

Topologically Reliable Approximation of Curves and Surfaces

by

Wonjoon Cho

M.S. in Ocean Engineering, M.I.T., June, 1992

M.S. in Naval Architecture, Seoul National University, February, 1988

B.S. in Naval Architecture, Seoul National University, February, 1986

Submitted to the Department of Ocean Engineering
in partial fulfillment of the requirements for the degree of


Doctor of Philosophy

at the

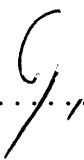
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1997

© Massachusetts Institute of Technology 1997. All rights reserved.

Author 
Department of Ocean Engineering
January 31, 1997

Certified by
Nicholas M. Patrikalakis
Professor of Ocean Engineering
Thesis Supervisor

Accepted by 
J. Kim Vandiver
Chairman, Departmental Committee on Graduate Students

Eng.

APR 29 1997

LIBRARIES

Topologically Reliable Approximation of Curves and Surfaces

by

Wonjoon Cho

Submitted to the Department of Ocean Engineering
on January 31, 1997, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Many application programs driven from modern CAD/CAM systems often operate on approximate lower order representations of the exact geometric definition. These applications include visualization, fundamental geometric calculations such as integrals, finite element or boundary element meshing for analysis and data exchange between various geometric modelers or design and fabrication systems. For this reason, a robust piecewise linear approximation method of high order and procedural curves and surfaces is necessary.

This thesis presents a piecewise linear approximation method of edges and faces of complicated boundary representation (B-rep) models within a user specified geometric tolerance. The proposed method uses numerically robust interval geometric definitions and computations and also addresses the problem of topological consistency between the exact geometry and its approximation. These are among the most important outstanding issues in geometry approximation problems. The possible inappropriate intersections of the approximating elements are efficiently checked and removed by local refinement of the approximation to guarantee a homeomorphism between the exact geometry and its approximation. We extract important differential geometric features of input geometry for use in the approximation. Our surface tessellation algorithm is based on the unstructured Delaunay mesh approach which leads to an efficient adaptive triangulation. A robustness problem in the conventional Delaunay triangulation is discussed and robust decision criteria are also introduced to prevent possible failures in the Delaunay test. To satisfy the prescribed geometric tolerance, an efficient node insertion algorithm is utilized and furthermore, a new efficient method to compute a tight upper bound of the approximation error is proposed. Unstructured triangular meshes for free-form surfaces frequently involve triangles with high aspect ratio. Our proposed surface triangulation algorithm constructs 2D triangulation domains which sufficiently preserve the shape of triangular elements when mapped into 3D space and furthermore, the algorithm provides an efficient method explicitly controlling the aspect ratio of the triangles on the triangulation domain.

Thesis Supervisor: Nicholas M. Patrikalakis

Title: Professor of Ocean Engineering

Acknowledgments

This thesis is dedicated to my wife Kunyoung, whose love and encouragement have made it possible. I also thank my parents for their love and support throughout my life. Especially, I would like to dedicate this thesis to my father who passed away during my doctoral exam. I would also like to share my joy with other members of my family.

I express my sincere gratitude to my advisor, Professor Nicholas M. Patrikalakis for his support and patient guidance for the last three years. I am also grateful to my thesis committee members, Professor Jaime Peraire, who provided valuable advice in the crucial steps of my research; Dr. Takashi Maekawa, who tried to understand my rather stubborn and unusual work habits and also kindly showed me better ways; and, Dr. Xiuzi Ye, who provided useful comments on the manuscript even during his vacation in Florida.

I would like to thank my former advisors; Professor Chryssostomos Chryssostomidis for his support and generosity during my first half life at MIT. I also express my sincere appreciation to Dr. Nikiforos Papadakis, who shared my most painful time with me. Without his encouragement, my academic life might not be continued. I am thankful to my advisor in Korea, Professor Keuk-Chun Kim, who retired from Seoul National University last year.

I would also like to thank Professor Jin S. Chung in Colorado, who has taken care of me from the beginning of my life in the States.

I wish to thank all the members of MIT Design Laboratory; Mr. Todd R. Jackson, Mr. Guolin Shen, Mr. Michael O. Jastram, Mr. Guoxin Yu and Ms. Julie Chalfant and also former members; Mr. Michael S. Drooker, Dr. Seamus T. Tuohy, Dr. Jingfang Zhou, Dr. Evan C. Sherbrooke, Dr. Chun-Yi Hu and Mr. George D. Margetis. Special thanks to Mr. Stephen L. Abrams who kindly answered my questions on computer systems and helped me generate examples for the thesis. I also thank Ms. Kristin Gunst for making arrangements for my seminars.

I appreciate the assistance of Dr. Nickolas S. Sapidis with the literature identifi-

cation in the early stages of this work and I thank Professor Thomas J. Peters and Professor Panajiotis Sakkalis for their comments on space homeomorphism.

I also thank Mr. Donald Danmeier, Professor Josef Hoschek and Dr. Steffan Schalck for providing example surface patches.

Financial support for this work was provided in part from ONR and NSF under grant numbers N00014-94-1-1001 and DMI-9500394. Funding for the development of the nonlinear system solver used in this thesis was obtained in part from MIT Sea Grant, ONR and NSF under grant numbers NA90AA-D-SG-424; N00014-91-J-1014 and N00014-94-1-1001; and DMI-9215411.

Contents

| | |
|---|-----------|
| Abstract | 2 |
| Acknowledgements | 3 |
| List of Figures | 9 |
| List of Tables | 14 |
| 1 Introduction | 17 |
| 1.1 Motivation and Objective | 17 |
| 1.2 Thesis Organization | 20 |
| 2 Literature Review | 23 |
| 2.1 Introduction | 23 |
| 2.2 Curve Approximation | 23 |
| 2.3 Surface Tessellation | 25 |
| 2.4 Topological Issues | 30 |
| 2.5 Summary of Shortcomings in Existing Methods | 32 |
| 3 Topologically Reliable Approximation of Composite Curves | 33 |
| 3.1 Introduction | 33 |
| 3.2 Differential Geometry of a Curve | 35 |
| 3.2.1 Concept of a Curve | 35 |
| 3.2.2 Curvature and Torsion | 36 |
| 3.3 Polynomials in Bernstein Form and Bézier Curves | 42 |

| | | |
|----------|---|-----------|
| 3.4 | Significant Points of a Regular Curve | 44 |
| 3.4.1 | Significant Points of a Planar Curve | 44 |
| 3.4.2 | Significant Points of a Space Curve | 46 |
| 3.4.3 | Solution Methodology | 50 |
| 3.5 | Piecewise Linear Approximation of Composite Bézier Curves | 52 |
| 3.5.1 | Preliminary Approximation | 52 |
| 3.5.2 | Main Approximation | 54 |
| 3.6 | Intersection Test and Final Approximation | 56 |
| 3.6.1 | Homeomorphism between the Exact and Approximating Curve | 57 |
| 3.6.2 | Construction of Buckets | 59 |
| 3.6.3 | Preprocessing – Putting Linear Segments into Buckets | 61 |
| 3.6.4 | Intersection Check and Final Approximation | 65 |
| 3.7 | Linear Approximation of Interval Bézier Curves | 67 |
| 3.7.1 | Introduction | 67 |
| 3.7.2 | Preliminary and Main Approximation | 68 |
| 3.7.3 | Intersection Test and Final Approximation | 69 |
| 3.8 | Complexity Analysis | 71 |
| 3.8.1 | Preliminary Approximation | 71 |
| 3.8.2 | Main Approximation | 72 |
| 3.8.3 | Intersection Test and Final Approximation | 72 |
| 3.9 | Examples | 73 |
| 4 | Topologically Reliable Approximation of Composite Surfaces | 85 |
| 4.1 | Introduction | 85 |
| 4.2 | Differential Geometry of a Surface | 87 |
| 4.2.1 | Concept of a Surface | 88 |
| 4.2.2 | First and Second Fundamental Forms | 89 |
| 4.3 | Tensor Product and Triangular Bézier Surfaces | 92 |
| 4.3.1 | Tensor Product Polynomial Bézier Surfaces | 92 |
| 4.3.2 | Triangular Polynomial Bézier Surfaces | 94 |

| | | |
|--------|---|-----|
| 4.3.3 | Extraction of Triangular Sub-Bézier Surfaces from a Tensor Product Bézier Surface | 96 |
| 4.4 | Stationary Points of a Root Mean Square Curvature | 102 |
| 4.4.1 | Root Mean Square Curvature κ_{rms} of a Surface | 102 |
| 4.4.2 | Solution Methodology | 103 |
| 4.5 | Approximate Locally Isometric Mapping | 105 |
| 4.5.1 | Isometric Mapping | 105 |
| 4.5.2 | Mapping Error Function | 106 |
| 4.5.3 | Minimization of Mapping Error Function | 107 |
| 4.6 | Delaunay Triangulation based on Interval Delaunay Test (IDT) | 114 |
| 4.6.1 | Dirichlet Tessellation and Delaunay Triangulation | 115 |
| 4.6.2 | Bowyer-Watson Algorithm | 117 |
| 4.6.3 | Robustness Problems in Delaunay Triangulation | 118 |
| 4.6.4 | Interval Delaunay Test (IDT) | 120 |
| 4.7 | Piecewise Planar Approximation of Composite Bézier Surfaces | 122 |
| 4.7.1 | Construction of Triangulation Domain | 122 |
| 4.7.2 | Initial Triangulation | 125 |
| 4.7.3 | δ -Improving Triangulation | 130 |
| 4.7.4 | AR-Improving Triangulation | 131 |
| 4.8 | Intersection Test and Final Triangulation | 145 |
| 4.8.1 | Homeomorphism between the Exact and Approximating Surface | 146 |
| 4.8.2 | Preprocessing-Putting Triangles into Buckets | 148 |
| 4.8.3 | Intersection Check and Final Triangulation | 151 |
| 4.9 | Approximation of Interval Bézier Surfaces | 152 |
| 4.10 | Robustness Issues | 154 |
| 4.11 | Complexity Analysis | 155 |
| 4.11.1 | Initial Triangulation | 155 |
| 4.11.2 | δ -Improving Triangulation | 156 |
| 4.11.3 | AR-Improving Triangulation | 157 |
| 4.11.4 | Intersection Test and Final Triangulation | 157 |

| | | |
|----------|--|------------|
| 4.12 | Examples | 158 |
| 4.12.1 | Wave-Like Surface | 159 |
| 4.12.2 | High Degree Surface | 162 |
| 4.12.3 | Ship Hull | 164 |
| 4.12.4 | Airfoil | 165 |
| 4.12.5 | Utah Teapot | 166 |
| 4.12.6 | Stem with a Bulbous Bow | 167 |
| 5 | Conclusions and Recommendations | 207 |
| 5.1 | Summary and Contributions | 207 |
| 5.2 | Future Research | 209 |

List of Figures

| | | |
|------|---|----|
| 3-1 | Frenet frame | 38 |
| 3-2 | A quadratic Bézier curve and its control polygon | 43 |
| 3-3 | Significant points on a planar Bézier curve of degree 5 | 46 |
| 3-4 | Significant points on a space curve and its projections | 48 |
| 3-5 | A preliminary approximation of a planar Bézier curve of degree 5 | 53 |
| 3-6 | Construction of an upper bound δ of the approximation error | 55 |
| 3-7 | Subdivision of a Bézier curve segment at $t = 0.5$ | 55 |
| 3-8 | A main approximation of a planar Bézier curve of degree 5 with $\epsilon = 10^{-2}$ | 56 |
| 3-9 | A composite Bézier curve involving a constriction and the corresponding linear approximating segments with $\epsilon = 10^{-1}$ | 57 |
| 3-10 | 2D and 3D buckets | 59 |
| 3-11 | A rectangle which bounds the linear segments in xy coordinate system | 61 |
| 3-12 | Putting a 2D linear segment $\tilde{\mathbf{l}}_p$ into buckets B_{ij} | 62 |
| 3-13 | A 2D bucket associated with 3 transformed linear segments | 64 |
| 3-14 | 2×2 2D buckets containing 8 transformed linear approximating segments | 64 |
| 3-15 | Input curves and linear approximations involving overlapping intersection | 65 |
| 3-16 | 3×3 2D buckets containing 10 transformed linear approximating segments | 66 |
| 3-17 | A composite curve with constriction and its approximation guaranteeing a prescribed <i>tolerance</i> and the existence of a <i>homeomorphism</i> between the exact and approximating curves | 66 |
| 3-18 | A 2D interval linear segment $\mathbf{l}(t)$ | 68 |
| 3-19 | Interval linear approximating segments near constriction | 70 |

| | | |
|------|---|-----|
| 3-20 | Two pieces of input curve segments and their approximation, $0 < \eta \ll 1$ | 70 |
| 3-21 | A planar Bézier curve of degree 5 | 79 |
| 3-22 | Profile of a human face: 12 planar cubic Bézier curves | 80 |
| 3-23 | Quartic and parabolic planar Bézier curves | 81 |
| 3-24 | A simple space Bézier curve of degree 5 | 82 |
| 3-25 | Boundary curves of a biquartic surface | 83 |
| 3-26 | Boundary curves of a biquartic surface | 84 |
| 4-1 | A bicubic tensor product Bézier surface together with its control polygon net | 93 |
| 4-2 | A triangular Bézier surface of degree 3 together with its control polygon net | 95 |
| 4-3 | Approximating triangular element $\mathbf{s}(u, v, w)$ together with its control points $\mathbf{s}_{i,j,k}$ and control points $\mathbf{r}_{i,j,k}$ of exact triangular Bézier surface patch $\mathbf{r}(u, v, w)$ of degree 3 | 97 |
| 4-4 | A sub-triangle $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ on the unit st -parametric space | 98 |
| 4-5 | Neighboring points | 106 |
| 4-6 | Given bi-cubic surface and the corresponding surface net | 108 |
| 4-7 | Procedure to get approximately developed surface net D | 109 |
| 4-8 | Bisected surface nets and their developed surface nets | 111 |
| 4-9 | Self-intersecting developed surface net | 112 |
| 4-10 | Mappings \mathbf{f} , \mathbf{f}^{-1} and \mathbf{r} | 114 |
| 4-11 | Mappings \mathbf{f} and \mathbf{f}^{-1} | 114 |
| 4-12 | Dirichlet tessellation (dotted) and Delaunay triangulation (solid) . . . | 115 |
| 4-13 | Bowyer-Watson algorithm | 117 |
| 4-14 | Incorrect Delaunay triangulation \mathcal{T} | 118 |
| 4-15 | Incorrect Delaunay test | 119 |
| 4-16 | Interval Delaunay test (IDT) | 120 |
| 4-17 | A composite surface composed of 2 trimmed bi-cubic Bézier patches . | 123 |
| 4-18 | Developed surfaces together with trimming curves mapped on them . | 124 |

| | | |
|------|--|-----|
| 4-19 | Triangulation domains with $\epsilon = 10^{-2}$ | 124 |
| 4-20 | Triangulation domain for a trimmed patch | 126 |
| 4-21 | Closeness test | 127 |
| 4-22 | Repositioning a node | 127 |
| 4-23 | Initial triangulation | 129 |
| 4-24 | An internal ABC and external DBA triangle together with a boundary edge AB and center V of circumcircle ABC | 131 |
| 4-25 | Node (P) insertion at the midpoint of the exact boundary segment and the resulting triangulation | 132 |
| 4-26 | δ -improving triangulation | 133 |
| 4-27 | A triangle ABC | 133 |
| 4-28 | A boundary triangle ABC | 135 |
| 4-29 | Node insertion in AR-improving triangulation | 136 |
| 4-30 | An isosceles triangle ABC | 136 |
| 4-31 | A new node P not on the Voronoi edge V_1V_2 : case 1 | 139 |
| 4-32 | A new node P not on the Voronoi edge V_1V_2 : case 2 | 140 |
| 4-33 | A uniform triangulation on the unit uv -parametric space | 142 |
| 4-34 | AR-improving triangulation | 144 |
| 4-35 | A composite Bézier surface and its topologically inconsistent approxi- mation with $\epsilon = 10^{-1}$ and $\tau_{AR} = 3$ | 147 |
| 4-36 | A $10 \times 6 \times 1$ bucket set (front view) | 149 |
| 4-37 | A topologically consistent approximation to the surface of Figure 4-35-(a) | 152 |
| 4-38 | Wave-like surface; exact | 169 |
| 4-39 | Wave-like surface; mesh with κ_{rms} stationary points, $\epsilon = 10^{-1}$ (see also Tables 4.2, 4.3) | 169 |
| 4-40 | Wave-like surface; mesh with κ_{rms} local maximum points, $\epsilon = 10^{-1}$ (see also Table 4.2) | 170 |
| 4-41 | Wave-like surface; mesh without κ_{rms} stationary points, $\epsilon = 10^{-1}$ (see also Table 4.2) | 170 |
| 4-42 | Wave-like surface; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.3, 4.4) | 171 |

| | | |
|------|---|-----|
| 4-43 | Wave-like surface; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.3) | 171 |
| 4-44 | Wave-like surface; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.3) | 172 |
| 4-45 | Wave-like surface; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.3) | 172 |
| 4-46 | Wave-like surface; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.3) | 173 |
| 4-47 | Wave-like surface; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.4) | 173 |
| 4-48 | Wave-like surface; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 5$ (see also Table 4.4) | 174 |
| 4-49 | High degree surface; exact | 176 |
| 4-50 | High degree surface; mesh with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.6) | 176 |
| 4-51 | High degree surface; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.6, 4.7) | 177 |
| 4-52 | High degree surface; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.6) | 177 |
| 4-53 | High degree surface; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.6) | 178 |
| 4-54 | High degree surface; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.6) | 178 |
| 4-55 | High degree surface; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.6) | 179 |
| 4-56 | High degree surface; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.7) | 179 |
| 4-57 | Ship hull; exact | 181 |
| 4-58 | Ship hull; mesh with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.9) | 181 |
| 4-59 | Ship hull; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.9, 4.10) | 182 |
| 4-60 | Ship hull; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Figure 4-62 and Table 4.9) | 182 |
| 4-61 | Ship hull; absolute curvature map | 183 |
| 4-62 | Ship hull; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (magnified) | 183 |
| 4-63 | Ship hull; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.9) | 184 |

| | |
|--|-----|
| 4-64 Ship hull; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.9) | 184 |
| 4-65 Ship hull; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.9) | 185 |
| 4-66 Ship hull; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.10) | 185 |
| 4-67 Ship hull; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 5$ (see also Table 4.10) | 186 |
| 4-68 Airfoil; exact | 188 |
| 4-69 Airfoil; mesh with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.12) | 188 |
| 4-70 Airfoil; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.12, 4.13) | 189 |
| 4-71 Airfoil; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.12) | 189 |
| 4-72 Airfoil; mesh near leading edge with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ | 190 |
| 4-73 Airfoil; mesh near leading edge with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ | 190 |
| 4-74 Airfoil; mesh near trailing edge with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ | 191 |
| 4-75 Airfoil; mesh near trailing edge with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ | 191 |
| 4-76 Airfoil; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.12) | 192 |
| 4-77 Airfoil; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.12) | 192 |
| 4-78 Airfoil; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.12) | 193 |
| 4-79 Airfoil; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.13) | 193 |
| 4-80 Airfoil; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 5$ (see also Table 4.13) | 194 |
| 4-81 Teapot; exact | 196 |
| 4-82 Teapot; mesh with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.15) | 196 |
| 4-83 Teapot; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.15, 4.16) | 197 |
| 4-84 Teapot; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.15) | 197 |
| 4-85 Teapot; approximation near handle with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ | 198 |
| 4-86 Teapot; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.15) | 198 |

| | |
|--|-----|
| 4-87 Teapot; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.15) | 199 |
| 4-88 Teapot; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.15) | 199 |
| 4-89 Teapot; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.16) | 200 |
| 4-90 Teapot; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 5$ (see also Table 4.16) | 200 |
| 4-91 Stem; exact | 202 |
| 4-92 Stem; mesh with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.18) | 202 |
| 4-93 Stem; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.18, 4.19) | 203 |
| 4-94 Stem; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.18) | 203 |
| 4-95 Stem; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.18) | 204 |
| 4-96 Stem; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.18) | 204 |
| 4-97 Stem; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.18) | 205 |
| 4-98 Stem; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.19) | 205 |
| 4-99 Stem; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 5$ (see also Table 4.19) | 206 |
| 5-1 A composite <i>unknotted space</i> curve and its approximations | 210 |

List of Tables

| | | |
|------|--|-----|
| 3.1 | Intrinsic equations for special curves ($c_1, c_2 = \text{constant} \neq 0$) | 40 |
| 3.2 | Result for Figure 3-21 | 77 |
| 3.3 | Result for Figure 3-22 | 77 |
| 3.4 | Result for Figure 3-23 | 77 |
| 3.5 | Result for Figure 3-24 | 78 |
| 3.6 | Result for Figure 3-25 | 78 |
| 3.7 | Result for Figure 3-26 | 78 |
| 4.1 | Complexity comparison between our proposed method and polar form based method | 102 |
| 4.2 | Wave-like surface; dependency of time and space cost upon κ_{rms} stationary points and ϵ (see also Figures 4-38, 4-39 – 4-41) | 168 |
| 4.3 | Wave-like surface; $\tau_{AR} = 4$ (see also Figures 4-38, 4-39, 4-42 – 4-46) . | 168 |
| 4.4 | Wave-like surface; AR- Δ , $\epsilon = 10^{-2}$, $N_\delta = 229$ (see also Figures 4-42, 4-47, 4-48) | 168 |
| 4.5 | Wave-like surface; $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ | 168 |
| 4.6 | High degree surface; $\tau_{AR} = 4$ (see also Figures 4-49 – 4-55) | 175 |
| 4.7 | High degree surface; AR- Δ , $\epsilon = 10^{-2}$, $N_\delta = 132$ (see also Figures 4-51, 4-56) | 175 |
| 4.8 | High degree surface; $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ | 175 |
| 4.9 | Ship hull; $\tau_{AR} = 4$ (see also Figures 4-57 – 4-65) | 180 |
| 4.10 | Ship hull; AR- Δ , $\epsilon = 10^{-2}$, $N_\delta = 390$ (see also Figures 4-59, 4-66, 4-67) | 180 |
| 4.11 | Ship hull; $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ | 180 |

| | | |
|------|--|-----|
| 4.12 | Airfoil; $\tau_{AR} = 4$ (see also Figures 4-68 – 4-78) | 187 |
| 4.13 | Airfoil; AR- Δ , $\epsilon = 10^{-2}$, $N_\delta = 2218$ (see also Figures 4-70, 4-79, 4-80) | 187 |
| 4.14 | Airfoil; $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ | 187 |
| 4.15 | Teapot; $\tau_{AR} = 4$ (see also Figures 4-81 – 4-88) | 195 |
| 4.16 | Teapot; AR- Δ , $\epsilon = 10^{-2}$, $N_\delta = 1520$ (see also Figures 4-83, 4-89, 4-90) | 195 |
| 4.17 | Teapot; $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ | 195 |
| 4.18 | Stem; $\tau_{AR} = 4$ (see also Figures 4-91 – 4-97) | 201 |
| 4.19 | Stem; AR- Δ , $\epsilon = 10^{-2}$, $N_\delta = 967$ (see also Figures 4-93, 4-98, 4-99) | 201 |
| 4.20 | Stem; $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ | 201 |

Chapter 1

Introduction

1.1 Motivation and Objective

Modern CAD/CAM systems allow users to access specific application programs for performing several tasks, such as rendering objects on a graphical display, fundamental geometric calculations and finite element or boundary element meshing for analysis. These application programs often operate on approximate piecewise lower order representations of the exact geometric definition. For this reason and also in order to support reliable data exchange between various geometric modelers, a robust piecewise linear approximation method of high order and procedural curves and surfaces is necessary.

For complicated 3D curved objects, a boundary representation (B-rep) model coupled with constructive solid geometry (CSG) is regarded as the most promising representation method in the current CAD/CAM systems [63]. Parametric curves and trimmed parametric surfaces have a fundamental role in B-rep [14, 29, 65] since they represent edges and faces of the complicated 3D curved objects. One of the most appropriate ways to accomplish a piecewise planar approximation of trimmed parametric patches is a *triangular* tessellation within a specified geometric tolerance, following a piecewise linear approximation of the interior and exterior boundary loops.

The triangular tessellation allows topological simplicity which enables the local *adaptivity* of the *unstructured* triangular network or mesh [8, 55, 70, 71, 90] and it

also provides a unique database i.e., the same triangular facets can be used for *rendering* [79] as well as for other geometric computations or general analysis. Solid free-form fabrication methods for example, currently use surface faceting for data exchange and manufacturing [54]. We need to note the rendering of surfaces on the basis of a faceted model is becoming increasingly advantageous in comparison with the scan-line technique, owing to the increasing power of graphics workstations to handle facets [84]. The local adaptivity feature of the unstructured triangular faceting provides an efficient tool for discretizing a geometry which has rapidly changing intrinsic characteristics. Furthermore, for the finite element (FE) meshing in computational fluid dynamics (CFD) and computational mechanics in general, this feature is particularly important where not only the geometries involved tend to be complex but also the flows are characterized by the appearance of localized features such as shocks and boundary layers. The location of these features is often unknown a priori and adaptivity represents the only viable option for their effective resolution.

Identification of *geometrically significant points* of input surfaces as well as boundary curves reflects important differential geometric properties of input geometry. It is meaningful because first of all, such significant points provide the most important geometric information of the exact input geometry and they play a role as the most appropriate initial nodes in case fully automatic surface tessellation process is required. Moreover, they permit quite optimal coarse approximation especially when the approximation tolerance is relatively loose or input geometry has an irregular distribution of intrinsic features. For polynomial or rational parametric curves and surfaces, those equations to compute geometrically significant points result in *high* degree nonlinear polynomial equation systems and hence, a robust as well as efficient solution methodology should be employed [46, 58, 59, 85].

Automatic mesh generators are designed so that they can be used as a black box and hence can be integrated into larger systems. One of the most important issues concerning existing generators is *robustness* and *consistency*. For complex three dimensional geometries, the sources of failure fall into the following two categories:

- inaccuracies in the geometric definition/computation and

- topological inconsistency between the input geometry and its triangulation.

All state-of-the-art CAD systems used to create and interrogate curved objects are based on geometric solid modeling systems, operating in floating point arithmetic (FPA), that frequently fail. The ultimate reason for this failure is the limited precision of the geometrically crucial computations [41]. In our approach, we employ a new representation based on *interval* parametric curves and surfaces [46, 47, 48, 49, 59, 81, 82, 91] implemented in rounded interval arithmetic (RIA) [66] in order to overcome the lack of robustness in FPA. Interval curves and surfaces differ from classical ones in that the real numbers representing control point coordinates are replaced by intervals. Of course, our approximation methodology is directly applicable to the conventional definition of parametric curves and surfaces operating in FPA. In addition to the numerical robustness, the possible topological inconsistency between the input geometry and its approximation should be carefully checked since such topological inconsistency immediately results in system failure during solution of a computational mechanics problem. For those simple composite curves and surfaces which do not have any inappropriate intersection, corresponding approximations should not have inappropriately intersecting linear or planar elements either to guarantee the existence of a *homeomorphism* [2] between the exact input geometry and its approximation. Therefore, robust intersection test relying on RIA is necessitated for each approximating element and, furthermore, such intersection test should be performed efficiently with low time and space complexity.

Unstructured triangular meshes for free-form surfaces frequently suffer from badly shaped triangles with *high aspect ratio* (AR) [73]. If such low quality elements are used for further geometric computations or general analysis, they cause serious numerical problems [7, 88] such as ill-conditioned matrices or unacceptable convergence. Most existing triangulation algorithms for trimmed parametric surfaces perform triangulation in the original parametric spaces and map each triangular element onto the 3D space through surface equations. Even if quality of triangular elements is assumed to be sufficiently high in the parametric space, it can easily deteriorate through the mapping process due to highly irregular distribution of intrinsic properties or due to

the actual relative size of different edges of the trimmed patch. Most current surface triangulation algorithms do not present a method which *explicitly* controls the AR of triangular elements. Our proposed triangulation algorithm constructs 2D triangulation domain which sufficiently preserves the shape of triangular elements during mapping process and furthermore, provides an efficient method explicitly controlling the AR of triangular elements on the 2D triangulation domain. We note that our AR-improving process can be immediately applied to not only trimmed surface meshing but also another well-known computational geometry problem i.e., triangulation of a multiply connected *planar* region.

Based on the motivations mentioned above, the objective of this thesis is the development of a piecewise linear approximation method of edges and faces of complicated B-rep models within a user specified geometric tolerance, which uses numerically robust geometric definitions and computations and addresses the problems of:

- topological consistency between the exact geometry and its approximation;
- extraction of important differential geometric features of input geometry for use in the approximation;
- efficient adaptive triangulation based on the unstructured mesh approach;
- well-shaped high quality triangulation in 3D space.

1.2 Thesis Organization

In chapter 2, we review recent curve and surface approximation methods and some literature on topological issues in the context of the geometry approximation problem.

Chapter 3 begins with brief review of fundamental theory of differential and computational geometry of curves. Equations to compute geometrically significant points of curves are derived and a solution methodology is also presented. A piecewise linear approximation method of interval as well as conventional composite Bézier curves is presented. A robust and efficient intersection test algorithm is also described in order to guarantee the existence of a homeomorphism between the exact input curves

and their approximations. Complexity analysis for each approximation stage is performed next. This is followed by several examples and numerical results illustrating the method.

In Chapter 4, the fundamental theory of differential and computational geometry of surfaces is briefly reviewed. A method of constructing a 2D triangulation domain for trimmed composite Bézier patches is next presented. This is a crucial step to achieve a well-shaped high quality mesh in 3D space. A triangulation method is then developed suitable for the approximation of interval as well as conventional trimmed surface patches which are the faces of a complex 3D curved B-rep model. An efficient method of extracting sub-triangular Bézier surfaces from a tensor product Bézier surface patch is developed, which significantly reduces time cost for evaluating a tight upper bound of the approximation error of each triangular facet. Two types of node insertion algorithms are discussed to refine the mesh in terms of the approximation error and aspect ratio of triangular elements. A robust and efficient test is also performed to identify possible inappropriate intersections between approximating triangular facets. If such intersections exist, a method to execute local mesh refinement to avoid inappropriate facet intersection is developed. We also analyze complexity of each step of the triangulation process. Finally, the proposed triangulation algorithm is implemented and demonstrated with several illustrative examples.

Chapter 5, the final chapter of this thesis, contains conclusions and a summary of the main contributions of this work. Recommendations for future research are also presented.

Chapter 2

Literature Review

2.1 Introduction

In this Chapter, we review recent curve and surface approximation algorithms and some literature on topological issues in the context of the geometry approximation problem.

2.2 Curve Approximation

A number of low order approximation methods of high order and procedural curves have been developed for example, [24, 69, 74, 96]. A piecewise linear approximation of a planar or space curve is a special instance of the general approximation problem.

A multitude of methods have been developed in the last two decades to solve various kinds of piecewise linear approximation problems. However, most literature has been limited to *digitized planar* curves. Progress on efficient heuristic algorithms concerning such a 2D approximation problem is well described in [26].

Farouki [31] described hierarchical segmentations of planar algebraic curves based on identifying points of a curve where successive differential characteristics vanish. Given a planar polynomial curve $\mathbf{r}(t) = (x(t), y(t))$, [31] computes turning, inflection and curvature stationary points of $\mathbf{r}(t)$. By hierarchical segmentations based on those characteristic points, three types of segments i.e., monotone, convex and

primitive segments are obtained. By utilizing the upper bound of an approximation error derived by Filip *et al.* [32] and the characteristics of the primitive segment where curvature is monotonously varying, subsequent segmentations are performed until the approximation error is within the prescribed geometric tolerance. The method presented in [31] uses meaningful differential geometric features of input curves; however, it does not describe how to solve those characteristic equations in a robust manner and also it does not address the topological validity of the approximation. Moreover, Farouki [31] does not extend the segmentation scheme to general space curves.

Hamann and Chen [39] proposed a 2D piecewise linear approximation technique, which weighs the points on the input planar curve with respect to a local curvature measure and the arc length of parametric segment. It then iteratively removes points of smallest curvature weights until a piecewise linear approximation is obtained within a specified geometric tolerance. Their method may result in near optimal approximation; however, it can be suitably applied to *only digitized planar* curves and furthermore, it does not consider the topological problem possibly caused by the large data reduction.

Among very few papers on 3D curve discretization, Kehtarnavaz and deFigueiredo [51] presented a scheme for segmentation of a space curve which uses the norm of the Darboux vector [53] as a criterion for segmentation. A quintic B-spline curve $\mathbf{r}(s)$ is constructed for noisy data followed by the formulation of curvature, torsion and their derivatives. Total curvature $\omega(s)$ which is the norm of the Darboux vector is then formulated. Among those solutions for $\omega'(s) = 0$, the points of maximum $\omega(s)$ are selected and placed as break points. This method utilizes one of the most appropriate differential geometric features of a space curve; however, as described in Section 3.4.2 of this thesis, the degree of the polynomial equation $\omega'(s) = 0$ for a quintic space curve is 63. Despite such high degree for $\omega'(s)$, reference [51] does not seriously consider the numerical robustness problem in solving high degree nonlinear polynomial equations, such as $\omega'(s) = 0$, which is their only criterion equation providing segmentation nodes.

Ihm and Naylor [50] introduced an iterative heuristic algorithm for piecewise linear approximations of digitized space curves based upon the notions of curve length and

spherical image. Given a digitized space curve and an approximation tolerance, they recursively bisect the curve segment until the approximation error is less than the tolerance. At each subdivision level i , they compute a bisecting point $c_i = \frac{\int_0^l s\kappa(s) ds}{\int_0^l \kappa(s) ds}$ by using forward difference approximation followed by discrete summation, where l , s , $\kappa(s)$ are total length of the digitized sub-curve at level $(i-1)$, arc length parameter and curvature at s , respectively. This method appropriately utilizes a discrete version of the differential geometric features of the input digitized space curve; however, the application is restricted to the approximation of a *digitized* space curve and also it does not consider the topological relation of the input geometry and the corresponding approximation.

2.3 Surface Tessellation

Surface tessellation of a boundary representation (B-rep) model is an essential element of various algorithms for general graphic display, stereo-lithography applications, finite element (FE) mesh generation and reliable data exchange between various geometric modelers.

Several tessellation algorithms are found which deal with *implicit* surfaces e.g., [3, 38, 92]. Main drawback of their algorithms is they are not directly applicable to surface tessellation of B-rep models, where faces are represented by trimmed *parametric* surfaces in general.

Filip *et al.* [32] present theorems giving bounds on the maximum deviation between parametric surfaces and their triangular tessellations using the second partial derivatives of C^2 input parametric surfaces. We note that the theorem is frequently utilized in other literature such as [73, 84]. Based on the theorem, Filip *et al.* [32] construct *uniform* grids on the parametric space and consequently a set of right triangles. Corresponding surface mesh satisfying the specified tolerance is obtained by mapping each right triangle onto 3D space through the surface equations. For surfaces of degree less than 3, the method presented in the literature can obtain the bound of the approximation error in a straightforward manner; however, it becomes

much harder to determine the bound for surfaces of *higher* degree. The bound could also be arbitrarily *loose* even for bi-quadratic surfaces, as demonstrated by Elber [25]. Furthermore, the presented method may generate *unnecessarily many* triangular elements, because their error bound is a global one. In other words, given a parametric surface, the maximum norms of the second partial derivatives are computed for the *whole untrimmed* surface. If an input surface is, for example, mostly flat except for the very *localized* high curvature regions, it is obvious that the resulting mesh has unnecessarily many elements. An even worse result would happen in case the locally high curvature region is *trimmed* away. Even if the element shape on the parametric space in [32] is a right triangle, the *mesh quality* could be unsatisfactory due to highly irregular distribution of intrinsic properties or due to the actual relative size of different edges of the surface patches. Furthermore, the algorithm is neither *adaptive* nor *incremental* and it does not address topological consistency issues between the input surface patches and their approximation.

In another paper, Filip [33] develops an adaptive triangulation algorithm. Given a set of polynomial parametric surfaces, each surface from the set is converted into a triangular Bézier surface. The original triangular Bézier surfaces are then recursively subdivided until every triangular facet corresponding to each triangular sub-Bézier surface passes a flatness test. Several different subdivisions are exploited in terms of efficiency and triangulation quality. Similar to other triangulation algorithms, optimizing the number of triangles causes low quality tessellation, and vice versa. The paper suggested an efficient flatness test; however, it does not actually guarantee that the absolute position difference between the exact surface and its approximation at isoparametric points is within a prescribed approximation tolerance, because they consider the maximum distance between each control point of the exact curve (surface) and its *projection* onto the corresponding linear approximating segment (triangle), respectively. Moreover, in case a control polygon of an exact surface intersects itself, the flatness test is not adequately applicable as reported in [33]. The algorithm checks possible cracks between adjacent triangular facets and inserts additional nodes to prevent them; however, possible inappropriate *intersections* between triangular

facets are not considered. The surface tessellation algorithm is also applicable to only *untrimmed* composite polynomial surfaces.

