GPU, we need to first transfer T-spline data into GPU and then perform extraction process there. T-spline data involve control points and their connectivity. In general, the connectivity of T-spline surfaces are flexible and arbitrary, which makes them difficult for parallel implementation. We propose a specially designed data structure for T-splines which is compact and also suitable for parallel implementation on GPU. Then the extraction is implemented using three main kernel functions: a control point based parallel procedure for C^∞ region identification, a grid based parallel procedure for Bézier region determination, and a Bézier region based parallel procedure for Bézier patch generation.

Thus the input to our algorithm is the specially encoded T-spline surfaces and some offset information for multiple T-spline surfaces. The offset information is used to indicate the starting position of each surface in the encoded T-spline list. The output is a set of Bézier patches. The overall algorithm is given below, followed by the detailed elaboration of each steps.

Input T-spline surfaces and offset information

Output Bézier patches

Control point based parallel procedure For all the T-points:

- extract its knot vector;
- label all its knot lines on the T-mesh;
- identify the new connection types of all points (both T-point and N-points) on the T-mesh.

Grid based parallel procedure Traverse all the surface knots of each T-spline surface and generate a set of NURBS grids. From each grid (suppose its minimal parameter is (s, t)):

- use Breadth-first search to find all connected grids (regular or irregular);
- record the parameter range $(s_{min}, t_{min}, s_{max}, t_{max})$ of the connected region;
- if it is an irregular region: do irregular region partition;
- update the parameter range of irregular regions to the parameter range of newly generated regular regions;
- if $(s_{min}, t_{min}) \neq (s, t)$, label current region as invalid.

Bézier region based parallel procedure For all valid Bézier regions:

- find all the control points contributing to this region;
- extract Bézier patches from the B-spline surfaces defined by each control point;
- sum up all the Bézier patches corresponding to this region.

3.3.1 Special encoding of T-splines for GPU

Due to the arbitrary pattern of T-mesh, a typical data structure for T-splines is a half-edge-like structure which includes T-points, edges and faces [80]. Such data structure is quite different from NURBS which uses a rectangular point grid to store the control points. It is inconvenient for GPU implementation. Removing or inserting a point in the T-mesh usually results in the changes of T-points, edges and faces.

To overcome this drawback, we propose a new data structure for T-splines. We represent a T-mesh as a rectangular point grid of size $S_s \times S_t$, where S_s and S_t are the dimensions along the s and t directions, respectively. Different from NURBS, not all points on the T-mesh are control points (as shown in Figure 3.8). We classify the T-mesh points into two categories: N-points and T-points. A T-point is a control point on the T-mesh (square points in 3.8(b), including points No.2, No.3, No.5, and No.6) while an N-point is a virtual point (point No.1 and No.4 in Figure 3.8(b)). By transferring all N-points into T-points, the T-mesh becomes a NURBS control mesh and the T-spline surface becomes a NURBS surface. Each T-mesh point has a value c indicating its connectivity with its neighboring points. The value c is an accumulated value of

- 0x0001 if it connects to the next point along s - direction,
- 0x0010 if it connects to the next point along $s+$ direction,
- 0x0100 if it connects to the next point along t - direction,
- 0x1000 if it connects to the next point along $t+$ direction.

Thus, $c \in 0, 1, \dots, 15$. In Figure 3.8(b), $c = 3$ for N-point No.1, $c = 0$ for N-point No.4, $c = 11$ for T-point No.2, and $c = 6$ for T-point No.6. The storage of a T-mesh is different from the control mesh of a NURBS. For N-points, no x, y, z coordinates are required. In our data structure, each T-point is represented by 5 values (c, x, y, z, w) with $c > 0$, while an N-point only needs 1 value of c with $c < 0$ indicating that it is an N-point. By assigning a connectivity value for each T-mesh point, we do not need to store the edges and the

Figure 3.8: The pre-image (row for the s direction and column for the t direction) (a) for a NURBS surface; and (b) for a T-spline surface (No.1, No.2 and No.3 are on the same column; N-point No.4 is located on the same column as No.5 and the same row as No.6).

faces, which can be automatically derived from the connectivity values. The proposed data structure is intuitive for implementation.

We pass the connectivity information and all the positions of the T-points to GPU in two arrays. One stores the connectivity types of all T-points and N-points; the other stores the positions of all T-points. To fetch the position of corresponding T-points, we also assign each T-point an index in the T-point list. For T-points, to compress the connectivity value c and the index value id into a single value, we record them as the integer part and the decimal part of a floating-point number f , respectively. Also, we need n , the number of digits of the index when recovering it back from the decimal part. Therefore, we record the number of digits in the last digit of the integer part of the number. Then the f can be computed as:

$$f = \begin{cases} c, & c < 0 \\ 10c + n + id/10^n, & c > 0 \end{cases} \quad (3.3)$$

where n is the digit of id . For N-points, since there are no control points associated with them, no id value is needed. We just make them negative, which can distinguish them from

T-points.

$$\begin{aligned} f \leq 0 &\Rightarrow T - point \quad c = -f, \\ f > 0 &\Rightarrow N - point \quad c = \lfloor f \rfloor / 10, m = f - 10c, n = \lfloor m \rfloor, id = \lfloor (m - n) \times 10^n \rfloor. \end{aligned} \quad (3.4)$$

3.3.2 Kernel function 1: Topology construction

In the first step, we extract the knot vectors of all T-points based on the T-mesh information. This step is executed in a control point-based parallel way. All the points on the T-mesh, including both T-points and N-points, are stored in a column-wise order. The knot vectors of each control point can be directly inferred from the connectivity types of adjacent points. Those adjacent points can be found easily in the connection type array by indexing. After that the knot lines of each control point are figured out. We take the whole T-mesh as a regular grid on which some of the edges are connected and some are disconnected. An array is allocated to record the edge connectivity type of the whole T-mesh in a column-wise order. What we need to do here is to label all the edges that on a knot line as connected. Those will be further used for continuous regions identification. As illustrated in Figure 3.9(a), the small white quads indicate T-points on the T-mesh and the blue edges are all of their knot lines. Suppose all the surface knots traverse the T-spline surface, it is converted to a NURBS grid as in Figure 3.9(b). To ease the further computation, we also sort the connectivity information for all grids on the T-mesh. An array is allocated to record the connectivity information of the surrounding four edges of each grid. For a T-mesh with dimension m rows and n columns, the edge tag list allocated on the global memory should be $(m - 1) \times n + (n - 1) \times m$. Similarly, the array size of grid-edge tags, domain ranges and region valid tags are shown in Figure 3.10. To save spaces, we only consider the grids inside the surface domain.

Figure 3.9: T-mesh grids (a) continuous regions of a T-mesh (b) T-mesh grids

3.3.3 Kernel function 2: Bézier domains identification

Given the edge tags and grid-edge tags of all the T-meshes, the next step is to identify the continuous regions on the surface. In this step each thread will start from a grid. Taking each grid as seed, the seed expansion algorithm (Breadth-first-search) is adopted to obtain the continuous regions. We first check its four neighbors. If there is no edge between two regions and it is an unvisited neighbor, we add it to the current seed list, tag it as visited and combine it to the current region. Meanwhile, we check its four corner points from the grid-edge tag array to see whether they are on the contour of the region. For a corner point, if there are edges connecting to it, it is a contour point. Then we further check whether the contour point is a concave point. If yes, this region is classified as an irregular region. Repeat the above process until all connected regions are visited. In this step, each thread produces a list of unorganized contour points of a continuous region. The results will not be written to the global memory of GPU but are temporally stored on registers for further processing. If it is an irregular region, we firstly sort all the contour points row by row according to their coordinates. Based on the well-organized contour points, we start from the left-bottom corner, along each row and each column they are connected two by two (in pairs) to form contour edges of the region. Since each contour point is exactly connected to two edges, starting from one point, it is easy to traverse the contour in a specified direction. According to the direction of its two connected edges, a

Edge tag array, size: $(m-1) \times n + (n-1) \times m$

1	1	1	0	...	...	...	...	...	...	...	...	...	1	0	1
---	---	---	---	-----	-----	-----	-----	-----	-----	-----	-----	-----	---	---	---

Grid-edge tag array, size: $(m-3) \times (n-3) \times 4$

(1,0,0,1)	(0,1,1,1)	(1,1,1,1)		(1,1,1,1)
-----------	-----------	-----------	-------	-----------

Domain ranges lead by all grids, size: $(m-3) \times (n-3) \times 4$

($s_{min0}, t_{min0},$ s_{max0}, t_{max0})	($s_{min0}, t_{min0},$ s_{max0}, t_{max0})	($s_{min0}, t_{min0},$ s_{max0}, t_{max0})		($s_{min0}, t_{min0},$ s_{max0}, t_{max0})
---	---	---	-------	---

Region validity tag array, size: $(m-3) \times (n-3)$

1	1	0		1
---	---	---	-------	---

Figure 3.10: Global memory allocation

point can be distinguished as concave or convex. Given all the concave vertices, firstly find the maximum set of non-intersection concave lines using the graph algorithm, and then create partition edges from remaining concave points. Finally, we add the newly generated partition edges to the edge tag list. The parameter range of irregular regions will be updated to the ones of a regular region. At the end of this stage, each grid leads to a rectangle domain labeled with a domain range.

For all connected grids, certainly they will lead to the same continuous region. This will make the results have some redundancy. We provide another array to tag the validity of each region to avoid duplicating tessellation of the same region. For all the grids in the same continuous region, we only let the left-bottom grid be corresponding to a valid region. Other grids in this continuous region are tagged as invalid.

3.3.4 Kernel function 3: Bézier patch computation

After obtaining all the Bézier regions on the parameter domain, we calculate the geometry representations of the Bézier patches. This step operates in a Bézier region-based parallel

way. For each region, we firstly find the control points whose domain overlaps this region. Unlike NURBS surface, we cannot directly predict which control points overlap with a specified region. On a T-mesh, this is usually done by iteratively checking the domain of each control point one by one. However this is very time-consuming since one mesh may contain thousands of control points. In our implementation, this tedious process is replaced by hash table functions. This is implemented in an open source library: cudpp [83]. Its input and output are shown in Figure 3.11. The input of the hash function is a set of key-value pairs. A key-value pair is one control point id and one Bézier patch id. We use the left-bottom corner point index as the Bézier patch id. The output is two arrays. One array successively stores each patch and the control points that cover it; the other one indicates the offset for each patch to fetch its control points in the first array. Compared to iteratively checking method, our method can speed up the execute time considerably, especially on large T-meshes.

Figure 3.11: The input and output of hash function

After finding out all the control points over each Bézier region, to extract a specified Bézier region, a sub-Bézier patch is extracted from each control point. The final Bézier representation is obtained by summing all the sub-patches together. Finally, if we want render the surfaces, we apply a uniform tessellation to all the Bézier patches. The generated vertices and triangles on device will be directly rendered using OpenGL. All the vertices

are stored in a vertex buffer; all the triangles are stored in an index buffer. These buffers are created by OpenGL as graphics resource. Those graphics resources can be direct accessed by CUDA. Once the CUDA program finishes writing to those buffers, the OpenGL functions will be called to render the data in those buffers directly.

Excluding the initial input data, the memory allocated on GPU includes: edge tags, the face connectivity information, parameter domains of all Bézier regions, validity tags of the regions, key (region)-value (control points) pairs of the hash table functions, the final tessellated vertices, normal and triangles.

3.4 Experimental results and discussion

All the experiments were conducted on an Intel Xeon CPU at 2.5 GHz and an NVIDIA Quadro K5000 video card with memory bandwidth 173 GB/s. The video card features 8 multiprocessors and maximum 65536 registers in each block. Our proposed framework allows all computations to be proceed on the video card. Some of the tested examples are shown in Figure 3.14 to Figure 3.19. These models are composed of hundreds of T-spline surfaces. We have further refined them for performance testing. The most large model contains more than 110 thousands control points. As we can see in Table 3.2, the real-time extraction is achieved even for models with numerous control points. The overall run time is multiplied with respect to the increase of the model complexity. When the model is partially edited, only the modified control points will be passed to GPU. In this case, we do not need to re-extract all the Bézier patches. This allows the users to get the computation results on the fly. Figure 3.12 shows the CPU extraction time versus the GPU extraction time. The extraction time on GPU has a significant speed up (more than 1000x) compared to CPU. The total run time does not include the time consumed in data transferring between host and device. This is still a bottleneck for the overall GPU performance. They both use the same data structure. Since the CPU program successively extracts the Bézier patches one by one, it is undoubted that the GPU program will excel greatly.

Figure 3.12: Performance on CPU and GPU

Figure 3.13: Complex T-mesh with regions contain more than sixty grids

The execution time of each stag is reported in Table 3.2. It can be seen that the Bézier region identification and the control points computation dominate the overall execution time. That is mainly because the former stage adopts Breadth-first search and the latter stage needs to extract the sub-Bézier patches one by one. Table 3.3 gives the number of Bézier regions containing more than four grids (we call them large regions) and the number of irregular regions. We can see that the run time is closely related to the regularity of the model. In the dinosaur model, we refined the T-mesh to obtain some regions that contain several decades grids as shown in Figure 3.13. Its run time grows dramatically compared to the submarine model. The submarine contains more control points but with a more regular T-mesh topology. For the hash table functions, they need some necessarily setting up on

CPU and this is not involved in our total GPU run time. Theoretically, the hash functions have remarkable effect only when the model contains very large T-meshes. In the final stage, we performed a 5 by 5 uniform tessellation to each Bézier patch. The tessellation data is directly written to a vertex buffer and an index buffer for further processing or rendering. Since the vertices and triangles are a huge set, the writing to GPU global memory also consumes a certain amount of both time and memory. Actually the memory used for the geometry information of T-mesh is quite limited (as in Table 3.1). The memory consumed by all models are not significant which demonstrates the compactness of our T-spline data structure. Current GPUs are well qualified to store those data.

We adopt control points and grids as the unit of parallelism. The enormous data may lead to utmost utilization of the GPU. On the other hand, in our program, the overuse of registers limits the number of threads that run in each clock. In our results the occupancy of each multiprocessor is up to 50% which is relatively low. Therefore the run time increases significantly for extreme large models. Moreover, the run-time in each block is determined by the most time consuming thread. In the Bézier region identification stage, for all threads the time mainly diverges at the seed expansion stage. The threads spend much more time in the seed expansion stage and the overuse of the local resource in each block will further confine the performance. We test the worst case of seed expansion, with regions containing more than fifty grids. The time increases several milliseconds.

Figure 3.14: Bézier patch extraction from a chair model: (a) the control mesh; (b) Bézier patches with some Bézier control points; (c) a close view of some Bézier control points

Figure 3.15: Bézier patch extraction from a sun face model: (a) the control mesh (b) Bézier patches with some Bézier control points; (c) a close view of some Bézier control points

Table 3.1: Static of models

Model	# of control points	# of patches	# of vertices	# of triangles	Memory used (exclude tessellation data) (MB)	Memory used (include tessellation data)(MB)
shark	1347	1270	31,750	40,640	5.1	10.25
chair	1832	858	21,450	27,456	4.20	8.84
sun face	28751	9750	243,750	312,000	10.38	20.93
dinosaur	58048	40996	1,024,900	1,310,912	19.96	62.92
submarine	81865	52444	1,311,100	1,678,208	26.05	69.22
car	110976	75015	1,875,375	2,400,480	33.22	94.52

Table 3.2: Execute time (ms)

Model	Knot vector extraction	hash table sorting	Bézier region identification	Bézier patch computation	Tessellation
shark	0.16	0.62	2.91	1.23	0.39
chair	0.14	0.59	0.03	1.04	0.42
sun face	0.26	1.83	8.29	8.18	0.77
dinosaur	0.34	2.35	19.60	11.59	7.79
submarine	0.43	2.70	6.83	15.71	9.17
car	0.78	3.75	25.14	16.08	13.84

Figure 3.16: Bézier patch extraction from a shark model: (a) the control mesh; (b) isocurves of the original model; (c) Bézier regions; (d) a close view of one Bézier patch

Figure 3.17: Bézier patch extraction from a dinosaur model: (a) the control mesh; (b) Bézier patches with some control points; (c) a close view of some Bézier control points

Table 3.3: the number of irregular regions

Model	# of large regions	# of irregular regions
shark	925	0
chair	9	0
sun face	19520	8
dinosaur	32460	16
submarine	6133	12
car	60850	25

Figure 3.18: Bézier patch extraction from a submarine model: (a) the control mesh; (b) Bézier patches with some Bézier control points; (c) a close view of some Bézier control points

Figure 3.19: Bézier patch extraction from a car model: (a) the control mesh; (b) Bézier patches with some Bézier control points; (c) a close view of some Bézier control points

Chapter 4

GPU-friendly Crack-free Adaptive Tessellation of B-rep Models

4.1 Introduction

This chapter considers the problem of how to adaptively, seamlessly tessellate compound T-spline models on GPU. The tessellation of spline surfaces is a fundamental process that can not only be used for rendering but also facilitate many other geometric processes. Also it provides a way to convert the spline models into polygonal models which can be used in game applications.

One common approach to tessellation of NURBS/T-spline surfaces is to decompose the spline surface into a set of Bézier patches and then tessellate the Bézier patches independently in parallel. In most previous approaches, the Bézier conversion is always done on CPU [20, 25, 15], which makes the whole tessellation of NURBS/T-spline models in real-time difficult. Another problem with these approaches is that cracks exist since Bézier patches are tessellated independently as shown in Figure 4.1. Therefore various methods are proposed to remove or prevent the cracks [4, 5, 1]. In general, to achieve real-time and crack-free tessellation, there are a few issues that need to be resolved:

- Conversion between NURBS/T-spline surfaces to Bézier patches is hard to implement

Figure 4.1: Crack caused by different tessellation densities

in real-time;

- Cracks may appear between adjacent surface boundaries because of different tessellation densities;
- Cracks may appear between adjacent trimmed boundaries.

The previous chapter has given a solution to real-time T-spline to Bézier conversion. Therefore, the first issue can be solved. The second issue is caused by different parameterizations between adjacent surfaces. Even with the same tessellation factor, two boundary curves may generate totally different sets of sample points. Prior work often assumes that one surface has only one neighbor surface at one boundary and adjacent surfaces have the same parametrization. The traditional crack-filling in the object space simply lets adjacent boundaries share the same tessellation factor. In this way, some patches are over-sampled at the boundary [14, 25]. Even computing sample points at the same parameter sampling rate, it may possibly result in difference points. The problem becomes more serious while applying a displacement map to the model [24]. Other methods stitch the sample points at adjacent patches [6, 19], which heavily depend on the topological information among patches. Therefore, they are not practical for GPU implementation. In the screen space, the cracks are filled using fat borders along boundaries [30] or adjusting the thickness of triangles dynamically [53]. This kind of approach arises for a faster rendering, but with artifacts caused by the extra geometries. Most recently, Claux et al. [15] filled the cracks using the ray-casting method. However, filling the cracks in the screen space cannot generate physically seamless polygon meshes. A recent novel work to handle the shared edges is

presented in [62]. To prevent cracks, it tessellated the shared edge using the edge information. Therefore, it does not rely on the topology information of patches. If adjacent patches share the same boundary, the resulting tessellation density at that boundary should be identical. However, the method still suffers from the bit-wise inconsistent problem. The third issue is the unavoidable gaps between trimmed patches which has troubled for a long time in industry. The trimming curves are 2D curves defined on the parametric domains. However when they are mapped to the 3D surface, they do not match with each other mathematically in general. Some approaches have been proposed to address this issue [20, 75, 53].

In this chapter we mainly address the second issue and present a new GPU-friendly tessellation algorithm. The main contributes lie in the following three aspects:

- We improve the tessellation factor estimation which results in fewer vertices to be generated to approximate the surface within the same tolerance and thus accelerates the tessellation process.
- The algorithm generates a crack-free polygonal mesh from a compound T-spline surface, which ensures an approximation error to be within the given tolerance. The surface is first decomposed into a set of Bézier patches plus a set of Bézier curves at the shared boundaries of those patches. All Bézier elements are tessellated according to their own variation. Two adjacent tessellations will connect to the same tessellation of Bézier curves. There is no over-sampling and bit-wise inconsistency problem at the shared boundary between patches.
- All Bézier patches and Bézier curves are extracted in parallel, followed by a parallel adaptive tessellation. As a result, we achieve tessellation of compound T-spline surfaces in an interactive rate.

Figure 4.2: Overview of the B-rep model rendering pipeline on GPU.

4.2 Algorithm overview

The **input** to our algorithm is a set of connected T-spline surfaces that contain exactly or partially matching boundaries. The input also includes all the geometric and topological information of those surfaces (in B-rep). The **output** is a watertight triangular mesh which approximates the input compound T-spline surface model within the given tolerance. Our algorithms takes great advantage of highly parallel resources on GPU. All the stages except the boundary merge are implemented on GPU using CUDA, with different parallel units. Figure 4.2 outlines the stages of the algorithm. Before passing the data to GPU, the T-mesh data are encoded as described in previous chapter. This data encoding method not only provides sufficient clue for data retrieving but also requires less memory.

Given a compound T-spline surface model, crack-free interactive tessellation works in the following stages. Firstly, in order to obtain a consistent sampling pattern for the two curves that may have totally different parameterizations or knot vectors, we split them into two sets of Bézier curves with identical length. This step is shown in Figure 4.3(a)(b). In the figure, two adjacent surfaces are purposely separated a little bit for a clearer visualization.

Figure 4.3: Algorithm overview (a) original surfaces and their boundary curves, (b) boundary merge, (c) Bézier patch extraction, (d) Bézier curve extraction, (e) transition region identification and central region tessellation, (f) the final mesh after transition region creation.

Since each pair of Bézier curves is in the same shape, the tessellation factor can be estimated by only using one curve in each pair. This stage is run on CPU. Then the original B-rep model data as well as the new split positions will be passed to GPU for further processing.

After that all the Bézier patches are extracted from the original T-spline surface as elaborated in Chapter 3. The Bézier regions are identified by T-point knot lines partition, followed by an irregular region partition. After this stage, all the continuous regions and continuous edges on the T-mesh are identified as Bézier elements. Moreover, the newly inserted positions at surface boundaries in the first stage are additional Bézier curve splitting positions.

The hash functions are adopted to find the control points which define each curve and each patch. From all the regions and curves each control point covers, we can obtain the group of T-points covering each region or curve by directly applying the parallel hash functions. After that the control points and the tessellation factors of all the Bézier elements will be computed. All the Bézier control points and tessellation factors will be stored on the global memory.

To prevent cracks between patches, instead of setting the edge tessellation factors to be the larger one of its two adjacent patches, we directly connect adjacent patches to their shared Bézier curves. Since the tessellation factors of the Bézier curves are computed independently, there is no over-sampling or bit-wise inconsistent problem at boundaries.

Based on the tessellation factors of all Bézier patches and Bézier curves, the prefix sum is applied to obtain the size of vertex buffer and index buffer. Simply connecting the central region of each patch and its surrounding curves cannot ensure the approximation error. A transition region is generated on the peripheral region of each patch. Its width is adjusted according to the tessellation factors of surrounding Bézier curves. Finally, all tessellation data are output to the vertex buffer and index buffer.

Figure 4.4: Boundary pre-process of adjacent boundary curves. (The colors are used to indicate individual Bézier curves) (a) split the boundary curves into a set of Bézier curves according to the original knots, (b) produce the same set of Bézier curves for adjacent boundaries with curve refinement.

4.3 Boundary merging

From the B-rep representation, we can obtain a list of boundary curve pairs that are aligned on adjacent surfaces. The boundary curves are a set of B-spline curves. For each pair of surface boundaries, even though they are exactly the same, they may be in different parameterizations. Hence even if applying the same tessellation factor, the sample points may be totally different. Therefore, our algorithm begins with a boundary merge stage which aims to solve the unmatched parameterization problem between adjacent surfaces.

Our problem is thus to convert each pair of B-spline curves into two sets of identical Bézier curves so that each pair of adjacent surfaces share the same set of Bézier curves. Take one pair for example. We label them as *cur1* and *cur2*. We first compute the 4D

points (x, y, z, w) corresponding to each knot of $cur1$. Since the two curves have the same shape, those points should be on $cur2$ too. We compute their parameters on $cur2$, and then refine $cur2$ at those parameters. Similarly, $cur1$ is refined in the same way. After that, if we split the two B-spline curves at their knots, it will generate two sets of Bézier curves which are exactly the same.

Given a point on the curve, we use the Bézier inversion algorithm described in [84] to get its parameter, where the derivation is however for planar curves. To adapt the algorithm to 3D curves, we arbitrarily choose two components of (x, y, z) and treat them as the x, y coordinates of a planar curve. It is possible that the solution is not unique when projecting the 3D curve to 2D. In this case, the inversion equation gives 0/0. We then select other two components to avoid this issue. Since there is no self-intersections in our surfaces, there is a unique solution. The inversion of the Bézier curve is given as follows:

$$t = \frac{l_b(x, y)}{l_b(x, y) - l_a(x, y)} \quad (4.1)$$

where

$$l_a(x, y) = c_1 l_{31}(x, y) + c_2 [l_{30}(x, y) + l_{21}(x, y)] + l_{20}(x, y),$$

$$l_b(x, y) = c_1 l_{30}(x, y) + c_2 l_{20}(x, y) + l_{10}(x, y),$$

$$l_{ij}(x, y) = \binom{n}{i} \binom{n}{j} w_i w_j \begin{vmatrix} x & y & 1 \\ x_i & y_i & 1 \\ x_j & y_j & 1 \end{vmatrix}, \quad n \text{ is the degree of the curve, and } x_i, y_i, w_i \text{ are the co-}$$

ordinates and weight of the i_{th} control point. If P_1, P_2 and P_3 are not collinear, there exist constants c_1 and c_2 :

$$c_1 = \frac{w_0 \begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}}{3w_2 \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}}, c_2 = \frac{w_0 \begin{vmatrix} x_0 & y_0 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}}{3w_1 \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}}$$

For each pair of Bézier curves, if their shapes are exactly the same and both parameterized properly, theoretically they can be re-parameterized to get the same parametrization. Therefore, using any one of them to compute their common tessellation factor is fine. Their tessellation factors are stored with their surface id, boundary id and the starting parameter of each Bézier curve. Figure 4.4 gives an example of boundary merge. In Figure 4.4(a), the Bézier segments of the original curves are displayed in different colors; Figure 4.4(b) shows the resulting Bézier curves after boundary merge. It is clear that along the adjacent boundaries among surfaces, they have exactly the same set of Bézier segments.

4.4 Surfaces decomposition

We now propose a method that computes the tessellation factor of each Bézier patch independently without comparing with its neighbor patches, which makes our method suitable for parallel implementation on GPU. Moreover, each edge should be treated as a separated element, so that it is independent of the patches. The two adjacent Bézier patches then connect to the same set of edges, which can avoid the crack problem.

4.4.1 Bézier patch extraction

The previous chapter has presented a CUDA-based algorithm to extract Bézier patches from T-spline surfaces. It guarantees a minimum number of Bézier patches. The whole algorithm is split into several parallel stages to fit the CUDA architecture maximally. The details can

be found in Chapter 3. Here we just briefly summarize it for completeness of the algorithm.

At beginning, we encode the T-spline surfaces in a memory efficient way. The data structure is compact for a fast data transfer from host to device as well as efficient for the information retrieving.

Firstly, we need to identify the Bézier regions on the domain. This is done by appending all the knot lines of all T-points to the T-mesh and run in T-point parallel. After that, we obtain all continuous regions, but some of them are not rectangles. Then an irregular region partition algorithm is applied to partition them into minimal number of rectangles. The obtained rectangular regions are ensured to be Bézier patch domains. With the knowledge of each control point defining which patches, the control points defining each patch can be obtained by the hash functions. Take each control point associated with its blending function as a point-based B-spline surface. A sub-Bézier patch at specified domain is extracted from each point-based B-spline surface. The final control points of a Bézier patch are the sum of all sub-Bézier patches over its domain.

Since the Bézier extraction step can be realized in the parallel tessellation process, the whole tessellation is accelerated greatly.

4.4.2 Bézier curve identification

We extract one Bézier curve corresponding to each edge on the partitioned T-mesh. In particular, the Bézier representations of the edges at the T-mesh boundaries can be directly retrieved from the boundary Bézier curves obtained in Section 4.3. The retrieval is done using the surface index, boundary index and the starting edge parameter.

In the partitioned T-mesh, except all the original T-points, there are some N-points changed from none, horizontal edge or vertical edge to T-junctions or cross junctions. They are labeled as *new T-points*. One example of such a point is shown in Figure 4.5(b). Both new T-points and original T-points are end points of Bézier curves. Starting from each T-point (both new and old T-points) on the partitioned T-mesh, we find their edges connected

to the right and top T-points. Two threads are occupied by each T-point. One searches in $s+$ direction; the other searches in $t+$ direction where s and t are the two parameter directions of the surfaces.

Figure 4.5: Bézier edge identification. (a) the original T-mesh, (b) the partitioned T-mesh(the circled point is a *new T-point*), (c) edges associated with some T-points (the $s+$ direction edge is in red; the $t+$ direction edge is in green).

In previous chapter, the patch identification simply runs in grid-point (all T-points and N-points) parallel since we cannot predict which point leads to a Bézier patch. In this case, we allocate memories for each grid-point to store the possible Bézier region it leads to. Since not all grid points lead to a valid Bézier region, there are much redundant memory space allocated. Hence we do improvement to let the Bézier edge identification run in T-point parallel. Some preparation work is needed to achieve this. Given the connection types of all grid points, firstly a prefix sum is applied to obtain the total number of T-points, num_T . Besides this, according to the grid-point index, T-point label array and the prefix sum array, we obtain a grid-point index for each T-point. This can be efficiently done in grid point parallel level. The procedure is shown in Figure 4.6. With those information, it is easy to get the T-point index for each grid point, and vice versa. This also provides convenience when later we use the surrounding grid-point indices of each patch to retrieve its boundary curves. To identify the Bézier curve from each T-point, firstly, find its grid-point index from the grid-point index array. Then its right and top connection can be identified by checking the connection types of points in the specified direction.

Figure 4.6: Preparation for running in T-point parallel

Figure 4.5(c) selects five T-points as examples to show their associated edges. By finding the right and top edges of each T-point, all the Bézier edges on the T-mesh can be traversed. The total memory allocated to store edges is $\text{sizeof}(\text{integer}) * 2 * \text{num}_T$. Each edge is uniquely identified by the grid-point indices of its two end points. For each T-point, it acts as one end point whose grid-point index can be easily obtained. Therefore we only need two integers to record the other ends of the two edges. The index of an end point located at the i_{th} row and j_{th} column is computed as $j * \text{num}_{col} + i$, where num_{col} is the number of columns of current T-mesh. If there is no valid edge, we fill the indices as -1. After estimating the two end points of all edges, the parameter range of each curve can be easily obtained by looking up the two knot vectors of current surface.

4.4.3 Bézier curve computation

After estimating the parameter range of each Bézier curve, we compute the Bézier control points. This step is done by finding all the control points defining each curve, extracting iso-curves from each point-based B-spline surface and then summing them up.

In the Bézier patch computation, searching for control points defining a curve takes up a large proportion of time among the whole process, especially for very large T-meshes. The hash functions are also applied here. The key-value pair becomes one old T-point-one edge segment pair since only old T-points are associated with non-zero control points. We find all the possible edges segments it covers and we use the starting T-point index (possibly

old or new T-points) to indicate each edge. In Figure 4.7, take the red point for example. Inferred from its polar label, its domain is shaded in blue. The control point itself with each T-point index it covers (both new and old T-points) forms a key-value pair. All the key-value pairs are obtained in parallel from all control points on the T-mesh. Given all the key value pairs, the parallel hash function returns two arrays. One array lists each T-point and which control points cover it. To fetch the influenced control points over a specified edge segment from the first array, we also need the offset and length information which are indicated in the second array. Figure 4.8 illustrates the output. For example, in the first array, the control points defining each edge are stored successively. If we want to fetch the control point indices defining it, we start from 0 element and retrieve the successive six control points in the first array.

Figure 4.7: Input of the hash functions (a) a control point (the red point) with its domain, (b) all T-points it covers (highlighted in green and red).

Figure 4.8: Output of the hash functions

To obtain the Bézier representation of those curves, we adapt the Bézier patch extrac-

tion framework with only one difference. After obtaining all the control points over an edge, in the Bézier patch extraction, we extract a sub-Bézier patch from each point-based NURBS defined by each control point. In the Bézier curve extraction, we extract a sub-Bézier curve from each point-based NURBS. As in Section 3.2.3 of Chapter 3, the Bézier curve extraction from a B-spline curve can be deduced using an operator:

$$\begin{bmatrix} Pb_0 \\ Pb_1 \\ Pb_2 \\ Pb_3 \end{bmatrix} = \mathbf{C} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \quad (4.2)$$

where $Pb_i, (0 \leq i \leq 3)$ are the Bézier control points and $P_i, (0 \leq i \leq 3)$ are the related control points on the B-spline curve; $\mathbf{C}$ is the linear Bézier curve conversion operator.

To obtain a Bézier curve within interval $[s'_0, s'_1]$ and iso-parameter t' on the surface, firstly the operator is applied along the s direction on each row. Then the curve is set to be the iso curve at parameter t' . This Bézier curve function becomes much concise when it is applied to a point-based NURBS surface defined by a control point:

$$C^b(s) = \mathbf{C} \mathbf{P} N(s) N(t') \quad (4.3)$$

where $\mathbf{C}$ is the Bézier conversion operator and $\mathbf{P}$ is an matrix of dimension 4 by 4 which contains only one non-zero row: the control point (x, y, z, w) . The position of this non-zero row is set according to the extraction interval. Then the Bézier representation of the curve can be obtained by summarizing all the sub-Bézier curves together. Finally, the tessellation intervals for those Bézier curves are computed using the criterion described in Section 4.5.