Sheng and Hirsch [84] present an algorithm for triangulating trimmed composite polynomial surfaces. The basic strategy is to perform the triangulation completely in parametric space as in [32]. They first split the parametric space along the common boundaries of the corresponding surface and generate sub-polygons to be triangulated. After completing boundary approximation for each sub-polygon within specified approximation tolerance, they merge polygonal boundary loops of two matching surfaces to prevent possible cracks in the triangulation; however, possible inappropriate *intersections* of triangular facets are not considered. Based on the theorem developed in [32], uniform grids are generated and the corresponding triangulation is performed on each sub-polygon. Although [84] utilizes the same theorem on the upper bound of the approximation error as in [32], they evaluate the upper bound differently by using Chebyshev polynomials previously suggested in [23]. The authors convert Bézier surfaces into Chebyshev polynomials and use the extremal properties of the Chebyshev polynomial to compute the maximum norms of the second partial derivatives of the surface. Although they subdivide the original surface into sub-patches along the common boundaries, their upper bound of the approximation error is still *global* for each sub-patch. Therefore, it is expected that an unnecessarily large number of triangular elements may be generated in case high curvature regions are very localized or trimmed away. Furthermore, since the triangulation is entirely performed in the original parametric space, *triangulation quality* can be unacceptably low after mapping each triangular facet onto 3D space. This algorithm also has the disadvantage of not being *adaptive*.

Chew [16] extended his two-dimensional mesh generator [15] to meshing curved surfaces defined by a single patch by generalizing planar Delaunay triangulation. In his previous two-dimensional triangulation [15], he assumes that all boundary edges have lengths between s and $\sqrt{3}s$, where s is a predefined feature size. This condition can be enforced by subdividing edges. Chew starts with the constrained Delaunay triangulation (CDT) of the input and then, while there is a triangle with circumcircle

of radius greater than s , he adds an additional node at the center of the circle, and re-computes the CDT. He also proved that the algorithm results in a triangulation in which all angles of the elements are between 30° and 120° assuming the boundary loop does not form a sharp angle. He modified the two-dimensional algorithm by extending the circumcircle concept to three-dimensional space. He defines the circumcircle of a triangular element whose vertices are on a free-form surface as: *Given three vertices on a curved surface, consider the infinite set of spheres through the three vertices. The center of all the spheres lie on a single line. A sphere whose center is on the surface is chosen and the circumcircle of the three vertices is defined by the set of points where the sphere intersects the surface.* The algorithm repeatedly inserts a node at the center of the circumcircle defined above, until every triangular element satisfies prescribed size and shape criteria. However, this definition of circumcircle has some difficulties; in particular, the line on which the sphere centers lie may intersect the surface more than once or it may not intersect the surface at all. Similar to the two-dimensional case, his algorithm produces pretty high quality mesh in terms of elements' shape for input surfaces with simple shape. However, there is need of a starting triangulation in that the surface normals do vary by more than $\frac{\pi}{2}$ in a region about each triangle. Furthermore, there is no *direct control of the approximation error*, which is a crucial factor in surface tessellation.

Given a set of parametric surfaces, Boender *et al.* [11] perform an incremental boundary-conforming triangulation on the parametric spaces and map each triangle into three-dimensional space. They adapt a texture-mapping procedure for use in mapping a parametric space triangulation onto 3D space with minimal distortion. The authors also state that the generated mesh should be topologically correct. However, they do not explain their node insertion criteria and approximation error measure.

Piegl and Richard [73] present a piecewise planar approximation of a trimmed NURBS surface. The overall their algorithm is similar to [32] explained earlier in this Section except that [73] deals with the more general NURBS surface and the estimation method of the maximum norms of the second partial derivatives of a NURBS surface is novel. However, the method in [73] inherits most disadvantages

of [32], e.g., it is neither incremental nor adaptive, and it also has the possibility of generating an unnecessarily large number of triangular elements and may produce low quality triangulation in case of surface patches with highly irregular distribution of intrinsic properties or with different actual relative size of edges.

Klein and Straßer [52] develop a surface meshing algorithm for B-rep geometries. They generate initial triangulation of the parametric domain which contains nodes on the exterior and interior loops. These nodes on the boundary loops are found from a *discrete* set of parametric points uniformly and densely distributed on each boundary loop segment with the property that the corresponding point on the boundary of the surface has the maximum distance to the approximating loop segment until the maximum distance is smaller than a given tolerance. Similarly, a point is found from a *discrete* grid of interior parametric points and inserted stepwise into the parametric space with the property that the corresponding point on the surface has the maximum distance to the triangulation until the maximum distance is smaller than a given tolerance. This method provides adaptivity and easily checks neighborhood information by using B-rep data structure during triangulation. However, the algorithm is based on a discrete measure of the approximation error which does not fully satisfy a true approximation error bound. It does not consider differential geometric features of the input geometry nor topological aspects such as inappropriate intersections between the approximating triangles.

Elber [25] presents an incremental free-form surface triangulation algorithm utilizing an auxiliary bilinear surface in the computation of the approximation error bound. Given an untrimmed parametric surface, two initial triangles are generated by simply dividing the uv -parametric space along its diagonal. The distance bound between an approximating triangle and a bilinear surface is measured. The bilinear surface is the one corresponding to the four corner points of the parametric space. Another distance bound is computed between the bilinear surface and the exact surface. He defines the approximation error bound as the sum of two distance bounds. In case the approximation error does not meet a given tolerance, the algorithm subdivides the surface along the direction that minimizes the approximation error from the two

subdivision possibilities, u or v and repeats the procedure. This algorithm is quite efficient since both the error measure and the subdivision along isoparametric lines are performed quickly. However, the algorithm easily creates cracks between adjacent triangular facets; e.g., the original patch is subdivided into two along $u = u_0$, and assuming two triangles existing in the left sub-patch pass the error test, while the other two triangles corresponding to the right sub-patch do not. If, at least one of further subdivisions should be performed along $v = v_0$, then it is obvious that there exist *cracks*, as shown in Figure 3-(b) of [25]. The author does not discuss how to remedy this problem.

Shimada and Gossard [86] generate adaptive 2D/3D meshes for a given boundary representation of the geometry of a space region and a density distribution over the region. This method uses a physically-based approach that simulates the attraction and repulsion forces between bubbles covering the region. An advantage of the method is creation of a conformal triangulation with well-shaped triangles although the method does not *directly* control the aspect ratio of triangular elements.

Gürsoy and Patrikalakis [35, 36, 37] develop a medial axis transform (MAT) based method for surface tessellation. The strong points of such a method are that they can perform feature recognition such as identification of symmetries and constrictions and extraction of local length scales, which generally facilitates the automatic specification of geometrically meaningful meshes. However, computation of the medial axis has not been solved completely for a planar region with arbitrary curved boundaries such as the parametric space of a trimmed surface. For example, Gürsoy and Patrikalakis [68] compute the MAT of a region bounded by straight line segments and circular arcs.

2.4 Topological Issues

For applications such as computational mechanics, it is natural to require that a *perturbed* object (meshed domain) should have the *same topological* form as a given original object (exact domain).

In *solid modeling* field, Mäntylä [62] demonstrates that the validity criteria of a boundary model should include the following conditions:

- The set of faces of the boundary model *closes*, i.e., forms the complete *skin* of the solid with no missing parts.
- Faces of the model do not intersect each other except at the common vertices or edges.
- The boundaries of faces do not intersect themselves.

Andersson *et al.* [4] use the precise mathematical condition that two objects have the same topological form provided that there is a *homeomorphism* from \mathbb{R}^3 onto \mathbb{R}^3 which carries one object onto the other with applications in the *tolerancing* and *metrology* fields. For example, a cube might be transformed into a solid spherical ball, but a torus can not become a torus with a knot in it. They also suggest that as computer modules become completely *autonomous*, the verification of topological correctness is a crucial problem.

Hoppe *et al.* [42] solve a *surface reconstruction* problem such that given a set of data points scattered in three dimensions and an initial triangular mesh M_0 , they produce a mesh M of the same topological type as M_0 that fits the data well and has a small number of vertices. However, they indicate that the verification of correctness of topological form may not be simple if an algorithm is producing a large number of surface elements.

Surface tessellation is another example of generating perturbed objects and hence, its topological correctness has to be taken into account. Peters *et al.* [72] describe the topological requirements that should be satisfied in a surface tessellation: each triangle should be correctly *connected* to other triangles through their common edges and vertices. This means that no cracks are allowed between neighboring triangles and *no* two triangles shall *intersect* each other at positions other than their explicitly stored common edges or vertices. This ensures that the surface is not self-intersecting.

2.5 Summary of Shortcomings in Existing Methods

In this Section, we summarize the conceptual shortcomings of all current state-of-the-art approaches in surface tessellation as follows:

1. Meshing systems which address the topological aspects of the automated meshing problem do not invoke differential geometric properties.
2. Grid generation systems with considerable emphasis on local differential geometric methods do not have a topological concept which is necessary to mesh arbitrary domains automatically.
3. Most meshing systems (except those based on MAT) do not capitalize on global feature recognition such as identification of symmetries, constrictions and local length scales.
4. All meshing systems suffer from the lack of numerical robustness in geometric modeling systems.
5. There are considerable difficulties in adaptive meshing problems e.g., in boundary value problems with time dependent boundaries.
6. Unstructured triangular meshes for free-form surfaces frequently suffer from badly shaped triangles with high aspect ratio.

Chapter 3

Topologically Reliable Approximation of Composite Curves

3.1 Introduction

Piecewise linear approximation of planar and space curves, a special instance of the general approximation problem, plays a vital role in the successful discretization of general solid models such as meshing applications. The piecewise linear approximation of an input curve should be completed within a certain user specified global tolerance. To permit the formulation of valid idealizations of physical problems on discretized geometric representations, the discrete representation of the exact curve has to be topologically identical to the exact curve. This means that a *homeomorphism* [2] should exist between the exact curve and its approximation.

In this Chapter, we present an efficient method of *approximating a set of mutually non-intersecting simple composite planar and space Bézier curves* within a user specified global tolerance and *ensuring the existence of a homeomorphism* between the linear approximating segments and the actual nonlinear curves. A summary of Chapter 3 of this thesis has been presented in a paper by Cho *et al.* [18]. Prior liter-

ature had not addressed *topological issues* in the context of geometric approximation problems.

Given a set of planar and space Bézier curves either open or closed, we compute geometrically significant points using the projected polyhedron algorithm [46, 58, 59, 85] for solving systems of nonlinear polynomial equations implemented in floating point arithmetic (FPA) or rounded interval arithmetic (RIA) [66]. By using these significant points as well as the boundary points on a set of curves as nodes, we can create a *preliminary* piecewise linear approximation of the input curves. The preliminary approximation, reflecting the global differential properties of the input curves, is valuable especially when a *coarse* approximation of good quality is required such as in *finite element meshing* applications. *Main* approximation begins with computing the error for each line segment by evaluating a tight upper bound for the deviation between a piece of the exact Bézier curves and the corresponding line segment. A convex hull method is effectively employed to compute such an upper bound. We adaptively subdivide each segment until the error is within a specified tolerance. Finally, for each pair of linear approximating segments, an *intersection check* is performed to identify possible inappropriate intersections arising, for example, from global distance function features of the curve such as *constrictions*. A brute force approach takes $\mathcal{O}(n^2)$ time for computation where n is the number of segments. A superior alternative method is use of the *bucket sort* [5, 19] which runs in $\mathcal{O}(n)$ time on the average. If those inappropriate intersections exist they necessitate further refinement of the approximation to capture the global features of the curve relating to minima of the self-distance function.

State-of-the-art CAD systems, operating in FPA, frequently fail. The ultimate reason is the *limited precision* of geometric computations. To cope with this adversity, we consider *interval* Bézier curves evaluated using RIA as our input and also apply our approximation scheme to those *interval* Bézier curves. Such curves and the corresponding interval Bézier surfaces are the basis of a robust interval solid modeling technique described in Hu *et al.* [46, 47, 48, 49].

3.2 Differential Geometry of a Curve

We summarize the essential differential geometric concepts of a curve well described in the literature on classical differential geometry such as [22, 53].

3.2.1 Concept of a Curve

By a *regular parametric representation* we mean a vector function of t in an interval I ,

$$\mathbf{r} = \mathbf{r}(t), \quad t \in I \quad (3.1)$$

with the property that

- $\mathbf{r}(t)$ is of class C^m , $m \geq 1$ in I ,
- $\mathbf{r}'(t) \neq \mathbf{0}$, $\forall t$ in I .

A real valued function $t = t(\theta)$ on an interval I_θ is an *allowable change of parameter* if

- $t(\theta)$ is of class C^m , $m \geq 1$ in I_θ ,
- $t'(\theta) \neq 0$, $\forall \theta$ in I_θ .

A regular parametric representation $\mathbf{r} = \mathbf{r}(t)$, $t \in I_t$, is *equivalent* to a regular parametric representation $\mathbf{r}^* = \mathbf{r}^*(\theta)$, $\theta \in I_\theta$, if there exists an allowable change of parameter $t = t(\theta)$ such that

- $t(I_\theta) = I_t$,
- $\mathbf{r}(t(\theta)) = \mathbf{r}^*$.

We define a *regular parametric curve* to be an equivalent class of regular parametric representations. *Unless otherwise stated, a curve $\mathbf{r}(t)$ in the following will denote a regular parametric curve.* Furthermore, a curve of class C^m is a collection of regular

parametric representations of class C^m any two of which are related by an allowable change of parameter of class C^m .

A curve $\mathbf{r}(t)$, $t \in I$, is said to be *simple* if there are no multiple points; that is, if $t_1 \neq t_2$ implies $\mathbf{x}(t_1) \neq \mathbf{x}(t_2)$.

Given $t \in I$ the *arc length* s of a curve $\mathbf{r}(t)$ from t_0 is defined by

$$s(t) = \int_{t_0}^t \left| \frac{d\mathbf{r}}{dt} \right| dt. \quad (3.2)$$

Since $\mathbf{r}'(t) \neq \mathbf{0}$, the arc length $s(t)$ has a continuous nonvanishing derivative given by

$$\frac{ds}{dt} = \left| \frac{d\mathbf{r}}{dt} \right|. \quad (3.3)$$

Hence $s = s(t)$ is an allowable change of parameter on I . Also $s(t)$ is of class C^m on I if $\mathbf{r}(t)$ is of class C^m . Thus the arc length s can be introduced along the curve as a parameter and such a representation $\mathbf{r} = \mathbf{r}(s)$ on I_s is called a *natural representation*.

3.2.2 Curvature and Torsion

Let $\mathbf{r} = \mathbf{r}(s)$ be a natural representation of a curve C . The vector

$$\mathbf{T}(s) = \frac{d\mathbf{r}}{ds} = \dot{\mathbf{r}}(s) \quad (3.4)$$

is called the *unit tangent vector* to the curve C at the point $\mathbf{r}(s)$. Introducing any allowable parameter t , we have

$$\mathbf{T}(t) = \frac{d\mathbf{r}}{dt} \frac{dt}{ds} = \frac{\mathbf{r}'(t)}{|\mathbf{r}'(t)|}. \quad (3.4')$$

The straight line through a point \mathbf{r} on a curve C parallel to the unit tangent vector at \mathbf{r} is called the *tangent line* to C at \mathbf{r} .

The totality of all vectors bound at a point P of a curve C which are orthogonal to the corresponding unit tangent vector \mathbf{T} lie in a plane. This plane is called the *normal plane* to C at P , as shown in Figure 3-1.

Let P_0, P_1 and P_2 be three noncollinear points on a curve $\mathbf{r} = \mathbf{r}(t)$ of class $m \geq 2$. Then we can uniquely define a plane E passing through those three points. The limit position of the plane E as P_1 and P_2 both tend to P_0 is determined by two vectors $\mathbf{r}'(t)$ and $\mathbf{r}''(t)$, if linearly independent. The plane spanned by $\mathbf{r}'(t)$ and $\mathbf{r}''(t)$ is called the *osculating plane* of the curve at P_0 , see Figure 3-1.

Let $\mathbf{r} = \mathbf{r}(s)$ be a natural representation of a curve C of class C^m with $m \geq 2$. The derivative of the unit tangent vector

$$\frac{d\mathbf{T}}{ds} = \dot{\mathbf{T}}(s) = \ddot{\mathbf{r}}(s) \quad (3.5)$$

is called the *curvature vector* $\mathbf{k}(s)$ of C at $\mathbf{r}(s)$ i.e., $\mathbf{k}(s) \equiv \ddot{\mathbf{r}}(s)$. Since the tangent vector has unit length, the norm of the curvature vector measures the rate of change of the angle which neighboring tangents make with the tangent at s and it is called the *curvature* $\kappa(s)$ of C at $\mathbf{r}(s)$ i.e.,

$$\kappa(s) = |\mathbf{k}(s)| = |\ddot{\mathbf{r}}(s)|. \quad (3.6)$$

Therefore, the curvature is a measure of how rapidly the curve pulls away from the tangent line at s , in a neighborhood of s . Notice that under a change of orientation of the curve, \mathbf{k} and the curvature κ remain invariant, and hence κ is an *intrinsic* property of the curve. For any allowable parameter t ,

$$\kappa(t) = \frac{|\mathbf{r}'(t) \times \mathbf{r}''(t)|}{|\mathbf{r}'(t)|^3}. \quad (3.6')$$

If the curvature vector $\mathbf{k}(s)$ is not the null vector, it is orthogonal to the unit tangent vector $\mathbf{T}(s)$ and consequently lies in the normal plane to the curve C at the point under consideration; \mathbf{k} also lies in the osculating plane. For nonvanishing \mathbf{k} the unit vector

$$\mathbf{N}(s) = \frac{\mathbf{k}(s)}{|\mathbf{k}(s)|} \quad (3.7)$$

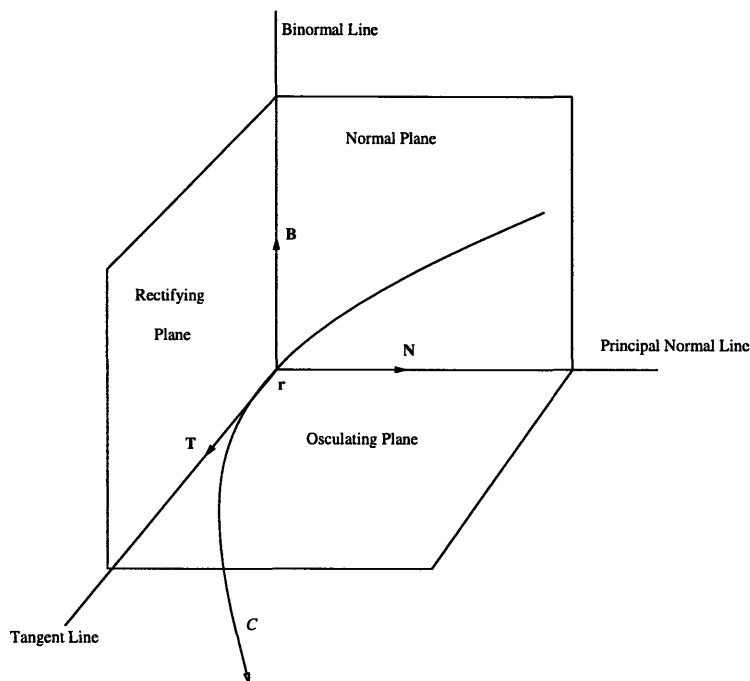


Figure 3-1: Frenet frame

which has the direction and sense of $\mathbf{k}(s)$ is called the *unit principal normal vector* to C at the point $\mathbf{r}(s)$. The straight line passing through this point and containing $\mathbf{N}(s)$ is called the *principal normal line* to the curve C at $\mathbf{r}(s)$.

We again consider a curve of class $m \geq 2$. To every point of the curve at which $\mathbf{k} \neq \mathbf{0}$, we have associated two unit vectors, $\mathbf{T}(s)$ and $\mathbf{N}(s)$. These two vectors are orthogonal. We now introduce a unit vector

$$\mathbf{B}(s) = \mathbf{T}(s) \times \mathbf{N}(s) \quad (3.8)$$

which is orthogonal to both $\mathbf{T}(s)$ and $\mathbf{N}(s)$ and is called the *unit binormal vector* of the curve at $\mathbf{r}(s)$. The straight line passing through a point $\mathbf{r}(s)$ of a curve C in the direction of the corresponding $\mathbf{B}(s)$ is called the *binormal line* to C at $\mathbf{r}(s)$. The plane spanned by \mathbf{B} and \mathbf{T} is called the *rectifying plane*, see Figure 3-1.

To every point of a curve at which $\mathbf{k} \neq \mathbf{0}$ we have now associated three orthogonal unit vectors. The triplet $(\mathbf{T}, \mathbf{N}, \mathbf{B})$ is called the *moving trihedron* of the curve, see Figure 3-1.

We now suppose that $\mathbf{r} = \mathbf{r}(s)$ is a natural representation of a curve C of class C^m with $m \geq 3$. For the nonvanishing curvature, we can differentiate the binormal vector $\mathbf{B}(s)$, obtaining

$$\dot{\mathbf{B}}(s) = -\tau(s)\mathbf{N}(s). \quad (3.9)$$

The continuous function $\tau(s)$ is called the *torsion* of C at $\mathbf{r}(s)$ and is determined by

$$\tau(s) = -\dot{\mathbf{B}}(s) \cdot \mathbf{N}(s) = \frac{\ddot{\mathbf{r}}(s) \cdot (\dot{\mathbf{r}}(s) \times \ddot{\mathbf{r}}(s))}{|\ddot{\mathbf{r}}(s)|^2}. \quad (3.10)$$

Since the binormal vector \mathbf{B} is the normal vector to the osculating plane, the torsion measures the magnitude and sense of deviation of a curve from the osculating plane in the neighborhood of the corresponding point of the curve, or in other words, the rate of change of the osculating plane. Imposing on $\mathbf{r}(s)$ an allowable change of parameter of class C^m with $m \geq 3$ we obtain from Eq. (3.10)

$$\tau(t) = \frac{\mathbf{r}'''(t) \cdot (\mathbf{r}'(t) \times \mathbf{r}''(t))}{|\mathbf{r}'(t) \times \mathbf{r}''(t)|^2}. \quad (3.10')$$

Note the sign of τ is independent of the sense of \mathbf{N} and the orientation of a curve, and hence τ is an *intrinsic* property of the curve. The sign has been chosen so that *right-handed* curves have a *positive* torsion.

The first derivatives $\dot{\mathbf{T}}(s)$, $\dot{\mathbf{N}}(s)$ and $\dot{\mathbf{B}}(s)$ of the unit tangent vector \mathbf{T} , the unit principal normal vector \mathbf{N} and the unit binormal vector \mathbf{B} can be represented as a linear combination of these three linearly independent unit vectors; the corresponding formulae are called *Serret-Frenet formulae*:

$$\begin{pmatrix} \dot{\mathbf{T}} \\ \dot{\mathbf{N}} \\ \dot{\mathbf{B}} \end{pmatrix} = \begin{pmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{pmatrix} \begin{pmatrix} \mathbf{T} \\ \mathbf{N} \\ \mathbf{B} \end{pmatrix} \quad (3.11)$$

We observe that the Serret-Frenet formulae form a system of three vector differential equations of the first order in \mathbf{T} , \mathbf{N} and \mathbf{B} with the coefficient matrix in terms of

| | |
|----------------|--|
| circle | $\kappa = c_1, \tau \equiv 0$ |
| circular helix | $\kappa = c_1, \tau = c_2$ |
| general helix | $\kappa = \kappa(s), \tau = c_1\kappa$ |

Table 3.1: Intrinsic equations for special curves ($c_1, c_2 = \text{constant} \neq 0$)

the curvature κ and the torsion τ . A little elaboration on the Serret-Frenet formulae yields the following *fundamental existence and uniqueness theorem for space curves* [22].

Theorem 3.2.1 *Let $\kappa(s) > 0$ and $\tau(s)$ be arbitrary continuous functions on the interval I_s . Then there exists, except for position in space, one and only one space curve C for which $\kappa(s)$ is the curvature, $\tau(s)$ is the torsion and s is a natural parameter along C .*

Furthermore, the equations $\kappa = \kappa(s), \tau = \tau(s)$ which give the curvature and torsion of a curve as functions of s are called the *intrinsic* equations of a curve, since they completely define the shape of the curve. Table 3.1 shows the intrinsic equations of some special curves.

When a point moves along a curve C the corresponding trihedron defined by \mathbf{T} , \mathbf{N} and \mathbf{B} undergoes a rigid body motion. This consideration turns out to be a kinematic interpretation of the Serret-Frenet formulae. We exclude the translation of the moving trihedron from our investigation and consider only its rotation. Thus we assume the trihedron undergoes a translation and is then bound at a fixed point, say at the origin of the Cartesian coordinate system in space. We will now determine the rotation vector Ω of the trihedron, assuming that the curve C under consideration is of class C^m with $m \geq 3$ and has nonvanishing curvature κ . The position vector of any point P of the trihedron is of the form

$$\mathbf{x} = u\mathbf{T} + v\mathbf{N} + w\mathbf{B}. \quad (3.12)$$

Assuming that the point moving along the curve C has a constant speed equal to 1,

we may equate the arc length s of C with the time t . The velocity vector \mathbf{v} of P thus is

$$\mathbf{v} = \dot{\mathbf{x}} = u\dot{\mathbf{T}} + v\dot{\mathbf{N}} + w\dot{\mathbf{B}}. \quad (3.13)$$

As follows from Eq. (3.11) the vectors $\dot{\mathbf{T}}$, $\dot{\mathbf{N}}$ and $\dot{\mathbf{B}}$ lie in the same plane $(\mathbf{N}, \dot{\mathbf{N}})$. Since the rotation vector Ω is orthogonal to the velocity vector \mathbf{v} , Ω has the direction of

$$\dot{\mathbf{T}} \times \dot{\mathbf{N}} = \kappa\mathbf{N} \times (-\kappa\mathbf{T} + \tau\mathbf{B}) = \kappa^2(\mathbf{T} \times \mathbf{N}) + \kappa\tau(\mathbf{N} \times \mathbf{B}) = \kappa(\tau\mathbf{T} + \kappa\mathbf{B}), \quad (3.14)$$

hence it is of the form

$$\Omega = c(\tau\mathbf{T} + \kappa\mathbf{B}). \quad (3.15)$$

From $\mathbf{v} = \Omega \times \mathbf{x}$ and by setting $v = w = 0$ in Eq.'s (3.12), (3.13), $\dot{\mathbf{T}} = \Omega \times \mathbf{T}$. By inserting Eq. (3.15) in this expression, we find $\dot{\mathbf{T}} = c(\tau\mathbf{T} + \kappa\mathbf{B}) \times \mathbf{T} = c\kappa\mathbf{N}$. Comparing this with Eq. (3.11) we have $c = 1$ and thus finally the following result; *The rotation vector of the moving trihedron of a curve $C: \mathbf{r}(s)$ of class C^m with $m \geq 3$ and with nonvanishing curvature, when a point moves along C with constant velocity 1, is given by the expression*

$$\Omega(s) = \tau\mathbf{T}(s) + \kappa\mathbf{B}(s). \quad (3.15')$$

Here the rotation vector Ω is often called the *vector of Darboux* and its norm $|\Omega|$ indicates the angular speed ω of the moving trihedron. The angular speed ω is called *total curvature* of a curve and defined by

$$\omega(s) = \sqrt{\kappa^2(s) + \tau^2(s)}. \quad (3.16)$$

If a curve is planar then ω reduces to κ and the binormal line becomes the axis of rotation.

3.3 Polynomials in Bernstein Form and Bézier Curves

We introduce some fundamental definitions and properties associated with polynomials in Bernstein basis and Bézier curves which will be used in the following Sections [28, 44].

The Bernstein polynomial basis of degree n on the unit interval $u \in [0, 1]$ are defined by

$$B_{i,n}(u) = \binom{n}{i} u^i (1-u)^{n-i}, \quad (3.17)$$

where the binomial coefficients are given by

$$\binom{n}{i} = \begin{cases} \frac{n!}{i!(n-i)!} & \text{if } 0 \leq i \leq n, \\ 0 & \text{else.} \end{cases} \quad (3.18)$$

Any polynomial function $f(t)$ of degree n on an arbitrary interval $[t_l, t_u]$ can be defined in the Bernstein basis on the unit interval $[0, 1]$ by *affine reparametrization* [28],

$$u = \frac{t - t_l}{t_u - t_l}. \quad (3.19)$$

And the corresponding transformed polynomial function is

$$\tilde{f}(u) = \sum_{i=0}^n \tilde{f}_i B_{i,n}(u), \quad u \in [0, 1]. \quad (3.20)$$

In general, the use of Bernstein form has advantages over the familiar power basis in that it possesses a natural geometric interpretation and is substantially less sensitive to arithmetic round-off error or perturbations in initial data [30]. Some of the important properties of the Bernstein basis functions for $u \in [0, 1]$ are as follows

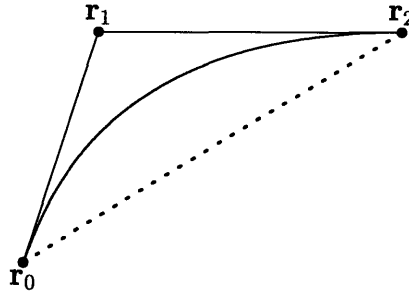


Figure 3-2: A quadratic Bézier curve and its control polygon

[28, 44].

$$0 \leq B_{i,n}(u) \leq 1 \quad : \text{Positivity} \quad (3.21)$$

$$\sum_{i=0}^n B_{i,n}(u) = 1 \quad : \text{Partition of unity} \quad (3.22)$$

$$\sum_{i=0}^n \frac{i}{n} B_{i,n}(u) = u \quad : \text{Linear precision} \quad (3.23)$$

Furthermore, the *derivatives* $f^{(r)}(u)$ of a Bernstein polynomial $f(u)$ and *arithmetic operations* between two Bernstein polynomials can be easily achieved in Bernstein form, [30] i.e.,

$$f^{(r)}(u) = \sum_{i=0}^{n-r} g_i B_{i,n-r}(u), \quad (3.24)$$

where n is the degree of $f(u)$ and g_i is computed explicitly in terms of coefficients f_i of $f(u)$ and also

$$f_1(u) \odot f_2(u) = \sum_{i=0}^m g_i B_{i,m}(u), \quad (3.25)$$

where \odot is an arithmetic operator $+$, $-$ or \times and m is the degree of the resulting polynomial after \odot operation and g_i is computed in terms of f_{1i} and f_{2i} .

An *integral Bézier curve* $\mathbf{r}(u)$ is determined by *repeated linear interpolation* on the successive vertices \mathbf{r}_i of a control polygon, called *de Casteljau algorithm* [28]. The de Casteljau algorithm is compactly represented by the combination of \mathbf{r}_i and the

Bernstein basis function $B_{i,n}(u)$ i.e.,

$$\mathbf{r}(u) = \sum_{i=0}^n \mathbf{r}_i B_{i,n}(u), \quad u \in [0, 1], \quad (3.26)$$

where each vertex \mathbf{r}_i of the control polygon is called a control point. Figure 3-2 shows a quadratic Bézier curve and its control points.

Because the Bézier curve uses the Bernstein basis, several properties of Bézier curves are immediately known [28, 44], such as convex hull property following from Eq.'s (3.21), (3.22), linear precision in Eq. (3.23) and invariance under affine reparameterization, etc.

3.4 Significant Points of a Regular Curve

Curvature and torsion are the most meaningful intrinsic features of a curve. Curvature measures turning of the tangent vector while torsion measures twist out of the osculating plane, when moving along the curve. We base the first step of our approximation scheme upon the differential characteristics of curvature and torsion.

3.4.1 Significant Points of a Planar Curve

Given an interval $I = [t_l, t_u] \subset \mathfrak{R}$, a planar curve \mathbf{r} is defined by a map,

$$\begin{aligned} \mathbf{r} &: I \longmapsto \mathfrak{R}^2 \\ t &\mapsto \mathbf{r}(t) = (x(t), y(t)). \end{aligned} \quad (3.27)$$

We assume $\mathbf{r}(t)$ is regular so that $|\mathbf{r}'(t)| \neq 0$ and of class C^m with $m \geq 2$ in I . Since a planar curve lies in its osculating plane, its torsion vanishes identically. Therefore, we consider only the curvature to determine the significant points on $\mathbf{r}(t)$. In general, the curvature function $\kappa(t)$ is defined by Eq. (3.6'). However, for a planar curve a

signed curvature function can be defined as, [22]

$$\kappa(t) = \frac{x'(t)y''(t) - x''(t)y'(t)}{(x'^2(t) + y'^2(t))^{\frac{3}{2}}} \quad (3.28)$$

by using relation (3.27).

A significant point on a planar curve $\mathbf{r}(t)$, where the curvature $\kappa(t)$ changes sign, is called an *inflection* point, since the center of curvature moves from one side of the curve to the other as we traverse them. Due to the regularity of $\mathbf{r}(t)$, a necessary condition to determine inflection points is

$$x'y'' - x''y' = 0, \quad t \in I. \quad (3.29)$$

For a polynomial parametric curve of degree n , Eq. (3.29) is a polynomial equation of degree $2n - 3$. Figure 3-3 shows an inflection point, labeled by \times , on a planar curve.

Another significant point on $\mathbf{r}(t)$ is defined to be a *stationary point* of curvature function $\kappa(t)$ i.e., the point of $\kappa'(t) = 0$, where $\kappa(t)$ have a local maximum or minimum. The stationary point of the curvature function satisfies the equation,

$$(x'^2 + y'^2)(x'y''' - x'''y') - 3(x'x'' + y'y'')(x'y'' - x''y') = 0, \quad t \in I. \quad (3.30)$$

For a polynomial parametric curve of degree n , Eq. (3.30) is a polynomial equation of degree $4n - 6$. Figure 3-3 also shows the corresponding significant points on $\mathbf{r}(t)$, labeled by \circ 's. Furthermore, comparing Eq.'s (3.29) and (3.30) we see both $\kappa(t)$ and $\kappa'(t)$ of a regular planar curve $\mathbf{r}(t)$ vanish for some t if and only if $x'y'' = x''y'$ and $x'y''' = x'''y'$, simultaneously.

Finally, for the exceptional cases where $\kappa(t)$ is constant or vanishes identically - $\mathbf{r}(t)$ being a part of a circle or a linear segment respectively, as described in Table 3.1 - no significant points need to be considered.

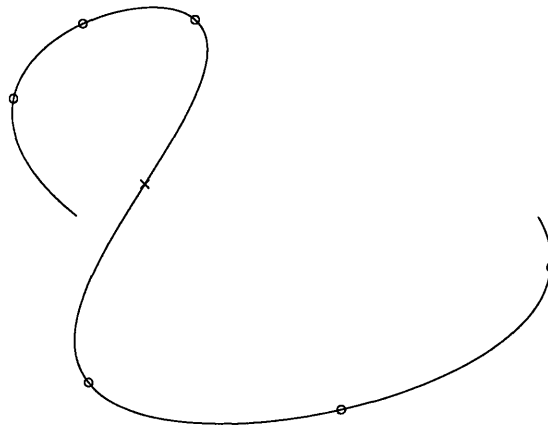


Figure 3-3: Significant points on a planar Bézier curve of degree 5

3.4.2 Significant Points of a Space Curve

A space curve \mathbf{r} is defined by a map,

$$\begin{aligned} \mathbf{r} &: I \mapsto \mathfrak{R}^3 \\ t &\mapsto \mathbf{r}(t) = (x(t), y(t), z(t)). \end{aligned} \quad (3.31)$$

We now assume $\mathbf{r}(t)$ is regular and of class C^m with $m \geq 3$ in $I = [t_l, t_u] \subset \mathfrak{R}$. As described in Section 3.2, a general space curve $\mathbf{r}(t)$ does not lie in its osculating plane since its unit binormal vector $\mathbf{B}(t)$ has an angular velocity while moving along the curve. The magnitude of this angular velocity is called torsion $\tau(t)$. Hence it is necessary to consider the effects of both curvature and torsion in determining the significant points of a space curve.

The curvature $\kappa(t)$ of a space curve is given by Eq. (3.6'). The formula can be expressed as,

$$\kappa(t) = \frac{[(x'y'' - x''y')^2 + (y'z'' - y''z')^2 + (z'x'' - z''x')^2]^{\frac{1}{2}}}{(x'^2 + y'^2 + z'^2)^{\frac{3}{2}}}, \quad (3.32)$$

by using relation (3.31). Since $\mathbf{r}(t)$ is regular, a condition to determine a significant

point on $\mathbf{r}(t)$ where the curvature $\kappa(t)$ vanishes, is

$$K_0(t) = 0, \quad t \in I, \quad (3.33)$$

where

$$K_0(t) \equiv (x'y'' - x''y')^2 + (y'z'' - y''z')^2 + (z'x'' - z''x')^2, \quad (3.34)$$

or

$$x'y'' - x''y' = y'z'' - y''z' = z'x'' - z''x' = 0, \quad t \in I. \quad (3.33')$$

For a polynomial parametric curve $\mathbf{r}(t)$ of degree n , $K_0(t)$ is a polynomial function of degree $4n - 6$. Such significant points on $\mathbf{r}(t)$ are shown in Figure 3-4, marked by \times 's.