The edge identification, control point computation and tessellation factor estimation are all run successively by one kernel function. To avoid register spilling that may decrease the performance, this kernel can be split into several kernel functions. In this way, we need more global memory to store the intermediated variables. There is always a tradeoff

between run time and memory size.

The pseudocode of parallel Bézier extraction from a set of T-meshes is given below.

Input All the T-points (c, x, y, z, w) and N-points (c) (**partitioned T-mesh**)

Two knot vectors of all surfaces

Surface offset information

Output All Bézier curves corresponding to the partitioned T-mesh edges

kernel 1: hash function

input: indices of edge segments covered by each control point;

output: indices of control points that cover each edge segment.

kernel 2: identify edge segments For each T-point on the T-mesh:

identify the edge segments in the $s+$ and $t+$ direction.

estimate their parameter ranges if applicable;

retrieve all the control points over the two edges if applicable;

compute the Bézier control points of each Bézier curve.

compute its tessellation factor.

4.5 Tessellation factor estimation

Figure 4.9: Error between rational curve and its approximation line segment

After the surface decomposition, the tessellation factor of each patch and each curve is computed independently according to their own variation. To reduce the data set and

improve the performance, we strive to represent the whole model using as few triangles as possible while ensuring the approximation to be within the given error tolerance. In this stage, we estimate a good tessellation factor for rational Bézier patches and curves. In our following description, $tessellationfactor = ceil(1/tessellationinterval)$.

Many methods have been proposed to estimate the tessellation intervals of rational curves or surfaces [85, 86, 87]. Compared to other methods, Zheng and Sederberg's approach [87] provides a relatively precise results. Naturally, the tessellation interval should be only related to the curve/surface variance. However, their results rely not only on the variation but also the position of the surface. This is not intrinsic but inevitable in their method since they performed perspective projection from rational space to non-rational space. While they proposed a heuristic approach to overcome the problem, here we improve their method by alleviating the impact of the curve/surface position.

A rational curve over the domain $[\alpha, \beta]$ can be represented as:

$$r(t) = \frac{R(t)}{w(t)}, t \in [\alpha, \beta] \quad (4.4)$$

Let $L(t)$ be a fractional-linearly parameterized line segment that connects $r(\alpha)$ and $r(\beta)$. The approximation error (as shown in Figure 4.9) can be guaranteed not higher than ε if the step size δ satisfies:

$$\delta = \beta - \alpha \leq \begin{cases} \sqrt{\frac{8 \inf_t \{w(t)\} \varepsilon}{\sup_t (\|R''(t)\| + (r - \varepsilon) |w''(t)|)}}, & \varepsilon < r, \\ \sqrt{\frac{8 \inf_t \{w(t)\} \varepsilon}{\sup_t \|R''(t)\|}}, & r \leq \varepsilon < 2r, \\ 1, & 2r < \varepsilon. \end{cases} \quad (4.5)$$

where r is defined as the furthest point on the curve $r = \sup_t \|r(t)\|$ with respect to the origin and $R''(t)$ is the second order derivative of the nonrational curve. Applying the above results to a rational Bézier curve of degree n

$$r(t) = \frac{R(t)}{w(t)} = \frac{\sum_{i=0}^n P_i w_i B_i^n(t)}{\sum_{i=0}^n w_i B_i^n(t)}, t \in [0, 1] \quad (4.6)$$

the bounds of the second-order derivatives of the nonrational Bézier curve can be estimated by

$$\|R''(t)\| \leq n(n-1) \max_i \|\Delta^2(w_i P_i)\| \quad (4.7)$$

$$\|R''(t)\| + (r - \varepsilon)|w''(t)| \leq n(n-1) \max_i (\|\Delta^2(w_i P_i)\| + (r - \varepsilon)|\Delta^2 w_i|) \quad (4.8)$$

The qualified step size for the Bézier curve can be obtained by substituting the above results to formula 4.5.

In their method, they estimated the effect of perspective transformation from rational space to non-rational space first and then defined the approximation line segment $L(t)$ also in a rational form. After that, the approximation error between $r(t)$ and $L(t)$ can be determined in non-rational space which is less computationally intensive. It can be seen that the results are related to the position of the origin of the perspective projection. They gave a solution by translating the curve and made r be minimal with respect to the new origin of the coordinate system. From formula 4.5, it can be learnt that the smaller formula 4.7 and 4.8 are, the larger the step size is. However, other parts in the formula 4.8, $\|\Delta^2(w_i P_i)\|$ also changes with the translation, which makes the final results non-optimal.

We improve their result by computing the minimal of the whole formula $\|\Delta^2(w_i P_i)\| + (r - \varepsilon)|\Delta^2 w_i|$. Assume O is the new origin; after translation, the right parts of formula 4.8 becomes:

$$\begin{aligned} & n(n-1) \max_i (\|\Delta^2(w_i(P_i - O))\| + (r_0 - \varepsilon)|\Delta^2 w_i|) \\ = & n(n-1) \max_i (\|\Delta^2 w_i P_i - \Delta^2 w_i O\| + (r_0 - \varepsilon)|\Delta^2 w_i|) \\ = & n(n-1) \max_i |\Delta^2 w_i| (\|\frac{\Delta^2 w_i P_i}{\Delta^2 w_i} - O\| + (r_0 - \varepsilon)) \end{aligned} \quad (4.9)$$

Moreover, since ε is a constant provided by the user and r can be roughly defined as the distance from the origin to the furthest control point $r = \max_j \|P_j\|$ rather than the furthest point on the curve, the above formula changes to

$$\begin{aligned} & n(n-1) \max_i |\Delta^2 w_i| (\|\frac{\Delta^2 w_i P_i}{\Delta^2 w_i} - O\| + \|P_j - O\| - |\Delta^2 w_i| \varepsilon) \\ & i = 0, \dots, n-2; j = 0, \dots, n \end{aligned} \quad (4.10)$$

Given two groups of discrete points, forward difference points $\frac{\Delta^2 w_i P_i}{\Delta^2 w_i}$ and control point P_j , to find a point O and make the sum of the distance from O to the furthest point in each group, $\|\frac{\Delta^2 w_i P_i}{\Delta^2 w_i} - O\| + \|P_j - O\|$ be minimized, one reasonable approach is finding a tight bounding box or smallest circle that encloses all the points and then taking the center of the bounding box or smallest circle as the new origin. However, one additional element in the formula $\Delta^2 w_i$ varying with different point makes the final results unpredictably. Therefore, when estimating O , the geometric positions of these points are not the only determinant factors. For the discrete points with higher weights, there is a high probability that one of them leads to the maximal value of the distance. Hence, the O should be closer to them, to balance the effects of their weights. Based on the above analysis, we set the following objective function to determine O :

$$\min_O \left(\sum_{j=0}^n (P_j - O)^2 + \sum_{i=0}^{n-2} |\Delta^2 w_i| \left(\frac{\Delta^2 w_i P_i}{\Delta^2 w_i} - O \right)^2 \right) \quad (4.11)$$

Let

$$F = \sum_{j=0}^n (P_j - O)^2 + \sum_{i=0}^{n-2} |\Delta^2 w_i| \left(\frac{\Delta^2 w_i P_i}{\Delta^2 w_i} - O \right)^2 \quad (4.12)$$

Then O can be solved by letting the first-order derivative of the function equal to zero:

$$\frac{\partial F}{\partial O} = 2 \sum_{j=0}^n (O - P_j) + 2 \sum_{i=0}^{n-2} |\Delta^2 w_i| \left(O - \frac{\Delta^2 w_i P_i}{\Delta^2 w_i} \right) = 0 \quad (4.13)$$

$$O = \frac{\sum_{j=0}^n P_j + \sum_{i=0}^{n-2} |\Delta^2 w_i| \frac{\Delta^2 w_i P_i}{\Delta^2 w_i}}{n + 1 + \sum_{i=0}^{n-2} |\Delta^2 w_i|} \quad (4.14)$$

In addition, we subdivide the rational Bézier curve once to obtain a tighter bounding box of the curve. r will be reduced with a high probability. We represent the subdivision control points as P_s :

$$O = \frac{\sum_{j=0}^k P_{s_j} + \sum_{i=0}^{n-2} |\Delta^2 w_i| \frac{\Delta^2 w_i P_i}{\Delta^2 w_i}}{k + 1 + \sum_{i=0}^{n-2} |\Delta^2 w_i|} \quad (4.15)$$

If $|\Delta^2 w_i| = 0$, the point $\frac{\Delta^2 w_i P_i}{\Delta^2 w_i}$ becomes infinity. We ignore this kind of points and

apply the algorithm to the rest points. Another merit is that when the error tolerance is changed, there is no need to re-compute O . New tessellation interval can be obtained very quickly by only changing some constant values. This is beneficial in dynamically tessellation according to the view.

The above method can be easily extended to rational Bézier surfaces. According to Zheng and Sederberg's method, the tessellation intervals in the two parameter directions are confined by the following inequality:

$$D_{ss}\delta_s^2 + 2D_{st}\delta_s\delta_t + D_{tt}\delta_t^2 \leq 8\varepsilon \inf_{(s,t) \in T} w(s, t) \quad (4.16)$$

where

$$r \geq \sup_T (\|r(s, t)\|)$$

$$D_{ss} = \begin{cases} \sup_T (\|R''_{ss}(s, t)\| + (r - \varepsilon)|w''_{ss}(s, t)|), & \varepsilon < r, \\ \sup_T \|R''_{ss}(s, t)\|, & r \leq \varepsilon < 2r, \\ 0, & 2r < \varepsilon. \end{cases}$$

The other two elements D_{st}, D_{tt} are defined in a way similar to D_{ss} ; δ_s and δ_t are the tessellation step sizes along the two parameter directions. Under a specified error tolerance, in order to maximize the step sizes, D_{ss}, D_{st} and D_{tt} should be made as small as possible. For a rational Bézier surface defined by

$$r(s, t) = \frac{R(s, t)}{w(s, t)} = \frac{\sum_{i=0}^n \sum_{j=0}^m w_{i,j} P_{i,j} B_i^n(s) B_j^m(t)}{\sum_{i=0}^n \sum_{j=0}^m w_{i,j} B_i^n(s) B_j^m(t)}, s, t \in [0, 1]. \quad (4.17)$$

The bounds of its D_{ss}, D_{st}, D_{tt} can be computed as

$$D_{ss} = n(n-1) \max_{0 \leq i \leq n-2; 0 \leq j \leq m} \{ \|w_{i+2,j} P_{i+2,j} - 2w_{i+1,j} P_{i+1,j} + w_{i,j} P_{i,j}\| + (r - \varepsilon) |w_{i+2,j} - 2w_{i+1,j} + w_{i,j}| \} \quad (4.18)$$

$$D_{st} = nm \max_{0 \leq i \leq n-1; 0 \leq j \leq m-1} \{ \|w_{i+1,j+1}P_{i+1,j+1} - w_{i+1,j}P_{i+1,j} - w_{i,j+1}P_{i,j+1} + w_{i,j}P_{i,j}\| + (r - \varepsilon)|w_{i+1,j+1} - w_{i+1,j} - w_{i,j+1} + w_{i,j}|\} \quad (4.19)$$

$$D_{tt} = m(m-1) \max_{0 \leq i \leq n; 0 \leq j \leq m-2} \{ \|w_{i,j+2}P_{i,j+2} - 2w_{i,j+1}P_{i,j+1} + w_{i,j}P_{i,j}\| + (r - \varepsilon)|w_{i,j+2} - 2w_{i,j+1} + w_{i,j}|\} \quad (4.20)$$

Let $r = \max_{0 \leq i \leq n; 0 \leq j \leq m} \|P_{ij}\|$. The objective function to compute the new origin can be derived in the same manner as the curve case:

$$\min_O \sum_{k,h} (P_{k,h} - O)^2 + \sum_{i,j} |\Delta^2 W_{i,j}| (Q_{i,j} - O)^2 \quad (4.21)$$

In this function, $Q_{i,j}$ is the combination of

$$\begin{aligned} & \frac{w_{i+2,j}P_{i+2,j} - 2w_{i+1,j}P_{i+1,j} + w_{i,j}P_{i,j}}{w_{i+2,j} - 2w_{i+1,j} + w_{i,j}}, 0 \leq i \leq n-2; 0 \leq j \leq m \\ & \frac{w_{i+1,j+1}P_{i+1,j+1} - w_{i+1,j}P_{i+1,j} - w_{i,j+1}P_{i,j+1} + w_{i,j}P_{i,j}}{w_{i+1,j+1} - w_{i+1,j} - w_{i,j+1} + w_{i,j}}, 0 \leq i \leq n-1; 0 \leq j \leq m-1 \\ & \frac{w_{i,j+2}P_{i,j+2} - 2w_{i,j+1}P_{i,j+1} + w_{i,j}P_{i,j}}{w_{i,j+2} - 2w_{i,j+1} + w_{i,j}}, 0 \leq i \leq n; 0 \leq j \leq m-2 \end{aligned} \quad (4.22)$$

The results are further improved by subdividing the surface once to obtain a smaller r . Besides, the weighted point method [88] is used to find a smaller convex hull for the surface. Using these points to compute the discrete forward difference points $Q_{i,j}$ is better.

After getting the tessellation factors for all patches and curves, a prefix sum is applied to compute the total number of vertices and allocated memory for both the vertex buffer and the index buffer. The offset for each surface in the two buffers can also be obtained. Since we will create transition regions on the patch later, we set the vertex number of each patch to be $tessfactor_x * tessfactor_y + additionalvers$. The number of additional vertices is set heuristically to make sure that there are sufficient spaces for all patches.

4.6 Mesh generation

Straightforward mesh generation according to the Bézier patch and curve tessellation factors cannot ensure the approximation error near the Bézier patch boundaries. The Bézier curves are all tessellated solely with its own information. It is obvious that the Bézier curve only reflects partial variation information of the Bézier patch it belongs to. Therefore, the tessellation intervals of Bézier curves are generally larger than that of the Bézier patches' along the same parameter direction. We should estimate a transition region with a small enough width to guarantee the approximation error between the interior region of each patch and its surrounding curves.

4.6.1 Transition region identification

In tessellation, the approximation error is estimated as the maximum distance from the tessellated polygon mesh to the parametric surface. Theoretically, with fixed approximation error and tessellation interval in one direction, we certainly can find a small enough tessellation interval in the other direction to ensure the approximation error. In Zheng and Sederberg's work [87], they mentioned that the two tessellation intervals in the two parameter directions of a surface are under a constrain function, as described by the inequality (4.16).

In this inequality, ε is a constant given by the user. If the tessellation interval in t direction: δ_t is fixed, this becomes a quadratic inequality with only one variable. Then the range of δ_s can be obtained by,

$$\delta_s = \frac{\sqrt{D_{st}^2 \delta_t^2 - D_{ss} D_{tt} \delta_t^2 + 8 D_{ss} \varepsilon \inf w(s, t)} - D_{st} \delta_t}{D_{ss}} \quad (4.23)$$

If the function under the square root is not negative, δ_s has a solution. To ensure that, the D_{ss} , D_{st} and D_{tt} should be computed with respect to the local region near this boundary curve. If using the ones with respect to the whole surface, δ_s may have no solutions. That

is mainly because the D_{ss} , D_{st} and D_{tt} are all continuous functions at the surface. If the approximation error at the curve can be guaranteed, there must be a solution at a small enough region near this curve.

4.6.2 Transition region creation

For all Bézier curves, the tessellation is performed according to the tessellation factors obtained. The central region of each patch is re-estimated using Equation (4.23) and then tessellated using the patch tessellation factors. To create the transition regions, we should find the tessellation vertices of surrounding curves and the outer layer vertices of the central region of the patch.

In T-spline surfaces, one Bézier patch may be surrounded by more than four Bézier curves. Previous stages possess the two diagonal corner point indices of each Bézier patch (Section 4.4.1) and the two end point indices of each Bézier curve (Section 4.4.2). With this information, it is not hard to fetch the surrounding vertices for each patch. Given the corner grid point indices of a patch, we can find all grid-point indices along the patch boundary. If a grid point is a T-point, we then find its location in the T-point list in the prefix sum array according to the T-point label array and the grid-point index, as shown in Figure 4.6. Afterwards, the Bézier curve vertices can be obtained in the vertex buffer according to their offset information and the number of vertices of each curve.

At last, the vertices of the surrounding curves are connected with the outer layer vertices of the patch interior region. It is noteworthy that the transition region is divided into four separate sections which are triangulated respectively. If taking the four sections as a whole, it may generate disordered triangles at patch corners. The four transition sections are generated using an approach similar to Bresenham's algorithm for drawing lines [89].

For relatively flat Bézier patches, the width of the transition region along one boundary or the sum of two opposite transition region is occasionally larger than 1. In this case, the triangulation inside the patch is performed in another pattern. The central region will be

Figure 4.10: Illustration of transition region creation

only a line or even a point. If both opposite transition regions is wider than 1, the two boundaries of this Bézier patch are connected directly without creating transition regions, as shown in Figure 4.11 (b). Our algorithm successfully prevents cracks without affecting

Figure 4.11: Illustration of the transition region creation with large shrink intervals. (a) the sum of opposite shrink interval > 1 and at least one shrink interval is < 1 , (b) both opposite shrink intervals > 1

the parallelism and only utilizes very little topological information that can be acquired easily.

4.7 Results and discussion

In this section, we show the crack-free tessellation effects as well as the interactive performance of our methods when tessellating large scale models. To realize interactive mesh

update when the tessellation densities are changed according to the view, the tessellation results are passed to a *VertexBuffer* and an *IndexBuffer* for tessellation. The buffers are OpenGL bounded resources on GPU. The update of vertices are performed using a kernel function which executes at per-vertex parallel. Similarly, the updates of triangles are run at per-triangle level.

We compare the algorithm execution time on different models to see how the model structure affects the performance. The tessellation intervals are computed using our new method. A comparison between the previous and the new tessellation method is also given.

4.7.1 Tessellation effect

Comparing the tessellation quality of our method and the one adopted in most CAD systems is done by showing two images of the same region at the same zoom-in level. In normal rendering, the artifacts can be seen visually along surface boundaries. Some systems improve this by tessellating the models in a very high density. The crack-problems can be ameliorated. However, the render will become much slower. Our methods are crack-free over the whole surface and can run in an interactive speed.

In one prior work the crack is well filled by object-space ray-casting [15]. The cracks should be detected with care to avoid wrongly report. On the other hand, the cracks are changed with the view variation. The whole routine need to re-run in this case. After the crack-filling, their rendering has almost equivalent effects with ours, and their algorithm is also applicable to trimmed boundaries. However, one superiority of our method is that we generate real watertight geometries without cracks. The approximation error is controlled to ensure that the whole mesh is smooth enough. Two examples are shown in Figure 4.12 and Figure 4.13. We can see that our mesh generation pattern stitches all adjacent meshes together without any cracks.

Table 4.1: Average step size of a set of Bézier patches with error tolerance 0.1

*	Zheng and Sederberg's method	our method	no. of patches our interval is larger
[-10, 10]	0.02475	0.02807	851
[-100, 100]	0.00698	0.00815	835
[-1000, 1000]	0.00225	0.00257	846

4.7.2 Comparison of tessellation factor estimation

To get the tessellation statistics of the overall performance, we run the algorithm on a set of randomly generated degree 3 rational Bézier surfaces. The x, y coordinates are randomly distributed in three intervals, [-10, 10], [-100, 100], [-1000, 1000], each interval contains 1000 surfaces. The weight of each control point is the ratio of two random numbers between 1 and 10,000. In most cases, our method is better. For all intervals, the average step size using our method is larger. Our scheme evaluates all vertices on a patch or curve in sequence. Hence less vertices to be evaluated can increase the performance to some extent. The results are shown in Table 4.1.

4.7.3 More examples

To test the performance, we apply the algorithm on several models with millions of control points on a NVIDIA Quadro K5000 GPU. The model statistics has been given in Chapter 3. The run time is broken down into each stage and listed in Table 4.2 and Table 4.3. The performance is highly related to the complexity of the T-meshes. In our experiment, we use three models with different T-mesh structures. The stegosaurus model only contains a single large T-spline surface. Both the car and the submarine models are composed of a host of small patches. The submarine model contains lots of T-junctions while the car is a models only containing NURBS surfaces.

At the stage of edge identification, the time is mainly determined by the edge length. To identify an edge, from one T-point, we need to find the other connected T-point by traversing all the grid-points along one direction. A similar situation occurs in the stage

Table 4.2: Break-down run time (ms) for each stage (without hash Functions)

model	preprocess	patch idt	patch cmp	cur idt	cur cmp	cur tess	patch tess
car(NURBS)	2.66	0.26	37.39	0.05	36.27	0.49	15.20
submarine(T-spline)	5.13	8.05	73.75	0.16	58.62	0.85	19.99
steg	0.67	1.05	8.54	0.07	6.54	0.05	0.96

Table 4.3: Break-down run time (ms) for each stage (with hash Functions)

model	preprocess	patch ident	patch comp	cur ident	cur comp	cur tess	patch tess
car(NURBS)	2.60	0.28	34.37	0.05	55.47	0.45	12.69
submarine(T-spline)	5.12	9.00	68.15	0.16	51.64	0.85	18.25
steg	0.67	1.048	2.831	0.065	1.847	0.047	0.96

of Bézier region identification. Threads may be divergent when identifying regions with different sizes. Taking the submarine model for instance, the refined model contains more T-junctions. Not surprisingly, it costs more time on those two stages. Though the NURBS car model is apparently larger than the stegosaurus, its Bézier edge and patch identification are actually very simple, thereby less time consuming. In the run time table, the transition region identification and generation are included in the patch tessellation stage.

The hash functions can improve the performance significantly only when there are large T-meshes inside the model. In this case, the check of control points over a Bézier curve or a Bézier patch is prohibitively time-consuming. For models that only contain small patches, the hash function does not show remarkable predominance. Therefore, if the model is composed of many small patches, using the hash function to pre-estimate the control points for each Bézier curve/patch is not necessary. By contrast, since the results of the hash functions are stored in the global memory, it may be more time-consuming to read data from global memories than directly checking the whole mesh in the kernel function. For the NURBS car model, using hash functions consumes more time in computing the control points of Bézier curves.

In the table, patch idt refers to Bézier patch region identification; patch cmp is to compute the control points of Bézier patches; cur idt refers to Bézier curve identification; cur

cmp is to compute the control points of Bézier curves; cur tess indicates Bézier patches tessellation; patch tess indicates Bézier curves tessellation .

Figure 4.12: (a) A car model, (b) cracks between several surfaces, (c) zoom in of those cracks, (d) the normal tessellation pattern, (e) the tessellation pattern using our algorithm, (f) our rendering effect.

Figure 4.13: (a) A submarine model, (b) cracks between several surfaces, (c) zoom in of those cracks, (d) the normal tessellation pattern, (e) the tessellation pattern using our algorithm, (f) our rendering effect.

Chapter 5

Parallel T-spline Multi-point Removal

5.1 Introduction

This chapter considers the problem of removing redundant control points from a compounded T-spline model, either keeping the shape unchanged or restricting the shape change within a given tolerance. The knot removal (or control point removal) is a fundamental operation in the NURBS and T-spline geometry, and often serves as a building block for other processes such as the surface simplification and decomposition. While the knot removal for a NURBS surface is performed in a simple pattern that causes the removal of a whole row or column of control points. The knot removal for a T-spline has been shown much more complicated.

There are mainly two previous methods for the T-spline simplification. The first one uses the iterative refinement [11]. From an over-simplified version of the original model, this method progressively inserts knots in the region where the error is greater than the given tolerance until a desired approximation to the original surface is achieved. This approach can generate a simplified surface with a quite regular topology. However, this is an approximation method and the topology of the final T-mesh may be far different from the T-mesh of the original T-spline surface. The second method is iterative simplification [42], which intends to remove those control points that can be removed and thus also keeps the

shape unchanged. It checks the whole T-mesh iteratively until none of the control points could be removed. Nonetheless, the method may fail to remove some control points even if they are created by knot insertions. Hence a challenging problem is: how to remove the points as many as possible. This problem is hard due to the following reasons:

- The number of extra point insertions caused by inserting one point varies. There are many different patterns [11].
- The removal of one point may require knot insertion, thus causing even more points in the final result.

In this chapter we propose a multi-point removal strategy and present an efficient knot removal algorithm on GPU.

5.2 Complexity of T-spline knot removal

The preliminaries of T-spline knot removal can be found in Section 2.2 of Chapter 2. In this section, we give some knot insertion and removal cases to demonstrate the complexity of the T-spline knot removal problem. As aforementioned, the complexity of T-spline knot removal comes from the following observations: firstly, the inserted points may not be removable; secondly, one control point may not be removed until some other control points are removed first; thirdly, different removing orders may lead to different results; and finally, the removal of one point may also cause extra knot insertions.

In the following example, inserting point No.0 into a T-spline surface causes insertion of totally 3 points (Figure 5.1(b)). Now we check all the three points. Unexpectedly, none of them is removable one by one. If we force to remove them one by one, error occurs and different single-point removal orders provide different results (see Table 5.1). Following the order 2,1,0, the removal of point No.2 (Figure 5.1(c)) causes an error of 0.04. Then, the error of removing No.1 is 0.13 (Figure 5.1(d)). Finally, the removing of point No.0 produces not only an error of 0.16 but also two new points (Figure 5.1(e)). Similar results

Figure 5.1: The single-point removal procedure cannot remove inserted control points introduced by refinement: (a) part of the pre-image, (b) inserting one control point introduces three control points, (c-e) knot removal following the order 2,1,0, and (f-h) knot removal following the order 0,1,2.

can be obtained for the order 0,1,2. All the 3 points can not be removed using the single-point removal method. After tracing the insertion and removal procedure we find that when inserting point No.0, the four shaded red points in Figure 5.2(a) have contributed to it. However, when removing it from the mesh, the residual is obtained from the four orange points as shown in Figure 5.2(b). In case the set of the control points contributing to the inserted point is different from the set of control points contributing to its residual, there is an extremely small chance that this point can be removed. For additionally inserted point No.1 and No.2, they should be not removable. Otherwise there is no need to introduce them when inserting point No.0.

Table 5.1: Removal results for Figure 5.1

Order	Error at No.0	Error at No.1	Error at No.2	Removable points
2,1,0	0.16	0.13	0.04	0
0,1,2	0.05	0.09	0.1	0
Multiple	4×10^{-7}	5×10^{-7}	5×10^{-7}	3

In some cases, one control point is not removable. However, after some control points on the T-mesh have been removed, it becomes removable again, as shown in Figure 5.3. In

Figure 5.2: (a) control points contribute to the newly inserted point No.0, (b) control points contribute to the residual when removing point No.0

another example (Figure 5.4), one point is removable at the beginning. However, after removing another point in the T-mesh, it becomes not removable. Therefore, the final simplification results are closely related to the removing order. This is mainly because after removing one point, both the topology and the geometry are changed at this local region. Besides, the knot removal may also introduce extra knot insertions. This has been

Figure 5.3: (a) a point is not removable, (b) the point can be removed after another point is removed, (c) the final T-mesh

demonstrated in the knot removal algorithm in Section 2.2 of Chapter 2.

All the above examples show the challenges and complexity of efficient knot removal. Previous point removal algorithms [42, 90] for T-splines are based on the single-point removal. The inserted points, especially the points successively generated by the refinement, may not be removed. Refer to Figure 5.1. To overcome this drawback, we turn to the

Figure 5.4: (a) a removable point, (b) removing another point from the T-mesh, (c) this point becomes not removable

multi-point removal in which we remove a group of points together. This raises another challenge: how to effectively form a set of points for one removal. This is still an open problem.

Currently, to identify all the removable control points on a T-mesh at a time is impossible. In general, all the T-mesh knot removal methods need many iterations to complete the process which is very time-consuming. Moreover, in each iteration, even only removing one point, all the control points on the T-mesh are checked one by one to see whether there are violations. For a model with thousands of control points, it may cost several minutes to complete the simplification process which is unacceptable in interactive operations. When applying the knot removal technique to industry applications, it should be highly efficient. Inspired by the great success achieved by exploring the power of GPU, we intend to design the removal algorithm to fit in CUDA architecture for the performance improvement. However, regarding to the challenges in the T-spline knot removal algorithm, different removal orders may lead to different removal results. Therefore, to make a problem be solvable in parallel is not straightforward, especially when the results depend on the removing order.

Moreover, the exact removal of control points is really a rare occurrence in common NURBS or T-spline surfaces in practice. As an extension of [42], an approximation method [90] is proposed to remove the restriction. To output a relatively regular control grid for an approximate T-spline, some criteria are also suggested. The edge removal directions for each

point are switched to avoid slender faces.

As a solution to these challenges, this chapter presents a GPU-based multi-point removal algorithm. Some strategies are proposed to complete the removal process using much fewer iterations, compared to both the single point removal algorithm [42] and the trial-and-error approach [90]. The general idea is to extend the single-point removal [42] to the multi-point removal which detects groups of control points that can be removed together. Since the insertion patterns are various depending on the T-mesh topology, we attempt to only find the most possible insertion groups on the T-mesh and try to remove them all at once. The group removal reduces the removing order dependency. Furthermore, we extend the method to approximate point removal to remove more points and meanwhile confine the surface change within a error tolerance. For compound T-spline models, special attention should be paid to the boundaries of adjacent surfaces, to avoid crack occurrence. The whole process is implemented in a series of parallel schemes on GPU. Though our algorithm still needs many iterations to complete the whole process, each iteration is significantly accelerated by harnessing the power of GPU. The number of iterations is greatly reduced by removing more points once.

5.3 GPU-based multi-point removal

While all previous single-point removal methods are realized on CPU, with regard to their long time-cost, the graphics hardware acceleration is worth an elaborate investigation. In this section we present our multi-point removal algorithm and elaborate how to parallelize it on GPU.

The main idea of the multi-point removal method is the removal groups. After removing a group of control points together, each control point will generate a residual. We say that the removal group is removable if all their residuals are zero. On a T-mesh, removing a group points one by one and removing them together can generate different results. The reason is that the set of control points contributing to each residual in single-point removal

is different from the group point removal. More specifically, if the point is removed one by one, the residual of previous removed control points will be fixed and won't change anymore no matter how many points are removed later. While in group removal, all the removed points will contribute to all the residuals.

In our approach, a local removal group is derived for each control point. The *local removal group* is identified as the possible insertion groups caused by inserting current control point. In each iteration, the multi-point removal method contains six phases:

- Computing the residual of each control point;
- Identifying the *local removal groups* for each control point;
- Computing the residual of each *local removal groups*;
- Assembling a global removal group for this iteration,
- Error evaluation after removal,
- Reselecting the global removal group according to all the residuals obtained.

The input data are the encoded T-mesh which has been explained in Chapter 3. All the above six phrases are executed in control point based parallel way except for the third phrase which runs in parallel based on local removal-group. At first, each thread copes with one control point to identify the local removal group with which it is associated; then we compute the residual of itself and its local removal group. The global removal group is then selected from all removable candidates (including control points and removal groups). After removing them from the T-mesh, the error over the whole surface is computed by resolving the violations of each polar label, thereby still in a control point based parallel way. Though each selected candidate is removable individually, we cannot guarantee that removing them together can also ensure a zero error. Therefore if the residuals located at some points are not zero. we exclude them from the global removal group and re-calculate the residuals. The last two steps are repeated until we get a qualified global removal group.

The key to realize knot removal in parallel is eliminating the data dependency between all the parallel units. In the following sections, we will explain the strategies used to avoid this problem.

5.3.1 Local removal group identification

In our approach, the local removal group is directly inferred from the topology of the T-mesh. From a point p , a removal group $\Omega(p)$ can be derived by:

- Step 1: find the four adjacent control points, each side by two.
- Step 2: estimate the range of the removal group.
- Step 4: find the faces along the central line and in range.
- Step 5: for each point in these faces, if it is horizontally or vertically connected to p , add it into $\Omega(p)$.

This is inspired from one knot insertion pattern. If we insert one point on an edge, it acts in the following manners. Along this edge, there are maximal four control points whose blending functions will be changed. According to the refinement rules, they will be refined to involve the new knot finally. In this process, the knot vectors in the other direction move associated with the refinement(the blue vertical knot vectors in Figure 5.5(a)). Starting from the outermost vertical knot vector, if it is finer than its adjacent inner knot vector, insert the missing knots into the inner one. On the contrary, if the inner one is finer, refine current blending function, which may introduce more control points. This comparison operation terminates at the central line that contains the inserted point(the red dotted line in Figure 5.5(a)). Moreover, newly introduced points start new insertion procedures. In Figure 5.5(b), the insertion of No.0 introduces point No.2; The insertion of No.2 introduces point No.1. The variety of the insertion patterns makes it difficult to exactly identify the set of control points introduced by the refinement. Inferring backward, in Figure 5.6, all the highlighted points or a subset of these points may be caused by inserting the point No.0.

Here we only select the points along the central line as a *local removal group*. The range of the group is then estimated as the outmost two knots of the knot vectors as shown in Figure 5.5(a). We cannot guarantee this kind of group can definitely be removed but with high potential.