We now consider a significant point on $\mathbf{r}(t)$ where the curvature $\kappa(t)$ is *stationary* i.e., $\kappa'(t) = 0$. Such a significant point satisfies the equation,

$$K_1(t) \equiv G_1(t)G_2(t) - 3G_3(t)K_0(t) = 0, \quad t \in I - I^*, \quad (3.35)$$

where $K_0(t)$ is defined in Eq. (3.34) and

$$G_1(t) \equiv x'^2 + y'^2 + z'^2, \quad (3.36)$$

$$G_2(t) \equiv (x'y'' - x''y')(x'y''' - x'''y') + (y'z'' - y''z')(y'z''' - y'''z') + (z'x'' - z''x')(z'x''' - z'''x'), \quad (3.37)$$

$$G_3(t) \equiv x'x'' + y'y'' + z'z'', \quad (3.38)$$

and $I^* \equiv \{t | \kappa(t) = 0\} \cap I$. For $t \in I^*$, $\kappa'(t) = 0$ is equivalent to

$$x'y''' - x'''y' = y'z''' - y'''z' = z'x''' - z'''x' = 0, \quad t \in I^*, \quad (3.35')$$

since the numerator of $\kappa'(t)$ for $t \in I^*$ should vanish faster than the denominator.

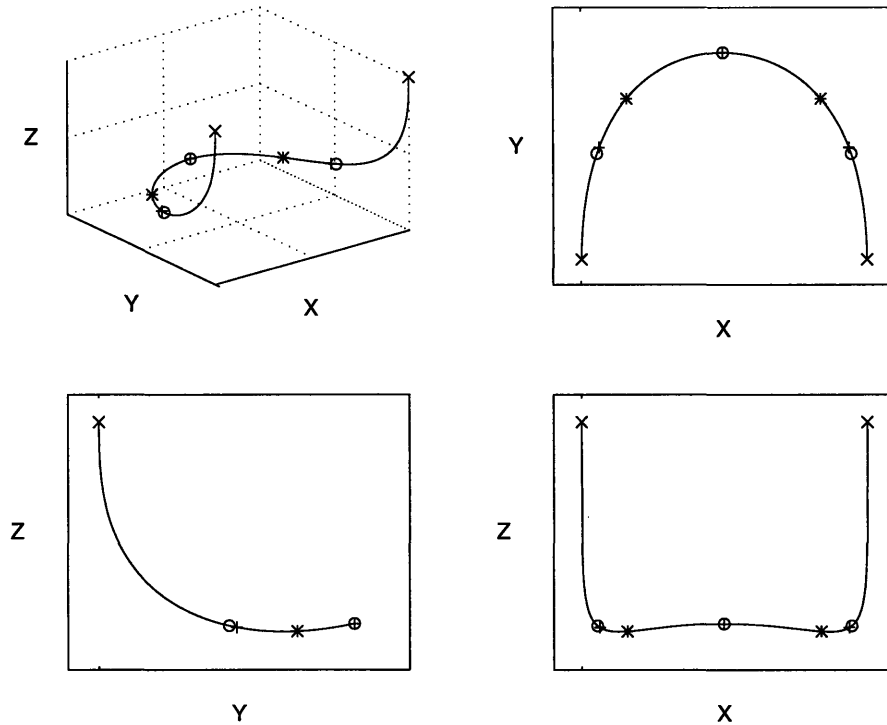


Figure 3-4: Significant points on a space curve and its projections

Given a polynomial curve of degree n , $K_1(t)$ is a polynomial function of degree $6n - 9$. Three points in Figure 3-4 satisfy Eq. (3.35). Two of them are marked by \circ 's and one is marked by \oplus at the midpoint of the curve.

As mentioned in **Theorem 3.2.1**, two continuous functions $\kappa = \kappa(t) > 0$ and $\tau = \tau(t)$ define uniquely (except for rigid body motions) a curve that has curvature κ and torsion τ . We have already assumed that a space curve $\mathbf{r}(t)$ under consideration is regular and of class C^m with $m \geq 3$ in I . In addition to these, we assume $t \in I - I^*$ in the following differential geometry equations associated with a torsion $\tau(t)$. A torsion function $\tau(t)$ of a space curve $\mathbf{r}(t)$ is defined in Eq. (3.10') and by components,

$$\tau(t) = \frac{x'''(y'z'' - y''z') + y'''(z'x'' - z''x') + z'''(x'y'' - x''y')}{(x'y'' - x''y')^2 + (y'z'' - y''z')^2 + (z'x'' - z''x')^2}. \quad (3.39)$$

We now consider a significant point on $\mathbf{r}(t)$ where the torsion $\tau(t)$ vanishes. From

Eq. (3.39), a condition for zero torsion is

$$T_0(t) = 0, \quad t \in I - I^*, \quad (3.40)$$

where

$$T_0(t) \equiv x'''(y'z'' - y''z') + y'''(z'x'' - z''x') + z'''(x'y'' - x''y'), \quad (3.41)$$

which is the polynomial function of degree $3n - 6$ for a polynomial parametric curve of degree n . The sign of torsion has geometric significance as explained in Section 3.2. If $\tau(t)$ changes its sign from $+(-)$ to $-(+)$ when passing the significant point satisfying Eq. (3.40), $\mathbf{r}(t)$ also changes its features from the right-handed (left-handed) curve to the left-handed (right-handed) one, respectively. Figure 3-4 shows a significant point of zero torsion, marked by \ominus at the midpoint of the curve.

Another significant point on $\mathbf{r}(t)$ is the point where the twist out of its osculating plane is *stationary*, namely $\tau'(t) = 0$. Such a significant point satisfies the equation,

$$T_1(t) \equiv G_4(t)K_0(t) - 2G_2(t)T_0(t) = 0, \quad t \in I - I^*, \quad (3.42)$$

where

$$G_4(t) \equiv x^{iv}(y'z'' - y''z') + y^{iv}(z'x'' - z''x') + z^{iv}(x'y'' - x''y'), \quad (3.43)$$

and $K_0(t)$, $G_2(t)$ and $T_0(t)$ are defined in Eq.'s (3.34), (3.37) and (3.41), respectively. Given a polynomial parametric curve of degree n , $T_1(t)$ is of degree $7n - 13$. Figure 3-4 shows the corresponding significant points, marked by *'s. Moreover, comparing Eq.'s (3.40) and (3.42), both $\tau(t)$ and $\tau'(t)$ of a regular $\mathbf{r}(t)$, $t \in I - I^*$, vanish if and only if $T_0(t) = G_4(t) = 0$.

We have explained the geometric significance of a Darboux vector $\Omega(t)$ and total curvature $\omega(t)$ defined in Eq.'s (3.15') and (3.16), respectively. The Darboux vector turns out to be a rotation vector of the Frenet frame while moving along a space curve

$\mathbf{r}(t)$ with a constant velocity 1 and therefore its Euclidean norm, total curvature $\omega(t)$ indicates the angular speed of the moving local frame. We notice the total curvature captures the *coupled* effect of both intrinsic features of a space curve, and hence we consider it as a criterion function for detecting a significant point on $\mathbf{r}(t)$. As mentioned before, we only consider $t \in I - I^*$, where $\kappa(t) > 0$ always. Therefore, only positive $\omega(t)$ needs to be considered. A significant point where the moving frame has its locally highest or lowest angular speed satisfies the equation $\omega'(t) = 0$ i.e.,

$$\Omega_1(t) \equiv K_0^3(t)K_1(t) + G_1^4(t)T_0(t)T_1(t) = 0, \quad t \in I - I^*, \quad (3.44)$$

where $K_0(t)$, $K_1(t)$, $G_1(t)$, $T_0(t)$ and $T_1(t)$ are defined in Eq.'s (3.34), (3.35), (3.36), (3.41) and (3.42), respectively. For a polynomial parametric curve of degree n , Eq. (3.44) is a polynomial equation of degree $18n - 27$. Comparing each function we can roughly see each contribution of $\kappa(t)$, $\kappa'(t)$, $\tau(t)$ and $\tau'(t)$ to Eq. (3.44). For a special example, if $\kappa'(t)$ and one of $\tau(t)$ or $\tau'(t)$ vanish at some t , Eq. (3.44) is also satisfied there. We note $K_0(t)$ and $G_1(t)$ are always positive for $t \in I - I^*$. Three points in Figure 3-4 satisfy Eq. (3.44). Two points, marked by +’s, are located close to the points of curvature extrema, marked by o’s, and the other point, marked by \oplus , is located at the midpoint of the curve where $\kappa'(t)$ and $\tau(t)$ also vanish.

Finally, for the special case where $\kappa(t)$, $\tau(t)$ and consequently $\omega(t)$ are constant - $\mathbf{r}(t)$ being a *circular helix* as mentioned in Table 3.1- no significant points need to be considered.

3.4.3 Solution Methodology

If a given planar or space curve $\mathbf{r}(t)$ is a *polynomial* parametric curve, each equation for obtaining significant points on $\mathbf{r}(t)$ derived in Sections 3.4.1, 3.4.2 is reduced to a *single univariate nonlinear polynomial* equation. Such a polynomial equation $f(t) = 0$ of degree m over an interval $I = [t_l, t_u] \subset \mathfrak{R}$ can be expressed as.

$$\tilde{f}(u) \equiv \sum_{i=0}^m \tilde{f}_i B_{i,m}(u) = 0, \quad u \in [0, 1], \quad (3.45)$$

by Eq.'s (3.19), (3.20). Using the linear precision property (3.23) and the Bézier function $\tilde{f}(u)$, we can create an explicit Bézier curve $\tilde{\mathbf{F}}(u)$ given by, [58]

$$\tilde{\mathbf{F}}(u) = \begin{pmatrix} u \\ \tilde{f}(u) \end{pmatrix} = \sum_{i=0}^m \binom{i/m}{\tilde{f}_i} B_{i,m}(u). \quad (3.46)$$

Therefore, the problem of $\tilde{f}(u) = 0$ can be transformed into a geometric problem of finding the intersections of the Bézier curve $\tilde{\mathbf{F}}(u)$ with the parameter u -axis. Furthermore, in case of $\mathbf{r}(t)$ being a Bézier curve, m and \tilde{f}_i are computed explicitly in terms of the degree and each component of the control points of the Bézier curve $\mathbf{r}(t)$, as stated in Section 3.3.

For the solution of the transformed geometric intersection problem, we use the *interval projected polyhedron algorithm* [46, 58, 59, 85]. The algorithm effectively finds intervals containing *all* real roots of overconstrained, underconstrained and balanced systems of nonlinear polynomial equations using the *de Casteljau* subdivision method coupled with *rounded interval arithmetic* (RIA). A floating point version of the algorithm is also available; however, we need to keep in mind that *floating point arithmetic* (FPA) computations frequently miss ill-conditioned roots.

Before we start the root-solving process we need to decide if an input curve is a planar or space curve and if the curve is planar we solve the equations described in Section 3.4.1 and otherwise solve the equations in Section 3.4.2. We now illustrate how to detect an input curve $\mathbf{r}(t)$, $t \in I$ is a planar or space curve. In case of a planar curve, its torsion $\tau(t)$ vanishes identically, and hence $T_0(t)$ in Eq. (3.41) is identically zero $\forall t \in I$. As stated above, any polynomial function can be transformed into a Bézier form by Eq. (3.45). Thus we can express $T_0(t)$, $t \in I$ as

$$\tilde{T}_0(u) = \sum_{i=0}^m \tilde{f}_i B_{i,m}(u), \quad u \in [0, 1]. \quad (3.47)$$

In order for $\tilde{T}_0(u)$ to vanish $\forall u \in [0, 1]$, each \tilde{f}_i for $i = 0, \dots, m$ should be identically

zero. Therefore, in practical algorithms we compute each \tilde{f}_i and if

$$\text{(FPA)} \quad |\tilde{f}_i| < \epsilon \ll 1, \quad (3.48)$$

$$\text{(RIA)} \quad \tilde{f}_{i_l} \leq 0, \quad \tilde{f}_{i_u} \geq 0 \quad \text{where } \tilde{f}_i \equiv [\tilde{f}_{i_l}, \tilde{f}_{i_u}] \quad (3.49)$$

for each $i = 0, \dots, m$ then we decide the input curve $\mathbf{r}(t)$ is planar.

In a similar way, we can handle the exceptional cases that equations finding significant points vanish identically for e.g., if an input curve is a circular helix, then $K_1(t)$, $T_1(t)$ and $\Omega_1(t)$ in Eq.'s (3.35), (3.42) and (3.44) vanish identically. We first formulate each equation in Bézier form and check whether every Bézier ordinate \tilde{f}_i vanishes identically by using decision criteria in (3.48), (3.49). If it does we do not attempt to solve such equations in order for the algorithm not to be trapped in an exhaustive root solving process.

3.5 Piecewise Linear Approximation of Composite Bézier Curves

Our piecewise linear approximation technique comprises of three steps. The *preliminary* step reflects the differential geometric properties of the input Bézier curves and the *main* approximation efficiently ensures that the approximation error is within a user specified tolerance. The *final* step guarantees the existence of a homeomorphism between a set of exact input curves and their approximation.

3.5.1 Preliminary Approximation

We compute the significant points for each input Bézier curve using the method described in Section 3.4. By using these significant points as well as the boundary points as nodes, we can create a *preliminary* piecewise linear approximation of the set of composite Bézier curves.

In practical algorithms, we actually obtain a list L_{pseg} of parametric segments

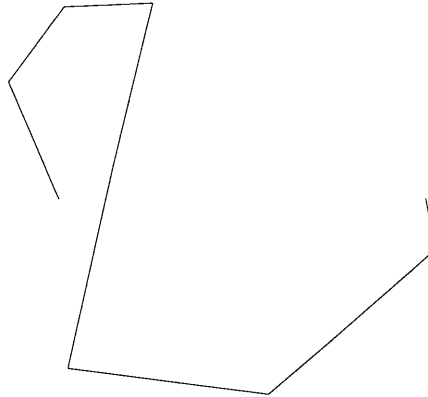


Figure 3-5: A preliminary approximation of a planar Bézier curve of degree 5

$pseg_i$ through the preliminary approximation. By making data structure of the item $pseg_i$ have a pointer to its parent input curve, its corresponding linear segment in 3D space is easily achieved by mapping end vertices of $pseg_i$ through the parent Bézier curve equation $\mathbf{r}(t)$. An example of the preliminary approximation corresponding to the curve in Figure 3-3 is shown in Figure 3-5.

This preliminary approximation, providing the most significant geometric information of input curves, is valuable especially when a *coarse* approximation of good quality is required such as in *finite element meshing* applications. The algorithm for the preliminary approximation would be as follows;

Preliminary-Approximation (L_c)

```

Init (  $L_{pseg}$  )  ▷ initialize the list  $L_{pseg}$  that will store parametric segments  $pseg_i$ 

 $c \leftarrow head[L_c]$   ▷ pick up the first curve from the input Bézier curve list  $L_c$ 

while  $c \neq NIL$   ▷ for each input curve

    Solve (  $c, L_{pseg}$  )  ▷ compute the significant points and append corresponding
                           parametric segments  $pseg_i$  to  $L_{pseg}$ 

     $c \leftarrow next[c]$   ▷ get the next curve element from  $L_c$ 

return  $L_{pseg}$   ▷ return the list  $L_{pseg}$  of parametric segments  $pseg_i$ 

```

3.5.2 Main Approximation

The main approximation begins with computing the approximation error for each 3D linear approximating segment by evaluating a tight upper bound for the deviation between a piece of the exact Bézier curve and the corresponding approximating segment. In general, suppose $\mathbf{r}(t)$, $\mathbf{s}(t)$ are integral Bézier curves of degree n and \mathbf{r}_i , \mathbf{s}_i ($i = 0, 1, 2, \dots, n$) the corresponding control points. Using the *convex hull property* as in Eq.'s (3.21) and (3.22), the absolute position difference $\delta(t)$ at isoparametric points is given by, [69]

$$\delta(t) = \left| \sum_{i=0}^n (\mathbf{r}_i - \mathbf{s}_i) B_{i,n}(t) \right| \leq \max_i |\mathbf{r}_i - \mathbf{s}_i|. \quad (3.50)$$

This convex hull method can be effectively employed to compute an approximation error bound under consideration. We first find an exact Bézier curve segment corresponding to a parametric segment $pseg_j \equiv [t_j, t_{j+1}]$ obtained in preliminary approximation. The exact Bézier curve segment $\mathbf{r}(t)$ is easily achieved by subdividing the parent input Bézier curve of $pseg_j$ at t_j , t_{j+1} using the de Casteljau algorithm, [28, 44]. Let \mathbf{r}_i , \mathbf{s}_i be the control points of the exact Bézier curve segment and those of an approximating linear segment, respectively. In fact, \mathbf{s}_i can be chosen as $n + 1$ points uniformly distributed on the linear segment, as shown in Figure 3-6. Since both end points $\mathbf{s}_0 = \mathbf{r}_0$ and $\mathbf{s}_n = \mathbf{r}_n$, \mathbf{s}_i are simply obtained by

$$\mathbf{s}_i = \mathbf{r}_0 \left(1 - \frac{i}{n}\right) + \frac{i}{n} \mathbf{r}_n, \quad i = 0, 1, 2, \dots, n. \quad (3.51)$$

The upper bound given in Eq. (3.50) is determined only by computing $|\mathbf{r}_i - \mathbf{s}_i|$ for each $i = 1, 2, \dots, n - 1$. We consider this upper bound as a criterion of our approximation error δ , see also Figure 3-6 i.e.,

$$\delta \equiv \max_{1 \leq i \leq n-1} |\mathbf{r}_i - \mathbf{s}_i|. \quad (3.52)$$

If the approximation error δ is not within a user specified tolerance ϵ , a subdivision

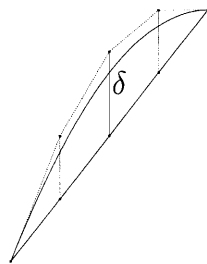


Figure 3-6: Construction of an upper bound δ of the approximation error

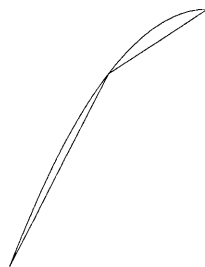


Figure 3-7: Subdivision of a Bézier curve segment at $t = 0.5$

of the parametric segment $pseg_j$ and corresponding Bézier curve segment $\mathbf{r}(t)$, is performed at the midpoint, as shown in Figure 3-7. In practical algorithms, we actually compute δ^2 and compare it with ϵ^2 . Our preliminary approximation described in Section 3.5.1 ensures that every subdivided curve segment is *convex* because the magnitudes of κ , τ and ω are monotonously increasing or decreasing along each piece of curve segments. As we found experimentally, this convexity makes the midpoint to be an appropriate breakpoint of the curve segmentation. Successive subdivisions are performed until every linear approximating segment satisfies the specified tolerance ϵ and finally, an updated list L_{pseg} of the parametric segments $pseg_j$ is obtained. The algorithm may be given as follows:

Main-Approximation (ϵ, L_{pseg})

$c \leftarrow head[L_{pseg}]$ \triangleright pick up the first element from the list L_{pseg} of parametric segments $pseg_i$

while $c \neq NIL$ \triangleright for each $pseg_i$

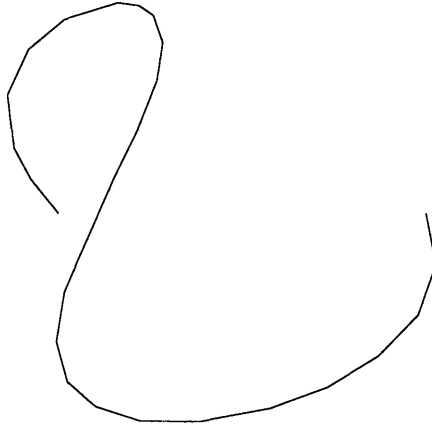


Figure 3-8: A main approximation of a planar Bézier curve of degree 5 with $\epsilon = 10^{-2}$

Error (c, δ) ▷ compute the approximation error δ by using parent Bézier curve of c and Eq. (3.52)

if $\delta^2 > \epsilon^2$ ▷ the error is not within a specified tolerance ϵ

Subdivide (c, L_{pseg}) ▷ subdivide c into two at the midpoint and insert them next to c

$c \leftarrow next[c]$ ▷ get one of the subdivided segments

Delete ($prev[c], L_{pseg}$) ▷ delete the old parametric segment which failed in error test

else

$c \leftarrow next[c]$ ▷ get the next parametric segment from L_{pseg}

return L_{pseg} ▷ return the updated L_{pseg}

For the preliminary approximation illustrated in Figure 3-5, corresponding main approximation with $\epsilon = 10^{-2}$ is shown in Figure 3-8.

3.6 Intersection Test and Final Approximation

Supposing the exact input curves constitute a set of mutually non-intersecting simple composite curves, the linear approximating segments should not have any inappropriate intersection in order to achieve a homeomorphism between the exact and approximated curves [2, 10]. The term *simple composite curve* here denotes a com-

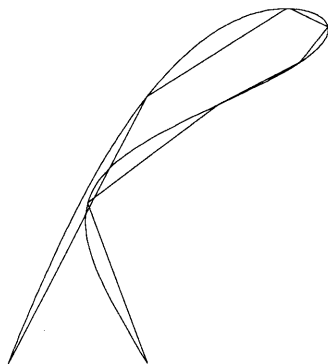


Figure 3-9: A composite Bézier curve involving a constriction and the corresponding linear approximating segments with $\epsilon = 10^{-1}$

posite regular curve without self-intersections. Therefore, the intersection test should be performed to identify possible inappropriate intersections of the linear segments arising from, for example, global distance function features of the input curve such as *constrictions* as depicted in Figure 3-9. If inappropriate intersections exist, they necessitate further local refinement of the approximation until the linear approximating segments do not have any inappropriate intersection. A bucketing technique [5, 19] is used to identify such inappropriate intersections, which runs in $\mathcal{O}(n)$ time on the average, where n is the number of the linear approximating segments. The idea of the bucketing technique is, first to divide the domain into equal-sized $\mathcal{O}(n)$ subdomains, and next to perform the intersection check for linear segments within the same bucket. Since the subdomains are uniformly distributed, we expect $\mathcal{O}(1)$ linear segments on the average to fall into each bucket.

3.6.1 Homeomorphism between the Exact and Approximating Curve

Before we proceed to the final approximation scheme, we prove the existence of a *homeomorphism* between a set of mutually non-intersecting simple composite curves and the corresponding linear approximating segments without any inappropriate intersection. The homeomorphism under consideration is easily constructed by using **Definition 3.6.1** and **Lemma 3.6.2** below.

Definition 3.6.1 We say that topological spaces X and Y are homeomorphic, if there exists a continuous one-to-one mapping $f : X \rightarrow Y$ which has a continuous inverse. Such a map f is called a homeomorphism [2].

Lemma 3.6.2 Let $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$ be topological spaces such that X_1 , X_2 are closed in X and Y_1 , Y_2 are closed in Y . Let $f_1 : X_1 \rightarrow Y_1$, $f_2 : X_2 \rightarrow Y_2$ be homeomorphic which restrict to the same homeomorphism $f_0 : X_1 \cap X_2 \rightarrow Y_1 \cap Y_2$. Then the function

$$f : X \rightarrow Y$$

$$x \mapsto \begin{cases} f_1(x), & x \in X_1 \\ f_2(x), & x \in X_2 \end{cases}$$

is well-defined and a homeomorphism [13].

Theorem 3.6.3 A simple composite curve X comprising of n curve segments is homeomorphic to the corresponding heap Y of n linear approximating segments which do not have any inappropriate intersection.

Proof. The proof is easily achieved by induction. Let the simple composite curve X and the corresponding heap Y of linear segments without inappropriate intersections be $X = \bigcup_{i=1}^n X_i$ and $Y = \bigcup_{i=1}^n Y_i$, where X_i and Y_i are the i^{th} curve and its approximation, respectively. To prove a map $f : X (= \bigcup_{i=1}^n X_i) \rightarrow Y (= \bigcup_{i=1}^n Y_i)$ is a homeomorphism, we first need to show that a curve segment X_1 is homeomorphic to the corresponding linear segment Y_1 , namely a map $f_1 : X_1 \rightarrow Y_1$ is a homeomorphism. This is clearly true because any non-self-intersecting arc described by the continuous motion of a particle moving from an end point to the other end point is *homeomorphic to a line segment* [17]. Assume now a map $f^* : X^* (= \bigcup_{i=1}^k X_i) \rightarrow Y^* (= \bigcup_{i=1}^k Y_i)$ is a homeomorphism for $1 < k < n$ and consider a map $f^{**} : X^{**} (= \bigcup_{i=1}^{k+1} X_i) \rightarrow Y^{**} (= \bigcup_{i=1}^{k+1} Y_i)$. Since $X^{**} = X^* \cup X_{k+1}$ and $Y^{**} = Y^* \cup Y_{k+1}$, each subspace X^* , X_{k+1} are closed in X^{**} and Y^* , Y_{k+1} are also closed in Y^{**} . Furthermore, the map $f^* : X^* \rightarrow Y^*$ is homeomorphic by the assumption and so is a map $f_{k+1} : X_{k+1} \rightarrow Y_{k+1}$ as described in

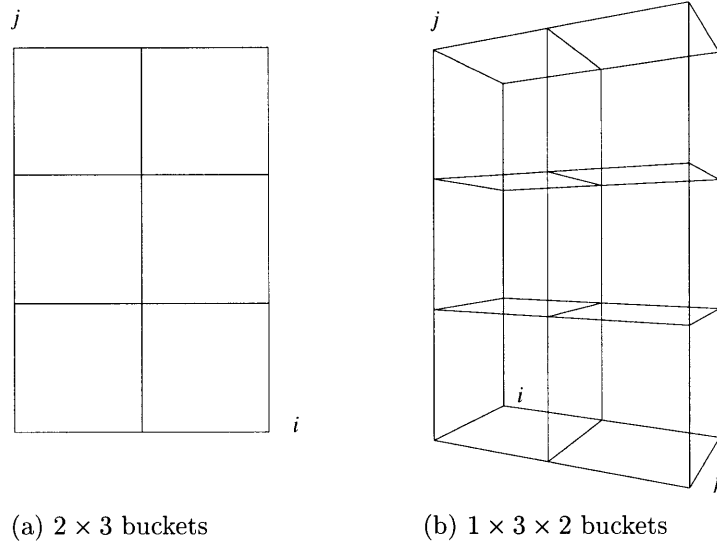


Figure 3-10: 2D and 3D buckets

the proof of f_1 . Moreover, $X^* \cap X_{k+1}$ and $Y^* \cap Y_{k+1}$ is a *node*, say N_k , connecting X^* , X_{k+1} and Y^* , Y_{k+1} . We note the node N_k is the only possible $X^* \cap X_{k+1}$ and $Y^* \cap Y_{k+1}$ since X^{**} is a simple composite curve and Y^{**} does not have any inappropriate intersection. Therefore, a map $f_0 : X^* \cap X_{k+1} \rightarrow Y^* \cap Y_{k+1}$ is equivalent to $N_k \rightarrow N_k$, which is obviously homeomorphic. By Lemma 3.6.2, a map $f^{**} : X^{**} \rightarrow Y^{**}$ is accordingly a homeomorphism. Therefore, a map f is a *homeomorphism* from a simple composite curve X comprising of n curve segments to the corresponding heap Y of n linear approximating segments which do not have inappropriate intersections. \square

3.6.2 Construction of Buckets

The main approximation described in Section 3.5.2 renders a list of linear approximating segments whose approximation error is within a user specified tolerance by mapping the corresponding list L_{pseg} of parametric segments $pseg_i$. Suppose a set of 2D (3D) linear approximating segments falls into a rectangular (rectangular parallelepiped) frame \mathcal{R} with sides parallel to the xy (xyz) axes, respectively. The frame \mathcal{R} is partitioned into $n_x \times n_y$ ($n_x \times n_y \times n_z$) rectangular (rectangular parallelepiped) buckets of equal size. As shown in Figure 3-10, a 2D bucket which is the i^{th} from left and the j^{th} from bottom is denoted by B_{ij} . A 2D *bucket coordinate system* is the

coordinate system obtained by *rescaling* the x and y axes such that the left-bottom corner of bucket B_{ij} has coordinates (i, j) . In a similar manner, a 3D bucket B_{ijk} and the corresponding coordinates (i, j, k) can be defined. We note that a bucket to which an end point of a linear approximating segment belongs is easily found by just taking the integer parts of the bucket coordinates of the end point. Given the number of linear approximating segments n , corresponding n_x and n_y of the 2D frame \mathcal{R} are determined by the formulas :

$$n_x = \lfloor \alpha_x \sqrt{n} \rfloor, \quad n_y = \lfloor \alpha_y \sqrt{n} \rfloor, \quad (3.53)$$

with properly chosen weighting parameters α_x and α_y , where $\lfloor \bullet \rfloor$ denotes the integer part of \bullet . Those parameters α_x and α_y can be appropriately obtained by the following two equations:

$$\frac{\alpha_y}{\alpha_x} = \frac{L_y}{L_x}, \quad \alpha_x \alpha_y = 1, \quad (3.54)$$

where L_x and L_y are the width and height of a rectangle which bounds the set of linear approximating segments in xy coordinate system, as depicted in Figure 3-11. By solving Eq. (3.54), α_x and α_y are determined as

$$\alpha_x = \sqrt{\frac{L_x}{L_y}}, \quad \alpha_y = \sqrt{\frac{L_y}{L_x}}. \quad (3.55)$$

Similarly, n_x , n_y and n_z of the 3D frame \mathcal{R} are determined by the formulas:

$$n_x = \lfloor \alpha_x n^{\frac{1}{3}} \rfloor, \quad n_y = \lfloor \alpha_y n^{\frac{1}{3}} \rfloor, \quad n_z = \lfloor \alpha_z n^{\frac{1}{3}} \rfloor, \quad (3.56)$$

where n is the number of linear approximating segments and

$$\alpha_x = \left(\frac{L_x^2}{L_y L_z} \right)^{\frac{1}{3}}, \quad \alpha_y = \left(\frac{L_y^2}{L_z L_x} \right)^{\frac{1}{3}}, \quad \alpha_z = \left(\frac{L_z^2}{L_x L_y} \right)^{\frac{1}{3}}. \quad (3.57)$$

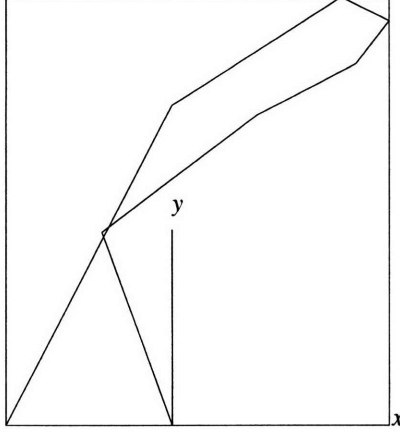


Figure 3-11: A rectangle which bounds the linear segments in xy coordinate system

If n_x , n_y or n_z in Eq.'s (3.53) and (3.56) happen to be zero, we reset them to be one. The number of buckets constructed is obviously $\mathcal{O}(n)$ and consequently the number of linear approximating segments to fall into each bucket will be $\mathcal{O}(1)$ on the average.

3.6.3 Preprocessing – Putting Linear Segments into Buckets

At the preprocessing stage, we associate each linear approximating segment with the buckets constructed in Section 3.6.2. For each linear approximating segment \mathbf{l}_p in xy (xyz) coordinate system, we first perform a transformation with respect to bucket coordinate system ij (ijk), respectively, i.e.,

$$\mathbf{l}_p \equiv (1 - s)\mathbf{b}_p + s\mathbf{e}_p \xrightarrow{T} (1 - s)\tilde{\mathbf{b}}_p + s\tilde{\mathbf{e}}_p \equiv \tilde{\mathbf{l}}_p, \quad s \in [0, 1], \quad p = 1, 2, \dots, n, \quad (3.58)$$

where vectors \mathbf{b}_p , \mathbf{e}_p are end points of the linear segment \mathbf{l}_p and the tilde $\tilde{}$ represents the corresponding transformed version. Those transformed end points $\tilde{\mathbf{b}}_p$, $\tilde{\mathbf{e}}_p$ in 2D case are obtained by the following relations:

$$\tilde{\mathbf{b}}_p = (\tilde{b}_{pi}, \tilde{b}_{pj}), \quad \tilde{\mathbf{e}}_p = (\tilde{e}_{pi}, \tilde{e}_{pj}), \quad (3.59)$$

and

$$\tilde{b}_{pi} = \frac{n_x}{L_x}(b_{px} - x_{\min}), \quad \tilde{b}_{pj} = \frac{n_y}{L_y}(b_{py} - y_{\min}), \quad (3.60)$$

$$\tilde{e}_{pi} = \frac{n_x}{L_x}(e_{px} - x_{\min}), \quad \tilde{e}_{pj} = \frac{n_y}{L_y}(e_{py} - y_{\min}), \quad (3.61)$$

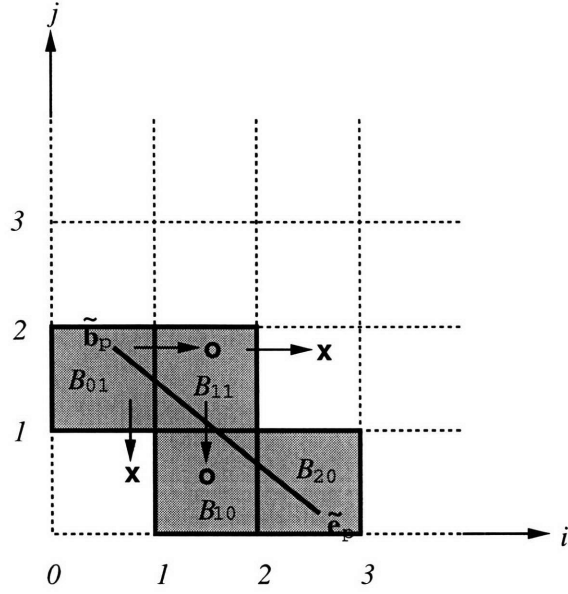


Figure 3-12: Putting a 2D linear segment $\tilde{\mathbf{I}}_p$ into buckets B_{ij}

where b_{px} , b_{py} , e_{px} , e_{py} are x , y components of the end points \mathbf{b}_p , \mathbf{e}_p and (x_{\min}, y_{\min}) are coordinates of the left-bottom corner of a bounding rectangle shown in Figure 3-11 and n_x , n_y , L_x , L_y are described in Eq.'s (3.53) and (3.54). In a similar manner, the transformed end points $\tilde{\mathbf{b}}_p$, $\tilde{\mathbf{e}}_p$ in 3D case are determined as:

$$\tilde{\mathbf{b}}_p = (\tilde{b}_{pi}, \tilde{b}_{pj}, \tilde{b}_{pk}), \quad \tilde{\mathbf{e}}_p = (\tilde{e}_{pi}, \tilde{e}_{pj}, \tilde{e}_{pk}), \quad (3.62)$$

where

$$\tilde{b}_{pk} = \frac{n_z}{L_z}(b_{pz} - z_{\min}), \quad \tilde{e}_{pk} = \frac{n_z}{L_z}(e_{pz} - z_{\min}), \quad (3.63)$$

and \tilde{b}_{pi} , \tilde{b}_{pj} , \tilde{e}_{pi} , \tilde{e}_{pj} are defined in Eq.'s (3.60) and (3.61).

We now put each 2D (3D) transformed linear approximating segment $\tilde{\mathbf{I}}_p$ into buckets B_{ij} (B_{ijk}), respectively. Buckets containing the end points $\tilde{\mathbf{b}}_p$ and $\tilde{\mathbf{e}}_p$ are easily determined by just taking the integer parts of the end points for example, buckets B_{01} and B_{20} in Figure 3-12. To determine intermediate buckets through which the transformed linear segment $\tilde{\mathbf{I}}_p$ passes, a *line-type* of $\tilde{\mathbf{I}}_p$, associated with the *signs* of Δi , Δj and Δk , is identified, where

$$\Delta i = \tilde{e}_{pi} - \tilde{b}_{pi}, \quad \Delta j = \tilde{e}_{pj} - \tilde{b}_{pj}, \quad \Delta k = \tilde{e}_{pk} - \tilde{b}_{pk}. \quad (3.64)$$

Once the line-type is identified, we can easily trace the linear segment $\tilde{\mathbf{l}}_p$. As an illustrative example, we consider a 2D linear segment $\tilde{\mathbf{l}}_p$ shown in Figure 3-12. It is obvious that $\tilde{\mathbf{l}}_p$ has positive Δi and negative Δj . Therefore, starting from the bucket B_{01} which contains the starting end point $\tilde{\mathbf{b}}_p$, we need to check whether $\tilde{\mathbf{l}}_p$ crosses the bottom edge or the right edge of each bucket it passes through. This procedure is repeated until an intermediate bucket, adjacent to B_{20} which contains the other end point $\tilde{\mathbf{e}}_p$, is determined. As illustrated in Figure 3-12, $\tilde{\mathbf{l}}_p$ crosses the right edge of the bucket B_{01} . An intermediate bucket B_{11} is thus simply determined. Similarly, we next check if $\tilde{\mathbf{l}}_p$ crosses the bottom edge or the right edge of the bucket B_{11} , and consequently we obtain another intermediate bucket B_{10} . We can easily extend the idea of this procedure to 3D transformed linear approximating segment $\tilde{\mathbf{l}}_p$ and the corresponding intermediate buckets B_{ijk} . The only difference is that we have more possible line-types of $\tilde{\mathbf{l}}_p$ associated with Δk defined in Eq. (3.64) and we need to consider face-crossing of $\tilde{\mathbf{l}}_p$ instead of edge-crossing in 2D case. We note that in general, very few numbers of intermediate buckets exist due to the characteristics of the bucketing technique - the number of uniformly distributed equally sized buckets $\sim \mathcal{O}(n)$, where n is the number of linear approximating segments.