Figure 5.5: The single-point insertion: (a) the point insertion process and the possible range of the insertion group, (b) inserting point No.0 introduce points No.1, 2.

Figure 5.6: A possible insertion pattern

Figure 5.7 shows two simple examples of the local removal groups. If the point p (in red) is previously inserted onto the edge, it is quite possible that all the points (in red and green) are inserted by refinement. In such a case, these points may be eliminated without changing the shape of the surface. For a point contained by both an s -edge and a t -edge, we can set

Figure 5.7: Identify a removal group: (a) a point on an s -edge; and (b) a point on a t -edge

Figure 5.8: A T-mesh refinement by inserting four control points (No.0 point in each figure)

up two different local removal groups from this point. In this case, the one with more points is selected. The single-point removal algorithm cannot remove all the numbered points in Figure 5.1. According to the statistics in Table 5.1, the multi-point removal algorithm can exactly remove them all. Thus, in such a case, our method is a reverse operation of the knot insertion.

In practice, the designers refine the models in order to add details. We argue that even if the insertion points are perturbed, the local removal group method is still effective. Firstly, the refinement may introduce much more control points than the plan, where there are many redundant points. Secondly, we provide an example to show that, even if the insertion groups are perturbed, the multi-point removal method is still more effective than the single-point removal method. In Figure 5.8, we insert four control points successively into the mesh. Four insertion groups are generated. Then we perturb some control points and simplify the mesh using the single-point removal and multi-point removal method. The results are shown in Figure 5.9. From Figures 5.9 (g) and (h), the identified removal groups may not be exactly the same as the original insertion pattern like Figure 5.8. This shows the flexibility of our algorithm.

Figure 5.9: The flexibility of the multi-point removal method. (a)(b)refined surface and its pre-image (Figure 5.8(d)), (c)(d) the surface after perturbation, (e)(f) the single-point removal results and its pre-image, (g)(h) the multi-point removal results and its pre-image

On GPU, each control point is assigned with a local group label to indicate whether a removal group is formed from it. Spaces are allocated for each of them to store the control point indices within its group.

5.3.2 Removal error assessment

This stage is to find all the potential removable points on the whole mesh.

The removal error is evaluated by the residual which is proposed and explained in [42]. In the knot removal, reverse blending function transformation is an atomic operation. After removing one control point, each blending function containing the removed knot is decomposed into two new ones:

$$B_{i1}(u) = \frac{1}{c_1} B_i(u) - \frac{c_2}{c_1} B_{i2}(u). \quad (5.1)$$

The decomposing process continues until all the blending functions match the new T-mesh. In this process, the knot insertion or the blending function refinement may be also needed. For the blending functions that have the same knot vector with the removed point, we add them to the removed point and get a residual. If and only if the residual is zero, the point can be exactly removed. The detail of the residual computation can be found in Section 2.2.3 of Chapter 2.

To reduce the divergency between threads, when computing the residual of each single point and each local removal group from this point, it is split to two kernel functions. Firstly, the program runs in control point based parallel and then in local removal group based parallel. Each thread will find and resolve all the blending function violations after removing a single point or a local group. In our implementation, one problem is that all the control points on the same T-mesh will read the T-mesh information in the global memory which is shared by all the threads. However, none of them can modify the shared data. When trying to compute the residual, we need to update the connection types and the coordinates of some control points in this process. We let each thread record their own changed

control points with their indices in the registers or the local memory. In case these data will be further used, search on the local memory first. If nothing is found on the local memory, then fetch the data on the global memory.

5.3.3 Global removal group assemble

In this step, we are going to assemble a new removal group using those candidates for one removal.

The candidates include all the single points and all the identified removal groups. All candidates with no intersection will be assembled into a new removal group. Since the domain of a removal group definitely has intersections with the domain of single points within it, it is impossible to select a single point and a removal group containing it in the same new removal group.

The strategy in GPU implementation is that, there is a removal tag array in the global memory, which records whether a control point will be removed in this iteration. Each thread deals with one removable candidate. To find a set of removal candidates without intersection is not easy to realize on GPU since all the candidates would like to change the removal tag array simultaneously. Previously, it is implemented on CPU by determining whether one candidate intersects with the already selected ones. On GPU we implement this using a lock with two CUDA functions: `atomicCAS` and `atomicExch`. When a thread is accessing the global memory, all the other threads will be locked. Until the current thread completes its write to the global memory, other threads can have the opportunity to modify the global memory. This kind of implementation almost convert the parallel process to a sequential one. However, there is no need to copy the data to CPU for calculation and then pass back to GPU. After a thread obtains the lock, before writing to the removal tag array, it must determine whether its candidate has intersections with existing removal candidates. If one of its control points has been removed by other threads, the current removal candidate is discarded in current iteration. There is one issue of current CUDA architecture for this

lock implementation. All threads cannot be run in the same warp. However, this limitation caused very limit influence in our test.

val = 0;

Lock : *while*(*atomicCAS*(&(val), 0, 1) != 0);

Unlock : *atomicExch*(&(val), 0);

5.3.4 Global removal group update

After confirming the group of control points that will be eliminated at once, a prefix sum is applied to obtain the total number of removal points and then spaces are assigned to store their residuals. We remove them from the T-mesh and change the connection types of related control points. This removal is done in control point based parallel. It is known that each point has a connection type value c in the T-mesh, with 0x0001 indicating its connection along $s-$ direction; 0x0010 indicating $s+$; 0x0100 indicating $t-$ direction; and 0x1000 indicating $t+$ direction. If we want to remove the edge on its $s+$ direction, an *and* operation is applied as $c \& 0x1101$. In the multi-point removal, several removal points may remove the same edge of a point. Here *atomicAnd* function in CUDA is used. The topology modification runs in control point based parallel based on the removal tag array obtained in the previous step.

After getting the updated T-mesh, for each control point, we check whether its blending function violates the mesh topology. If yes, we resolve it to match the new T-mesh. In the previous single-point removal method, after one point removal, all the blending functions on the mesh will be checked one by one. Here all of them are processed by the multiprocessors simultaneously which are significantly accelerated. This time we still do not change the initial T-mesh data, but record the topology changes in a copy of the T-mesh data. Maintaining a copy of the initial T-mesh data is for possible future global removal since we may reselect the global removal group again.

Though each selected candidate is removable individually, we cannot guarantee that

removing them together can also generate a zero error. If the residual is not zero at some removed control points or the removal of some points causes new insertions, we exclude them from the removal group and then re-calculate the residuals. The exclusion is done by changing the label in the removal tag array to be 0.

5.4 Approximate multi-point removal

Since the exact removal is only under rare circumstances, we ease the restriction by allowing the approximate removal. After removal, the surface error can be estimated as the distance between the original shape and the approximated shape [38]. Such a method is somehow computationally expensive. Instead, we use residuals to evaluate the perturbation error directly. By removing a removal group Ω , each point has a residual. The T-spline surface,

$$P(s, t) = \sum_{i=0}^n P_i B_i(s, t) \quad (5.2)$$

can be re-formulated as

$$P(s, t) = \sum_{i \notin \Omega} \bar{P}_i \bar{B}_i(s, t) + \sum_{j \in \Omega} Q_j \bar{B}_j(s, t) = \bar{P}(s, t) + \bar{Q}(s, t), \quad (5.3)$$

where $\bar{P}_i$ are control points after the knot removal, Q_j are residuals at the removed points after the knot removal, $\bar{B}_i(s, t)$ are blending functions matching the T-mesh after knot removal, $\bar{P}(s, t) = (x(s, t), y(s, t), z(s, t), w(s, t))$ is the T-spline after knot removal, and $\bar{Q}(s, t) = (\bar{x}(s, t), \bar{y}(s, t), \bar{z}(s, t), \bar{w}(s, t))$ is the residual for Ω . According to (5.3), the error is given by

$$\bar{Q}(s, t) = \sum_{j \in \Omega} Q_j \bar{B}_j(s, t), \quad (5.4)$$

Since the blending function $\bar{B}_j(s, t)$ is defined over a rectangular domain D_j , the perturbation error yielded by $Q_j \bar{B}_j(s, t)$ is smaller to Q_j and is confined to D_j . If m domains cover

the same sub-domain (Figure 5.10), the error over this sub-domain is

$$error_{overlap} = \sum_{r=1}^m Q_r. \quad (5.5)$$

The maximum error of all sub-domains is the error for Ω , which is confined to the domain $D_\Omega = \cup_{j \in \Omega} D_j$.

The method in [90] restricts $\bar{x}, \bar{y}, \bar{z}, \bar{w}$ within a given tolerance ε . We want to point out two facts:

- Compared to the change in $\bar{x}, \bar{y}, \bar{z}$, the change of the weight $\bar{w}$ affects more on the change of the surface in $\mathbb{R}^3$,
- (Eq.5.3) gives $w(s, t) + \bar{w}(s, t) = \sum w_i B_i(s, t)$, where w_i is the weight of P_i . If the T-spline is in the polynomial form as $\sum w_i B_i(s, t) = 1$, $\bar{w}(s, t) \neq 0$ will turn the T-spline into a rational form with $w(s, t) \neq 1$.

Therefore, in the knot removal algorithm, it is better to use a much smaller tolerance for $\bar{w}$. Our removal algorithm sets the tolerance for $\|(\bar{x}, \bar{y}, \bar{z})\|$ and $\bar{w}$ as ε and 0.1ε . Moreover, for polynomial T-splines, we only remove the points with $\bar{w} = 0$ to keep it in a polynomial form.

Figure 5.10: Two intersecting domains (dash lines are from the pre-image).

To make the resulted mesh be regular with few slender or irregular faces we introduce a regularity measurement. The regularity is a linear combination of two components: the

regularity of the pre-image R_1 and the regularity of the T-mesh R_2 as

$$regularity = \alpha R_1 + \beta R_2, \quad (5.6)$$

where α, β are two parameters. R_1 reflects the quality of the parametrization, different parameterizations lead to different values. R_2 is controlled by the geometry of the T-mesh. Each component is computed as the mean value of all length-width ratios of its faces:

$$R_x = \frac{1}{n} \sum_{i=1}^n length_i / width_i. \quad (5.7)$$

On a face of a T-mesh, the two opposite edges in a direction (s -direction or t -direction) may be of different length. In Figure 5.11, the left one is considered as more regular than the right one due to that the mean length-width ratio of the left one is smaller than that of the right one. I-junctions are not allowed in the T-mesh since a single line in the T-mesh is not intuitive for editing. As pointed out in [90], the single-point removal method [42] usually removes edges and points following the same direction. Thus it reduces the regularity of the pre-image and the T-mesh. Our method can overcome this drawback. Users can manually set values for α and β . Increasing the value of α achieves better results on pre-image and increasing the value of β reduces the number of slender faces on the T-mesh.

Figure 5.11: The T-mesh regularity

Similar to the exact removal, the candidates also include all the single points and all the identified removal groups in the mesh. An array is allocated to record the error in the global memory on each grid region. After one removal iteration, the error will be accumulated. If

the accumulated error exceeds the tolerance, we exclude the control points that affect this region from the global removal group and re-calculate the error again.

Besides, to avoid gaps between surfaces, our method prohibits the knot removal near the boundary. Particularly, the points that define the boundary.

5.5 Results and discussion

In this section, we will present some experiments to demonstrate the effectiveness of our algorithm. All the results were obtained on a machine with Intel Xeon CPU at 2.5 GHz and an NVIDIA Quadro K5000 video card with memory bandwidth 173 GB/s. Some of the models are obtained from the T-spline company [91]. We have further refined them to test the performance. In the following, we will compare the removal number of exact single-point removal (CPU) with exact multi-point removal (GPU), as well as their run time. The run time is broken down at each step to show the dominating computation in our algorithm. We also provide examples to verify the effective of the regularity control. Like previous single point removal algorithms, our multiple-point removal algorithm is also a heuristic and iterative method. However, we are able to remove more control points in each iteration. Besides, based on the parallel implementation, each iteration can be completed very fast.

In the following examples, both the single point and multi-point removal methods were employed. We can see that the multi-point removal obtains better results than single-point removal, in both the removal point number and the run time. Figure 5.12 is an example where the T-spline has 1148 control points. Figure 5.12(c,d) are the results obtained by applying the single-point removal method and the multi-point removal method. The statistics on the number of removed points, the number of iterations and their run time are collected in Table 5.2. Compared to the single-point removal, our new method performs more than a factor of thousands faster and can remove more control points from the T-mesh. Figure 5.13 examines another compound T-spline model that contains 9170 control points and consists of 42 surfaces. The statistics in Table 5.2 show that under the same condition, the

Figure 5.12: Single-point removal vs. multi-point removal (a) original model with 1148 control points, (b) pre-image of the original model, (c) exact single-point removal results: 1057 control points, (d) exact multi-point removal results: 1028 control points (Some removal groups are circled out compared to single-point removal).

multi-point removal can remove more points with fewer iterations. The execution has been significantly accelerated in each iteration.

In Figure 5.14 and Figure 5.15, the full algorithm was executed on two models. One thing I'd like to mention is that, the input model of Figure 5.14 is an approximate T-spline model. Its not the original fandisk model. The T-spline fandisk does not have sharp edges. The control points of the first model are reduced by 164 out of 1969. The car body is simplified from 36317 control points to 33172 control points. The execution time is broken down for only one iteration. We do not show the run time of other iterations because the

Figure 5.13: Single-point removal vs. multi-point removal (a) original model with 9170 control points, (b) after exact single-point removal: 8354 control points, (c) after exact multi-point removal: 8321 control points, (d) the pre-image of the front surface on the model, (e) pre-image of the surface after exact single-point removal, (f) pre-image of the surface after exact multi-point removal.

dominant step is always the residual computation of single points and local removal groups. The run time of all steps in one iteration decreases gradually as the number of control points or removable points reduces. For models with a large number of control points, this part takes even more time. For example, the jar model contains much more control points than the stegosaurus model, but it is composed of many small patches. Therefore, the residual computation for the jar model is much faster than that for the stegosaurus model. Actually this part is not that GPU-friendly. Each thread computes the residual of a control point or a removal group. In the knot removal algorithm, we need an array to store all the violate blending functions with each blending function containing ten float numbers

Table 5.2: Run-time comparison: exact single-point removal (CPU) vs. exact multi-point removal(GPU)

Figure	Removal type	Removed points	Iterations	Run-time
Figure 5.12(c)	single-removal	91	91	more than 20 mins
Figure 5.12(d)	multi-removal	120	18	about 32s
Figure 5.13(e)	single-removal	816	816	more than 30mins
Figure 5.13(f)	multi-removal	849	16	about 22s

Figure 5.14: Exact multi-point removal (a) original model with 1969 control point,(b) multi-removal result with 1805 control points

(knot vector in the two directions). Moreover, all the changed points are stored in local registers for each thread. The registers are always overused and the over flow data will be carried to other memories. This will slow down the whole process. Besides, removing points from surfaces with a very huge number of control points, much time is cost on check violation blending functions. Nevertheless, the multi-point removal in conjunction with the GPU implementation could achieve considerably better performance compared to previous single-point removal method, in both run time and removing numbers. We also run the same algorithm while allowing an approximation error tolerance. In Figure 5.16, the model is simplified using different error tolerance.

In Figure 5.17, the multi-point removal is applied using a tolerance of 0.01. Three sets of values are used for regularity parameters (α, β) : $(1, 0)$, $(0.5, 0.5)$, and $(0, 1)$. Figures 5.17(b-d) show the same portion of the result T-mesh. In Figure 5.17(b), only the pre-image regularity is taken into account, and thus there may be some irregular faces on

(a)

(b)

Figure 5.15: Exact multi-point removal (a)original car model with 36317 control points . (b) multi-point removal result with 33172 control points (only car body is considered, not including the wheels)

the T-mesh. As we increase regularity value, the irregular faces can be removed (see Figures 5.17(c,d)).

Our proposed parallel multi-point removal scheme successfully prevents the data dependency problems in knot removal process. This is due to the following strategies adopted. In case several threads need to read and update the original T-mesh data, such as the residual computation step and the point removal step from the T-mesh topology, we save a copy of the changed data and retain the original data. Besides, knot insertion in the removal process is prohibited in our algorithm. All the kernel functions hand over the intermediated

Table 5.3: Run-time (ms) of each stage (first iteration)

Figure	compute single- point residual	identify removal group	compute local group residual	identify fi- nal group	group re- moval
Figure 5.12	126.11	0.235	35.17	0.184	21.04
Figure 5.13	27.67	0.612	10.30	0.225	91.38
Figure 5.14	135.20	0.262	39.2	0.17	202.85
Figure 5.15	991.0	0.81	158.35	0.87	124.53

results on device. Only the number of removal points used to allocate residual memory will be passed to the host. In the algorithm, sometimes many threads write to the same global memory address which is inevitable. For example, when removing the final group of control points from the T-mesh topology, an *atomicAnd* operation is applied to change the connection type of related point. Also in the final group removal, several threads may contribute to the same control points or residual, where *atomicAdd* is used. Since atomic operations' chief feature is locking the affected memory location until the operation is completed, too much writing to the same address can turn parallel algorithm into poorly performing sequential processes. Fortunately, only very limit number of threads write to the same location in our algorithm.

Figure 5.16: Approximate model simplification (a) the original model with 1148 control points, (b) the simplified model with 1008 control points (error tolerance 0.001), (c) the simplified model with 982 control points (error tolerance 0.01), (d) the simplified model with 728 control points (error tolerance 0.1).

Figure 5.17: The same portion of the T-mesh after knot removal (a) the T-spline surface, (b) $(\alpha, \beta) = (1, 0)$, (c) $(\alpha, \beta) = (0.5, 0.5)$, and (d) $(\alpha, \beta) = (0, 1)$.

Chapter 6

Construction of T-spline Surfaces with Complicated Boundaries

6.1 Introduction

Surface extension to a target curve while preserving the parameterizations and geometry of the original surface is a useful technique, which allows users to create new shapes from existing ones. Examples of the applications are surface repairing [92] and gap filling among surface patches [93]. Many surface extension algorithms are based on NURBS surfaces. In the extension, the extended surface boundary and the target curve should be consistent in both control points and parameterizations. Shetty and White [94] presented an extension algorithm, in which a point is taken as the connection between the original surface and the target curve. Hu et al. [95] presented another algorithm which starts from a clamped rational B-spline surface, interpolates the target curve by surface unclamping, and then clamps the new boundary again. Pan et al. [96] presented an explicit extension solution for curves or surfaces with arbitrary knot vectors. This method will be adopted in our T-spline surface extension algorithm.

NURBS has limitations in its overall parameter domain, which must be a rectangle. T-splines [97] is more flexible. It is possible to represent a model with complex boundaries

using a single T-spline surface while several surfaces are need if NURBS is used (see Figure 6.1). Meanwhile the local refinable property of T-splines allows designers to modify a local area of the model which helps to reduce the size of the data set and accelerate the design process as well.

Figure 6.1: Designing models with complicate boundaries (a) using a set of NURBS surfaces, (b) using one complex boundary T-spline surface, (c) one possible pre-image of the T-spline surface.

This chapter presents a scheme for interactively designing complex boundary T-spline surfaces (CBT-splines). By drawing or presenting a set of curves, a T-spline surface can be extended to interpolate the set of curves, while maintaining the shape and parameterization of original surface. The connection between original surface and the new part achieves C^2 continuity. Different from NURBS, partial boundary can be selected on a T-spline surface. The NURBS surface extension will be explained in detail in Section 6.2. The NURBS surface extension to a target curve can be decomposed to a set of curve extension to a target point. However, several practical questions arise when dealing with T-spline surfaces. Due to its generality of T-mesh topology, it is not straightforward to convert the T-spline

surface extension problem into a set of curve extension problems. The control points on the same row/column do not always have the same blending functions in the other parameter direction. Our objective is hence to develop a method to perform T-spline surface extension while keeping the original surface unchanged after the surface extension.

The remainder of this chapter is organized into four sections. Section 6.2 describes some preliminary knowledge of NURBS surface extension. Section 6.3 explains the algorithm of T-spline surface extension. Moreover, to construct T-spline surfaces with complex boundaries, we also introduce surface trimming in case users want to include holes in the surface. In section 6.4, the implementation details are discussed.

6.2 Preliminaries

6.2.1 Knot modification of NURBS curves

Given $n + 1$ control points P_i forming a control polygon, weights w_i , degree k and a knot vector $T = \{s_0, \dots, s_{n+k+1}\}$, where $s_0 \leq s_1 \leq \dots \leq s_{n+k+1}$ are the knots, a rational B-spline curve of order $k + 1$ is defined by

$$C(s) = \frac{\sum_{i=0}^n w_i P_i N_i^k(s)}{\sum_{i=0}^n w_i N_i^k(s)} \quad (6.1)$$

The domain of the curve is $[s_k, s_{n+1}]$, while other knots outside the domain are called exterior knots. The exterior knot can be freely replaced while keeping the curve unchanged [96]. If a multiple knot s_i has multiplicity k (assume $s_i = \dots = s_{i+k-1}$) then the B-spline curve interpolates control point P_{i-1} . If $s_1 = \dots = s_k$, the B-spline curve interpolates the first control point. If $s_{n+1} = \dots = s_{n+k}$, the B-spline curve interpolates the last control point. If the B-spline curve interpolates the end control points, it is said to meet the Bézier end condition.

Suppose the knot vector of the curve is changed to $T = \{s'_0, \dots, s'_{k-1}, \dots, s_{n+k+1}\}$. In order to keep the curve shape unchanged, we first identify the newly generated control

points and then get them by applying affine combinations of original control points. In this process, the results may not be achieved by one iteration. Some intermediate polar values will be computed first. In particular, the control points of the curve can be re-calculated as:

$$\begin{aligned}
 & \text{for}(i = 1; i \leq k - 1; i++) \{ \\
 & \quad \text{for}(j = 0; j \leq k - 1 - i; j++) \{ \\
 & \quad \quad (w_j P_j, w_j) = \frac{s_{j+k+1} - s'_{k-i}}{s_{j+k+1} - s_{j+i}} (w_j P_j, w_j) \\
 & \quad \quad + \frac{s'_{k-i} - s_{j+i}}{s_{j+k+1} - s_{j+i}} (w_{j+1} P_{j+1}, w_{j+1}) \\
 & \quad \quad \} \\
 & \quad \} \\
 & \}.
 \end{aligned}$$

Similarly, if the right hand exterior knots are changed to $T = \{s_0, \dots, s_{k-1}, \dots, s'_{n+k-1}, \dots, s'_{n+k+1}\}$, the control points are updated by

$$\begin{aligned}
 & \text{for}(i = 1; i \leq k - 1; i++) \{ \\
 & \quad \text{for}(j = 0; j \leq k - 1 - i; j++) \{ \\
 & \quad \quad (w_{n-j} P_{n-j}, w_{n-j}) = \frac{s_{n+k+1-i-j} - s'_{n+1+i}}{s_{n+k+1-i-j} - s_{n-j}} (w_{n-j-1} P_{n-j-1}, w_{n-j-1}) \\
 & \quad \quad + \frac{s'_{n+1+i} - s_{n-j}}{s_{n+k+1-i-j} - s_{n-j}} (w_{n-j} P_{n-j}, w_{n-j}) \\
 & \quad \quad \} \\
 & \quad \} \\
 & \}.
 \end{aligned}$$

6.2.2 NURBS surface extension

For NURBS surface extension to a target curve, the control points on the extended boundary and the control points on the target curve should be one-to-one corresponded. The NURBS

surface is defined as:

$$S(s, t) = \frac{\sum_{i=0}^m \sum_{j=0}^n P_{ij} w_{ij} N_i^k(s) N_j^l(t)}{\sum_{i=0}^m \sum_{j=0}^n w_{ij} N_i^k(s) N_j^l(t)} \quad (6.2)$$

or using homogenous representation,

$$S(s, t) = \sum_{i=0}^m \sum_{j=0}^n (P_{ij} w_{ij}, w_{ij}) N_i^k(s) N_j^l(t) \quad (6.3)$$

where $N_i^k(s)$, $N_j^l(t)$ are the basis functions associated with the point P_{ij} . To interpolate the target curve, the NURBS control mesh is decomposed into a set of rows/columns. Then the surface extension can be convert to a set of B-spline curve extension problem. Consider the problem of extending the curve defined by each row/column to interpolate its corresponding control point on the target curve. If we assume the extension is along t parameter direction, the curve defined by the i_{th} column on the mesh is:

$$c_i(t) = \sum_{j=0}^n (P_{ij} w_{ij}, w_{ij}) N_j^l(t) \quad (6.4)$$

After extension, each curve adds one control point. t_{end} is the new boundary parameter of the extended surface and it is set to be triple knots to meet the Bézier end condition. The new curve will interpolate the i_{th} control point Q_i on the target curve $C_{target}(s)$:

$$c_i(t_{end}) = \sum_{j=0}^{n+1} (P_{ij} w_{ij}, w_{ij}) N_j^l(t_{end}) = Q_i \quad (6.5)$$

Then the extended surface at the end of t parameter direction is equal to the target curve. The following equation demonstrates this.

$$\begin{aligned} C_{target}(s) &= \sum_{i=0}^m c_i(t_{end}) N_i^k(s) \\ &= \sum_{i=0}^m \sum_{j=0}^{n+1} (P_{ij} w_{ij}, w_{ij}) N_j^l(t_{end}) N_i^k(s) \\ &= S_{extend}(s, t_{end}) \end{aligned} \quad (6.6)$$

6.3 Construction of T-spline surfaces with complicated boundaries

This section explains how to construct T-spline surfaces with complex boundaries. To effectively manage the boundaries, we first propose a special data structure for T-spline surface with complex boundaries. Then we describe the T-spline surface extension algorithm, after which trimming and local refinement are explained.

6.3.1 Structure of complicated boundary T-spline

To construct a T-spline surface with complex boundaries, two layers of loops and a set of virtual boundary coefficients are appended to each normal T-spline surface:

- a T-spline surface,
- a set of domain boundary loops,
- a set of mesh boundary loops,
- a set of virtual boundary values.

Besides the geometry information of the T-spline surface, the domain boundary is used to determine the domain region of the T-spline surface. It contains (row_{id}, col_{id}) of all N-points and T-points along the domain boundary. In the surface evaluation, it is used to decide at which parameter vertices are generated. The mesh boundary is only used in implementation. It is obtained as one ring neighbor of the domain boundary which is also a set of row and column indices. Each pair of row and column indices indicates a point (T-point or N-point) on the T-mesh. Figure 6.2 gives an example of T-spline surfaces with a complicate boundary. The points on mesh boundary and domain boundary are labeled in green and red respectively. The solid rectangles are T-points while the empty ones indicate N-points. The outer loop indices are listed in counterclockwise direction while the inner

Figure 6.2: Domain boundary loop and mesh boundary loop.

loop is in clockwise direction. The mesh boundary must be a continuous connected loop. After T-spline surface extension, both the domain boundary and the mesh boundary are updated.

To extract the polar label from each control point, the method in [97] is adopted. This has been described in Section 2.2 of Chapter 2. To handle the boundary of the T-mesh when extracting polar label properly, at most two virtual boundaries are supplemented around the T-mesh boundary. We also set two virtual coefficients for each boundary of the T-mesh.

After extension, it should be kept in mind that the virtual knots of the new surface should follow the following criteria. Firstly, for a rectangular T-spline surface, the virtual knots can be selected arbitrarily as long as keeping an ascending order of the global knot vector. For T-spline surfaces with arbitrary boundaries, special attentions should be paid to the concave or holes. In those regions, the mesh boundary may not be the outmost edge of the T-mesh. In order to isolate the boundaries, we should not let the virtual knots of current boundary overlap the next adjacent surface boundary. Otherwise, the control points on the current mesh boundary will have influence on next surface boundary which is unexpected in the CBT-spline surface definition. Secondly, since the target curves are all B-spline curves, it

also has virtual knots at its two ends. The virtual knots of the target curve should be the same as the extended surface at that boundary. Otherwise, the surface will not interpolate the target curve. Thirdly, the surface at the extended boundary is set to have triple knots to meet the Bézier end condition. In this case, only the control points on the target curve define the new surface boundary. Except those control points, any other control points can not have influence on the new boundary. For instance, if the boundary is partially selected for extension, for the non-extend part along this boundary, its virtual knots should not exceed the target curve. Otherwise, there are extra control points influencing the new boundary other than the target curve control points, which makes the extended surface not interpolate the target curve.

6.3.2 T-spline surface extension algorithm

Our proposed T-spline surface extension procedure consists of four steps:

- Specify the boundary to be extended and the target curve;
- Make the parameter range of the selected boundary and the target curve be consistent;
- Build a new surface part that connects the original surface and interpolates the target curve;
- Perturb related control points to keep the original surface unchanged.

In the first step, the user specifies a part of the surface boundary and a B-spline curve that will be interpolated. To do the extension, only the iso-parameter boundary can be selected. The boundary crossing the corner of one patch is not allowed as shown in Figure 6.3. In the second step, since the target curve will be the boundary of the newly generated surface, it should have the same parameter range as the selected boundary. Theoretically, the linear scaling or translating of knots will not affect the shape of a B-spline curve. Based on this concept, we re-parameterize the target B-spline curve to match the selected boundary.

Given a curve with knot vector $T = \{s_0, s_1, s_2, s_3, s_4, s_5\}$, its domain is $[s_2, s_3]$. To map it to the parameter domain $[a, b]$, we do scaling and translation to the curve:

$$T' = T \times \frac{b-a}{s_3-s_2} = \left\{ \frac{s_0(b-a)}{s_3-s_2}, \frac{s_1(b-a)}{s_3-s_2}, \frac{s_2(b-a)}{s_3-s_2}, \frac{s_3(b-a)}{s_3-s_2}, \frac{s_4(b-a)}{s_3-s_2}, \frac{s_5(b-a)}{s_3-s_2} \right\} \quad (6.7)$$

$$s_{offset} = a - \frac{s_2(b-a)}{s_3-s_2}$$

$$\begin{aligned} T_{new} &= T' + s_{offset} \\ &= \left\{ \frac{s_0(b-a)}{s_3-s_2} + s_{offset}, \frac{s_1(b-a)}{s_3-s_2} + s_{offset}, a, b, \right. \\ &\quad \left. \frac{s_4(b-a)}{s_3-s_2} + s_{offset}, \frac{s_5(b-a)}{s_3-s_2} + s_{offset} \right\} \end{aligned} \quad (6.8)$$

Figure 6.3: Extension boundary selection (a) a T-spline surface and a target curve, (b) legal iso-parameter boundaries (c) illegal selected boundaries

Now the new domain of the curve becomes $[a, b]$. To absorb the target curve as a part of the surface, three more steps are needed. Firstly, to make sure the two end control points of the target curve be connected to the mesh, we may need to perturb them. Otherwise, they will become I-junctions on the T-mesh. Their parameters are set to be the parameters of the two most adjacent T-points outside of the selected boundary. After parameter change, the two boundary control points are adjusted via the method described in Section 6.2.1. Secondly, the knot vector of the original surface is extended by one knot. The extended knots can be any value, as long as it follows the virtual knot setting rules mentioned in Section 6.3.1.

For example, the new knot vector can be $\{s_0, s_1, \dots, s_{n+k-1}, s_{n+k}, s_{n+k+1}, s_{n+k+1} + coe1\}$ with domain $[s_k, s_{n+k-1}]$ or $\{s_0, s_1, \dots, s_{n+k-2}, s_{n+k-2} + coe1, s_{n+k-2} + coe1 + coe2, s_{n+k-2} +$

$coe1 + coe2 + coe3, s_{n+k-2} + coe1 + coe2 + coe3 + coe4\}$ with domain $[s_k, s_{n+k-2} + coe1]$. Then the interpolation of the target curve can be obtained by repeating the end knots s_{n+k-1} k times. Then the knot vector is changed to $\{s_0, s_1, \dots, s_{n+k-1}, \dots, s_{n+k-1}, s_{n+k-1} + coe1\}$ or $[s_k, s_{n+k-1}]$ or $\{s_0, s_1, \dots, s_{n+k-2}, s_{n+k-2} + coe1, s_{n+k-2} + coe1, s_{n+k-2} + coe1, s_{n+k-2} + coe1 + coe2\}$. Finally, the control points of the target curve are set as the last row/column of the extended surface.

After the extension, we guarantee that the surface interpolates the target curve. However, the virtual knots and the boundary knot of the original T-mesh may have been changed. Consequently the polar label of some control points may be changed. The surface shape will certainly have some perturbations. Our objective is to recover the original surface shape after boundary knots are changed.

The solution is to resolve the unmatched blending functions to fit the new knots. One way to this problem is the blending function splitting and reverse splitting. However, to resolve one unmatched blending function, sometimes both splitting and reverse splitting are needed. After the blending function decomposition, some blending functions out of the domain should be discarded. This process is very tedious. This situation happens when the point is at the original domain boundary. For example, the knot vector of one boundary point is changed from $[5, 6, 7, 8, 9]$ to $[5, 6, 7, 8.5, 9]$. The extended domain is $[7, 8.5]$. To simplified this, we divide the whole process into two stages. The first stage solves the violations caused by boundary knots changes; the second stage processes the violations inside the surface domain which is exactly the same as [97].

In the first stage, we only consider the polar values violation of all control points. That is based on the theory that if the exterior knots of a NURBS curve/surface are replaced, the new control points can be obtained using the simple affine combination of related old control points. The curve/surface shape is preserved as explained in Section 6.2.