Suppose the *preprocessing* step is completed for each 2D (3D) transformed linear approximating segment $\tilde{\mathbf{l}}_p$, each bucket B_{ij} (B_{ijk}) is associated with the corresponding linear segments. In practical algorithms, a bucket B_{ij} (B_{ijk}) is no other than a *pointer* to the list of associated linear segments. For example, provided that a 2D bucket B_{01} is associated with three 2D transformed linear approximating segments $\tilde{\mathbf{l}}_3$, $\tilde{\mathbf{l}}_4$ and $\tilde{\mathbf{l}}_{100}$ as shown in Figure 3-13, then data for the bucket B_{01} can be represented as

$$B_{01} \longrightarrow \tilde{\mathbf{l}}_3 \longrightarrow \tilde{\mathbf{l}}_4 \longrightarrow \tilde{\mathbf{l}}_{100}. \quad (3.65)$$

These data for the buckets can be constructed in $\mathcal{O}(n)$ time and space on the average. Finally, 2×2 buckets and eight transformed linear approximating segments corresponding to the linear segments shown in Figure 3-11 are depicted in Figure 3-14.

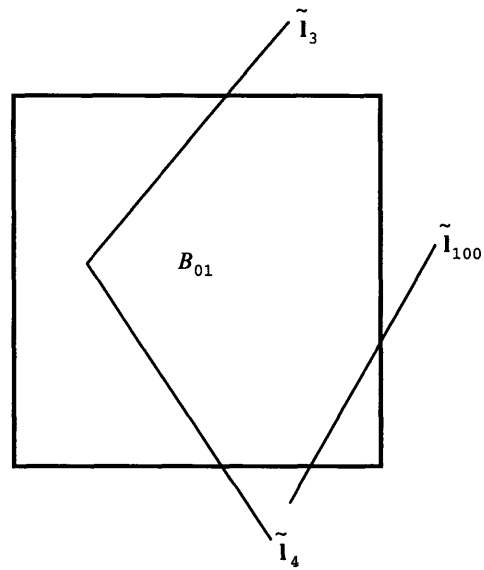


Figure 3-13: A 2D bucket associated with 3 transformed linear segments

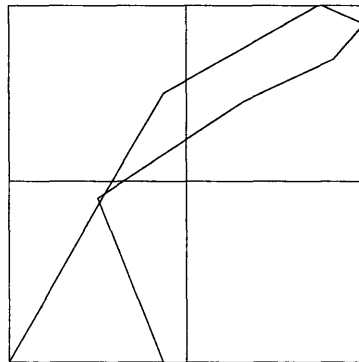


Figure 3-14: 2×2 2D buckets containing 8 transformed linear approximating segments

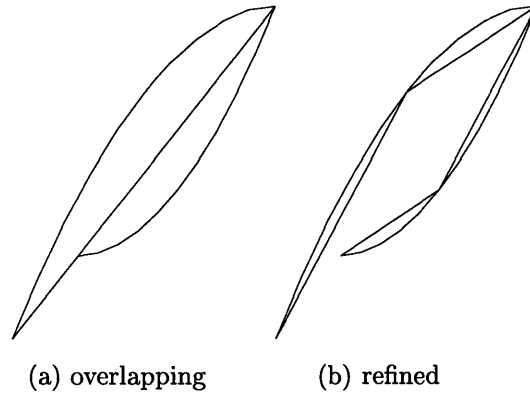


Figure 3-15: Input curves and linear approximations involving overlapping intersection

3.6.4 Intersection Check and Final Approximation

For each bucket B_{ij} (B_{ijk}), an intersection check is performed to identify inappropriate intersections of the linear approximating segments. The intersection test begins with inquiring whether at least two linear segments exist in a bucket B_{ij} (B_{ijk}). If it is true, an intersection check between each pair of transformed linear segments associated with B_{ij} (B_{ijk}) is performed. If a pair transformed linear approximating segments $\tilde{\mathbf{l}}_q(s)$ and $\tilde{\mathbf{l}}_r(t)$ inappropriately intersect at those parameter values $s = s_0$ and $t = t_0$, i.e.,

$$\tilde{\mathbf{l}}_q(s_0) \equiv (1 - s_0)\tilde{\mathbf{b}}_q + s_0\tilde{\mathbf{e}}_q = (1 - t_0)\tilde{\mathbf{b}}_r + t_0\tilde{\mathbf{e}}_r \equiv \tilde{\mathbf{l}}_r(t_0), \quad 0 \leq s_0, t_0 \leq 1, \quad (3.66)$$

further local refinement of the approximation is necessitated. We first pick up two parametric segments $pseg_q(s)$ and $pseg_r(t)$ from the list L_{pseg} corresponding to $\tilde{\mathbf{l}}_q(s)$ and $\tilde{\mathbf{l}}_r(t)$. Subdivision of corresponding Bézier curve segments $\mathbf{R}_q(s)$ and $\mathbf{R}_r(t)$ at $s = s_0$ and $t = t_0$ is performed. In case $\tilde{\mathbf{l}}_q(s)$ and $\tilde{\mathbf{l}}_r(t)$ *overlap*, the subdivision is performed at $s = t = 0.5$, as illustrated in Figure 3-15. The bucketing procedure described in Sections 3.6.2, 3.6.3 and the intersection test are then repeated for a new list of parametric segments until every linear approximating segment does not inappropriately intersect another linear segment. This procedure, called *final* approximation, is described in Figures 3-16 and 3-17 corresponding to the example shown in Figures 3-9 and 3-14.

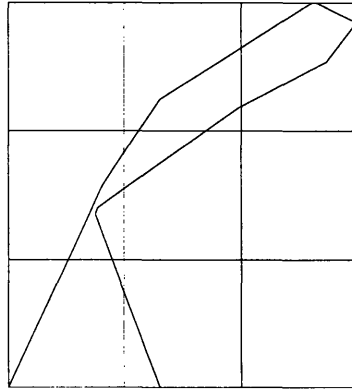


Figure 3-16: 3×3 2D buckets containing 10 transformed linear approximating segments



Figure 3-17: A composite curve with constriction and its approximation guaranteeing a prescribed *tolerance* and the existence of a *homeomorphism* between the exact and approximating curves

3.7 Linear Approximation of Interval Bézier Curves

3.7.1 Introduction

State-of-the-art CAD systems, operating in *floating point arithmetic* (FPA), frequently fail. The ultimate reason for this failure is the *limited precision* of geometric computations. For example, in the course of successive subdivisions of an exact curve, subdividing points evaluated using FPA are close to but not precisely on the exact curve. This gap increases as the subdivision progresses. Recalling our approximation scheme, tighter approximation tolerance necessitates more subdivisions of the exact curve and therefore, numerical errors associated with nodes of the linear segments are increased as the tolerance becomes tighter. However, in many practical applications of linear approximation, provided that an input curve is *precisely* defined and only a moderate approximation tolerance ϵ is required, the numerical error associated with FPA is frequently negligible compared with ϵ . On the other hand, suppose an input curve already has a significant geometric *uncertainty* with an ill-posed configuration, its approximation may result in more serious ambiguities propagating in the process of approximation. To cope with this adversity, we use *interval* Bézier curves computed in *rounded interval arithmetic* (RIA). RIA [66], which can be implemented effectively in object-oriented language such as C++, leads to numerical robustness and provides results with numerical certainty and verifiability.

Interval Bézier curves differ from classical Bézier curves in that the real numbers representing control point coordinates are replaced by *intervals*. Such curves have been used in [81, 82] for approximating parametric curves and their offsets, in [91] for the representation of functions with uncertainty, in [46, 59] for solving shape interrogation problems robustly. In these curves, the classical control points are replaced by rectangular boxes. This implies that, in 3D space, interval Bézier curves represent *slender tubes*, if the intervals are chosen sufficiently small. Linear approximating segments corresponding to such interval Bézier curves consequently have the form of *straight slender beams* connected at each subdividing interval point. An interval straight line is defined as the largest area (volume) obtained by connecting each ver-

text \mathbf{b}_i and \mathbf{e}_i of two interval points \mathbf{b} and \mathbf{e} where $i = 1, 2, 3, 4$ ($i = 1, 2, \dots, 8$) in 2D (3D) case, respectively. Figure 3-18 illustrates a 2D interval linear segment $\mathbf{l}(t)$ which can be still represented by a vector formula :

$$\mathbf{l}(t) = (1 - t)\mathbf{b} + t\mathbf{e}, \quad t \in [0, 1], \quad (3.67)$$

where \mathbf{b} and \mathbf{e} are interval end points of $\mathbf{l}(t)$. In the following two Sections, we apply

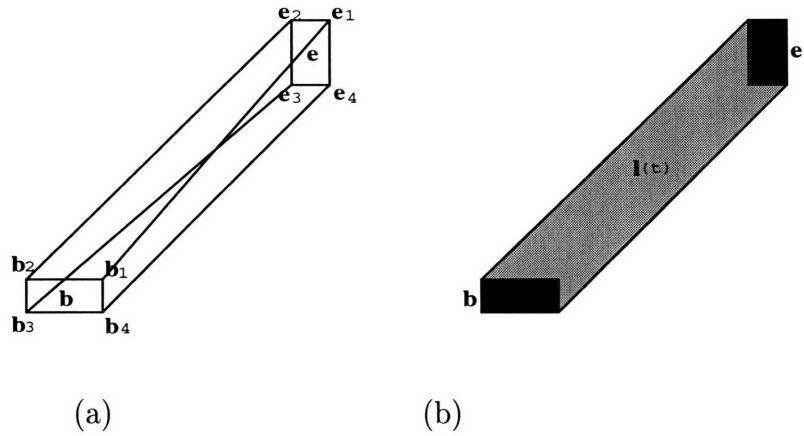


Figure 3-18: A 2D interval linear segment $\mathbf{l}(t)$

the approximation scheme described in Sections 3.4, 3.5 and 3.6 to the interval Bézier curves.

3.7.2 Preliminary and Main Approximation

Given a set of planar or space interval Bézier curves, we formulate the problem as in Section 3.4 using RIA to determine significant points of the input interval Bézier curves. The *interval* projected polyhedron algorithm mentioned in Section 3.4.3 is next employed to solve robustly single univariate nonlinear polynomial equations. This algorithm actually provides *intervals* containing t_j 's where $t_j = [t_{jl}, t_{ju}]$ containing roots. Averaging is then performed for each root t_j ,

$$\bar{t}_j = \frac{t_{jl} + t_{ju}}{2}. \quad (3.68)$$

Preliminary approximation is accordingly achieved by subdividing each interval Bézier curve at \bar{t}_j 's. For each preliminary linear approximating segment, an approximation error δ given in Eq. (3.52) is computed. We note here that the computed approximation error δ is also an *interval* number such as $\delta = [\delta_l, \delta_u]$. In order to guarantee a prescribed tolerance ϵ , we choose δ_u as an approximation error. The approximation error δ_u is clearly overestimated compared with δ obtained by FPA. Successive subdivisions are performed until every interval linear approximating segment satisfies the prescribed tolerance ϵ (as in the main approximation described in Section 3.5.2).

3.7.3 Intersection Test and Final Approximation

A bucketing technique described in Section 3.6 is now performed to identify inappropriate intersections of interval linear approximating segments. Supposing such an intersection exists, Eq. (3.66) gives the corresponding interval parametric roots $s_0 = [s_{0l}, s_{0u}]$ and $t_0 = [t_{0l}, t_{0u}]$. After averaging each interval root s_0 and t_0 , i. e. :

$$\bar{s}_0 = \frac{s_{0l} + s_{0u}}{2}, \quad \bar{t}_0 = \frac{t_{0l} + t_{0u}}{2}, \quad (3.69)$$

subdivisions of the corresponding interval Bézier curve segments are performed at \bar{s}_0 and \bar{t}_0 . This bucketing procedure and intersection test is repeated until every interval linear approximating segment does not involve intersections, as described in Section 3.6.4. As an example, we *artificially* impose $\mathcal{O}(10^{-3})$ interval on each control point coordinate of the curve shown in Figure 3-9. Figure 3-19 – see also Figure 3-18 (a) – shows locally refined interval linear approximating segments near the constriction.

We now illustrate the importance of RIA in the fully *robust* intersection test between the linear approximating segments. Even if our intersection problem is the simplest line to line intersection problem, blind use of FPA always has a possibility of missing root [41], which leads to a topological inconsistency between the exact and approximating curves. Such topological inconsistency eventually results in system failure provided that further geometric computations or analysis are performed on those topologically inconsistent approximations. For instance, we consider two pieces

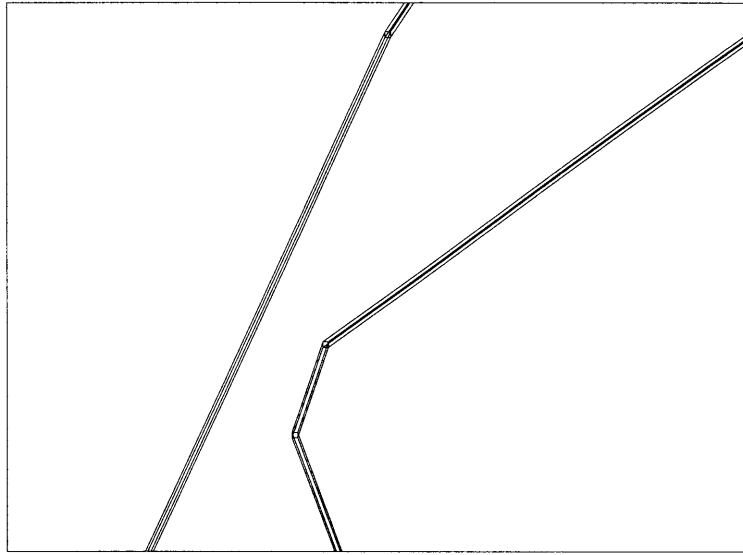


Figure 3-19: Interval linear approximating segments near constriction

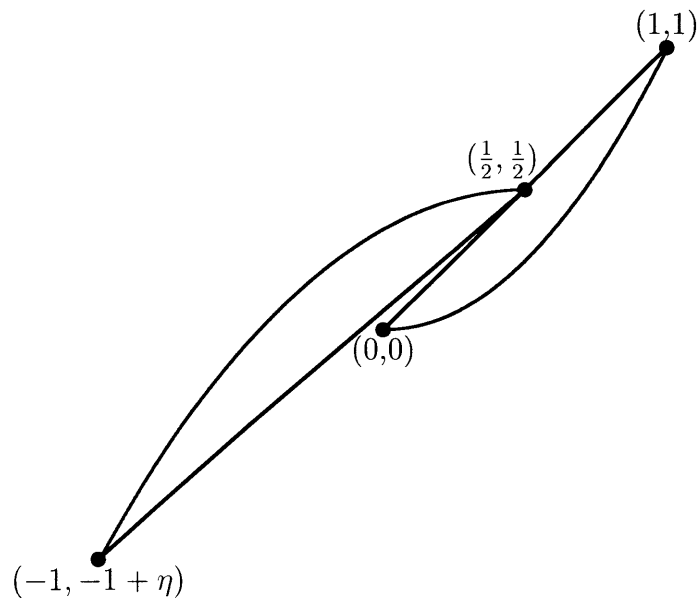


Figure 3-20: Two pieces of input curve segments and their approximation, $0 < |\eta| \ll 1$

of exact curve segments and the corresponding linear approximating segments shown in Figure 3-20. Each 2D approximating segment can be exactly described by the following formulas,

$$\mathbf{l}_1(s) = (1 - s)\mathbf{b}_1 + s\mathbf{e}_1, \quad \mathbf{l}_2(t) = (1 - t)\mathbf{b}_2 + t\mathbf{e}_2, \quad (3.70)$$

where $\mathbf{b}_1 = (0, 0)$, $\mathbf{e}_1 = (1, 1)$, $\mathbf{b}_2 = (-1, -1 + \eta)$, $\mathbf{e}_2 = (\frac{1}{2}, \frac{1}{2})$ and $0 \leq s, t \leq 1$, $0 < |\eta| \ll 1$. Obviously, intersection test must identify the inappropriate intersection at $s = \frac{1}{2}$ and $t = 1$. However, the order of η can make the problem highly ill-conditioned and correspondingly, blind use of Eq. (3.66) operating in FPA may miss the intersection root. For example, we set $\eta = 10^{-6}$ and apply single precision FPA and the corresponding root is $s = 0.625$, $t = 1.0625$. Since $t > 1$ the intersection module returns $\mathbf{l}_1 \cap \mathbf{l}_2 = \emptyset$ while RIA reports the inappropriate intersection at $s = [0.5, 0.5]$ and $t = [1.0, 1.0]$.

3.8 Complexity Analysis

In this Section, we perform complexity analysis for each approximation procedure, i.e., preliminary, main and final approximation.

3.8.1 Preliminary Approximation

In the preliminary approximation stage, the dominant part is the computation of significant points of input curves. For simplicity, *we assume that each input curve is of the same degree m* in the following analysis. Given a univariate polynomial equation of degree p , for example $p = 18m - 27$ in Eq. (3.44), the projected polyhedron algorithm takes $\mathcal{O}(p^2)$ for each iteration [85] and therefore, time complexity of the algorithm can be described as $\mathcal{O}(N_{iter}p^2)$ where N_{iter} is the number of iterations to get the solution. In fact N_{iter} depends on the root tolerance and the characteristics of the input curve, in other words as the root tolerance becomes tighter or the input curve causes tangential roots, N_{iter} is obviously increased. If the total number of

input curves is N_c and if N_{iter} is the average number of iterations for each input curve, then preliminary approximation takes $\mathcal{O}(N_c N_{iter} m^2)$ on the average.

3.8.2 Main Approximation

For each parametric segment $pseg_i$ ($i = 1, 2, \dots, N_{pre}$) obtained by the preliminary approximation, we perform $(N_i - 1)$ number of binary subdivisions to get N_i parametric segments which generate corresponding N_i linear approximating segments satisfying the approximation tolerance ϵ . For each subdivision, de Casteljau algorithm is employed to extract corresponding sub-Bézier curve segment, which runs in $\mathcal{O}(m^2)$ and the approximation error δ is computed in $\mathcal{O}(m)$. Therefore, main approximation requires $\mathcal{O}(N_{pre} N_{avg} m^2)$ time complexity on the average, where $N_{avg} \equiv \frac{1}{N_{pre}} \sum_{i=1}^{N_{pre}} N_i$. In fact, N_i depends on the approximation tolerance ϵ and intrinsic features of the curve segment $\mathbf{r}_i(t)$ obtained by the preliminary approximation. Unfortunately, the *exact* relationship between N_i and those parameters is not available for our main approximation scheme however, we can describe the *average lower bound* as $N_i = \Omega(\sqrt{\frac{\bar{\kappa}_i}{\epsilon}})$ by utilizing the theorem reported in [32] and assuming the curve segment is parametrized along the arc length s , where $\bar{\kappa}_i$ is the average curvature for $s \in [s_{il}, s_{iu}] \equiv pseg_i$. Here we note $\bar{\kappa}_i$ is bounded within $[\min(\kappa_i(s_{il}), \kappa_i(s_{iu})), \max(\kappa_i(s_{il}), \kappa_i(s_{iu}))]$ since our preliminary approximation produces sub-Bézier curve segment $\mathbf{r}_i(s)$ whose curvature $\kappa_i(s)$ is monotonously varying along $s_{il} \leq s \leq s_{iu}$.

3.8.3 Intersection Test and Final Approximation

Main approximation results in $N_{main} (\approx N_{pre} N_{avg})$ number of parametric segments $pseg_i$. We next construct corresponding buckets in $\mathcal{O}(N_{main})$ as described in Section 3.6.2. Since $\mathcal{O}(1)$ linear approximating segments are associated in each bucket, time complexity for the intersection test is also $\mathcal{O}(N_{main})$. In case that an inappropriate intersection between a pair of linear approximating segments is detected, we bisect such segments - the number of linear segments is increased by 2 - and immediately update the buckets for the further intersection test. Therefore, total complexity is

described as $\mathcal{O}(f)$ where

$$f = \sum_{i=0}^{N_{inter}} (N_{main} + 2i) = (N_{inter} + 1)N_{main} + (N_{inter} + 1)(N_{inter} + 2), \quad (3.71)$$

where N_{inter} is the total number of pairs of intersecting segments detected during the test. By taking the leading terms in Eq. (3.71), the total complexity will be $\mathcal{O}(N_{inter}N_{main} + N_{inter}^2)$ for $N_{inter} > 0$ and $\mathcal{O}(N_{main})$ in case $N_{inter} = 0$.

3.9 Examples

We have implemented our method on a graphics workstation running at 150 MHz. For each exact input Bézier curve, we perform the approximation with rounded interval arithmetic (RIA) as well as floating point arithmetic (FPA). In the following examples, we fix the *root tolerance* of the projected polyhedron algorithm to be 10^{-6} during the process of determining significant points of input curves, regardless of the approximation tolerance ϵ . A 10^{-6} root tolerance is sufficiently tight compared with the typical approximation tolerance ϵ such as in finite element meshing. Notations used in Tables 3.2 – 3.7 are summarized as follows:

- ϵ : User specified approximation tolerance (normalized)
- n : Number of linear approximating segments
- FPA : Floating point arithmetic
- RIA : Rounded interval arithmetic
- *Preliminary* : Preliminary approximation
- *Main* : Main approximation
- *Final* : Intersection test and final approximation
- **Total** : Sum of CPU-times in *Preliminary*, *Main* and *Final*

Tables 3.2 – 3.7 together with Figures 3-21 – 3-26 illustrate the performance for test curves. Overall time cost of FPA is very low for both 2D and 3D cases. However, there are two problems associated with FPA. The first one is the possibility of missing significant points due to numerical errors in the projected polyhedron algorithm. Such a situation is typically encountered when an input curve has locally *flat* configuration. For example, given a planar curve $y = (x - \alpha)^m$ of *high* degree m , the projected polyhedron algorithm – like other existing polynomial solvers – operating in FPA with *too tight* a tolerance, say 10^{-8} , fails to find a significant point $x = \alpha$ corresponding to the equations $\kappa = 0$, $\kappa' = 0$, while RIA finds it with a relatively high time cost. This unsatisfactory result comes from the ill-conditioned root $x = \alpha$ of high multiplicity. This fact could be also applied to the equations associated with torsion in 3D case. Nevertheless, as far as our linear approximation problem is concerned, this is not a serious adversity. Because, in general, too tight a root tolerance is not necessary in the preliminary approximation step, and moreover, in a practical sense, a locally *flat* region is not problematic for linear approximation. However, as discussed in Section 3.7.3, RIA is inevitably necessitated for the fully robust intersection test between the linear approximating segments. Otherwise, there is always a possibility of *topological inconsistency* between the exact input curves and their approximations. Another problem associated with FPA arises when a floating point linear approximation, corresponding to an input curve which inherently has a significant geometric uncertainty, results in serious ambiguities propagating in the approximation procedure. This is an apparent adversity of FPA. Operations in RIA for interval input curves are consequently necessitated to capture the propagation of ambiguities during the process of main and final approximations. Some conclusions drawn from our investigation associated with FPA and RIA are as follows : suppose an input curve is precisely defined and a user specified approximation tolerance is moderate, FPA is recommended for the whole process of approximation except for the *final* approximation stage in order to guarantee a robust intersection test. On the other hand, in case an input curve has a significant geometric uncertainty with an ill-posed configuration, FPA with moderate root tolerance may still be used in the root-finding stage

but RIA should be used for the rest of the approximation algorithms – subdivision in preliminary approximation, main and final approximations.

Figure 3-23 together with Table 3.4 demonstrate the importance of the *final* approximation stage. Were it not for the intersection test, a homeomorphism between the exact and approximating curves could not be guaranteed for $\epsilon = 10^{-1}$.

In general, provided that a prescribed approximation tolerance ϵ is loose enough, say 10^{-1} , the preliminary approximation satisfies the ϵ -tolerance with a satisfactorily small number of linear approximating segments. According to our numerical experiments, a *blind* uniform binary subdivision algorithm produces about 40 % more number of linear approximating segments, if the approximation tolerance is loose enough and the input curves are of high degree with irregular distribution of intrinsic features. However, sometimes in 3D case, the preliminary approximation produces unnecessarily many linear approximating segments despite the loose ϵ being required, as shown in Table 3.6 and Figure 3-25. The reason for this problem is that the number of local extrema points of curvature κ , torsion τ and total curvature ω is large. In such a case, some of the significant points obeying $\kappa' = 0$, $\omega' = 0$ or those of $\tau' = 0$, $\omega' = 0$ are located quite *close* to each other due to the characteristics of those equations described in Section 3.4.2. This feature results in a large number of unnecessary subdivisions of an input curve in a small region. To overcome this, we may actually omit solving $\kappa' = 0$ and $\tau' = 0$ in our preliminary approximation scheme, still capturing the effect of both intrinsic features of a space curve. This alternative preliminary approximation scheme, denoted by *Preliminary** in Table 3.7, may be described as follows: in 3D case, if our conventional preliminary approximation scheme attempts large number of unnecessary subdivisions in a small region of an input curve, despite only a very loose ϵ being required, we determine those significant points corresponding to $\kappa = 0$, $\tau = 0$ and $\omega' = 0$ with FPA and perform the subdivisions with FPA or RIA. Table 3.7 and Figure 3-26 show the *satisfactory* results based on the modified preliminary approximation scheme – see also Table 3.6 and Figure 3-25 for the comparison. In Table 3.7, (FPA/RIA) represents that the root-solving is performed in FPA and the corresponding subdivision of input curves

is executed in RIA.

| Tolerance (ϵ) | CPU-time (sec) | | | | | | | | n |
|--------------------------|--------------------|------|-------------|------|--------------|------|--------------|------|-----|
| | <i>Preliminary</i> | | <i>Main</i> | | <i>Final</i> | | Total | | |
| | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | |
| 10^{-1} | 0.05 | 1.49 | 0.00 | 0.12 | 0.00 | 0.07 | 0.05 | 1.68 | 8 |
| 10^{-2} | 0.05 | 1.49 | 0.02 | 0.43 | 0.01 | 0.20 | 0.08 | 2.12 | 24 |
| 10^{-3} | 0.05 | 1.49 | 0.11 | 1.63 | 0.04 | 0.69 | 0.20 | 3.81 | 81 |

Table 3.2: Result for Figure 3-21

| Tolerance (ϵ) | CPU-time (sec) | | | | | | | | n |
|--------------------------|--------------------|------|-------------|------|--------------|------|--------------|------|-----|
| | <i>Preliminary</i> | | <i>Main</i> | | <i>Final</i> | | Total | | |
| | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | |
| 10^{-1} | 0.16 | 2.50 | 0.02 | 0.24 | 0.01 | 0.21 | 0.19 | 2.95 | 37 |
| 10^{-2} | 0.16 | 2.50 | 0.03 | 0.28 | 0.02 | 0.25 | 0.21 | 3.03 | 44 |
| 10^{-3} | 0.16 | 2.50 | 0.08 | 0.85 | 0.04 | 0.59 | 0.28 | 3.94 | 102 |

Table 3.3: Result for Figure 3-22

| Tolerance (ϵ) | CPU-time (sec) | | | | | | | | n |
|--------------------------|--------------------|------|-------------|------|--------------|------|--------------|------|-----|
| | <i>Preliminary</i> | | <i>Main</i> | | <i>Final</i> | | Total | | |
| | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | |
| 10^{-1} | 0.04 | 0.65 | 0.01 | 0.09 | 0.02 | 0.33 | 0.07 | 1.07 | 16 |
| 10^{-2} | 0.04 | 0.65 | 0.02 | 0.30 | 0.01 | 0.19 | 0.07 | 1.14 | 32 |
| 10^{-3} | 0.04 | 0.65 | 0.09 | 1.03 | 0.04 | 0.68 | 0.17 | 2.36 | 112 |

Table 3.4: Result for Figure 3-23

| Tolerance (ϵ) | CPU-time (<i>sec</i>) | | | | | | | | <i>n</i> |
|--------------------------|-------------------------|------|-------------|------|--------------|------|--------------|-------|----------|
| | <i>Preliminary</i> | | <i>Main</i> | | <i>Final</i> | | Total | | |
| | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | |
| 10^{-1} | 0.29 | 8.23 | 0.02 | 0.19 | 0.01 | 0.09 | 0.32 | 8.51 | 10 |
| 10^{-2} | 0.29 | 8.23 | 0.03 | 0.58 | 0.02 | 0.32 | 0.34 | 9.13 | 26 |
| 10^{-3} | 0.29 | 8.23 | 0.10 | 1.56 | 0.04 | 0.84 | 0.43 | 10.63 | 74 |

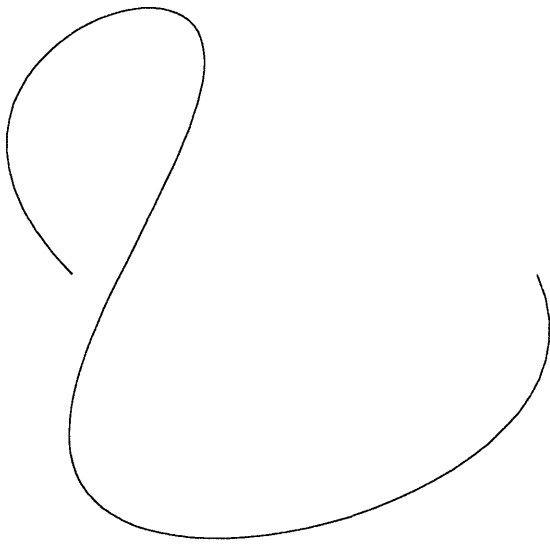
Table 3.5: Result for Figure 3-24

| Tolerance (ϵ) | CPU-time (<i>sec</i>) | | | | | | | | <i>n</i> |
|--------------------------|-------------------------|-------|-------------|------|--------------|------|--------------|-------|----------|
| | <i>Preliminary</i> | | <i>Main</i> | | <i>Final</i> | | Total | | |
| | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | |
| 10^{-1} | 0.65 | 15.65 | 0.04 | 0.38 | 0.01 | 0.29 | 0.70 | 16.32 | 38 |
| 10^{-2} | 0.65 | 15.65 | 0.04 | 0.54 | 0.02 | 0.37 | 0.71 | 16.56 | 49 |
| 10^{-3} | 0.65 | 15.65 | 0.11 | 1.37 | 0.05 | 0.94 | 0.81 | 17.96 | 106 |

Table 3.6: Result for Figure 3-25

| Tolerance (ϵ) | CPU-time (<i>sec</i>) | | | | <i>n</i> |
|--------------------------|----------------------------------|----------------------|-----------------------|---------------|----------|
| | <i>Preliminary*</i> (FPA/RIA) | <i>Main</i> (RIA) | <i>Final</i> (RIA) | Total* | |
| 10^{-1} | 0.59 | 0.19 | 0.13 | 0.91 | 18 |
| 10^{-2} | 0.59 | 0.47 | 0.26 | 1.32 | 37 |
| 10^{-3} | 0.59 | 1.33 | 0.85 | 2.77 | 98 |

Table 3.7: Result for Figure 3-26



(a) exact

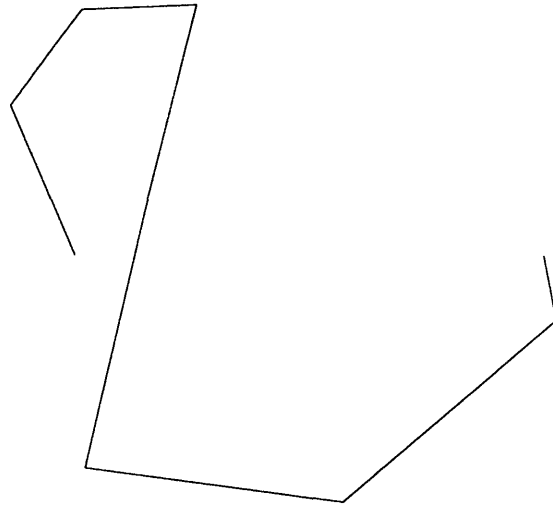
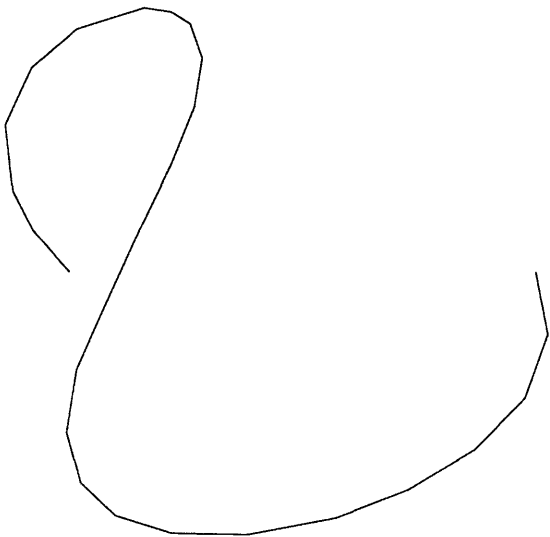
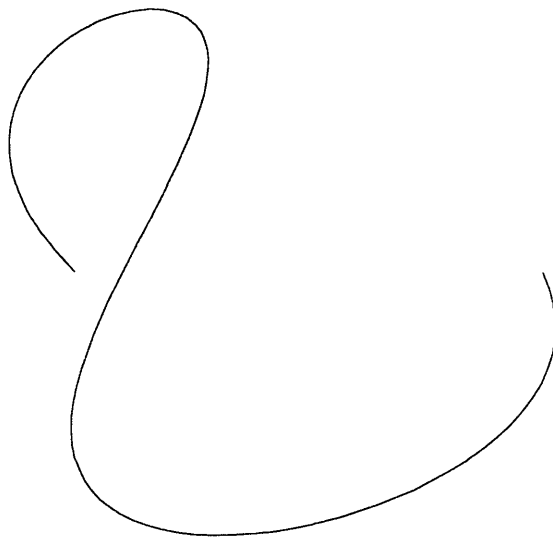
(b) $\epsilon = 10^{-1}$ (c) $\epsilon = 10^{-2}$ (d) $\epsilon = 10^{-3}$