On a B-spline curve/surface, if the exterior knots are changed, at most two layers of control points are affected. When applied to the T-spline surface, the two layers of control points become two layers of polar points along the extension direction. In the following ex-

ample (The two outmost knots are omitted in the knot vector since they won't influence the surface shape.), using the first extension knots, only one row of control points are affected (the shaded ones in Figure 6.4(b)); using the second knots, two layers will be affected (the shaded ones in Figure 6.4(c)). In the T-mesh, polar points refer to the points that contribute to polar labels. The polar points can be N-point or T-point. For T-points, their positions may be recalculated; for N-points, they may become T-points after perturbation. For all those control points whose polar values are changed, we recalculate them using the affine combination of old control points as mentioned in Section 6.2. In the computation, some adjacent polar points are involved. For example, in t direction the point $P_0(0.4, 0.6, 1.2)$ is

Figure 6.4: T-mesh extension with violation points in the shaded area. (a) the original T-mesh with knot vector $[-0.2, 0, 0.2, 0.22, 0.29, 0.34, 0.4, 0.6, 1.2]$ in t direction and its virtual boundaries, (b) the extended T-mesh with knot vector $[-0.2, 0, 0.2, 0.22, 0.29, 0.34, 0.4, 0.6, 0.6, 0.6]$ in t direction and its virtual boundaries, (c) the extend T-mesh with knot vector $[-0.2, 0, 0.2, 0.22, 0.29, 0.34, 0.4, 0.7, 0.7, 0.7]$ in t direction and its virtual boundaries.

changed to $P'_0(0.4, 0.6, 0.6)$ after the extension. P'_0 can be obtained by affine combination of $P_1(0.3, 0.4, 0.6)$ and $P_0(0.4, 0.6, 1.2)$. By doing this, we can resolve the polar value violation of P_0 . Using this method, we resolve all the polar label violations caused by exterior knots change.

However, on the T-mesh, each control point is associated with two polar labels. The polar label in s direction is also accompanied with the affine computation. This may result in polar label violations inside the domain. Therefore, after all the "exterior knots viola-

tion” are resolved, we turn to check the ones inside the mesh. In the second stage, all the violations will be resolved using the method of [11]:

- If a blending function is missing a knot dictated by current T-mesh, perform blending function splitting,
- If a blending function contains a knot not dictated by current T-mesh, insert an appropriate control point,
- Repeat above two steps until there is no more violations.

Therefore, it is possible to introduce new insertions into the mesh. In Figure 6.4(b), after extension, only one row control points on the original mesh boundary is changed. The affine combination only involves the second layer control points. No polar label violation inside the domain is introduced. However, in Figure 6.4(c), two layers are affected. In the affine combination, the third layer is also involved, which introduces the newly inserted points as shown in Figure 6.5(c).

6.3.3 Surface trimming

Surfaces with complicated boundaries are more flexible for designing complicated objects. To make the scheme more versatile, we add the trimming function, which is performed in the following procedure.

The trimmed domain is a closed rectilinear region sketched by the user. After eliminating all the control points and edges inside the trim region, the trimmed boundary emerges as a new domain boundary. The simple trimming may generate lots of I-junctions along the new generated domain boundary. To form a closed boundary loop, the original mesh is first refined at all the intersections of the trimming loop and the T-mesh edges, as well as the corner points of the trimming loop. The refinement will not change the surface shape but may introduce additional control points into the mesh. Based on the domain boundary obtained, we find the one ring loop of control points bounded by the domain boundary on the

Figure 6.5: T-spline surface extension results of Figure 6.4 (a) the original T-spline surface corresponding to Figure 6.4(a), and a target curve, (b) the extended surface corresponding to Figure 6.4(b), (c) the extended surface corresponding to Figure 6.4(c). (d)(e)(f)corresponding index graphes of (a)(b)(c).

T-mesh and take it as the newly generated mesh boundary. In order to make the trimming as a local operation, the inner one ring loop is not estimated on the original mesh. We insert one ring of control points with a small offset to the domain boundary. The whole procedure is illustrated in Figure 6.6.

6.3.4 Local refinement

On the complex boundary T-spline, the knot refinement must be given some special treatment. If we perform knot refinement as in the normal T-mesh and then remove the points in trimmed areas, there may be points occurring opposite trimmed area which is not expected. Since the fundamental operation of the knot refinement is the blending function refinement, we handle this problem by ignoring those blending functions that have no overlap with T-mesh domain in the refinement process. For control points having violation of virtual

Figure 6.6: Trimming of T-spline surfaces. (a) trimming loop(red lines), (b) refine the mesh to obtain the domain boundary (red points), (c) refine to get mesh boundary (blue lines and red points), (d) remove the trimmed points

knots outside the mesh, the violation will be handled using the polar value violation solving method as explained in the first stage of Section 6.3.

6.4 Experimental results

The input file contains the geometry information of the T-mesh and a set of domain boundary loops. The data structure of the T-mesh geometry data includes all the control points of the T-mesh, their connection types and the knot vectors of the T-spline surfaces. The domain boundary is a list of (row_{id}, col_{id}) . Given a domain boundary, the mesh boundary can be estimated easily as the one-ring neighbor of the domain boundary. Then all the points on the mesh boundary are labeled as mesh boundary. This label will be used for the polar label extraction. When encountering a mesh boundary point, corresponding virtual knots will be fetched. At each boundary, two virtual knots are assigned. Their values are dynamically adjusted after the surface is extended or trimmed. This is to make sure that the virtual knots setting meets the requirements mentioned in Section 6.3.1.

In the first example (Figure 6.7), we compare the original T-mesh and the extended T-mesh. The extended T-mesh is colored in black (or blue) while the original T-mesh is in red. The yellow points are newly inserted ones in the extended T-mesh. In Figure 6.7(b), to show the results more clearly, the original surface is shown in wire frame and the extended surface is smoothly shaded. We can see that they are exactly coincident at the original part.

Another example is given in Figure 6.8 where a surface is extended from its four directions to interpolate four target curves. Its pre-image is shown in Figure 6.8(c).

There are some cases where the extension may encounter obstacle control points. In the following example, the surface is extended in the right direction first and then in the top direction. The circled control points in Figure 6.9(b) block the way of the second extension. Those control points should be moved to the side. Since they correspond to an exterior knot along the row, they can be perturbed while maintaining the original shape of the surface. They will be perturbed according to the criteria mentioned in Section 6.3.

Finally we present an example that extends the surface from a trimmed hole. The extension from a hole is more complicated than the extension from outer boundaries. The difficulty lies on the fact that even setting triple knots at the target curve, the surface may still not interpolate the curve. This is because except the control points on the target curve, other control points of the T-spline surface may also have influence on it (the two points in yellow circles in Figure 6.10(f)). In order to let the surface interpolate the target curve, we have to reset the control points on the target curve. It should be $cur_{target} - cur_{extra}$. Figure 6.10(b) shows the result of direct extension without subtracting the influence of extra points. We can see that the surface does not interpolate the target curve. While in Figure 6.10(c), the points on the target curve are reset to $cur_{target} - cur_{extra}$, making the surface interpolate it perfectly. When the surface is extended inside a concave or a hole, the boundary knots will be set to be triple knots. In this case, the continuity of surface along this knots may be higher than it should be. The fan model in Figure 6.11 is constructed using a set of patches similar to Figure 6.10(c).

Figure 6.7: A T-spline surface extension example. (a) the original T-mesh and the extended T-mesh, (b) the original surface and the extended surface, (c) separate the original surface and the extended surface.

Figure 6.8: A T-spline surface extension. (a) the original surface and four target curves, (b) the extended surface, (c) its pre-image

Figure 6.9: T-spline surface extension that needs the shrink of a mesh boundary. (a) the original T-mesh, (b) surface extension along s direction, (c) shrink some boundary control points, (d) insert some necessary control points, (e) surface extension along t direction.

Figure 6.10: T-spline surface extension from a hole

Figure 6.11: Propeller model design using Complex Boundary T-splines.

Chapter 7

Summary and Future Work

7.1 Summary

In this thesis we have studied four geometric processes for compound T-spline surfaces. The four geometric processes are rational Bézier extraction from T-spline surfaces, crack-free adaptive T-spline tessellation, T-spline multi-point removal and complicated boundary T-spline construction. These are fundamental processes that are less explored before. They will enlarge the family of T-spline geometric processing. Our research objectives are to gain better understanding of these processes and develop effective and efficient techniques and algorithms for them. All these objectives have been fulfilled. In particular, we have completed the following research works.

First, we have presented a method for extracting Bézier patches from T-spline surfaces. While some previous work sometimes may not extract Bézier patches correctly, our method can extract all the Bézier patches correctly. Our method also minimizes the number of the extracted Bézier patches as much as possible. This helps reduce the size of data for various applications such as T-spline rendering and iso-geometric analysis, which use Bézier patches as elements. Moreover, our method is designed to fit the CUDA parallel architecture and entirely implemented on GPU. As a result, our method can achieve real time extraction even for large scale T-spline models. One problem of our method is, if the knot

lines are very intensive, it may generate numerous Bézier patches. This problem may be alleviated by approximating many small Bézier patches using a single patch. This will further minimize the number of Bézier patches generated but the parallel structure of the algorithm will be affected. More investigations are needed to handle this problem.

Second, we have proposed a novel framework for tessellating compound T-spline surfaces. The underlying techniques involve boundary merging, surface decomposition, tessellation estimation and mesh generation. Except for the boundary merging which can be viewed as a preprocess, all others are designed to be GPU-friendly. We improve the tessellation factor estimation for rational Bézier curves and surfaces, and design parallel strategies for curve and patch tessellation and mesh generation. As a result, our method can adaptively tessellate T-spline models into watertight triangular meshes in real time on GPU. The triangular meshes are guaranteed to approximate the T-spline models within the given tolerance. This process can be used for surface visualization, 3D model representation conversion, etc. One drawback of our method is memory intensive since we need to store the control points of all Bézier elements. Besides, just like the first work, if the knot vectors of the T-mesh are very intensive, it may generate numerous Bézier patches/curves.

Third, we have studied the T-spline knot removal problem extensively. We find that T-spline knot removal is much more difficult than B-spline knot removal, which is somewhat surprising. Based on the analysis of several T-spline knot removal patterns, we propose a T-spline multi-point removal algorithm that removes a group of points simultaneously. Compared to a single-point removal, the T-spline multi-point removal is more effective. We also consider the approximate T-spline multi-point removal, which can be used to simplify T-spline models. In our multi-point removal procedure, only a simple knot insertion pattern is considered for constructing the potential removal group. That's why the improvement of our method compared to single-point removal is not so significant. Actually the result depends on how many of those patterns the model contains. Another drawback of our method is, in the implementation, each thread consumes too much resource. It may limit the GPU performance. Currently, we haven't figured out a better solution. However, the

CPU algorithm takes orders of magnitude longer than our GPU algorithm.

Fourth, we have developed techniques for extending T-splines from partial boundaries, which results in T-spline models with complicated boundaries. This can alleviate the workload of handling cracks in compound models and reduce the number of surfaces used for modeling complicated shapes. We provide T-spline surface extension, trimming and splitting tools for the user to perform interactive design. The user can extend the surface to interpolate a curve or trim part of the surface. These operations keep the original parts unchanged. The complex boundary T-splines enhance the representation power of T-splines in modeling complicated shapes including those with complicated boundaries or holes. Our extension method has not considered more complex shapes, such as multiple extension from one boundary, closed shapes and models with genus more than 0.

7.2 Future Work

The research described in this thesis also opens venues for future investigation.

In our B-rep model tessellation algorithm, we only consider B-rep models that do not have trimming curves. Since trimmed surfaces are common in B-rep models, it is very useful to find a solution to generating a seamless mesh for trimmed surfaces while maintaining approximation accuracy. Previously, though many works have been done to handle the cracks along trimming curves, most of them are done in image space. The cracks are filled using ray-casting or additional geometries. To generate a seamless polygon mesh for trimmed surfaces, how to identify the tessellation density of the trimming curves and how to connect them to their adjacent patches are two problems that need to be investigated.

As for the simplification of T-spline models, the pattern of T-spline knot insertion depends on the T-mesh topology and thus is very complicated. The removal pattern still has many other possibilities. Hence many other possible removal group patterns can be explored in future. Moreover, the removing result depends on the removal order. We can not guarantee that the simplification results in the minimal number of control points. It may

be possible to develop more effective extensions if we can gain more insights of the knot insertion patterns and the knot removal process.

Another interesting research is the construction of arbitrary topology T-splines. Many objects in the real-world are of complex topology. Previous modeling methods employ segmentation and patching to represent a model using a set of surfaces. It is cumbersome to maintain the continuity between surfaces. In tessellation, they will have a risk of producing gaps. An appealing property of polygon meshes and subdivision surface is that they create watertight models with arbitrary topology. It would be nice if we could convert these meshes to a compact, explicit spline representation while satisfying smoothness requirements. It is thus of great interest theoretically and practically to have a smooth surface scheme that can represent objects with arbitrary topology and meanwhile is locally refinable.

There are a lot of literatures published on tri-variate volumetric T-spline in recent years [98, 99, 100]. In the future, its also feasible to extend the above algorithms to tri-variate volumetric T-splines, but with more complexity. For the first work, we may extract Bézier cubes from tri-variate volumetric T-spline. Previously, the Bézier volume extraction is obtained by knot insertion which is most likely time consuming and tedious. Our method is a parallel method and much faster. One possible difficulty when extracting Bézier volume is partitioning irregular volume to a minimal number of cubes. The second work is similar to the first work. Instead of extracting Bézier curves and Bézier patches, we need to extract Bézier patches and Bézier cubes. The difficulty still lies in the irregular volume partition. The point removal (T-mesh simplification) can also be applied to T-spline volume. Here if one point is removed, we need to handle the polar label violations in three parameter directions. To develop multi-point removal method on T-spline volume, we need to identify the insertion patterns in T-spline volume. They should be much more complicated than T-spline surfaces. The T-spline surface extension will become T-spline surface/volume extrusion in the tri-variate case. After extension, to maintain the original T-spline volume unchanged, we need to resolve polar label violations in three parameter directions and to be

more careful when setting the exterior knots of the volume.

References

- [1] T. W. Sederberg, G. T. Finnigan, X. Li, H. Lin, and H. Ipson, “Watertight trimmed nurbs,” *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 1–8, 2008.
- [2] I. Stroud, *Boundary representation modelling techniques*. Springer Science & Business Media, 2006.
- [3] J. Peters, “Patching catmull-clark meshes,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 255–258.
- [4] Y. He, K. Wang, H. Wang, X. Gu, and H. Qin, “Manifold t-spline,” in *Geometric Modeling and Processing-GMP 2006*. Springer, 2006, pp. 409–422.
- [5] X. Li, J. Deng, and F. Chen, “Surface modeling with polynomial splines over hierarchical t-meshes,” *Visual Computer*, vol. 23, no. 12, pp. 1027–1033, 2007.
- [6] F. Kahlesz, Á. Balázs, and R. Klein, “Multiresolution rendering by sewing trimmed nurbs surfaces,” in *Proceedings of the seventh ACM symposium on Solid modeling and applications*. ACM, 2002, pp. 281–288.
- [7] L. A. Piegl and W. Tiller, *The NURBS book*. Berlin ; New York : Springer, c1997. 2nd ed., 1997.

-
- [8] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs, “Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement,” *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 39-41, pp. 4135–4195, 2005.
- [9] Y. Bazilevs, V. M. Calo, J. A. Cottrell, J. A. Evans, T. J. R. Hughes, S. Lipton, M. A. Scott, and T. W. Sederberg, “Isogeometric analysis using t-splines,” *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 5-8, pp. 229–263, 2010.
- [10] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, “T-splines and t-nurccs,” *ACM Transactions on Graphics*, vol. 22, pp. 477–484, 2003.
- [11] T. W. Sederberg, D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, and T. Lyche, “T-spline simplification and local refinement,” *ACM Transactions on Graphics*, vol. 23, pp. 276–283, 2004.
- [12] M. Eck and H. Hoppe, “Automatic reconstruction of b-spline surfaces of arbitrary topological type,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 325–334.
- [13] T. DeRose, M. Kass, and T. Truong, “Subdivision surfaces in character animation,” in *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*, 1998, pp. 85–94.
- [14] M. Schwarz and M. Stamminger, “Fast gpu-based adaptive tessellation with cuda,” in *Computer Graphics Forum*, vol. 28, no. 2. Wiley Online Library, 2009, pp. 365–374.
- [15] F. Claux, L. Barthe, D. Vanderhaeghe, J.-P. Jessel, and M. Paulin, “Crack-free rendering of dynamically tessellated b-rep models,” in *Computer Graphics Forum*, vol. 33, no. 2. Wiley Online Library, 2014, pp. 263–272.

-
- [16] G. Barequet and S. Kumar, "Repairing cad models," in *Proceedings of Visualization'97*. IEEE, 1997, pp. 363–370.
- [17] J. Chhugani and S. Kumar, "View-dependent adaptive tessellation of spline surfaces," in *Symposium on Interactive 3 D Graphics: Proceedings of the 2001 symposium on Interactive 3 D graphics*, vol. 2001, 2001, pp. 59–62.
- [18] G. R. Kumar, P. Srinivasan, K. Shastry, and B. Prakash, "Geometry based triangulation of multiple trimmed nurbs surfaces," *Computer-Aided Design*, vol. 33, no. 6, pp. 439–454, 2001.
- [19] W. A. Stöger *et al.*, "Watertight tessellation of b-rep nurbs cad-models using connectivity information," in *Proceedings of the International Conference on Imaging Science, Systems and Technology, CISST*, vol. 2, 2003.
- [20] M. Guthe, A. Balázs, and R. Klein, "Gpu-based trimming and tessellation of nurbs and t-spline surfaces," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1016–1023, 2005.
- [21] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray tracing on programmable graphics hardware," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 703–712, 2002.
- [22] T. Foley and J. Sugerman, "Kd-tree acceleration structures for a gpu raytracer," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. ACM, 2005, pp. 15–22.
- [23] L. O. Simonsen, "A comparison of acceleration structures for gpu assisted ray tracing," Master's thesis, University of Aarhus, 2005.
- [24] M. Nießner, B. Keinert, M. Fisher, M. Stamminger, C. Loop, and H. Schäfer, "Real-time rendering techniques with hardware tessellation," in *Computer Graphics Forum*. Wiley Online Library, 2015.

- [25] Y. I. Yeo, L. Bin, and J. Peters, “Efficient pixel-accurate rendering of curved surfaces,” in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 2012, pp. 165–174.
- [26] E. G. Houghton, R. F. Emmett, J. D. Factor, and C. L. Sabharwal, “Implementation of a divide-and-conquer method for intersection of parametric surfaces,” *Computer Aided Geometric Design*, vol. 2, no. 1, pp. 173–183, 1985.
- [27] N. M. Patrikalakis, “Surface-to-surface intersections,” *IEEE Computer Graphics and Applications*, no. 1, pp. 89–95, 1993.
- [28] Y. Bazilevs, V. M. Calo, J. A. Cottrell, J. A. Evans, T. Hughes, S. Lipton, M. Scott, and T. Sederberg, “Isogeometric analysis using t-splines,” *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 5, pp. 229–263, 2010.
- [29] R. Concheiro, M. Amor, E. J. Padrón, and M. Doggett, “Interactive rendering of nurbs surfaces,” *Computer-Aided Design*, vol. 56, pp. 34–44, 2014.
- [30] Á. Balázs, M. Guthe, and R. Klein, “Fat borders: gap filling for efficient view-dependent lod nurbs rendering,” *Computers & Graphics*, vol. 28, no. 1, pp. 79–85, 2004.
- [31] R. N. Goldman and T. Lyche, *Knot insertion and deletion algorithms for B-spline curves and surfaces*. SIAM, 1993, vol. 36.
- [32] G. Farin, *Curves and surfaces for CAGD : a practical guide*. San Francisco, CA : Morgan Kaufmann ; London : Academic Press. 5th ed., 2002.
- [33] E. Cohen, T. Lyche, and R. Riesenfeld, “Discrete b- splines and subdivision techniques in computer-aided geometric design and computer graphics,” *Computer Graphics and Image Processing*, vol. 14, no. 2, pp. 87–111, 1980.
- [34] W. Boehm, “Inserting new knots into b-spline curves,” *Computer-Aided Design*, vol. 12, no. 4, pp. 199–201, 1980.

-
- [35] L. Ramshaw, “Blossoms are polar forms,” *Computer Aided Geometric Design*, vol. 6, no. 4, pp. 323–358, 1989.
- [36] M. Eck and J. Hadenfeld, “Knot removal for b-spline curves,” *Computer Aided Geometric Design*, vol. 12, no. 3, pp. 259–282, 1995.
- [37] W. Tiller, “Knot-removal algorithms for nurbs curves and surfaces,” *Computer-Aided Design*, vol. 24, no. 8, pp. 445–453, 1992.
- [38] T. Lyche and K. Morken, “Knot removal for parametric b-spline curves and surfaces,” *Computer Aided Geometric Design*, vol. 4, no. 3, pp. 217–230, 1987.
- [39] T. Lyche, E. W. C. C. K. Chui., and L. L. S. (eds.), *Approximation Theory VII*. Boston: Academic Press, 1993, ch. Knot removal for spline curves and surfaces, pp. 207–226.
- [40] Y. Wang and J. Zheng, “Adaptive t-spline surface approximation of triangular meshes,” in *6th International Conference on Information, Communications & Signal Processing*. IEEE, 2007, pp. 1–5.
- [41] J. Zheng, Y. Wang, and H. Seah, “Adaptive t-spline surface fitting to z-map models,” in *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. ACM, 2005, pp. 405–411.
- [42] Y. Wang and J. Zheng, “Control point removal algorithm for t-spline surfaces,” *Geometric Modeling and Processing*, pp. 385–396, 2006.
- [43] X. Li, J. Zheng, T. W. Sederberg, T. J. R. Hughes, and M. A. Scott, “On linear independence of t-spline blending functions,” *Computer Aided Geometric Design*, vol. 29, pp. 63–76, 2011.
- [44] E. Catmull and J. Clark, “Recursively generated b-spline surfaces on arbitrary topological meshes,” *Computer-Aided Design*, vol. 10, no. 6, pp. 350–355, 1978.

-
- [45] C. Loop and S. Schaefer, “Approximating catmull-clark subdivision surfaces with bicubic patches,” *ACM Transactions on Graphics*, vol. 27, no. 1, p. 8, 2008.
- [46] C. Loop, S. Schaefer, T. Ni, and I. Castaño, “Approximating subdivision surfaces with gregory patches for hardware tessellation,” *ACM Transactions on Graphics*, vol. 28, no. 5, p. 151, 2009.
- [47] J. A. Gregory, “Smooth interpolation without twist constraints,” *Computer Aided Geometric Design*, pp. 71–88, 1974.
- [48] X. Song, T. W. Sederberg, J. Zheng, R. T. Farouki, and J. Hass, “Linear perturbation methods for topologically consistent representations of free-form surface intersections,” *Computer Aided Geometric Design*, vol. 21, no. 3, pp. 303–319, 2004.
- [49] X. Gu, Y. He, and H. Qin, “Manifold splines,” *Graphical Models*, vol. 68, no. 3, pp. 237–254, 2006.
- [50] B. G. Baumgart, “Winged edge polyhedron representation,” DTIC Document, Tech. Rep., 1972.
- [51] I. Braid, *Notes on a geometric modeller*. University of Cambridge, 1980.
- [52] H. Chiyokura, *Solid modeling with DESIGNBASE: theory and implementation*. Addison-Wesley Longman Publishing Co., Inc., 1988.
- [53] S. Pavanaskar and S. McMains, “Filling trim cracks on gpu-rendered solid models,” *Computer-Aided Design*, vol. 45, no. 2, pp. 535–539, 2013.
- [54] E. Kilgariff and R. Fernando, “Gpu gem 2,” *Santa Clara (CA): Addison-Wesly*, vol. 5, 2006.
- [55] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

- [56] T. Kanai, “Fragment-based evaluation of non-uniform b-spline surfaces on gpus,” *Computer-Aided Design and Applications*, vol. 4, no. 1-4, pp. 287–294, 2007.
- [57] A. Krishnamurthy, R. Khardekar, S. McMains, K. Haller, and G. Elber, “Performing efficient nurbs modeling operations on the gpu,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 4, pp. 530–543, 2009.
- [58] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, “A survey of general-purpose computation on graphics hardware,” in *Computer graphics forum*, vol. 26, no. 1. Wiley Online Library, 2007, pp. 80–113.
- [59] C. Cuda, “Programming guide,” 2012.
- [60] C. Nvidia, “Nvidia cuda compute unified device architecture programming guide,” *NVIDIA Corporation*, 2007.
- [61] R. L. Cook, L. Carpenter, and E. Catmull, “The reyes image rendering architecture,” in *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4. ACM, 1987, pp. 95–102.
- [62] M. Fisher, K. Fatahalian, S. Boulos, K. Akeley, W. R. Mark, and P. Hanrahan, “Diagsplit: parallel, crack-free, adaptive tessellation for micropolygon rendering,” *ACM Transactions on Graphics*, vol. 28, no. 5, p. 150, 2009.
- [63] M. Kesson, “Pixar’s renderman,” in *ACM SIGGRAPH ASIA 2008 courses*. ACM, 2008, p. 16.
- [64] D. Wexler, L. Gritz, E. Enderton, and J. Rice, “Gpu-accelerated high-quality hidden surface removal,” in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. ACM, 2005, pp. 7–14.
- [65] A. Patney and J. D. Owens, “Real-time reyes-style adaptive surface subdivision,” in *ACM Transactions on Graphics*, vol. 27, no. 5. ACM, 2008, p. 143.

- [66] —, “Real-time reyes: Programmable pipelines and research challenges,” *ACM SIGGRAPH Asia 2008 Course Notes*, 2008.
- [67] K. Zhou, Q. Hou, Z. Ren, M. Gong, X. Sun, and B. Guo, “Renderants: interactive reyes rendering on gpus,” *ACM Transactions on Graphics*, vol. 28, no. 5, p. 155, 2009.
- [68] C. Woodward, “Ray tracing parametric surfaces by subdivision in viewing plane,” in *Theory and practice of geometric modeling*. Springer, 1989, pp. 273–287.
- [69] J. T. Kajiya, “Ray tracing parametric patches,” in *Proceedings of the 9th annual conference on Computer graphics and interactive techniques*. ACM, 1982, pp. 245–254.
- [70] D. L. Toth, “On ray tracing parametric surfaces,” *ACM SIGGRAPH Computer Graphics*, vol. 19, no. 3, pp. 171–179, 1985.
- [71] T. Nishita, T. W. Sederberg, and M. Kakimoto, “Ray tracing trimmed rational surface patches,” *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 4, pp. 337–345, 1990.
- [72] I. Wald, T. J. Purcell, J. Schmittler, C. Benthin, and P. Slusallek, “Realtime ray tracing and its use for interactive global illumination,” *Eurographics State of the Art Reports*, vol. 1, no. 3, p. 5, 2003.
- [73] H. F. Pabst, J. P. Springer, A. Schollmeyer, R. Lenhardt, C. Lessig, and B. Fröhlich, “Ray casting of trimmed nurbs surfaces on the gpu,” in *IEEE Symposium on Interactive Ray Tracing 2006*. IEEE, 2006, pp. 151–160.
- [74] A. Schollmeyer and B. Fröhlich, “Direct trimming of nurbs surfaces on the gpu,” *ACM Transactions on Graphics*, vol. 28, no. 3, p. 47, 2009.
- [75] R. Wu and J. Peters, “Correct resolution rendering of trimmed spline surfaces,” *Computer-Aided Design*, vol. 58, pp. 123–131, 2015.

- [76] E. Houghton, R. Emnett, J. Factor, and C. Sabharwal, “Implementation of a divide-and-conquer method for intersection of parametric surfaces,” *Computer Aided Geometric Design*, vol. 2, no. 1-3, pp. 173–183, 1985.
- [77] N. Patrikalakis, “Surface-to-surface intersections,” *Computer Graphics and Applications, IEEE*, vol. 13, no. 1, pp. 89–95, Jan 1993.
- [78] M. Scott, M. Borden, C. Verhoosel, T. Sederberg, and T. Hughes, “Isogeometric finite element data structures based on bzier extraction of t-splines,” *International Journal for Numerical Methods in Engineering*, vol. 88, no. 2, pp. 126–156, 2011.
- [79] M. Schwarz and M. Stamminger, “Fast gpu-based adaptive tessellation with cuda,” *Computer Graphics Forum*, vol. 28, no. 2, pp. 365–374, 2009.
- [80] T. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, “T-splines and t-nurccs,” *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 477–484, 2003.
- [81] R. Chadha and D. Allison, “Decomposing rectilinear figures into rectangles,” 1988.
- [82] J. E. Hopcroft and R. M. Karp, “An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs,” *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.
- [83] “cudpp open source library,” <http://cudpp.github.io/>, accessed: 2010-09-30.
- [84] T. W. Sederberg, “Computer aided geometric design,” Online class note, 2006, <http://tom.cs.byu.edu/557/text/ch1.pdf>.
- [85] D. Filip, R. Magedson, and R. Markot, “Surface algorithms using bounds on derivatives,” *Computer Aided Geometric Design*, vol. 3, no. 4, pp. 295–311, 1986.
- [86] F. Cheng, “Estimating subdivision depths for rational curves and surfaces,” *ACM Transactions on Graphics*, vol. 11, no. 2, pp. 140–151, 1992.

- [87] J. Zheng and T. W. Sederberg, “Estimating tessellation parameter intervals for rational curves and surfaces,” *ACM Transactions on Graphics*, vol. 19, no. 1, pp. 56–77, 2000.
- [88] G. Farin, “Tighter convex hulls for rational bézier curves,” *Computer Aided Geometric Design*, vol. 10, no. 2, pp. 123–125, 1993.
- [89] H. Moreton, “Watertight tessellation using forward differencing,” in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*. ACM, 2001, pp. 25–32.
- [90] L. David, “T-spline simplification,” Master’s thesis, Department of Computer Science, Brigham Young University, 2007.
- [91] T. splines company, “<http://www.tsplines.com/>.”
- [92] K. Bian and Y. Ke, “Topology recovery technique for complex freeform surface model after local geometry repair,” *Chinese Journal of Mechanical Engineering*, vol. 26, no. 1, pp. 197–206, 2013.
- [93] Y. Zang, Y.-J. Liu, and Y.-K. Lai, “Note on industrial applications of hu’s surface extension algorithm,” in *Advances in Geometric Modeling and Processing*. Springer, 2008, pp. 304–314.
- [94] S. Shetty and P. White, “Curvature-continuous extensions for rational b-spline curves and surfaces,” *Computer-Aided Design*, vol. 23, no. 7, pp. 484–491, 1991.
- [95] S.-M. Hu, C.-L. Tai, and S.-H. Zhang, “An extension algorithm for b-splines by curve unclamping,” *Computer-Aided Design*, vol. 34, no. 5, pp. 415–419, 2002.
- [96] J. Pan, J. Zheng, and G. Zhao, “Blind watermarking of nurbs curves and surfaces,” *Computer-Aided Design*, vol. 45, no. 2, pp. 144–153, 2013.

-
- [97] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, “T-splines and t-nurccs,” *ACM transactions on graphics*, vol. 22, no. 3, pp. 477–484, 2003.
- [98] W. Wang, Y. Zhang, G. Xu, and T. J. Hughes, “Converting an unstructured quadrilateral/hexahedral mesh to a rational t-spline,” *Computational Mechanics*, vol. 50, no. 1, pp. 65–84, 2012.
- [99] Y. Zhang, W. Wang, and T. J. Hughes, “Conformal solid t-spline construction from boundary t-spline representations,” *Computational Mechanics*, vol. 51, no. 6, pp. 1051–1059, 2013.
- [100] L. Liu, Y. Zhang, T. J. Hughes, M. A. Scott, and T. W. Sederberg, “Volumetric t-spline construction using boolean operations,” *Engineering with Computers*, vol. 30, no. 4, pp. 425–439, 2014.

Author's Publications

1. Y. Li, W. Chen, J. Zheng and Y. Cai, "T-spline multi-point removal", *Proceedings of Asian Conference on Design and Digital Engineering (ACDDE 2011)*, Shanghai, China, August 2011, pp.266-271.
2. W. Chen, Y. Li and J. Zheng and J. Pan and Y. Cai, "T-splines in VRML", *Proceedings of International Conference on Virtual Reality Continuum and its Applications in Industry (VRCAI'11)*, Hong Kong, China, December 2011, pp.257-264.
3. W. Chen, Y. Li, J. Pan, J. Zheng and Y. Cai, "Effective T-spline representation for VRML", *The International Journal of Virtual Reality*, Vol.11, No.1, 2012, pp.15-24.
4. Y. Li, W. Chen, J. Zheng, A. Nasri and Y. Cai, "Surface skinning using periodic T-spline in semi-NURBS form", *Journal of Computational and Applied Mathematics*, 273, 2015, pp.116-131.
5. Y. Li, X. Lu, W. Zhang and G. Wang, "Adaptive NURBS Tessellation on GPU" *GPU Computing and Applications*, Springer, Singapore, 2015, pp.69-84.
6. Y. Li and J. Zheng, "GPU-friendly Bezier extraction from T-spline surfaces", To be submitted to *International Journal for Numerical Methods in Engineering*, 2016
7. Y. Li and J. Zheng, "Crack-free rendering of B-rep models on GPU", To be submitted.
8. Y. Li and J. Zheng, "Construction of T-spline Surfaces with Complicated Boundaries", To be submitted.