Figure 3-21: A planar Bézier curve of degree 5

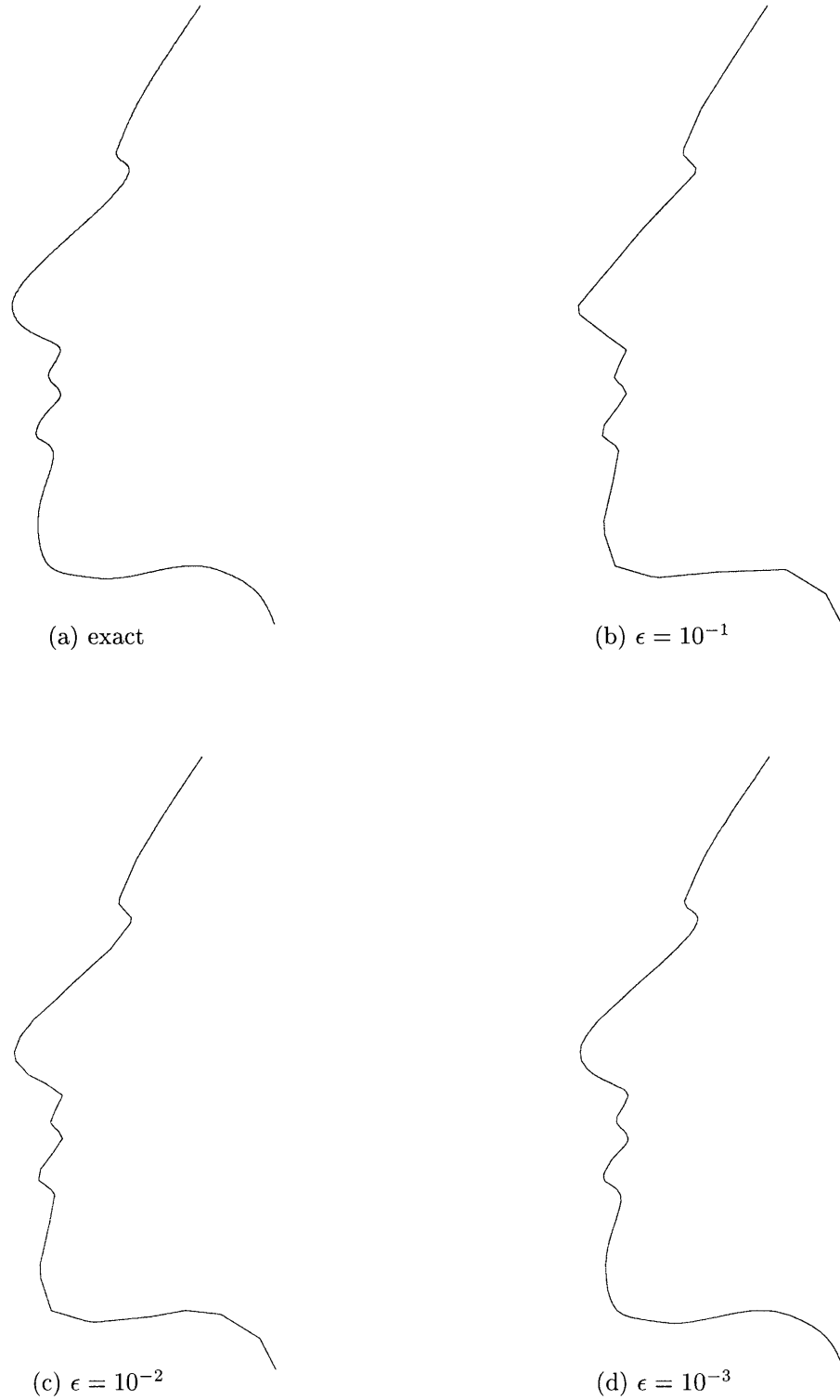


Figure 3-22: Profile of a human face: 12 planar cubic Bézier curves

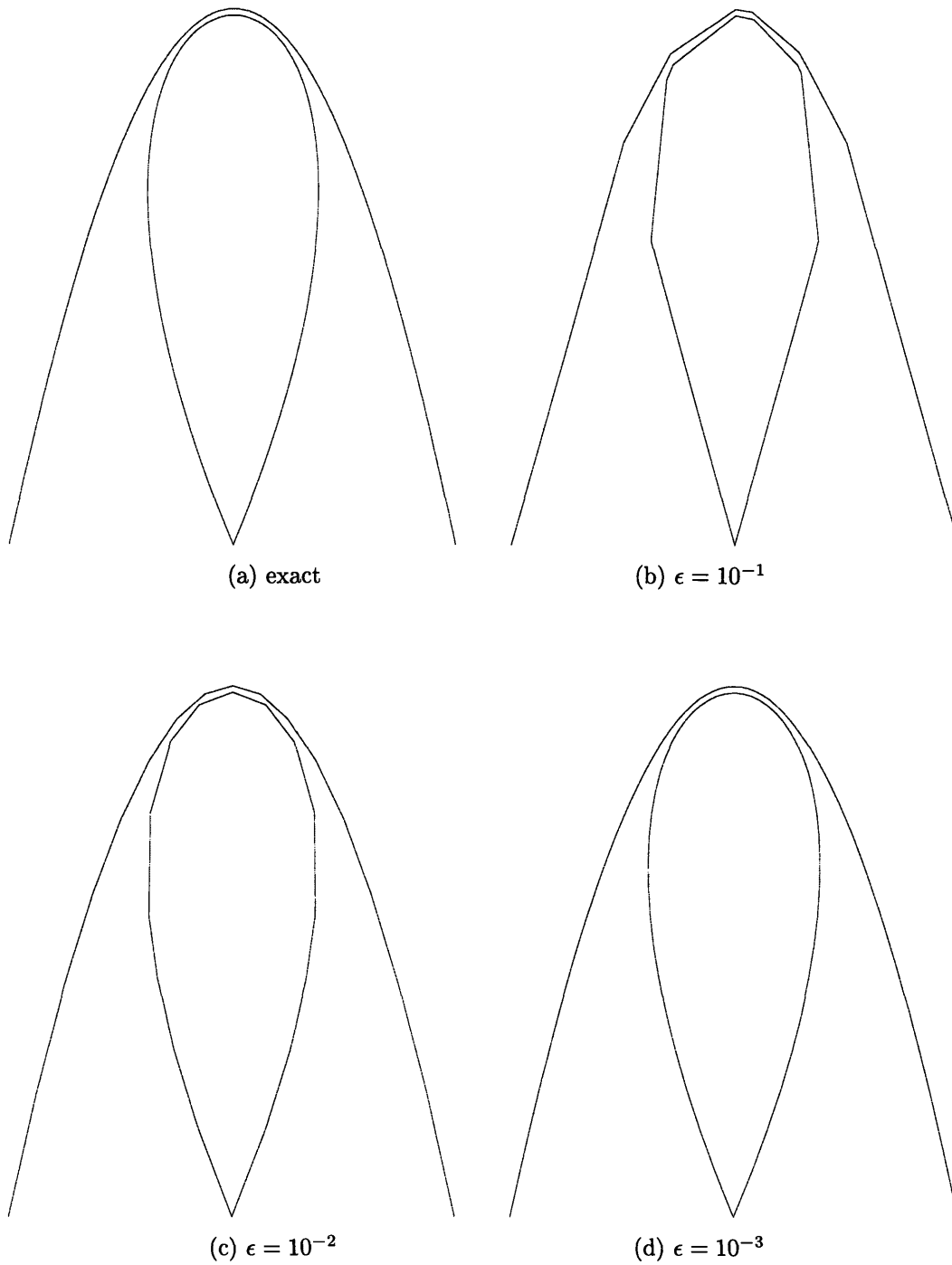


Figure 3-23: Quartic and parabolic planar Bézier curves

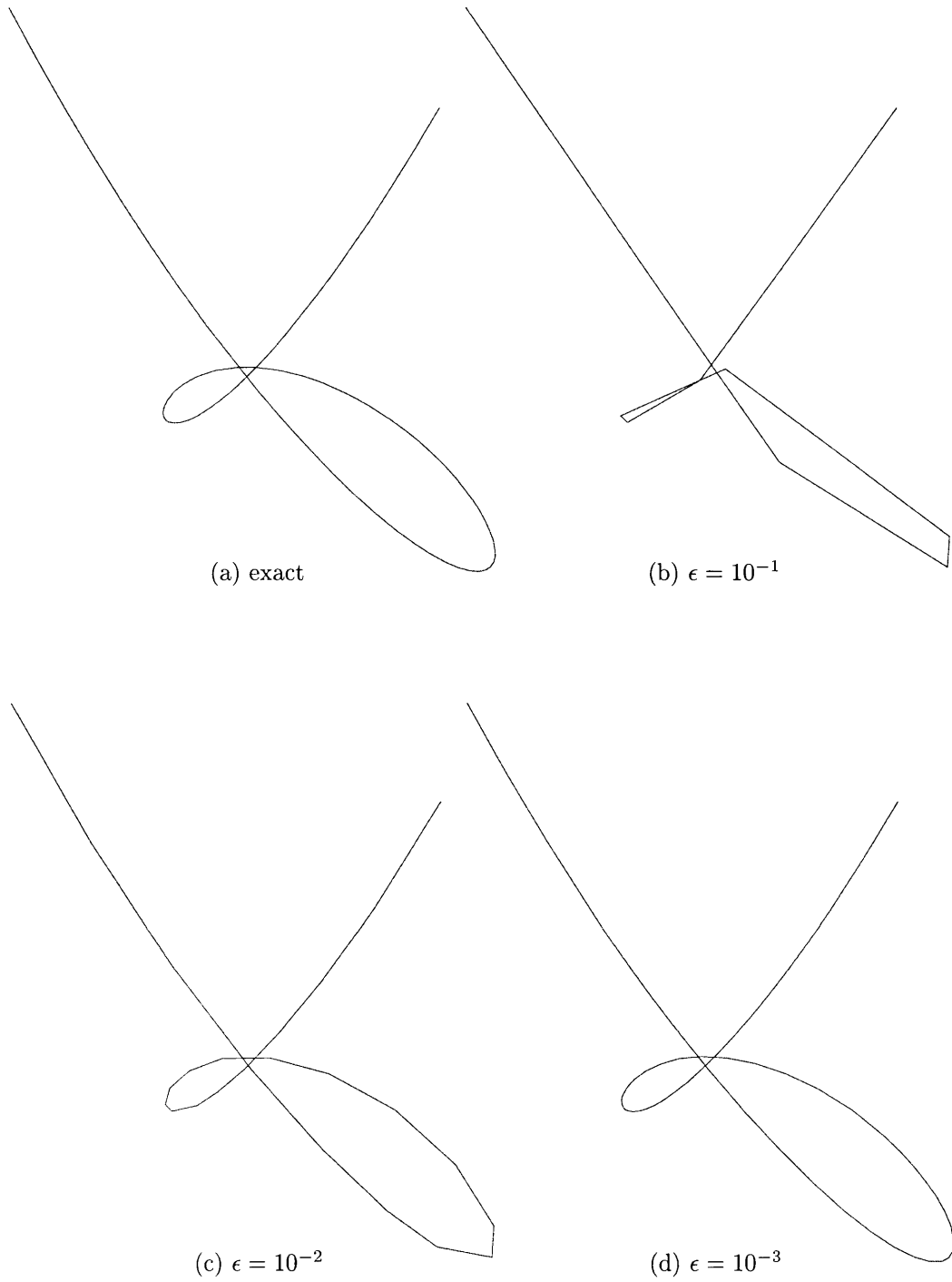
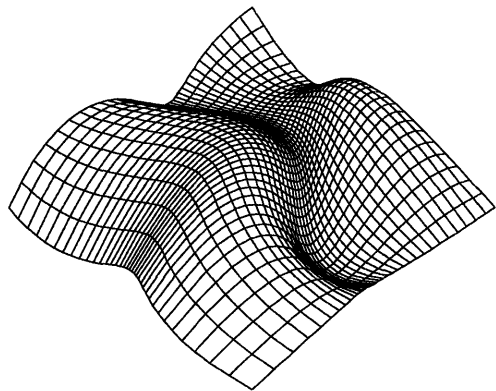


Figure 3-24: A simple space Bézier curve of degree 5



(a) surface net

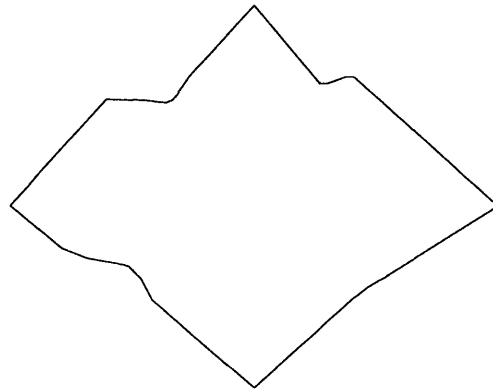
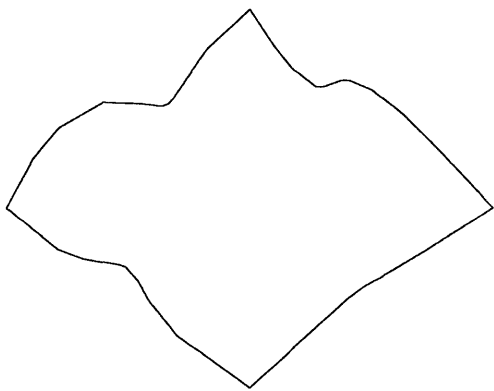
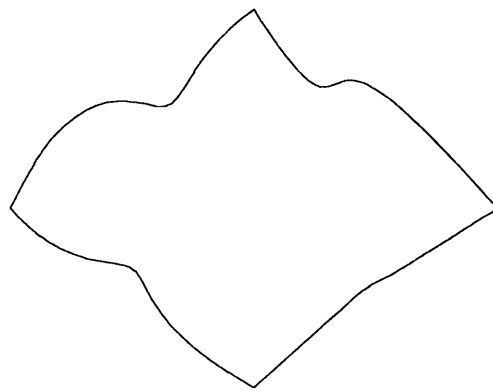
(b) $\epsilon = 10^{-1}$ (c) $\epsilon = 10^{-2}$ (d) $\epsilon = 10^{-3}$

Figure 3-25: Boundary curves of a biquartic surface

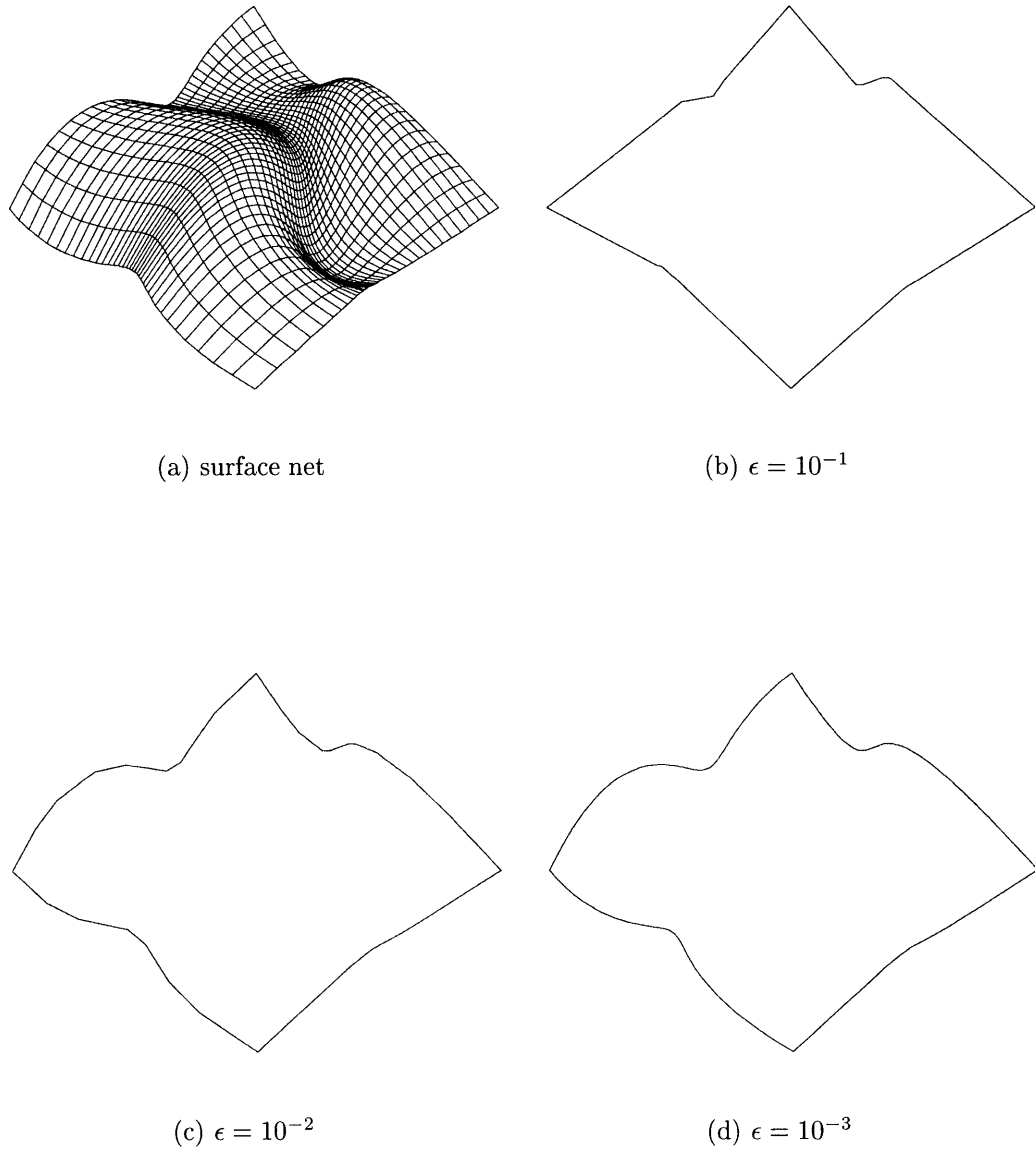


Figure 3-26: Boundary curves of a biquartic surface

Chapter 4

Topologically Reliable

Approximation of Composite

Surfaces

4.1 Introduction

Trimmed parametric surfaces play a fundamental role in boundary representation (B-rep) of solids [14, 29, 65] since they represent faces of complicated 3D curved objects. Tessellation of trimmed parametric surfaces is an essential part of various areas such as graphical display, stereo-lithography applications, finite element (FE) mesh generation and reliable data exchange between various geometric modelers. One of the most appropriate ways to accomplish a piecewise planar approximation of trimmed parametric surface patches is a *triangular* tessellation within a specified geometric tolerance.

The use of *unstructured* triangular mesh provides a very powerful tool for discretizing domains of complex shape. In addition, the unstructured triangular tessellation naturally offers the possibility of *adapting* the mesh to the geometry of rapidly changing intrinsic characteristics by locally modifying the mesh topology. This feature is particularly important for applications such as computational fluid dynamics (CFD),

where not only the geometries involved tend to be complex but also the solution variables are characterized by the appearance of localized features [8, 55, 90].

In this Chapter, we develop a method which approximates a set of trimmed parametric surface patches into a *homeomorphic* [2] set of *unstructured* planar triangular facets within a user specified *geometric tolerance*. Without loss of generality, we use *integral Bézier* patches for the illustration of our methodology. Our surface tessellation algorithm also addresses *numerical robustness* in the triangulation, *differential geometric features* of input geometries and *high quality* meshing in three-dimensional space.

Given a composite parametric surface, we first approximately develop each input trimmed surface patch onto two-dimensional space by employing the *isometric mapping concept*. This preliminary step is crucial to reduce distortion of triangles' shape when mapped into three-dimensional space. If the isometric mapping error is not satisfactory or the developed surface net self-intersects, we bisect the corresponding surface and repeat the surface developing process. Topologically reliable *boundary loop approximation* is next performed and consequently, initial nodes are obtained on the exterior and interior boundary loops of each trimmed surface. By locating those boundary nodes on the approximately developed surface, we obtain a planar *domain of triangulation* for each input surface. We also compute all the *stationary points* of root mean square curvature κ_{rms} of the input surfaces. Nodes are placed on the domain of triangulation corresponding to those stationary points of κ_{rms} . Once all the nodes are placed, boundary conforming Delaunay triangulation is employed to link those boundary and internal nodes to form an *initial* triangular mesh. Furthermore, robust *decision criteria* are introduced to prevent possible failures in the conventional Delaunay triangulation.

After performing the initial triangulation described above, we compute a *tight upper bound* of the deviation between a triangular facet and the corresponding triangular surface patch. In order to achieve the upper bound, we develop an efficient method to extract sub-triangular patches [28] from a tensor product patch. If the deviation is larger than the prescribed tolerance, a new node is placed on the Voronoi

vertex [78] of the approximating triangle and the triangulation is updated. We repeat this *Voronoi vertex point insertion algorithm* until every approximating triangle has an approximation error less than the prescribed tolerance.

We next improve the *aspect ratio* (AR) of the approximating triangles. Unstructured triangular mesh for free-form surfaces frequently suffers from badly shaped triangles with *high* AR [73]. If such low quality elements are used for further geometric computations or general analysis, they cause serious numerical problems [7, 88] such as ill-conditioned matrices or unacceptable convergence. We sort the approximating triangles whose AR's are greater than a given threshold τ . The worst triangle is chosen and a new node is inserted on the Voronoi segment [78] between the worst and the neighboring triangle to generate a new isosceles triangle having AR less than τ . We repeat this *Voronoi segment point insertion algorithm* until AR of the approximating triangles can not be improved any more.

Finally, we map each approximating triangle into three-dimensional space and check possible inappropriate intersections to achieve *topological consistency* between the exact input geometry and its approximation. A bucketing technique is employed again which offers efficiency in both time and space requirements [5, 19]. If such inappropriate intersections exist, we perform further local refinement until a *homeomorphism* between the exact and approximating surfaces is guaranteed. To overcome the lack of *numerical robustness* of floating point arithmetic (FPA) in geometric definitions and computations, our surface tessellation scheme is also applied to *interval trimmed parametric surfaces* evaluated with rounded interval arithmetic (RIA) [66].

4.2 Differential Geometry of a Surface

We summarize the fundamental theory of differential geometry of surfaces employed in our work. The following are well described in the classical literature on differential geometry [22, 40, 53, 89].

4.2.1 Concept of a Surface

A *regular parametric representation* of class C^m ($m \geq 1$) for a surface S in Euclidean space E^3 is a mapping $\mathbf{r} = \mathbf{r}(u, v)$ of an open set U in the uv -plane onto S such that

1. \mathbf{r} is of class C^m in U .
2. If $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ is a basis in E^3 and $\mathbf{r}(u, v) = r_1(u, v)\mathbf{e}_1 + r_2(u, v)\mathbf{e}_2 + r_3(u, v)\mathbf{e}_3$, then $\forall(u, v)$ in U rank of $\mathbf{J}(\mathbf{r})$ is 2, where $\mathbf{J}(\mathbf{r})$ is the Jacobian matrix of \mathbf{r} i.e.,

$$\mathbf{J}(\mathbf{r}) = \begin{pmatrix} \frac{\partial r_1}{\partial u} & \frac{\partial r_1}{\partial v} \\ \frac{\partial r_2}{\partial u} & \frac{\partial r_2}{\partial v} \\ \frac{\partial r_3}{\partial u} & \frac{\partial r_3}{\partial v} \end{pmatrix}. \quad (4.1)$$

Condition 2 states $d\mathbf{u} \equiv (du, dv)$ is mapped onto $d\mathbf{r}$ lying on a *plane* for all possible infinitesimal $d\mathbf{u}$ at a fixed (u, v) and the plane is called *tangent plane*, which will be discussed soon. Furthermore, condition 2 can be compactly described as $\mathbf{r}_u \times \mathbf{r}_v \neq \mathbf{0}$, where subscripts denote partial derivatives. *Unless otherwise stated, a surface represented by $\mathbf{r}(u, v)$ in the following will be a regular parametric surface of class C^m with $m \geq 1$.*

A *coordinate patch* of class C^m ($m \geq 1$) in S is a mapping $\mathbf{r} = \mathbf{r}(u, v)$ of an open set U into S such that

1. \mathbf{r} is a regular parametric representation of S of class C^m ,
2. \mathbf{r} is 1-1 and bi-continuous on U .

Thus a coordinate patch is a regular parametric representation of a *part* of S , which is 1-1 and bi-continuous.

Let S be a set of points in E^3 for which there exists a collection \mathcal{B} of coordinate patches of class C^m ($m \geq 1$) on S satisfying

1. \mathcal{B} covers S i.e., for every point P in S there exists a coordinate patch $\mathbf{r} = \mathbf{r}(u, v)$ in \mathcal{B} containing P .

2. Every coordinate patch $\mathbf{r} = \mathbf{r}(u, v)$ in \mathcal{B} is the intersection of an open set O in E^3 with S .

Then S together with the totality of coordinate patches of class C^m in S is a *simple surface* of class C^m in E^3 . It follows from the definition that a simple surface does *not intersect itself*.

A plane, through a point P on a surface S represented by $\mathbf{r} = \mathbf{r}(u, v)$, parallel to \mathbf{r}_u and \mathbf{r}_v at P is called *tangent plane* to S at P .

A vector \mathbf{N} defined by

$$\mathbf{N} = \frac{\mathbf{r}_u \times \mathbf{r}_v}{|\mathbf{r}_u \times \mathbf{r}_v|} \quad (4.2)$$

is called *unit normal vector* to the surface at a point P . Clearly \mathbf{N} is of unit length, perpendicular to the tangent plane at P , since it is perpendicular to both \mathbf{r}_u and \mathbf{r}_v and varies continuously through the patch since \mathbf{r} is at least of class C^1 and $\mathbf{r}_u \times \mathbf{r}_v \neq \mathbf{0}$.

4.2.2 First and Second Fundamental Forms

As described in Section 3.2, a curve in E^3 is uniquely defined by two local invariant quantities, curvature and torsion. Similarly, a surface in E^3 is uniquely determined by certain local invariant quantities called the *first and second fundamental forms*.

We now consider the quantity

$$\begin{aligned} I &\equiv d\mathbf{r} \cdot d\mathbf{r} = (\mathbf{r}_u du + \mathbf{r}_v dv) \cdot (\mathbf{r}_u du + \mathbf{r}_v dv) \\ &= Edu^2 + 2Fdudv + Gdv^2, \end{aligned} \quad (4.3)$$

where

$$E = \mathbf{r}_u \cdot \mathbf{r}_u, \quad F = \mathbf{r}_u \cdot \mathbf{r}_v, \quad G = \mathbf{r}_v \cdot \mathbf{r}_v. \quad (4.4)$$

I defined in Eq. (4.3) is called the first fundamental form of $\mathbf{r} = \mathbf{r}(u, v)$, which is a homogeneous function of second degree in du and dv with coefficients E , F and G ,

called the first fundamental coefficients. The first fundamental form I is *invariant* under an allowable parametric transformation and plays a basic role in calculating metric properties such as arc length, surface area and angles between two curves on a surface.

We now suppose $\mathbf{r} = \mathbf{r}(u, v)$ is a patch on a surface of class C^m with $m \geq 2$. Then, the unit normal vector \mathbf{N} defined in Eq. (4.2) is a function of u and v of class C^1 with differential $d\mathbf{N} = \mathbf{N}_u du + \mathbf{N}_v dv$. Since $d\mathbf{N}$ is orthogonal to \mathbf{N} , $d\mathbf{N}$ lies on the tangent plane at \mathbf{r} . We consider the quantity

$$\begin{aligned} II &\equiv -d\mathbf{r} \cdot d\mathbf{N} = -(\mathbf{r}_u du + \mathbf{r}_v dv) \cdot (\mathbf{N}_u du + \mathbf{N}_v dv) \\ &= Ldu^2 + 2Mdudv + Ndv^2, \end{aligned} \quad (4.5)$$

where

$$L = \mathbf{r}_{uu} \cdot \mathbf{N}, \quad M = \mathbf{r}_{uv} \cdot \mathbf{N}, \quad N = \mathbf{r}_{vv} \cdot \mathbf{N}. \quad (4.6)$$

II defined in Eq. (4.5) is called the second fundamental form of $\mathbf{r} = \mathbf{r}(u, v)$. Here again II is a homogeneous function of second degree in du and dv with second fundamental coefficients L , M and N . It is easily seen that II is *invariant* in the same sense that I is invariant under an allowable parametric transformation, which preserves the direction of \mathbf{N} ; otherwise II at most changes its sign. Furthermore, the second fundamental form II is a basic tool for analyzing local differential geometric properties of a surface.

Let P a point on a surface S of class C^m with $m \geq 2$, $\mathbf{r} = \mathbf{r}(u, v)$ a patch containing P , and $\mathbf{r} = \mathbf{r}(u(t), v(t))$ a regular curve C of class C^2 through P . The *normal curvature vector* to C at P , denoted by \mathbf{k}_n , is the vector projection of the curvature vector \mathbf{k} – see Eq. (3.6') – of C at P onto the unit normal vector \mathbf{N} at P i.e.,

$$\mathbf{k}_n = (\mathbf{k} \cdot \mathbf{N})\mathbf{N}. \quad (4.7)$$

Note that \mathbf{k}_n is independent of the sense of \mathbf{N} . It is also independent of the sense of C since \mathbf{k} is independent of the sense of C . The component of \mathbf{k}_n in the direction of \mathbf{N} is called *normal curvature* of C at P and is denoted by κ_n i.e.,

$$\kappa_n = \mathbf{k} \cdot \mathbf{N}, \quad (4.8)$$

where the sign of κ_n depends upon the sense of \mathbf{N} ; but it is independent of the sense of C . The normal curvature κ_n can be expressed in terms of I and II ,

$$\kappa_n = \frac{Ldu^2 + 2Mdudv + Ndv^2}{Edu^2 + 2Fdudv + Gdv^2} = \frac{II}{I}. \quad (4.8')$$

Two perpendicular directions for which the values of κ_n take on the maximum and minimum values are called the *principal directions*, and the corresponding normal curvatures κ_{\max} and κ_{\min} are called the *principal curvatures*. A real number κ is a principal curvature at P in the direction $du : dv$ if and only if κ , du and dv satisfy

$$\begin{aligned} (L - \kappa E)du + (M - \kappa F)dv &= 0, \\ (M - \kappa F)du + (N - \kappa G)dv &= 0, \end{aligned} \quad (4.9)$$

where $du^2 + dv^2 \neq 0$. In order for Eq. (4.9) to have a nontrivial solution du , dv

$$\begin{vmatrix} L - \kappa E & M - \kappa F \\ M - \kappa F & N - \kappa G \end{vmatrix} = 0, \quad (4.10)$$

and the corresponding roots are

$$\kappa_{\max} = H + \sqrt{H^2 - K}, \quad (4.11)$$

$$\kappa_{\min} = H - \sqrt{H^2 - K}, \quad (4.12)$$

where H , K are called *mean curvature* and *Gaussian curvature* respectively, defined

by

$$H = \frac{\kappa_{\max} + \kappa_{\min}}{2} = \frac{EN + GL - 2FM}{2(EG - F^2)}, \quad (4.13)$$

$$K = \kappa_{\max}\kappa_{\min} = \frac{LN - M^2}{EG - F^2}. \quad (4.14)$$

Provided that κ_{\max} and κ_{\min} have the same sign at a point P on a surface, K is positive at P and the point P is called *elliptic* point. In case either of κ_{\max} or κ_{\min} vanishes at P , K also vanishes there and the point P is called *parabolic* point. If the signs of κ_{\max} and κ_{\min} are different at P , K is negative at P and the point P is called *hyperbolic* point. If $\kappa_{\max} = \kappa_{\min} = \text{constant} \neq 0$ at an elliptic point P , then the point P is called an *elliptic umbilical* point. In case $\kappa_{\max} = \kappa_{\min} = 0$ at a parabolic point P , then the point is called *parabolic umbilical* or *planar* point, where K and H also vanish.

4.3 Tensor Product and Triangular Bézier Surfaces

Two common sets of basis functions for representing surfaces in computer-aided geometric design are tensor products of univariate Bernstein polynomials and the bivariate Bernstein polynomials in terms of barycentric parametric coordinates. A tensor product Bézier surface is one of the earliest surface representation methods in the CAD community. Linear combinations of the latter basis functions allow representation of triangular Bézier surfaces which provide a simpler topology and are better suited in describing *complex* overall surface geometries. We review fundamental definitions and properties associated with the tensor product and triangular Bézier surfaces.

4.3.1 Tensor Product Polynomial Bézier Surfaces

A tensor product polynomial Bézier surface of degree m, n in u, v parametric directions is obtained by *repeated bilinear* and possibly subsequent *repeated linear interpolation* on the m by n rectangular array of points called vertices of a control polygon

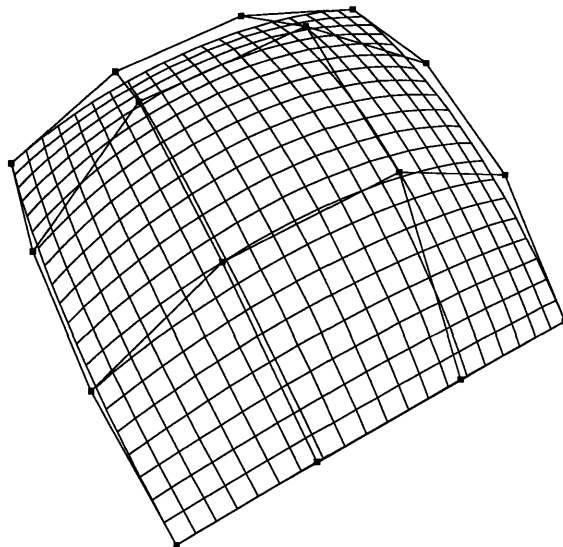


Figure 4-1: A bicubic tensor product Bézier surface together with its control polygon net

net [28]. It can be represented by the following well-known form [28, 80, 44],

$$\mathbf{r}(u, v) = \sum_{i=0}^m \sum_{j=0}^n \mathbf{r}_{i,j} B_{i,m}(u) B_{j,n}(v), \quad 0 \leq u, v \leq 1, \quad (4.15)$$

where $B_{i,m}(u)$, $B_{j,n}(v)$ are the Bernstein basis functions defined in Eq. (3.17) and $\mathbf{r}_{i,j}$ are the vertices of a control polygon net – see also Figure 4-1. The tensor product polynomial Bézier surface (4.15) can be interpreted as sweeping out the u -isoparametric Bézier curve along the v parametric direction and vice versa.

Most properties of Bézier patches follow in a straightforward way from those of Bézier curves – see also Section 3.3.

- *Affine invariance:* Since all those operations of repeated bilinear and linear interpolation are affinely invariant, tensor product polynomial Bézier surface has the property of affine invariance. We can also verify this property by considering

$$\sum_{i=0}^m \sum_{j=0}^n B_{i,m}(u) B_{j,n}(v) \equiv 1, \quad (4.16)$$

in other words, Eq. (4.15) is a barycentric combination of $\mathbf{r}_{i,j}$.

- *Convex hull property:* For $0 \leq u, v \leq 1$, the terms $B_{i,m}(u)B_{j,n}(v) \geq 0$. Then, taking Eq. (4.16) into account, $\mathbf{r}(u, v)$ in Eq. (4.15) is a convex combination of $\mathbf{r}_{i,j}$.
- *Boundary curves:* The boundary curves of the patch $\mathbf{r}(u, v)$ are polynomial Bézier curves. Their Bézier polygons are given by the boundary polygons of the control net.

A *partial derivative* of a tensor product polynomial Bézier surface $\mathbf{r}(u, v)$ with respect to u or v is accomplished by *differencing* the control points and the partial derivative is the tangent vector of each u or v isoparametric curve and furthermore, it is again represented in the form of a tensor product polynomial Bézier surface i.e.,

$$\mathbf{r}_u(u, v) = \sum_{i=0}^{m-1} \sum_{j=0}^n \mathbf{p}_{i,j} B_{i,m-1}(u) B_{j,n}(v), \quad (4.17)$$

$$\mathbf{r}_v(u, v) = \sum_{i=0}^m \sum_{j=0}^{n-1} \mathbf{q}_{i,j} B_{i,m}(u) B_{j,n-1}(v), \quad (4.18)$$

where $\mathbf{p}_{i,j} = m(\mathbf{r}_{i+1,j} - \mathbf{r}_{i,j})$, $\mathbf{q}_{i,j} = n(\mathbf{r}_{i,j+1} - \mathbf{r}_{i,j})$ and m, n are the degrees of $\mathbf{r}(u, v)$ in u, v parametric directions. Higher order partial derivatives can also be described in a similar manner.

4.3.2 Triangular Polynomial Bézier Surfaces

As described in [27, 28] the de Casteljau algorithm for triangular polynomial Bézier surface patches is a direct generalization of the corresponding algorithm for the polynomial Bézier curves mentioned in Section 3.3. The triangular de Casteljau algorithm is completely analogous to the univariate one, the main difference being notation. For a degree n triangular Bézier surface, the control polygon net consists of $\frac{(n+1)(n+2)}{2}$ vertices. The number $\frac{(n+1)(n+2)}{2}$ is referred to as a *triangle number*. A triangular polynomial Bézier surface $\mathbf{r}(u, v, w)$ of degree n is written in terms of bivariate Bernstein

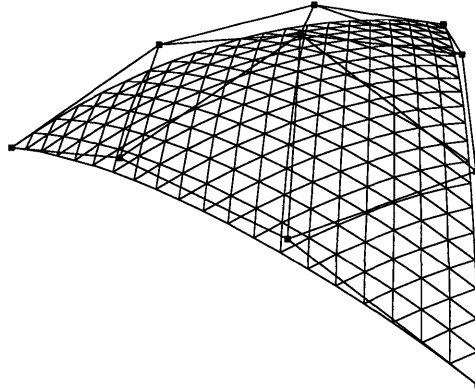


Figure 4-2: A triangular Bézier surface of degree 3 together with its control polygon net

polynomials $B_{i,j,k}^n(u, v, w)$ i.e.,

$$\mathbf{r}(u, v, w) = \sum_{i+j+k=n} \mathbf{r}_{i,j,k} B_{i,j,k}^n(u, v, w), \quad (4.19)$$

where i, j, k are non-negative integers and *barycentric coordinates* (u, v, w) satisfy $u + v + w = 1$ and moreover, $u, v, w \geq 0$. The bivariate Bernstein polynomials $B_{i,j,k}^n(u, v, w)$ in Eq. (4.19) are defined as

$$B_{i,j,k}^n(u, v, w) = \begin{cases} \frac{n!}{i!j!k!} u^i v^j w^k & \text{if } i, j, k \geq 0 \text{ and } i + j + k = n, \\ 0 & \text{else.} \end{cases} \quad (4.20)$$

A triangular Bézier surface of degree 3 together with its control polygon net is shown in Figure 4-2.

Based on the above descriptions, we can state some properties of the triangular Bézier surfaces as follows:

- *Affine invariance*: This follows since linear interpolation is an affine map and the de Casteljau algorithm is just a repeated linear interpolation. We can also

verify this property by considering

$$\sum_{i+j+k=n} B_{i,j,k}^n(u, v, w) \equiv 1, \quad (4.21)$$

in other words, Eq. (4.19) is a barycentric combination of $\mathbf{r}_{i,j,k}$.

- *Convex hull property:* Guaranteed since for $u + v + w = 1$ and $u, v, w \geq 0$, the terms

$$B_{i,j,k}^n(u, v, w) \geq 0. \quad (4.22)$$

Then, taking Eq. (4.21) into account, $\mathbf{r}(u, v, w)$ in Eq. (4.19) is a convex combination of $\mathbf{r}_{i,j,k}$.

- *Boundary curves:* The boundary curves of a triangular polynomial Bézier patch $\mathbf{r}(u, v, w)$ of degree n are polynomial Bézier curves of the same degree n . Their Bézier polygons are given by the boundary polygons of the control net.

4.3.3 Extraction of Triangular Sub-Bézier Surfaces from a Tensor Product Bézier Surface

Similar to the case of Bézier curves discussed in Section 3.5.2, suppose $\mathbf{r}(u, v, w)$, $\mathbf{s}(u, v, w)$ are triangular polynomial Bézier surfaces of degree p and $\mathbf{r}_{i,j,k}$, $\mathbf{s}_{i,j,k}$ ($i, j, k \geq 0$, $i + j + k = p$) the corresponding control points. Using the *convex hull property* in (4.21) and (4.22), the absolute position difference $\delta(u, v, w)$ at isoparametric points is given by,

$$\delta(u, v, w) = \left| \sum_{i+j+k=p} (\mathbf{r}_{i,j,k} - \mathbf{s}_{i,j,k}) B_{i,j,k}^n(u, v, w) \right| \leq \max_{i,j,k} |\mathbf{r}_{i,j,k} - \mathbf{s}_{i,j,k}|. \quad (4.23)$$

Similar upper bounds can be found in [69, 9, 18] for B-spline curves and surfaces and Bézier curves, respectively.

In a surface triangulation algorithm, this convex hull method can be effectively

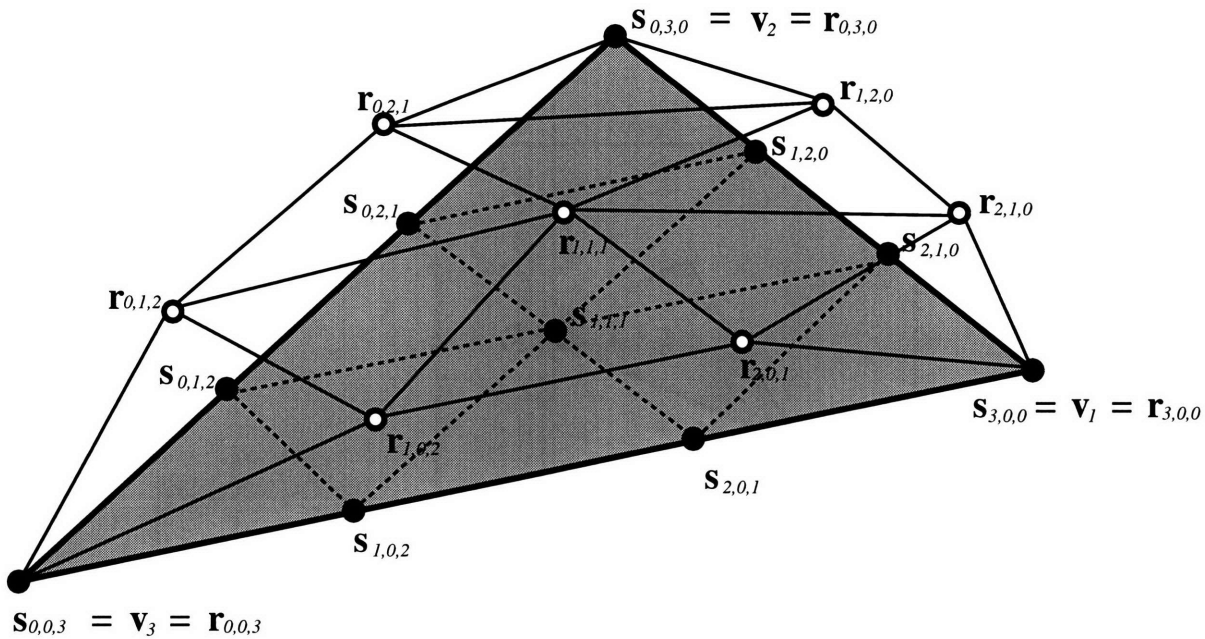


Figure 4-3: Approximating triangular element $s(u, v, w)$ together with its control points $s_{i,j,k}$ and control points $r_{i,j,k}$ of exact triangular Bézier surface patch $r(u, v, w)$ of degree 3

employed to compute a *tight upper bound* of the approximation error, if the control points $r_{i,j,k}$ of an exact triangular surface patch $r(u, v, w)$ and the control points $s_{i,j,k}$ of an approximating planar triangular patch $s(u, v, w)$ are available. In fact, the control points $s_{i,j,k}$ are easily computed by using the three vertices v_1 , v_2 and v_3 of the approximating triangular element. In other words, we set $s_{p,0,0} = v_1$, $s_{0,p,0} = v_2$, $s_{0,0,p} = v_3$ and distribute other $s_{i,j,k}$ on the approximating triangular element $s(u, v, w)$ uniformly for each parametric direction, where p is the degree of the corresponding exact triangular surface patch $r(u, v, w)$. This procedure corresponds to the *degree elevation* of a triangular planar Bézier surface patch of degree 1 up to p [27, 28]. Figure 4-3 shows each $s_{i,j,k}$ together with the corresponding control point $r_{i,j,k}$ of exact triangular Bézier surface patch of degree 3.

We now illustrate how to determine the control points $r_{i,j,k}$ of the triangular Bézier surface patch $r(u, v, w)$ corresponding to the approximating planar triangular patch $s(u, v, w)$. Suppose that the given surface has the form of a *tensor product polynomial Bézier surface*, this problem can be stated as follows – see also Figure 4-4.

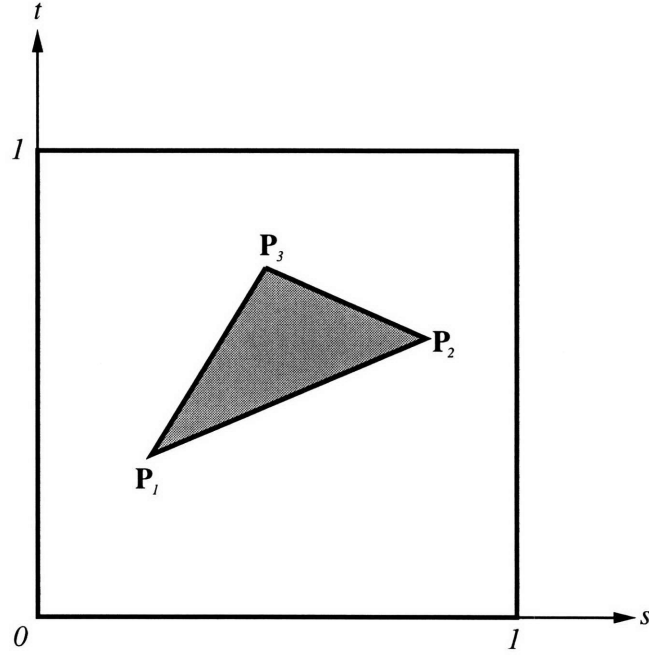


Figure 4-4: A sub-triangle $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ on the unit st -parametric space

Given : a degree m by n tensor product polynomial Bézier surface $\mathbf{R} = \mathbf{R}(s, t)$ and three points $\mathbf{P}_1(s_1, t_1)$, $\mathbf{P}_2(s_2, t_2)$, $\mathbf{P}_3(s_3, t_3)$ on the unit st -parametric space.

Find : control points $\mathbf{r}_{i,j,k}$ of the corresponding triangular sub-Bézier surface patch $\mathbf{r} = \mathbf{r}(u, v, w)$ of degree $p = m + n$.

To solve this problem, we use $N = \frac{(p+1)(p+2)}{2}$ points on $\mathbf{r}(u, v, w)$ *evaluated from the original Bézier surface patch* $\mathbf{R}(s, t)$. Let $\tilde{\mathbf{r}}_{\alpha,\beta,\gamma}$ be N points on $\mathbf{r}(u, v, w)$ with parametric point $(u, v, w) = (\frac{\alpha}{p}, \frac{\beta}{p}, \frac{\gamma}{p})$, where integers $\alpha, \beta, \gamma \geq 0$ and $\alpha + \beta + \gamma = p$. Those N points $\tilde{\mathbf{r}}_{\alpha,\beta,\gamma}$ are easily obtained by computing the corresponding points on the original tensor product Bézier surface $\mathbf{R}(s, t)$ i.e.,

$$\tilde{\mathbf{r}}_{\alpha,\beta,\gamma} = \mathbf{R}(s^*, t^*), \quad (4.24)$$

where each parametric point

$$(s^*, t^*) = \frac{\alpha}{p}(s_2, t_2) + \frac{\beta}{p}(s_3, t_3) + \frac{\gamma}{p}(s_1, t_1), \quad (4.25)$$

by setting three vertices $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ of the triangle on the st -parametric space to be

$$\mathbf{P}_1 = \mathbf{P}_1(s_1, t_1) = \mathbf{P}_1(0, 0, 1), \quad (4.26)$$

$$\mathbf{P}_2 = \mathbf{P}_2(s_2, t_2) = \mathbf{P}_2(1, 0, 0), \quad (4.27)$$

$$\mathbf{P}_3 = \mathbf{P}_3(s_3, t_3) = \mathbf{P}_3(0, 1, 0), \quad (4.28)$$

where the coordinates of $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ in the last column are with respect to the local barycentric coordinates (u, v, w) . Those $\tilde{\mathbf{r}}_{\alpha, \beta, \gamma}$ in Eq. (4.24) are described in terms of the triangular sub-Bézier surface patch $\mathbf{r}(u, v, w)$ in Eq. (4.19) i.e.,

$$\tilde{\mathbf{r}}_{\alpha, \beta, \gamma} = \mathbf{r}\left(\frac{\alpha}{p}, \frac{\beta}{p}, \frac{\gamma}{p}\right) = \sum_{i+j+k=p} \mathbf{r}_{i,j,k} B_{i,j,k}^p\left(\frac{\alpha}{p}, \frac{\beta}{p}, \frac{\gamma}{p}\right). \quad (4.29)$$

After rearranging Eq. (4.29), we obtain

$$\begin{aligned} \tilde{\mathbf{r}}_{\alpha, \beta, \gamma} - \tilde{\mathbf{r}}_{p,0,0} B_{p,0,0}^p\left(\frac{\alpha}{p}, \frac{\beta}{p}, \frac{\gamma}{p}\right) - \tilde{\mathbf{r}}_{0,p,0} B_{0,p,0}^p\left(\frac{\alpha}{p}, \frac{\beta}{p}, \frac{\gamma}{p}\right) - \tilde{\mathbf{r}}_{0,0,p} B_{0,0,p}^p\left(\frac{\alpha}{p}, \frac{\beta}{p}, \frac{\gamma}{p}\right) \\ = \sum^* \mathbf{r}_{i,j,k} B_{i,j,k}^p\left(\frac{\alpha}{p}, \frac{\beta}{p}, \frac{\gamma}{p}\right), \end{aligned} \quad (4.29')$$

where \sum^* denotes the summation over all (i, j, k) except for (i, j, k) being $(p, 0, 0)$, $(0, p, 0)$ and $(0, 0, p)$. We also use $\mathbf{r}_{p,0,0} = \tilde{\mathbf{r}}_{p,0,0}$, $\mathbf{r}_{0,p,0} = \tilde{\mathbf{r}}_{0,p,0}$ and $\mathbf{r}_{0,0,p} = \tilde{\mathbf{r}}_{0,0,p}$ in the derivation of Eq. (4.29'). Each component in the left hand side of Eq. (4.29') is determined by Eq. (4.24) and bivariate Bernstein polynomial Eq. (4.20). Therefore, our problem reduces to *solving a $N - 3$ by $N - 3$ system of linear equations* for each x, y and z component of the unknown control points $\mathbf{r}_{i,j,k}$ of the triangular sub-Bézier surface patch $\mathbf{r}(u, v, w)$ of degree p , where N is the triangle number $\frac{(p+1)(p+2)}{2}$. Notice that the solutions of our system of linear equations *exist* and are *unique* since, for any tensor product Bézier surface $\mathbf{R}(s, t)$, a triangular sub-piece of $\mathbf{R}(s, t)$ can always be described in terms of a triangular Bézier patch $\mathbf{r}(u, v, w)$ and furthermore, $\mathbf{r}(u, v, w)$ is *unique* if the degree of $\mathbf{r}(u, v, w)$ is fixed. To solve the system of linear equations, we employ the *LU*-decomposition method [75], which is especially appropriate for our problem because the *same coefficient matrix* is used for the solution of each x, y and

z component of the unknowns $\mathbf{r}_{i,j,k}$.

As a matter of fact, there is an alternative way to determine the control points $\mathbf{r}_{i,j,k}$. The alternative method can be achieved by using the *coordinates-free formula for polar forms* \mathbf{f} described by [76, 83],

$$\mathbf{f}(\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_p) = \frac{1}{p!} \sum_{|\mu_k|=i} (-1)^{p-i} i^p \mathbf{r}\left(\frac{1}{i}(\mu_1 \mathbf{u}_1 + \mu_2 \mathbf{u}_2 + \dots + \mu_p \mathbf{u}_p)\right), \quad (4.30)$$

where μ_k is 0 or 1 and the summation \sum is taken over a set of p indices $\mu_1, \mu_2, \dots, \mu_p$ and $|\mu_k| \equiv \mu_1 + \mu_2 + \dots + \mu_p$ and \mathbf{u}_j is the j^{th} parametric point (u_j, v_j, w_j) on the domain triangle. Using Eq. (4.30), the control points $\mathbf{r}_{i,j,k}$ in Eq. (4.29) are obtained by

$$\mathbf{r}_{i,j,k} = \mathbf{f}\left(\underbrace{\mathbf{P}_1, \mathbf{P}_1, \dots, \mathbf{P}_1}_i, \underbrace{\mathbf{P}_2, \mathbf{P}_2, \dots, \mathbf{P}_2}_j, \underbrace{\mathbf{P}_3, \mathbf{P}_3, \dots, \mathbf{P}_3}_k\right), \quad (4.31)$$

where $\mathbf{P}_1, \mathbf{P}_2$ and \mathbf{P}_3 are the vertices of a sub-triangle on the parametric space in terms of barycentric coordinates – see also Figure 4-4. The polar forms defined in Eq. (4.30) are a classical and useful mathematical tool for study of general polynomial based curves and surfaces; however, the polar forms are seriously inefficient for our specific problem because of cycling through 2^p possible summation index combinations.

The following analysis shows the comparison of time complexity C between the two alternative methods.

Our proposed method

$$C \approx N(N-3)C_1(p) + NC_2(m, n) + C_3(N), \quad (4.32)$$

where

- $N = \frac{(p+1)(p+2)}{2}$, $p = m + n$ and m, n are degrees of the original tensor product Bézier surface $\mathbf{R}(s, t)$ in u, v directions,
- $C_1(p) \propto \frac{p(p+1)}{2}$, which is the time cost for one evaluation of a bivariate Bernstein polynomial,

- $C_2(m, n) \propto \min\left(\frac{n(n+1)(m+1)}{2} + \frac{m(m+1)}{2}, \frac{m(m+1)(n+1)}{2} + \frac{n(n+1)}{2}\right)$, which is the time cost for one evaluation of a given Bézier surface $\mathbf{R}(s, t)$ of degree m, n ,
- $C_3(N) \propto (N - 3)^3$, time cost for solving $N - 3$ by $N - 3$ system of linear equations.

Polar form based method

$$C \approx N2^p C_2(m, n) \quad (4.33)$$

where N, p and $C_2(m, n)$ are defined above. In Eq. (4.32), the first and second terms are comparable and the third term is relatively smaller. Eq. (4.33) shows that the time cost of the polar form based method is approximately 2^p times higher than the cost of our proposed method, because Eq. (4.33) is equal to the second term of Eq. (4.32) multiplied by 2^p . Therefore, as the degrees of the given surface become higher, the time cost of the polar form based method increases unacceptably due to the exponential dependency.

We tested both methods and the results are summarized in Table 4.1. First, extraction of a triangular sub-Bézier patch is performed from a bi-cubic Bézier surface. In order to see the dependency of the time complexity upon the degrees m, n of an input tensor product Bézier surface, we elevate degrees of the bi-cubic ($m = n = 3, p = m + n = 6$) Bézier surface by 1, 2 and 3 in both s, t directions. The results clearly show 2^p times higher time complexity of the polar form based method compared with our proposed method. We perform the test operating in both floating point (FPA) and rounded interval arithmetic (RIA) and the corresponding results are also shown in Table 4.1.

| Degree (m, n) | CPU-time (<i>sec</i>) | | | | cost ratio | | 2^p |
|-----------------|-------------------------|------|----------------|-------|------------|------|-------|
| | (a) our method | | (b) polar form | | (b)/(a) | | |
| | FPA | RIA | FPA | RIA | FPA | RIA | |
| (3, 3) | 0.01 | 0.35 | 0.82 | 25.57 | 82 | 73 | 64 |
| (4, 4) | 0.04 | 1.19 | 10.96 | 350.0 | 274 | 294 | 256 |
| (5, 5) | 0.09 | 3.06 | 115.0 | 3757 | 1277 | 1228 | 1024 |
| (6, 6) | 0.22 | 7.47 | 1040 | 34052 | 4728 | 4559 | 4096 |

Table 4.1: Complexity comparison between our proposed method and polar form based method

4.4 Stationary Points of a Root Mean Square Curvature

We have discussed four different measures of surface curvature, the maximum and minimum principal curvatures κ_{\max} , κ_{\min} , Gaussian curvature K and mean curvature H in Section 4.2.2. It is worthwhile to include identification of highly curved regions as a preliminary step in the surface approximation procedure, since it provides geometrically meaningful information and it also increases the efficiency of the following triangulation procedures in terms of the number of approximating triangles. For this purpose, we define an appropriate intrinsic property of a surface and illustrate how to compute its stationary points as we did in the curve approximation algorithm. Those stationary points provide important differential geometric information of the input surfaces and play a role as the initial seed points of our triangulation algorithm.

4.4.1 Root Mean Square Curvature κ_{rms} of a Surface

The absolute curvature κ_{abs} [28] at a point of a regular surface of class C^m ($m \geq 2$) is defined as

$$\kappa_{abs} \equiv |\kappa_{\max}| + |\kappa_{\min}|, \quad (4.34)$$

where κ_{\max} , κ_{\min} are the maximum and minimum principal curvatures defined in Eq. (4.12). The absolute curvature may be one of the most appropriate curvature measures for the identification of highly curved regions of a surface. However by definition, it is *not differentiable*, which makes computation of its stationary points difficult in practice.

As an alternative curvature measure we use the *root mean square curvature* κ_{rms} , first proposed in [57]:

$$\kappa_{rms} \equiv \sqrt{\kappa_{\max}^2 + \kappa_{\min}^2} = \sqrt{4H^2 - 2K}, \quad (4.35)$$

where H and K are the mean and Gaussian curvatures. We note the root mean square curvature κ_{rms} is an *intrinsic* property since κ_{\max} and κ_{\min} at most change their signs under an allowable parametric transformation, as stated in Section 4.2.2.

4.4.2 Solution Methodology

For a regular parametric surface $\mathbf{r}(u, v)$ of class C^m ($m \geq 2$), the governing equations for computing the stationary points of κ_{rms} are formulated as [57]

$$\frac{\partial \kappa_{rms}}{\partial u} = \frac{(BB_u - A_u S^2)S^2 + (4AS^2 - 3B^2)\mathbf{s} \cdot \mathbf{s}_u}{S^5 \sqrt{B^2 - 2AS^2}} = 0, \quad (4.36)$$

$$\frac{\partial \kappa_{rms}}{\partial v} = \frac{(BB_v - A_v S^2)S^2 + (4AS^2 - 3B^2)\mathbf{s} \cdot \mathbf{s}_v}{S^5 \sqrt{B^2 - 2AS^2}} = 0, \quad (4.37)$$

where

$$\mathbf{s} = \mathbf{r}_u \times \mathbf{r}_v,$$

$$S = |\mathbf{s}| = \sqrt{EG - F^2},$$

$$A = \tilde{L}\tilde{N} - \tilde{M}^2,$$

$$B = 2F\tilde{M} - E\tilde{N} - G\tilde{L},$$

$$\tilde{L} = \mathbf{s} \cdot \mathbf{r}_{uu}, \quad \tilde{M} = \mathbf{s} \cdot \mathbf{r}_{uv}, \quad \tilde{N} = \mathbf{s} \cdot \mathbf{r}_{vv},$$

and the first fundamental coefficients E, F, G are defined in Eq. (4.4) and subscripts denote the corresponding partial derivatives. Notice that for a regular surface $\mathbf{r}(u, v)$, S does not vanish $\forall u, v$ and furthermore, $\sqrt{B^2 - 2AS^2}$ vanishes if and only if $\kappa_{\max} = \kappa_{\min} = 0$ i.e., only at *planar* points [60]. Therefore, for a regular parametric surface $\mathbf{r}(u, v)$ defined in a square domain \mathcal{U} such that $0 < u, v < 1$, we only need to solve

$$f(u, v) \equiv (BB_u - A_u S^2)S^2 + (4AS^2 - 3B^2)\mathbf{s} \cdot \mathbf{s}_u = 0, \quad (u, v) \in \mathcal{U} - \mathcal{U}^*, \quad (4.38)$$

$$g(u, v) \equiv (BB_v - A_v S^2)S^2 + (4AS^2 - 3B^2)\mathbf{s} \cdot \mathbf{s}_v = 0, \quad (u, v) \in \mathcal{U} - \mathcal{U}^*, \quad (4.39)$$

where \mathcal{U}^* denotes the set of parametric points corresponding to the planar points. If a polynomial surface $\mathbf{r}(u, v)$ is of degree m, n in u, v parametric directions, Eq.'s (4.38), (4.39) are of degrees $(14m - 9, 14n - 8), (14m - 8, 14n - 9)$ respectively in the u, v directions. In order to solve the 2×2 system of nonlinear polynomial equations of high degree, we first formulate $f(u, v), g(u, v)$ in terms of tensor products of univariate Bernstein polynomials and next utilize the projected polyhedron algorithm described in Section 3.4.3.

In case that either Eq. (4.38) or Eq. (4.39) vanishes identically i.e., $f(u, v) = 0$ or $g(u, v) = 0 \forall (u, v) \in \mathcal{U} - \mathcal{U}^*$, the roots of the system of two polynomial equations usually form *implicit curves* in the uv -parametric domain, which forces the solution algorithm to be trapped in an exhaustive root solving process. In order to prevent this, we check if $f(u, v)$ or $g(u, v)$ vanishes identically. Since $f(u, v)$ and $g(u, v)$ are expressed in terms of tensor product of univariate Bernstein polynomials i.e.,

$$f(u, v) = \sum_{i=0}^{14m-9} \sum_{j=0}^{14n-8} f_{ij} B_{i,14m-9}(u, v) B_{j,14n-8}(u, v), \quad (4.40)$$

$$g(u, v) = \sum_{i=0}^{14m-8} \sum_{j=0}^{14n-9} g_{ij} B_{i,14m-8}(u, v) B_{j,14n-9}(u, v), \quad (4.41)$$

we need to check if

$$\text{(FPA)} \quad |f_{ij}| < \epsilon \ll 1 \quad \text{or} \quad |g_{ij}| < \epsilon \ll 1, \quad (4.42)$$

$$\text{(RIA)} \quad f_{ij}^l \leq 0, \quad f_{ij}^u \geq 0 \quad \text{or} \quad g_{ij}^l \leq 0, \quad g_{ij}^u \geq 0, \quad (4.43)$$

$$\text{where} \quad f_{ij} \equiv [f_{ij}^l, f_{ij}^u] \quad \text{and} \quad g_{ij} \equiv [g_{ij}^l, g_{ij}^u],$$

for all possible i, j . Based on the above decision criteria, we do not attempt to solve the system of equations if $f(u, v)$ or $g(u, v)$ vanishes identically.

4.5 Approximate Locally Isometric Mapping

Most existing free-form surface meshing algorithms perform triangulation in the two-dimensional parametric space, typically the unit square, and map each triangular element into the three-dimensional space through the surface equation. It is well-known that such a mapping will inevitably produce *distortion* or *stretching* in general. To overcome this adverse effect, we introduce an auxiliary planar *domain of triangulation* by using an *approximate locally isometric mapping*.

4.5.1 Isometric Mapping

An isometric mapping is defined as [89]:

Definition 4.5.1 *A 1-1 mapping \mathbf{f} of a surface S onto a surface S^* is called an isometric mapping or isometry if the length of an arbitrary regular arc $\mathbf{r}(t)$ on S is identical to the length of its image $\mathbf{r}^* = \mathbf{r}^*(t) = \mathbf{f}(\mathbf{r}(t))$ on S^* .*

As also stated in [89]:

Theorem 4.5.2 *A surface is developable if there exists an isometric mapping \mathbf{f} of a surface S onto a plane D .*

It should be pointed out that the conditions for the isometric mapping are too strong and hence, only very special surfaces are developable. We then employ a *locally isometric mapping* defined as [22]:

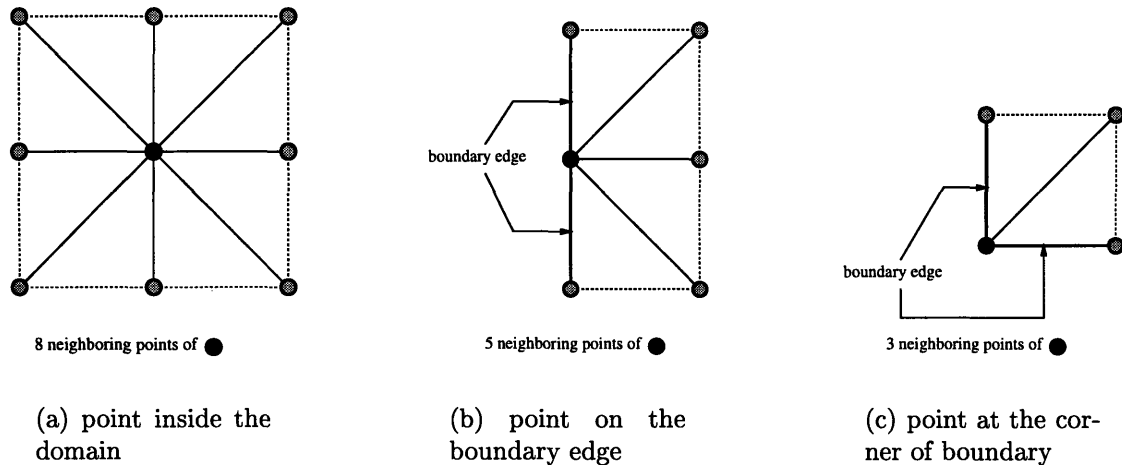


Figure 4-5: Neighboring points

Definition 4.5.3 A map $f : V \mapsto S^*$ of a neighborhood V of a point p on a surface S is a **local isometry** at p if there exists a neighborhood V^* of $f(p) \in S^*$ such that $f : V \mapsto V^*$ is an isometry. If there exists a local isometry into S^* at every $p \in S$, the surface S is said to be **locally isometric** to S^* .

Based on the above **Definitions 4.5.1, 4.5.3** and **Theorem 4.5.2**, a **locally developable** surface is defined as [56]:

Definition 4.5.4 A surface is **locally developable** if there exists a local isometry f of a neighborhood V of each point p on a surface S onto a plane D .

However in general cases, it is very difficult to find the locally isometric mapping either. Thus we define a global criterion to measure the *mapping error* in the following Section.

4.5.2 Mapping Error Function

A surface net S can be *approximately* developed onto a plane D by finding an *approximate* locally isometric mapping $f : S \mapsto D$ while minimizing the mapping error

function E defined as:

$$\begin{aligned} E &= E(v_{1x}, v_{1y}, v_{2x}, v_{2y}, \dots, v_{ix}, v_{iy}, \dots, v_{pqx}, v_{pqy}) \\ &= \sum_{k=1}^l [d_k^{3D} - d_k^{2D}(v_{1x}, v_{1y}, v_{2x}, v_{2y}, \dots, v_{ix}, v_{iy}, \dots, v_{pqx}, v_{pqy})]^2, \end{aligned} \quad (4.44)$$

where

- p, q : number of distributed points on the surface net S in the u, v parametric directions,
- v_{ix}, v_{iy} : **unknowns**, x and y coordinates of points on the approximately developed surface net D corresponding to p by q points on S
- d_k^{3D} : length of a straight line segment connecting each neighboring point on S ,
- d_k^{2D} : length of a straight line segment connecting each neighboring point on D .
- $l = 4pq - 3(p + q) + 2$, *

where the *neighboring points* are pictorially classified in Figure 4-5. Notice that E can not be minimized to zero unless S is developable.

As an example, we consider a bi-cubic Bézier surface and the corresponding surface net S , setting $p = q = 4$ as shown in Figure 4-6.

4.5.3 Minimization of Mapping Error Function

By minimizing the mapping error function E in Eq. (4.44), we determine the unknowns (v_{ix}, v_{iy}) and the corresponding approximately developed surface net D , as shown in Figure 4-7-(d). We use an existing minimization algorithm based on the *modified Powell's quadratically convergent method*, Chapter 10 of [75].

The minimization algorithm requires *initial starting points*. We now illustrate how to determine those initial starting points (v_{ix}^0, v_{iy}^0) for the minimization algorithm,

*For $p \times q$ array of points, there are $(p - 1)q + (q - 1)p + 2(p - 1)(q - 1) = 4pq - 3(p + q) + 2$ number of neighboring line segments.

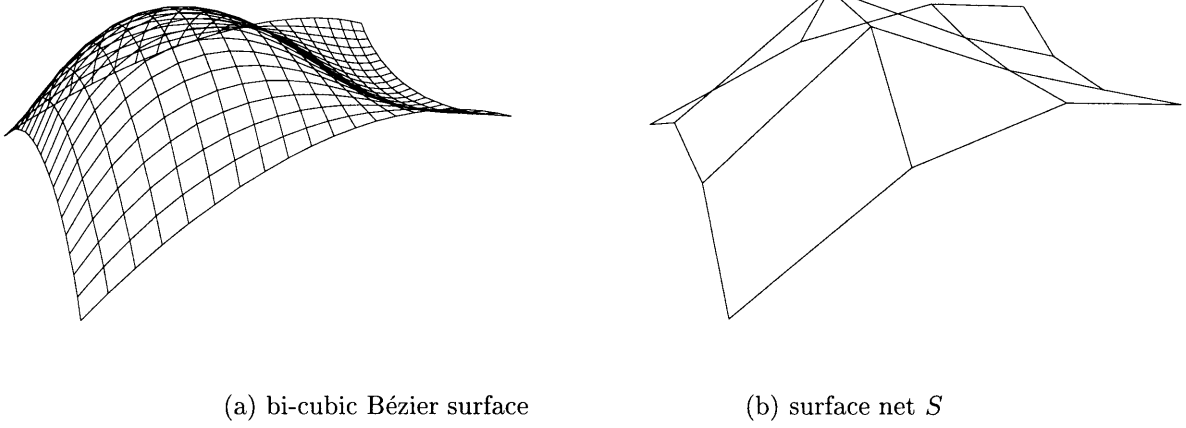


Figure 4-6: Given bi-cubic surface and the corresponding surface net

where $i = 1, 2, \dots, pq$ and p, q are defined in Eq. (4.44). We determine the initial starting points by using the length of surface net (S) edges and the angle between the neighboring edges of S , as detailed in **step a – step i**.

step a. Set $(v'_{1x}, v'_{1y}) = (0, 0)$.

step b. Set $(v'_{ix}, v'_{iy}) = (v'_{i-qx} + |\mathbf{r}_i - \mathbf{r}_{i-q}|, 0)$ for $i = q\alpha + 1$ where $\alpha = 1, 2, \dots, p-1$ and \mathbf{r}_i is a point on S corresponding to (v'_{ix}, v'_{iy}) .

step c. Set $(v'_{ix}, v'_{iy}) = (d \cos \gamma + v'_{i-1x}, d \sin \gamma + v'_{i-1y})$ for $i = \beta + 1$ where $\beta = 1, 2, \dots, q-1$, $d = |\mathbf{r}_i - \mathbf{r}_{i-1}|$ and γ is the angle between $\mathbf{r}_{i+q-1} - \mathbf{r}_{i-1}$ and $\mathbf{r}_i - \mathbf{r}_{i-1}$ with $0 \leq \gamma \leq \pi$.

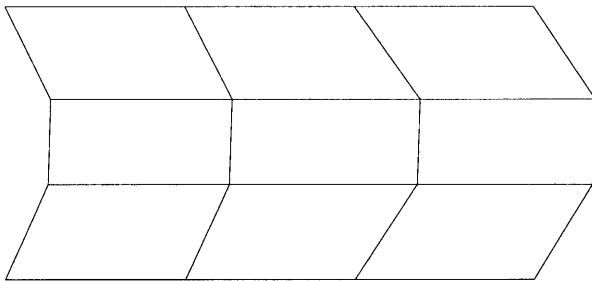
step d. Set $(v'_{ix}, v'_{iy}) = (|\mathbf{r}_i - \mathbf{r}_{i-q}| + v'_{i-qx}, v'_{\beta+1y})$ for $i = q\alpha + \beta + 1$ where $\alpha = 1, 2, \dots, p-1$ and $\beta = 1, 2, \dots, q-1$.

For a surface net S shown in Figure 4-6-(b), **step a – step d** results in Figure 4-7-(a).

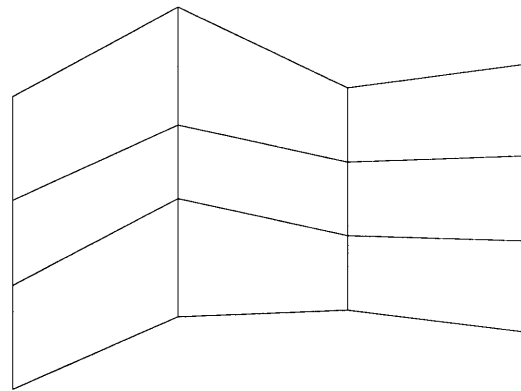
step e. Set $(v''_{1x}, v''_{1y}) = (0, 0)$.

step f. Set $(v''_{ix}, v''_{iy}) = (0, v''_{i-1y} + |\mathbf{r}_i - \mathbf{r}_{i-1}|)$ for $i = \beta + 1$ where $\beta = 1, 2, \dots, q-1$.

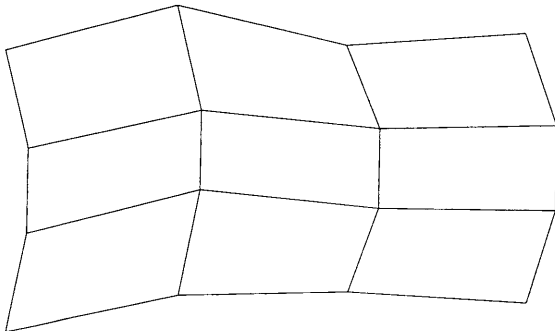
step g. Set $(v''_{ix}, v''_{iy}) = (d \sin \gamma + v''_{i-qx}, d \cos \gamma + v''_{i-qy})$ for $i = q\alpha + 1$ where $\alpha = 1, 2, \dots, p-1$, $d = |\mathbf{r}_i - \mathbf{r}_{i-q}|$ and γ is the angle between $\mathbf{r}_i - \mathbf{r}_{i-q}$ and $\mathbf{r}_{i-q+1} - \mathbf{r}_{i-q}$



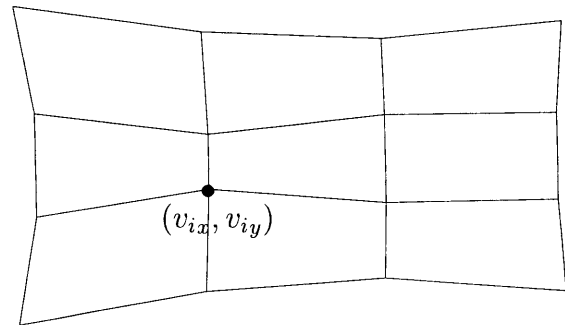
(a) step a. – step d.



(b) step e. – step h.



(c) **step i:** initial starting points by averaging (a), (b)



(d) developed surface net D

Figure 4-7: Procedure to get approximately developed surface net D

with $0 \leq \gamma \leq \pi$.

step h. Set $(v''_{i_x}, v''_{i_y}) = (v''_{q\alpha+1_x}, |\mathbf{r}_i - \mathbf{r}_{i-1}| + v''_{i-1_y})$ for $i = q\alpha + \beta + 1$ where $\alpha = 1, 2, \dots, p-1$ and $\beta = 1, 2, \dots, q-1$.

Figure 4-7-(b) shows the result of **step e – step h**.

step i. Determine those *initial starting points* $(v^0_{i_x}, v^0_{i_y})$ by averaging (v'_{i_x}, v'_{i_y}) and (v''_{i_x}, v''_{i_y}) i.e., $(v^0_{i_x}, v^0_{i_y}) = (\frac{v'_{i_x} + v''_{i_x}}{2}, \frac{v'_{i_y} + v''_{i_y}}{2})$ for $i = q\alpha + \beta + 1$ where $\alpha = 0, \dots, p-1$ and $\beta = 0, \dots, q-1$.

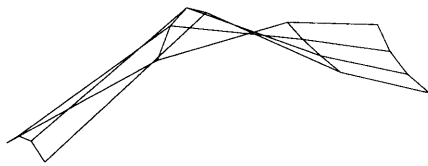
Figure 4-7-(c) together with Figure 4-7-(d) shows the initial starting points obtained by **step i** and the corresponding result of minimization, respectively.

In case that minimized mapping error $\min(E)$ is unsatisfactory or the developed surface net D self-intersects, we bisect the input surface and repeat the developing procedure until $\min(E)$ is within a certain threshold τ_{iso} and D does not intersect itself. The bisections are performed along the isoparametric lines $u = \frac{1}{2}$ and $v = \frac{1}{2}$ in turns. We define τ_{iso} as follows; we assume $|d_k^{3D} - d_k^{2D}|$ should be less than a certain tolerance ϵ , then $E \equiv \sum_{k=1}^l (d_k^{3D} - d_k^{2D})^2$ in Eq. (4.44) should be less than $\epsilon^2 l$. Therefore,

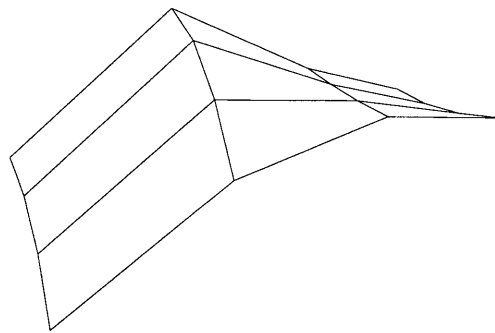
$$\tau_{iso} \equiv \epsilon^2 l, \quad (4.45)$$

where l is defined in Eq. (4.44). For a surface net S shown in Figure 4-6-(b), $\tau_{iso} \approx 0.12$ and the corresponding D shown in Figure 4-7-(d) has $\min(E) \approx 0.61$, if we set $\epsilon = 0.01$. Therefore $\min(E) > \tau_{iso}$ and further subdivision is required. We subdivide the input surface along an iso-parametric line $v = 0.5$ and the corresponding results are shown in Figure 4-8. After one subdivision, $\min(E) \approx 0.01 < \tau_{iso} \approx 0.12$ for each sub-surface net and hence, no further subdivisions are required.

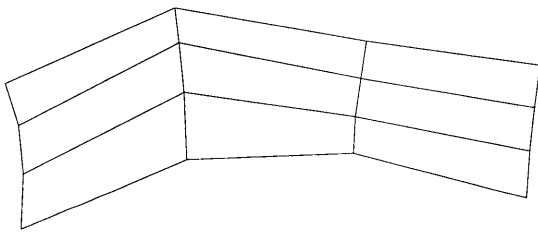
We now consider the latter case of subdivision being required i.e., the case of self-intersecting developed surface net D , as shown in Figure 4-9-(a,b). In such a case, there exist at least one pair of intersecting net edges and the 1-1 mapping between uv -parametric space and the developed surface net D is not achieved. Therefore, after



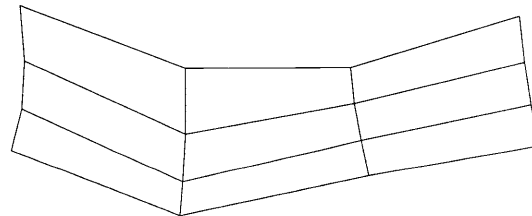
(a) sub-surface net 1



(b) sub-surface net 2

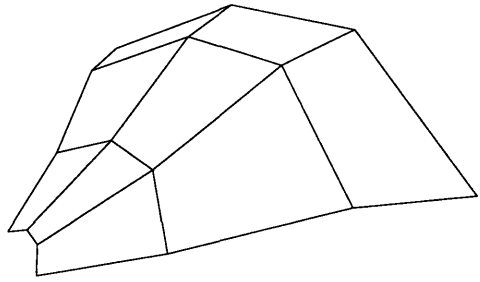


(c) developed surface net 1

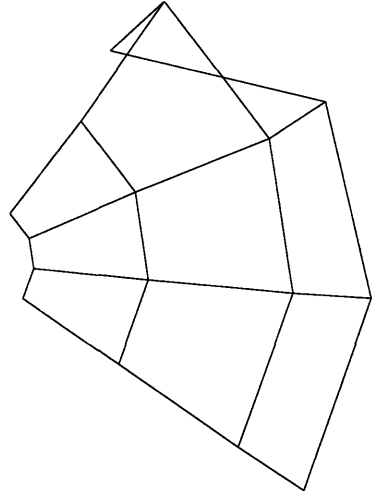


(d) developed surface net 2

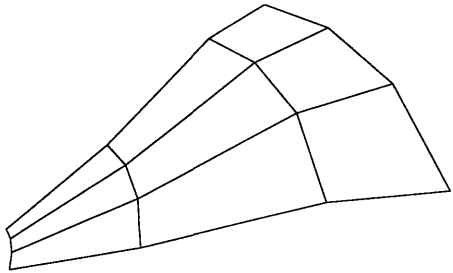
Figure 4-8: Bisected surface nets and their developed surface nets



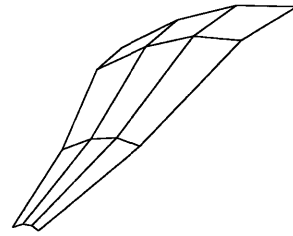
(a) original surface net



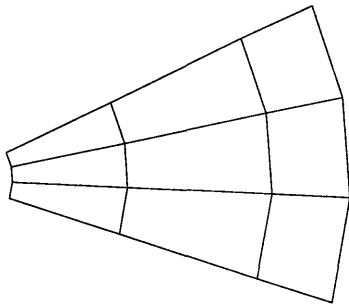
(b) developed surface net



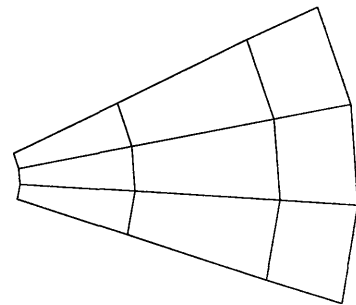
(c) sub-surface net 1



(d) sub-surface net 2



(e) developed surface net 1



(f) developed surface net 2

Figure 4-9: Self-intersecting developed surface net

approximately developing the original surface net S we should perform line to line intersection test for each pair of net edges of D and if such intersection is detected, we bisect input surface and follow the same procedure described in the former case until each developed sub-surface net does not intersect itself as illustrated in Figure 4-9-(c,d,e,f).

We finally illustrate how to define each mapping among the developed surface net, parametric space and the corresponding input parametric surface. Since a mapping \mathbf{r} from the unit uv -parametric space onto three-dimensional surface is defined by the surface equation $\mathbf{r}(u, v)$, we only need to determine the mapping \mathbf{f} from the parametric space onto the developed surface net and its inverse \mathbf{f}^{-1} as shown in Figure 4-10. The mappings \mathbf{f} and \mathbf{f}^{-1} are no other than the mappings between each sub-quadrilateral $\mathcal{U}_{i,j}$ on the unit uv -parametric space defined by $u_i \leq u \leq u_{i+1}$, $v_j \leq v \leq v_{j+1}$ and the corresponding sub-quadrilateral $\mathcal{V}_{i,j}$ on the developed surface net as illustrated in Figure 4-11 i.e.,

$$\mathbf{f} = \mathbf{f}(u, v) \equiv \sum_{p=0}^1 \sum_{q=0}^1 \mathbf{V}_{i+p,j+q} B_{p,1}(u^*) B_{q,1}(v^*), \quad (u, v) \in \mathcal{U}_{i,j} \quad (4.46)$$

$$\text{where} \quad u^* = m(u - u_i), \quad v^* = n(v - v_j), \quad (4.47)$$

m, n are the number of net segments in u, v parametric directions, $B_{p,1}(u^*), B_{q,1}(v^*)$ are the Bernstein basis functions of degree 1 defined in Eq. (3.17) and $\mathbf{V}_{i+p,j+q}$ are the vertices of $\mathcal{V}_{i,j}$ corresponding to the vertices (u_{i+p}, v_{j+q}) of $\mathcal{U}_{i,j}$. For any $\mathbf{V} = (V_x, V_y)$, $\mathbf{V} \in \mathcal{V}_{i,j}$, the mapping $\mathbf{f}^{-1} = \mathbf{f}^{-1}(\mathbf{V}) = \mathbf{f}^{-1}(V_x, V_y)$ can be achieved by solving the following system of equations for $0 \leq u^*, v^* \leq 1$;

$$\sum_{p=0}^1 \sum_{q=0}^1 [V_{i+p,j+q}^x - V_x] B_{p,1}(u^*) B_{q,1}(v^*) = 0, \quad (4.48)$$

$$\sum_{p=0}^1 \sum_{q=0}^1 [V_{i+p,j+q}^y - V_y] B_{p,1}(u^*) B_{q,1}(v^*) = 0, \quad (4.49)$$

where $V_{i+p,j+q}^x, V_{i+p,j+q}^y$ are the x, y components of the vertices $\mathbf{V}_{i+p,j+q}$ of $\mathcal{V}_{i,j}$. Eq.'s (4.48), (4.49) can be combined and reduced to a quadratic equation for u^* or v^* . After

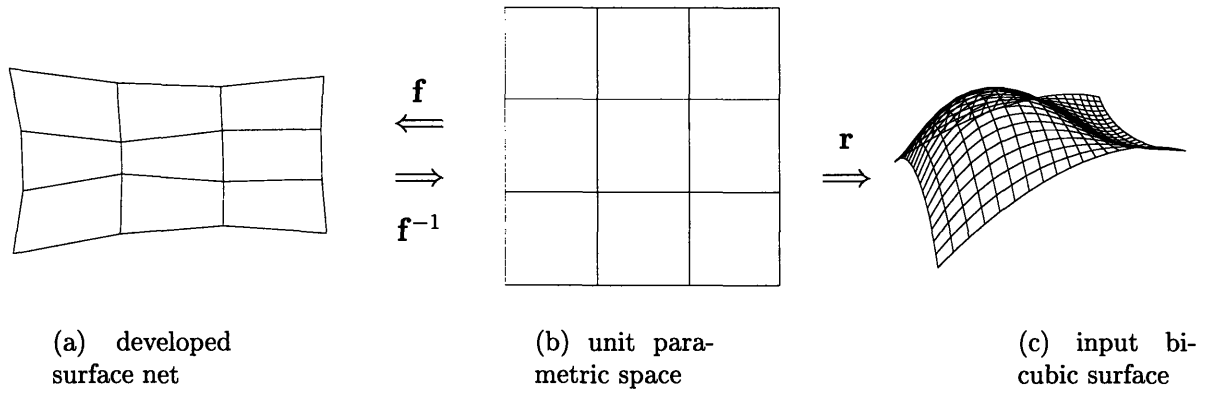


Figure 4-10: Mappings f , f^{-1} and r

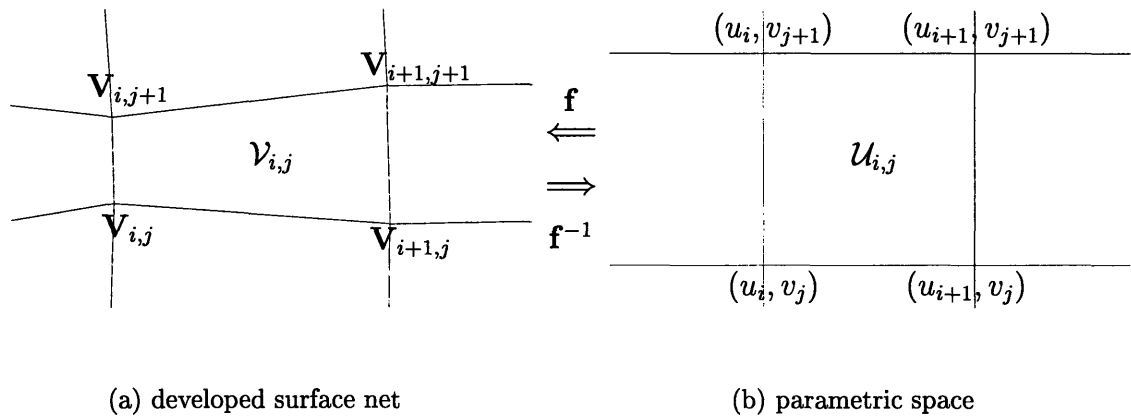


Figure 4-11: Mappings f and f^{-1}

computing u^* , v^* , we finally determine the corresponding u , v by Eq. (4.47) i.e.,

$$u = u_i + \frac{u^*}{m}, \quad v = v_j + \frac{v^*}{n}. \tag{4.50}$$

4.6 Delaunay Triangulation based on Interval Delaunay Test (IDT)

In this Section, we recall the well-known Delaunay triangulation in E^2 and associated Dirichlet tessellation. The Bowyer-Watson algorithm which is widely employed to construct an incremental Delaunay triangulation is also described. Furthermore, ro-

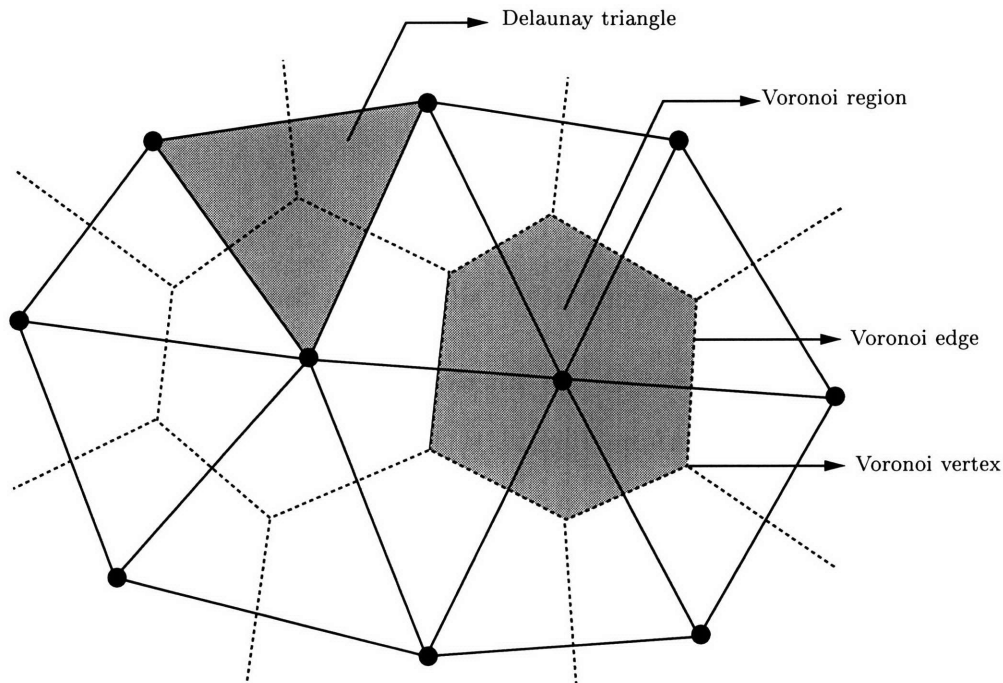


Figure 4-12: Dirichlet tessellation (dotted) and Delaunay triangulation (solid)

business problems in Delaunay triangulation are discussed and finally robust *interval Delaunay Test (IDT)* is introduced.

4.6.1 Dirichlet Tessellation and Delaunay Triangulation

The concept of Dirichlet tessellation is more than a century old, first introduced by Dirichlet [21] in 1850 and extensively discussed by Voronoi [93] in 1908. Surveys on Dirichlet tessellation are well performed in [6, 34].

Given a set of points $\{p_i\}$ in E^2 , we can define a convex polygon \mathcal{V}_i which consists of all the points *closer* to p_i than any other site p_j . These convex regions are called *Voronoi regions* and their union $\bigcup_i \mathcal{V}_i$ is called *Dirichlet tessellation* \mathcal{D} . Each vertex and edge of the Dirichlet tessellation are called *Voronoi vertex* and *Voronoi edge*, respectively. *Delaunay triangulation* \mathcal{T} [20], a dual structure of the Dirichlet tessellation \mathcal{D} , is obtained by connecting only those sites whose Voronoi regions have a common edge. Figure 4-12 shows a Dirichlet tessellation with the associated Delaunay triangulation for a small set of sites.

Some of the important properties of Dirichlet tessellation \mathcal{D} and Delaunay trian-

gulation \mathcal{T} are summarized as follows [67];

Properties of Dirichlet tessellation \mathcal{D}

- D1.** Each Voronoi region \mathcal{V}_i is convex.
- D2.** \mathcal{V}_i is unbounded if and only if the site p_i is on the convex hull of the point set.
- D3.** If v is a Voronoi vertex at the junction of \mathcal{V}_i , \mathcal{V}_j and \mathcal{V}_k , then v is the center of the circle $C(v)$ determined by p_i , p_j and p_k .
- D4.** $C(v)$ is the circumcircle of the Delaunay triangle corresponding to v .
- D5.** The interior of $C(v)$ contains no sites.
- D6.** If p_j is a nearest neighbor to p_i , then (p_i, p_j) is an edge of \mathcal{T} .
- D7.** If there is some circle through p_i and p_j which contains no other sites, then (p_i, p_j) is an edge of \mathcal{T} . The reverse also holds; for every Delaunay edge, there is some empty circle.

Properties of Delaunay triangulation \mathcal{T}

- T1.** \mathcal{T} is a triangulation if no four sites in the point set are cocircular; every face is a triangle, which is Delaunay's theorem. The faces of \mathcal{T} are called *Delaunay triangles*.
- T2.** \mathcal{T} is the straight-line dual of \mathcal{D} .
- T3.** The boundary of \mathcal{T} is the convex hull of the sites.
- T4.** Interior of each Delaunay triangle contains no sites.
- T5.** A triangulation is \mathcal{T} if it maximizes the minimum angle of triangles.

Among those interesting properties, **D5** and **T5** are especially notable for the *incremental* and *adaptive finite element meshing* application. The property **D5** is called *circumcircle property* and it provides an efficient incremental way to construct \mathcal{T} by

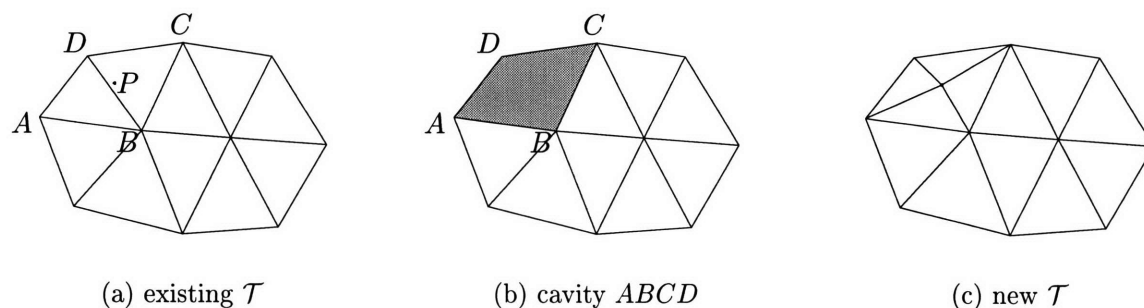
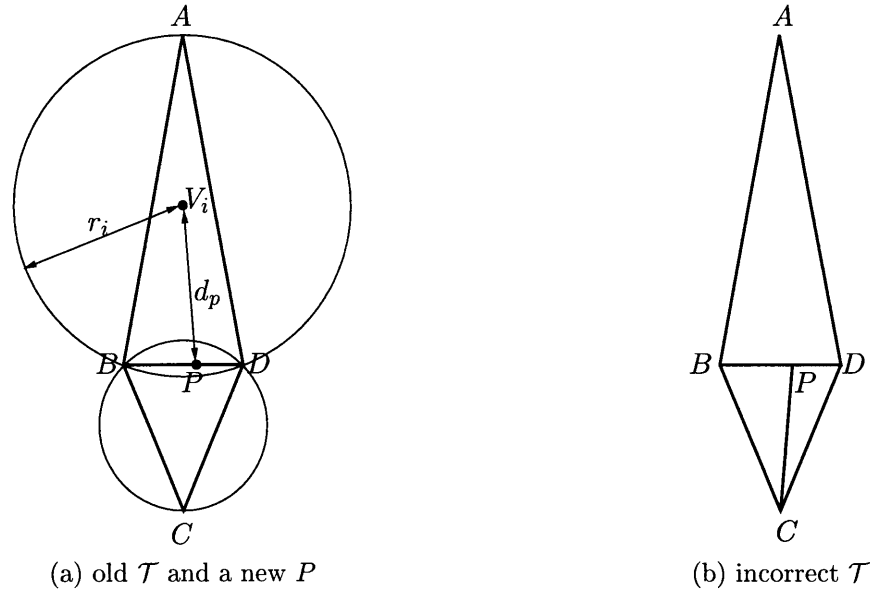


Figure 4-13: Bowyer-Watson algorithm

locally updating the existing \mathcal{T} structure. The Bowyer-Watson algorithm, which will be discussed in the following Section, is based on this property. The property **T5** is one of the most attractive features of \mathcal{T} for the finite element meshing application since, as mentioned in Sections 1.1 and 4.1, skinny triangular elements cause serious numerical problems in finite element analysis.

4.6.2 Bowyer-Watson Algorithm

Among the various Delaunay triangulation algorithms, the Bowyer [12] and/or Watson [94] algorithm is very convenient in a mesh generation procedure. The algorithm is based on the circumcircle property **D5** which guarantees that no point of a Delaunay triangulation \mathcal{T} can lie within the circle circumscribed to any Delaunay triangle. The process is *incremental*; each new point is introduced into the existing structure which is broken and then *reconnected* to form a new \mathcal{T} . The incremental procedure together with local reconnection provides a powerful tool especially for the *adaptive unstructured* meshing application. Figure 4-13-(a,b,c) illustrates the sequence of the algorithm. A new point P is found to lie inside the circumcircle of two existing Delaunay triangles ABD and BCD . These two triangles are therefore removed and a connected cavity $ABCD$ surrounding P is formed. By joining the vertices of $ABCD$ and P , a new Delaunay triangulation \mathcal{T} is *always* obtained as stated in [12, 94]. As a consequence, the Delaunay triangulation of an arbitrary set of points can be constructed in a purely sequential manner starting from a very simple initial Delaunay triangulation enclosing all points to be triangulated.

Figure 4-14: Incorrect Delaunay triangulation \mathcal{T}

4.6.3 Robustness Problems in Delaunay Triangulation

Apart from computing the radius of a circumcircle, the Delaunay algorithm in E^2 involves one of the few floating point operations, called *Delaunay test*;

$$d_p^2 = (x_p - x_i)^2 + (y_p - y_i)^2 \stackrel{?}{<} r_i^2 \quad (4.51)$$

where (x_p, y_p) is the coordinates of a newly inserted point P and r_i is the radius of a circumcircle centered at the Voronoi vertex V_i with coordinates (x_i, y_i) . For example, as shown in Figure 4-14-(a), suppose that two triangles ABD and BCD are part of a Delaunay triangulation \mathcal{T} in E^2 and that a new point P is introduced which lies on the edge BD . Since edge BD lies inside both circumcircle ABD and BCD , it will be removed by the Delaunay test and P will be joined to A, B, C and D to form a new \mathcal{T} .

Suppose, however, that owing to *floating point imprecision* the Delaunay test may not recognize P as falling inside circumcircle ABD , as reported by Baker [8] and Weatherill [95]. This could occur if the radius r_i of circumcircle ABD is extremely large and the distance d_p from P to the center V_i is only slightly less than r_i i.e., if $(r_i^2 - d_p^2)$ is the order of the *round-off error* of the computer. In such a case, the

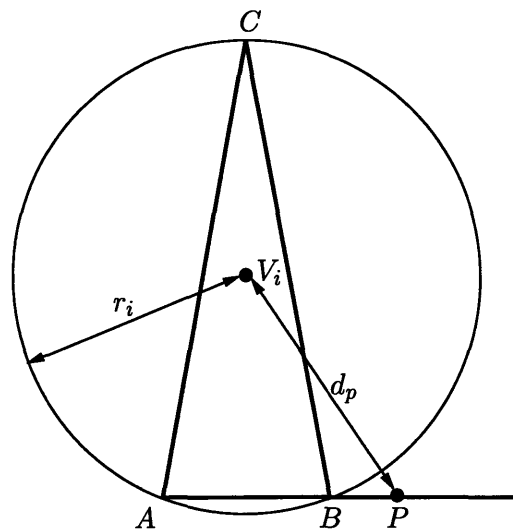


Figure 4-15: Incorrect Delaunay test

reconnections will be as shown in Figure 4-14-(b), which causes the following *two types of failures*. The three collinear points B , P and D will be regarded as forming one of the new Delaunay triangles. The algorithm will clearly fail when trying to compute the circumcenter of the *degenerate* triangle BDP because this calculation involves the solution of 2×2 system of linear equations, which is *singular* for degenerate triangles [8, 95]. The other failure, obviously seen in Figure 4-14-(b), is a *non-conforming* triangulation. In case of free-form surface meshing, triangles ABD , BCP and PCD will generate inappropriate *gaps* when they are mapped onto three-dimensional space through the surface equation.

Figure 4-15 shows a similar type of robustness problem, arising if a new point P that lies outside circumcircle ABC is incorrectly regarded as lying inside the circle, which happens if $(d_p^2 - r_i^2)$ is the order of the round-off error of the computer [8]. Point P will be connected to A , B and C and in particular, the three collinear points A , B and P will be regarded as forming one of the new Delaunay triangles. This incorrect Delaunay test causes the same *degeneracy* failure explained before. Notice here that this incorrect inclusion *alone* does not cause a failure in practice unless it is coupled with the *degeneracy*.

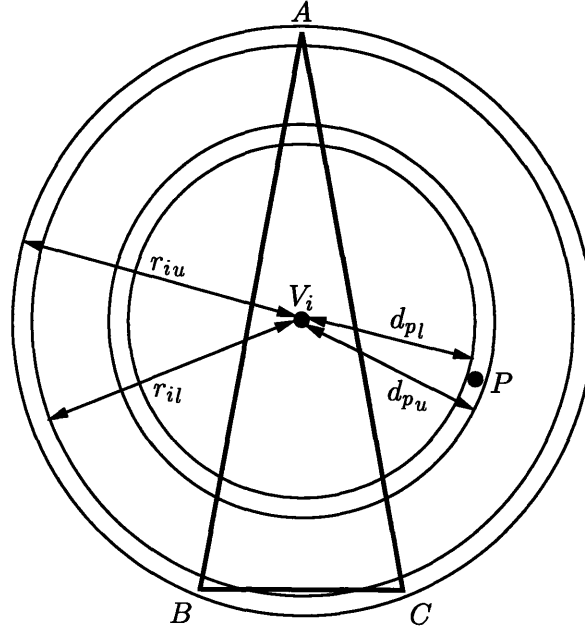


Figure 4-16: Interval Delaunay test (IDT)

4.6.4 Interval Delaunay Test (IDT)

As we discussed in the previous Section, floating point arithmetic (FPA) in Delaunay test – like other geometric computations – may cause serious failures such as degeneracies and non-conforming triangulations. To prevent those possible failures, we define new decision criteria, named *interval Delaunay test (IDT)* – see also Figure 4-16.

Definition 4.6.1 Given a newly inserted point $P(x_p, y_p)$, a Voronoi vertex $V_i(x_i, y_i)$ and the corresponding Delaunay triangle $T_i(ABC)$, we compute d_p^2 and r_i^2 defined in Eq. (4.51) using rounded interval arithmetic (RIA) and if the following two conditions **a**, **b** are satisfied, then we decide P is within the circumcircle of T_i :

a. $(d_p^2)_l \leq (r_i^2)_u$,

- b. $\left\{ \begin{array}{l} 1. \text{ all the triangles } ABP, BCP \text{ and } CAP \text{ will not degenerate, or} \\ 2. \text{ if a triangle degenerates e.g., } ABP, \text{ then } P \text{ should be within the edge } AB. \end{array} \right.$

where $(d_p^2)_l$, $(r_i^2)_u$ are the lower and upper value of interval numbers d_p^2 , r_i^2 and assuming that the newly inserted point P does not coincide with any other point in the existing point set in order to prevent a degenerate triangulation.

Condition **a** obviously removes the possibility of *incorrect exclusion*, while still possessing the possibility of *incorrect inclusion*. As stated in the previous Section, the incorrect inclusion results in a failure if it is coupled with the problem of *degeneracy*, discussed in the example of Figure 4-15. Condition **b** detects such failure. When performing the degeneracy test in condition **b**, we compute S using RIA defined by

$$S \equiv v_{1x}v_{2y} - v_{2x}v_{1y}, \tag{4.52}$$

which is equal to twice the *signed* area of a triangle defined by two vectors $\mathbf{v}_1 \equiv (v_{1x}, v_{1y})$, $\mathbf{v}_2 \equiv (v_{2x}, v_{2y})$ where $\mathbf{v}_1 = AB$, $\mathbf{v}_2 = AP$ for a triangle ABP . If S satisfies the following condition,

$$S_l \leq 0 \quad \text{and} \quad S_u \geq 0 \quad \text{where} \quad S \equiv [S_l, S_u], \tag{4.53}$$

then, we decide the corresponding triangle degenerates. Furthermore, in condition **b-2**, we decide P is within AB if

$$\begin{aligned} \min(A_x, B_x) &\leq P_x \leq \max(A_x, B_x), \\ \min(A_y, B_y) &\leq P_y \leq \max(A_y, B_y), \end{aligned} \tag{4.54}$$

where subscripts denote the corresponding coordinates of a point.

For example, a new point P in Figure 4-14 is included in the circumcircle ABD since conditions **a** and **b-2** are satisfied. Considering the example shown in Figure 4-15, P is excluded since condition **b-2** is violated.

4.7 Piecewise Planar Approximation of Composite Bézier Surfaces

We present a surface tessellation algorithm which consists of four major steps. In the first step, we approximately develop each input surface patch onto two-dimensional space by employing the isometric mapping concept discussed in Section 4.5, followed by linear approximation of boundary curves within a prescribed approximation tolerance as described in Chapter 3. By locating those boundary nodes on the approximately developed surface, we obtain a planar *domain of triangulation* for each input surface. We also identify stationary points of root mean square curvature κ_{rms} of input surfaces, as described in Section 4.4. Using those boundary and internal nodes, we achieve an *initial triangulation*.

We next refine the mesh by repeatedly inserting a node on the Voronoi vertex of each Delaunay triangle until every approximating triangle satisfies the approximation tolerance, called *δ -improving triangulation*.

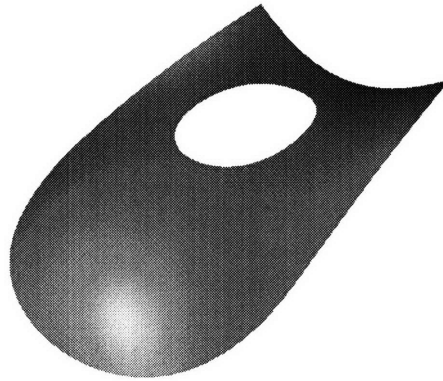
In the third step, we improve the aspect ratio (AR) of each triangular element until such AR is within the given threshold τ_{AR} or it is not improvable, and this is called the *AR-improving triangulation*.

Finally, for each approximating triangular element in E^3 , an *intersection test* is performed to identify and remove possible inappropriate intersections and to achieve a *homeomorphism* between the exact and approximating surfaces.

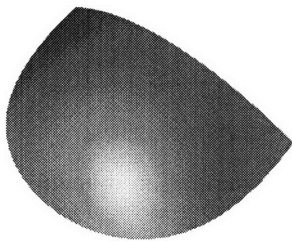
Without loss of generality, we use trimmed composite *integral Bézier* surfaces for the illustration of our methodology. We assume that each trimmed sub-patch is *simple* and of class C^m with $m \geq 2$. We also assume that it does not intersect other sub-patches except at the explicitly known common boundaries.

4.7.1 Construction of Triangulation Domain

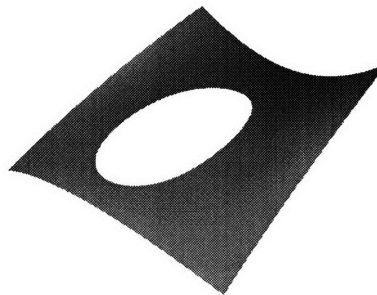
For each input surface, we first perform *approximate locally isometric mapping* described in Section 4.5 and consequently obtain an approximately developed surface.



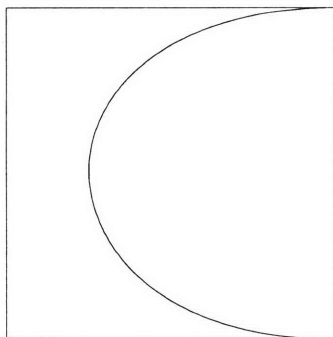
(a) a composite surface



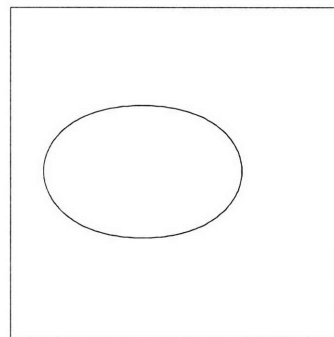
(b) trimmed surface 1



(c) trimmed surface 2



(d) parametric space 1



(e) parametric space 2

Figure 4-17: A composite surface composed of 2 trimmed bi-cubic Bézier patches

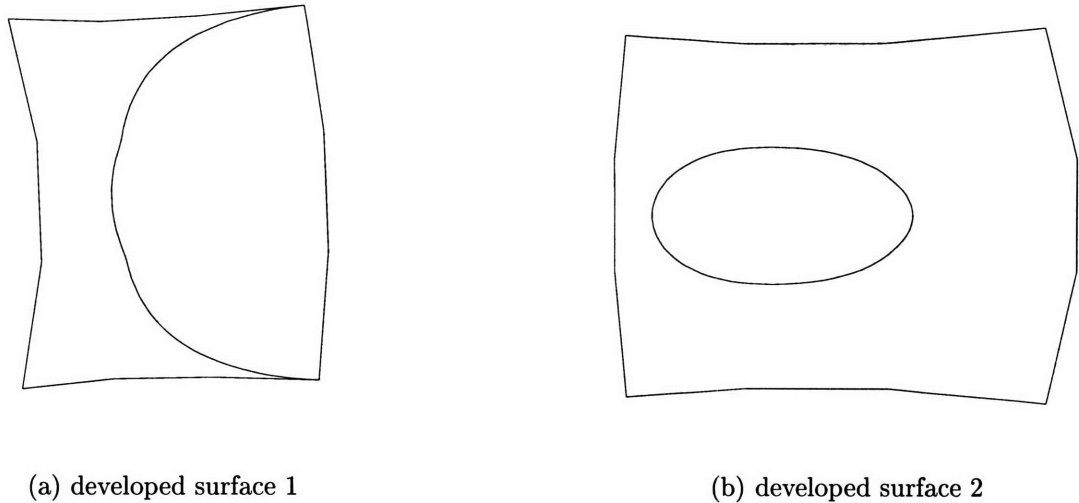


Figure 4-18: Developed surfaces together with trimming curves mapped on them

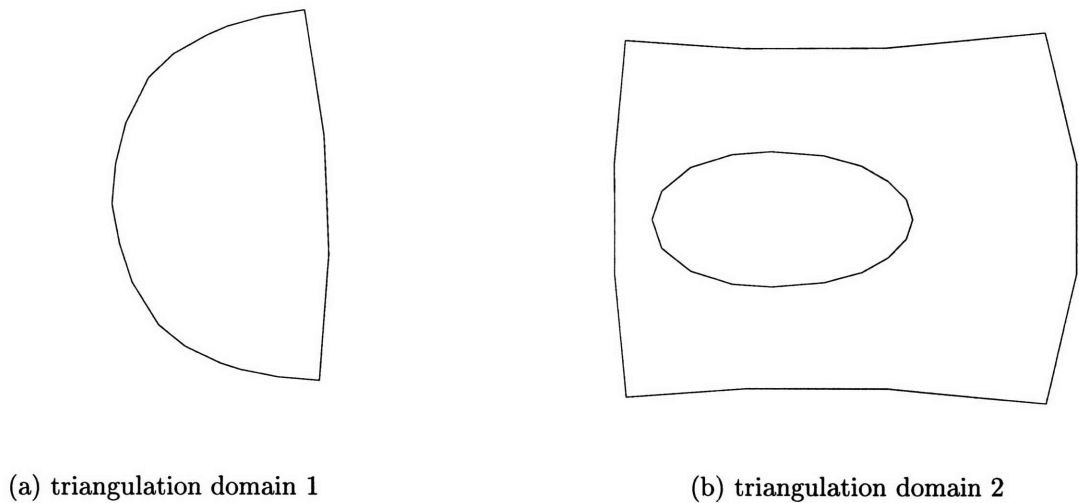


Figure 4-19: Triangulation domains with $\epsilon = 10^{-2}$

As an example, we consider a composite surface which consists of two trimmed bi-cubic Bézier surfaces, as shown in Figure 4-17. Each approximately developed surface together with mapped trimming curves are also shown in Figure 4-18. For this example, we set $p = q = 4$ in Eq. (4.44) and $\epsilon = 10^{-2}$ in Eq. (4.45).

We next perform the *topologically reliable linear approximation* of exterior and interior boundary loops of each input surface patch within a prescribed approximation tolerance ϵ , extensively discussed in Chapter 3. Notice that for a *common* boundary curve segment between the adjacent patches, we perform the linear approximation *once* and transfer the corresponding boundary data to the *mate* boundary segment

by utilizing *half-edge data structure* [62] coupled with input boundary representation (B-rep).

By mapping those boundary nodes onto each approximately developed surface, we obtain a *domain of triangulation*, as depicted in Figure 4-19 with $\epsilon = 10^{-2}$.

4.7.2 Initial Triangulation

As illustrated in Section 4.4, we compute stationary points of root mean square curvature κ_{rms} of input surfaces and use them as internal seed points in our surface tessellation algorithm.

If a stationary point of κ_{rms} exists outside the *exact* domain of triangulation, it should not be included in the triangulation. This has to be carefully checked in case of *trimmed* patches. Boundary loops of our triangulation domain are obtained by mapping each *linear approximating* boundary segment onto the triangulation domain. Therefore, for a trimmed patch, those *polygonal* boundary loops of our triangulation domain do not correspond to the *exact* boundary loops which are defined by mapping exact boundary curve segments onto the triangulation domain, as shown in Figure 4-20. For this reason, it is possible for a node to exist outside the exact domain of triangulation although it is inside our polygonal domain of triangulation e.g., node *A* in Figure 4-20.

To exclude the possibility of *incorrect node insertion*, we first check if the node is inside the *polygonal* domain of triangulation. This test can be done by inquiring if the node exists inside the exterior polygonal loop and outside the interior polygonal loops. This is well-known *vertex-loop containment* problem, which can be solved by a *ray-firing* technique [62, 45]. If those nodes associated with κ_{rms} stationary points are found to exist outside the polygonal domain of triangulation e.g., nodes *B*, *C* in Figure 4-20, we do not include such nodes in the triangulation. This decision may exclude a node which is inside the exact triangulation domain e.g., point *B* in Figure 4-20; however, it does not cause any serious problem in practice. Now, if we decide that a node exists inside the polygonal domain of triangulation, we further test if the node is

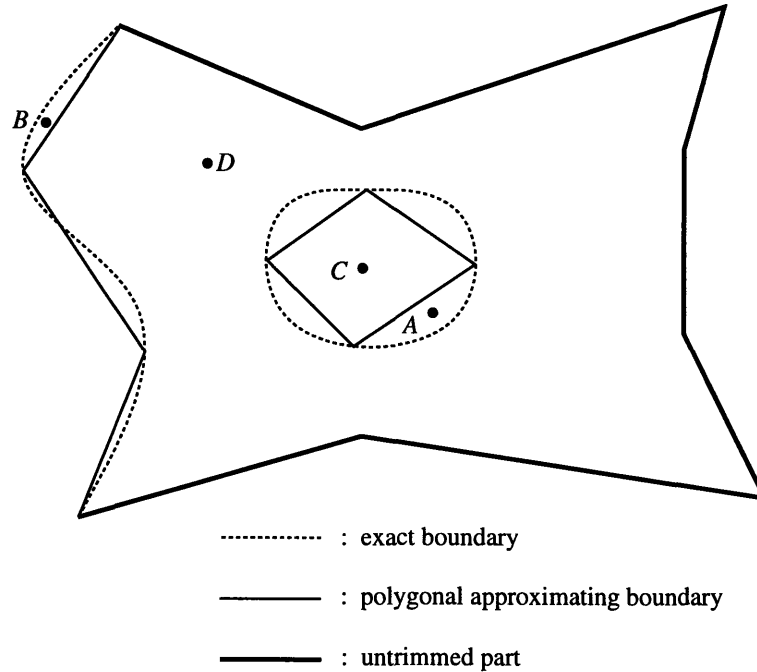


Figure 4-20: Triangulation domain for a trimmed patch

case a. *close* to the boundary or,

case b. *far enough* from the boundary.

Suppose we decide that a node is *close* to the boundary (**case a**); in such a case, we *reposition* such node to the appropriate place on the *exact* boundary segment and *update boundary information* of the triangulation domain. This procedure will prevent the possible incorrect node insertion explained before i.e., insertion of a node which exists outside the exact triangulation domain and inside our polygonal domain of triangulation e.g., node *A* in Figure 4-20. On the other hand, if we decide that a node is *far enough* from the boundary (**case b**), we insert the node *where it is*.

We now illustrate how to decide if a node is *close* to or *far enough* from the boundary. Consider a linear boundary segment \mathbf{l} on the uv -parametric space and the corresponding exact trimming curve segment \mathbf{r} , shown in Figure 4-21. We construct a control polygon for \mathbf{r} together with local xy -coordinates. We let the base x -coordinate correspond to the linear boundary segment \mathbf{l} . A *bounding box* ($BCDE$) for the control polygon of \mathbf{r} with respect to the local xy -coordinates is then determined. If a node P falls into the bounding box $BCDE$, then we decide P is *close* to the boundary

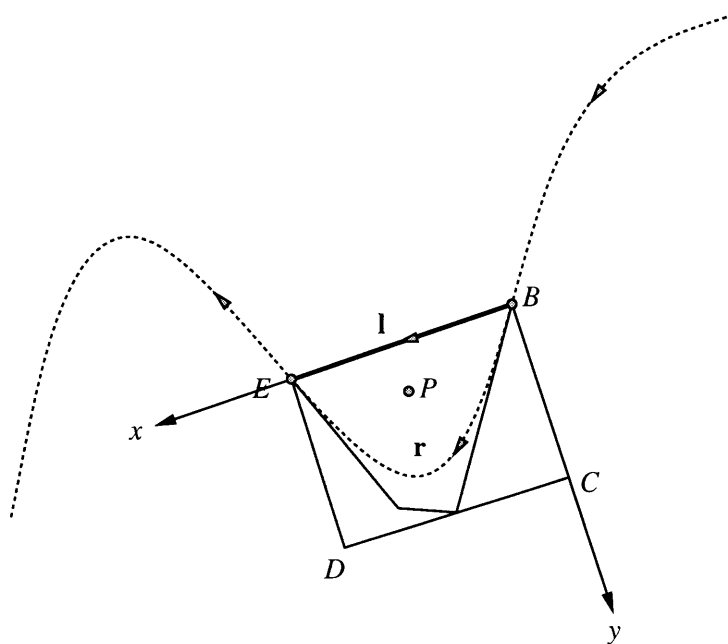


Figure 4-21: Closeness test

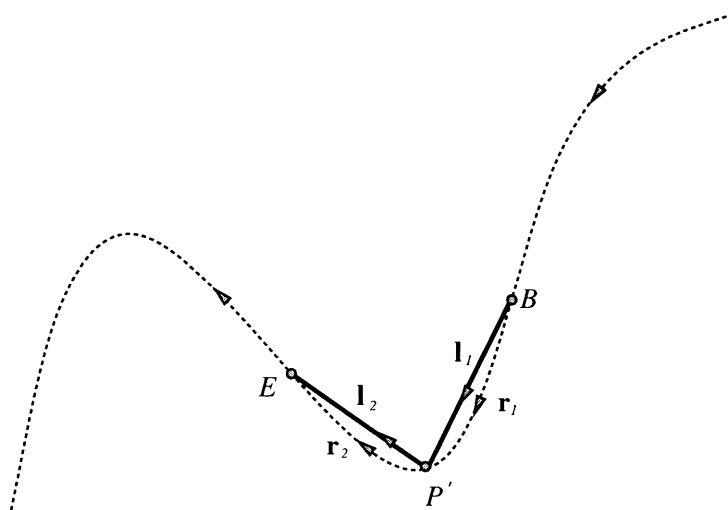


Figure 4-22: Repositioning a node

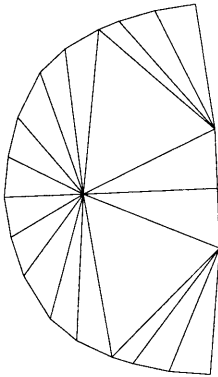
loop. Otherwise, we regard P is *far enough* from the boundary. Suppose that P is close to the boundary; P is now *repositioned* at the *midpoint* of the exact trimming curve segment i.e., at $P' = \mathbf{r}(0.5)$ and the corresponding boundary information is updated, as shown in Figure 4-22. We finally map all updated boundary data onto the triangulation domain.

Based on the node insertion criteria discussed so far, we place each node associated with κ_{rms} stationary point on the triangulation domain. With those internal as well as boundary nodes, we achieve an *initial triangulation* by performing a boundary conforming Delaunay triangulation coupled with Bowyer-Watson algorithm described in Section 4.6.2.

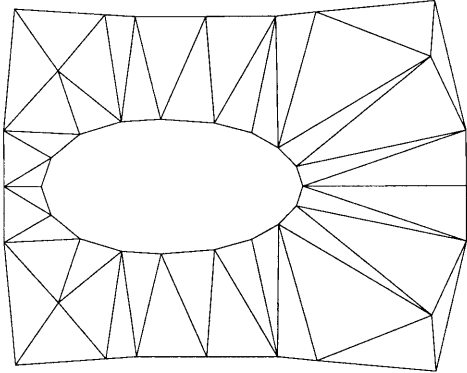
The initial triangulation should be *boundary conforming* [95] i.e., all boundary edges should be included in the triangulation. Since the Delaunay triangulation of a given point set is *unique*, there is no guarantee that the resulting triangulation will be boundary conforming for an arbitrary distribution of boundary nodes. Therefore, after the initial triangulation, we must check if each segment of the input polygonal boundary loops is included in the triangulation. It is proved in [95] that, by repeated insertions of new boundary nodes at the midpoint of the missing boundary edges, a boundary conforming triangulation is always achieved. Our node insertion method shown in Figure 4-22 is also employed when inserting a new boundary node at the midpoint of a missing boundary edge.

Two different types of triangles can be identified after the initial triangulation – the triangles *internal* to the domain of triangulation and the triangles *external* to it. Those external triangles belong to the trimmed part of the triangulation domain or in the background region outside the exterior polygonal boundary loop. We will keep track of the external triangles as well as the internal ones in order to check the necessity of boundary refinement, which will be discussed in the next Section. By simply removing the external triangles, we can achieve a true approximation of the exact trimmed surface patches.

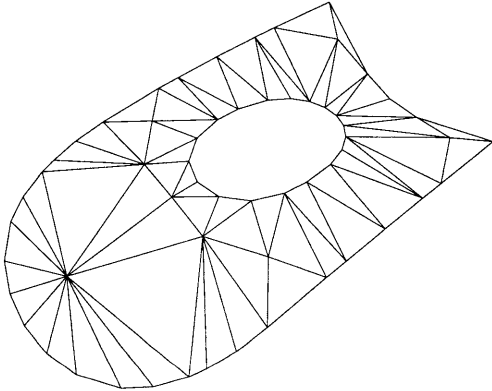
Figure 4-23 shows an initial triangulation after removing the external triangles for our example discussed in Figures 4-17, 4-18 and 4-19.



(a) triangulation domain 1



(b) triangulation domain 2



(c) 3D space

Figure 4-23: Initial triangulation

4.7.3 δ -Improving Triangulation

For each triangular approximating element obtained by the initial triangulation, we compute an approximation error. Using the method discussed in Section 4.3.3, we extract a triangular sub-Bézier surface patch $\mathbf{r}(u, v, w)$ corresponding to the triangular approximating element $\mathbf{s}(u, v, w)$. Then, the approximation error δ is easily determined by evaluating an upper bound of the absolute position difference at isoparametric points by Eq. (4.23) i.e.,

$$\delta \equiv \max_{i,j,k} |\mathbf{r}_{i,j,k} - \mathbf{s}_{i,j,k}|, \quad (4.55)$$

where integers $i, j, k \geq 0$, $i + j + k = m + n$ for an input tensor product Bézier surface patch $\mathbf{R}(s, t)$ of degrees m, n in the st -parametric directions and $\mathbf{s}_{i,j,k}$, $\mathbf{r}_{i,j,k}$ are control points of $\mathbf{s}(u, v, w)$ and $\mathbf{r}(u, v, w)$, respectively.

After computing δ^2 for each triangular element, a triangular element is chosen whose δ^2 is the largest. If the maximum δ^2 is not within the square of a given approximation tolerance ϵ^2 , a new node is placed on the Voronoi vertex – see also Figure 4-12 – of the corresponding triangle on the triangulation domain and the triangulation is updated using Bowyer-Watson algorithm. We then compute δ^2 for each newly generated triangle and repeat the same procedure until every approximating element satisfies ϵ -tolerance. We call this procedure *δ -improving triangulation*. A similar Voronoi-vertex point insertion algorithm was introduced by Rebay [78] in his *planar mesh generation algorithm*, whose triangular elements satisfy the prescribed *circumcircle radius limit*.

As shown in Figure 4-24, a Voronoi vertex V of a triangle ABC can exist inside the circumcircle of an *external* triangle DBA . This usually happens if the *longest* edge of an *optuse* triangle is on the boundary loop. In such a case, instead of Voronoi vertex V , we insert a node P at the midpoint of the corresponding *exact* boundary segment AB and update the triangulation, which is similar to the *node repositioning* step described in Section 4.7.2. Figure 4-25 together with Figure 4-24 illustrates the procedure. Note that if the updated boundary segment has a mate segment i.e., it

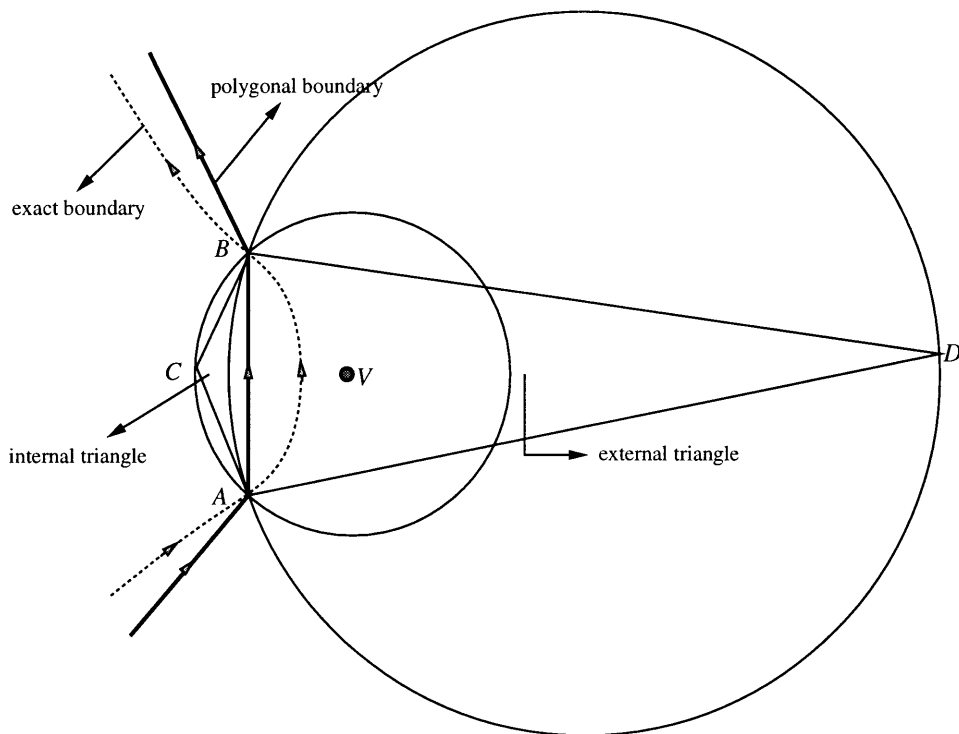


Figure 4-24: An internal ABC and external DBA triangle together with a boundary edge AB and center V of circumcircle ABC

is a part of common boundaries between two neighboring surface patches, the mate segment should be updated, too.

Figure 4-26 shows the result of δ -improving triangulation with $\epsilon = 10^{-2}$ corresponding to the initial triangulation shown in Figures 4-23. Total number of triangular elements is 118 and the worst and average approximation error are 0.96×10^{-2} and 0.55×10^{-2} , respectively.

4.7.4 AR-Improving Triangulation

As stated before, unstructured triangular meshes for free-form surfaces frequently suffer from badly shaped triangles with *high aspect ratio* (AR) [73]. If such low quality elements are used for further geometric computations or general analysis, they cause serious numerical problems [7, 88]. The aspect ratio (AR) of a triangle is defined by *the ratio of the length of the longest side to the height*, where the height is the minimum distance from a vertex to the opposite side i.e., in Figure 4-27 for a triangle

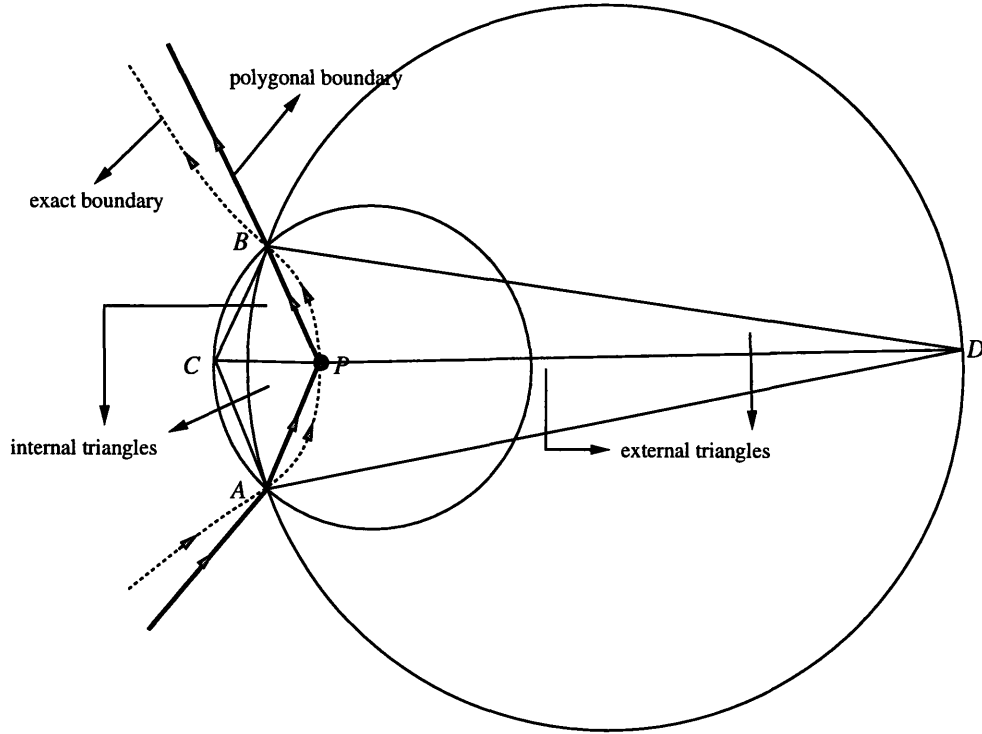


Figure 4-25: Node (P) insertion at the midpoint of the exact boundary segment and the resulting triangulation

ABC ,

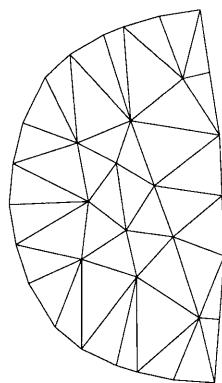
$$AR \equiv \frac{|\mathbf{p}|}{d} = \frac{|\mathbf{p}|^2}{|\mathbf{q} \times \mathbf{p}|}. \quad (4.56)$$

It is obvious that the *equilateral* triangle has the *least* $AR = \frac{2}{\sqrt{3}} \approx 1.155$.

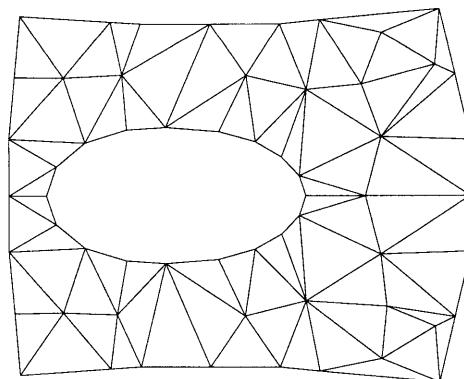
Our free-form surface triangulation algorithm constructs planar domain of triangulation which sufficiently preserves the shape of triangular elements when mapped into three-dimensional space. This feature enables us to achieve a well-shaped triangulation in three-dimensional space by improving AR of each triangle *on the planar triangulation domain*. The improvement of AR is performed in the triangulation domain until

condition a. AR of every triangle is below a given threshold τ_{AR} or,

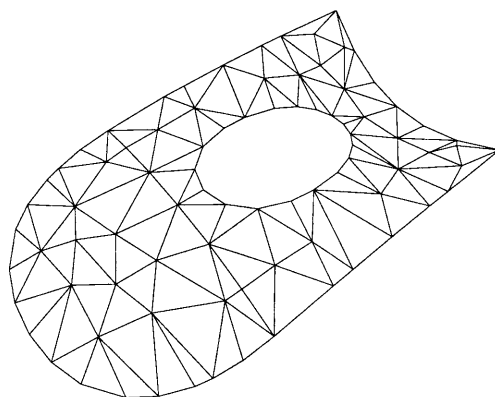
condition b. for those triangles having $AR > \tau_{AR}$, their AR's can not be improved any more.



(a) triangulation domain 1



(b) triangulation domain 2



(c) 3D space

Figure 4-26: δ -improving triangulation

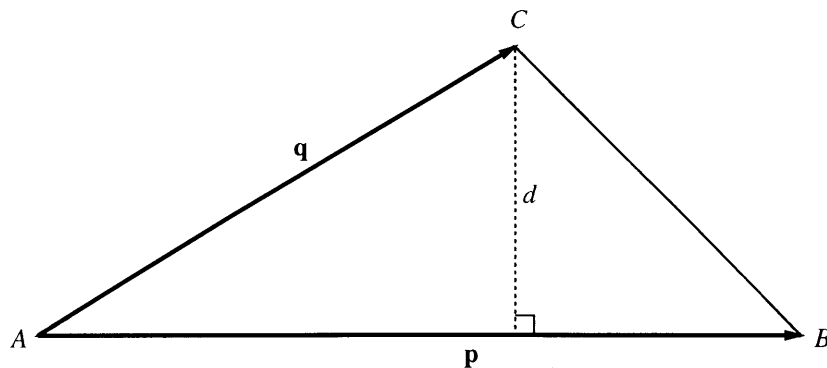


Figure 4-27: A triangle ABC

In condition **a**, given τ_{AR} should satisfy

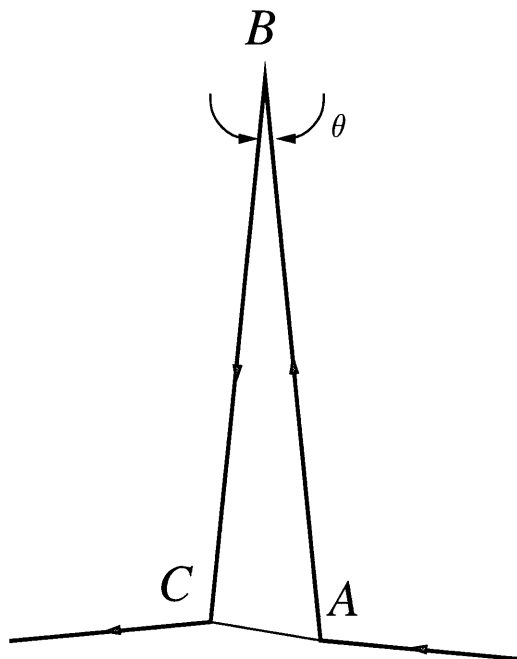
$$\tau_{AR} \geq \frac{2}{\sqrt{3}}, \quad (4.57)$$

since no triangles can have the aspect ratio less than that of an equilateral triangle. The condition **b** may appear vague but it will be explained later in detail. We note that our AR-improving process can be immediately applied to not only trimmed surface meshing but also another well-known computational geometry problem i.e., triangulation of a multiply connected *planar* region.

Our algorithm utilizes the concept of *Voronoi-segment point insertion algorithm* [78], where the author repeatedly inserts a new node on the Voronoi edge to generate a new triangle whose circumcircle radius is identical to the prescribed limit, where the Voronoi edge is an edge of a Voronoi region, as shown in Figure 4-12.

We now illustrate the procedure of *AR-improving triangulation* in detail. After the δ -improving triangulation, the *internal* triangles on the triangulation domain are further classified into two types of triangles depending on their AR – triangles already satisfying and not yet satisfying the threshold (τ_{AR}) of an aspect ratio. These two types of internal triangles are called *done* and *undone* triangles, respectively. Since we will insert a node by considering an undone triangle whose neighbors are done or external triangles, we need to introduce a further classification of the *undone* triangles namely, *active* and *waiting* triangles. An active triangle is an undone triangle having at least one *done* or *external* triangle among its neighbors. Finally, all the undone triangles which are *not active* are called waiting triangles. Therefore, a waiting triangle is surrounded by active or other waiting triangles.

Before we start the AR-improving triangulation for each triangle obtained from the previous δ -improving procedure, a *feasibility test* must be done for those triangles two of whose edges correspond to the boundary loop segments, as depicted in Figure 4-28. We first check if an angle θ made by two boundary loop segments *AB* and *BC* is the *minimum angle* of triangle *ABC* – this can be done by just checking if the *opposite edge CA* to the angle θ is the shortest one among three edges *AB*, *BC* and

Figure 4-28: A boundary triangle ABC

CA . This being true, a further test is performed. For a triangle whose least angle $\theta \ll 1$,

$$\text{AR} \approx \frac{1}{\sin \theta} \approx \frac{1}{\theta} \quad (4.58)$$

and thus, we compute θ and if

$$\theta < \frac{1}{\tau_{\text{AR}}} \quad (4.59)$$

then, we regard AR of the triangle ABC as *not improvable* and mark ABC to be a *done* triangle, in other words we do not attempt to improve AR of such triangle.

After classifying each internal triangle obtained from the δ -improving triangulation, we pick up a triangle whose AR is the *largest* among the *active* triangles on the triangulation domain. A new node is inserted by considering the Voronoi edge shared by the chosen active triangle and one of its *done* or *external* neighboring triangles, as depicted in Figure 4-29. If more than one done or external neighbors exist, the Voronoi edge associated with the *shortest* edge of the active triangle is chosen. As

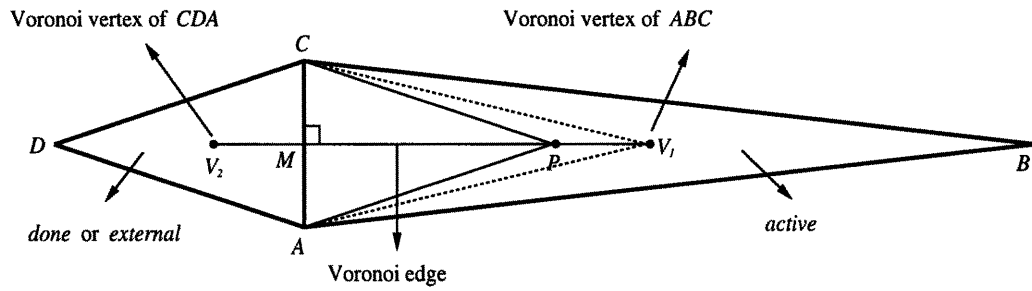
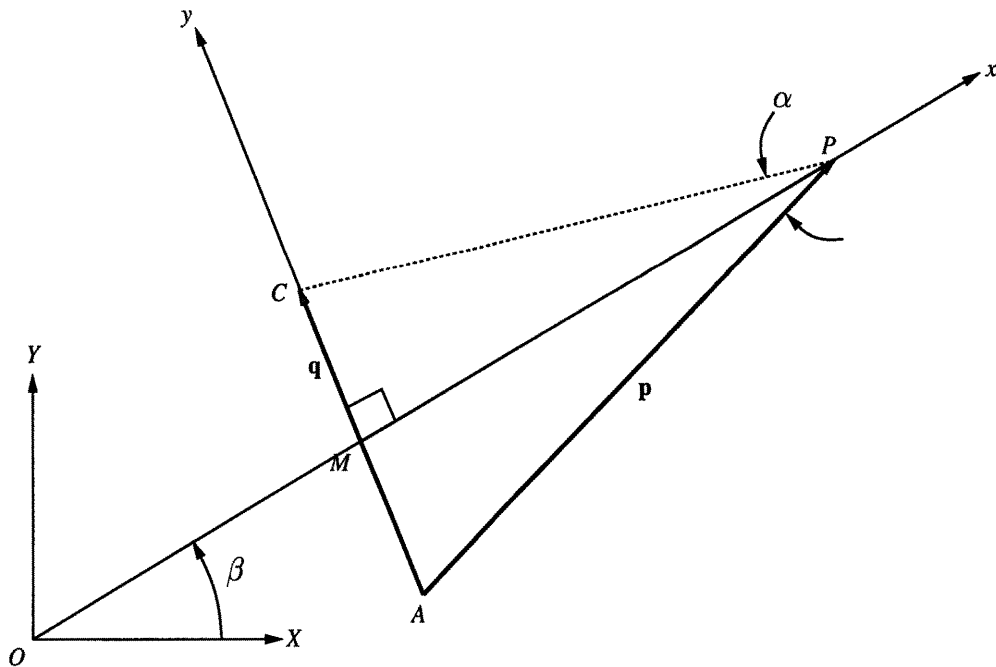


Figure 4-29: Node insertion in AR-improving triangulation

Figure 4-30: An isosceles triangle ABC

shown in Figure 4-29, the position of a new node P along the Voronoi edge V_1V_2 is chosen in an attempt to generate an *isosceles* triangle APC whose AR is equal to the prescribed threshold τ_{AR} ($\geq \frac{2}{\sqrt{3}}$). Furthermore, we assume the edge CA to be the shortest edge of the newly generated isosceles triangle APC , which means an angle between two edges AP and PC should not be greater than 60° . Hence, with reference to Figure 4-30, we need to solve the following problem.

Given : an edge CA and a local axis x which is perpendicular to CA with its origin equal to the midpoint M of the edge CA .

Find : a point P which makes an isosceles triangle APC having AR equal to the prescribed τ_{AR} ($\geq \frac{2}{\sqrt{3}}$).

Constraint : angle α between two edges AP and PC is not greater than 60° .

Two given points $C : (C_X, C_Y)$, $A : (A_X, A_Y)$ in terms of global XY -coordinates system can be expressed as $C : (0, c)$, $A : (0, -c)$ with respect to the local xy -coordinates system, where

$$c = \frac{|\mathbf{q}|}{2} \quad \text{and the vector} \quad \mathbf{q} = C - A = (C_X - A_X, C_Y - A_Y). \quad (4.60)$$

Furthermore, the point P can be represented by $P : (p, 0)$ with respect to the local xy -coordinates system. We now determine p . By Eq. (4.56), the aspect ratio (AR) of triangle ABC is

$$\text{AR} \equiv \frac{|\mathbf{p}|^2}{|\mathbf{q} \times \mathbf{p}|} = \frac{c^2 + p^2}{2cp}. \quad (4.61)$$

By setting $\text{AR} = \tau_{AR}$ in Eq. (4.61), we obtain the following quadratic equation with respect to p .

$$p^2 - 2\tau_{AR}cp + c^2 = 0, \quad (4.62)$$

whose roots are

$$p = (\tau_{AR} \pm \sqrt{\tau_{AR}^2 - 1})c, \quad (4.63)$$

where $\tau_{AR}^2 > 1$ *always* since $\tau_{AR} > \frac{2}{\sqrt{3}}$. Notice that we have to choose the bigger one as a true root in Eq. (4.63) i.e.,

$$p = (\tau_{AR} + \sqrt{\tau_{AR}^2 - 1})c. \quad (4.63')$$

The reason why the other root should be discarded is easily proved as follows; *assume* the other root $p = (\tau_{AR} - \sqrt{\tau_{AR}^2 - 1})c$ is a true root. For an equilateral triangle, $p = \sqrt{3}c$. Therefore, by the **Constraint** stated before,

$$p = (\tau_{AR} - \sqrt{\tau_{AR}^2 - 1})c > \sqrt{3}c. \quad (4.64)$$

By solving the inequality (4.64), we obtain

$$\tau_{AR} < \frac{2}{\sqrt{3}}. \quad (4.65)$$

However, this is *impossible* because no triangles can have AR less than that of equilateral triangle. Therefore, $p = (\tau_{AR} - \sqrt{\tau_{AR}^2 - 1})c$ can not be a true root. \square

Finally, we express $P : (p, 0)$ with respect to the given global XY -coordinates system i.e.,

$$P : (P_X, P_Y) = (M_X + p \cos \beta, M_Y + p \sin \beta), \quad (4.66)$$

where

$$(M_X, M_Y) = \left(\frac{C_X + A_X}{2}, \frac{C_Y + A_Y}{2} \right), \quad (4.67)$$

$$\cos \beta = \frac{C_Y - A_Y}{|\mathbf{q}|}, \quad \sin \beta = \frac{A_X - C_X}{|\mathbf{q}|}, \quad (4.68)$$

$$|\mathbf{q}| = \sqrt{(C_X - A_X)^2 + (C_Y - A_Y)^2}, \quad (4.69)$$

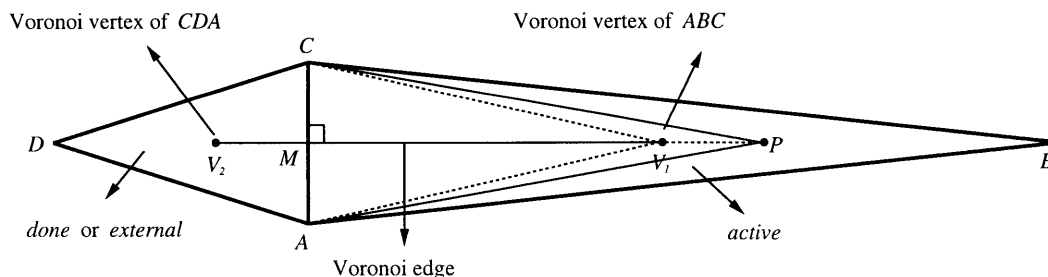


Figure 4-31: A new node P not on the Voronoi edge V_1V_2 : **case 1**

and p is defined in Eq. (4.63').

Notice that the new node P computed by Eq. (4.66) may not be within the Voronoi edge between the chosen active triangle and the neighboring done or external triangle, as shown in Figures 4-31 and 4-32. In other words, the Voronoi edge V_2V_1 is too short with respect to V_2P . This happens if

case 1 : the given τ_{AR} is large compared with the length of the Voronoi edge, as depicted in Figure 4-31 or

case 2 : the shortest edge of the chosen active triangle is much shorter than the other two edges and furthermore, the neighbor of the shortest edge is neither *done* nor *external* i.e., it is a *waiting* or another *active* triangle, as illustrated in Figure 4-32.

The **case 2** may cause an undesirable result, since the node P very likely does not delete the chosen active triangle ABC . This means that P is prone to exist outside the circumcircle of ABC . If such a case is encountered, we insert a new node at the Voronoi vertex V_1 of ABC instead of P . This is an appropriate alternative because the new node at Voronoi vertex V_1 will clearly delete the *currently worst active triangle* ABC by Bowyer-Watson algorithm. Furthermore, this alternative will be also applied to the **case 1** in an attempt to generate a new triangle AV_1C having a *better* AR than a triangle APC , as shown in Figure 4-31.

Another exceptional case occurs if an inserted new node is within the circumcircle of an *external* triangle, which was already seen in δ -improving triangulation of Section 4.7.3 – see also Figures 4-24 and 4-25. In AR-improving triangulation, this typically

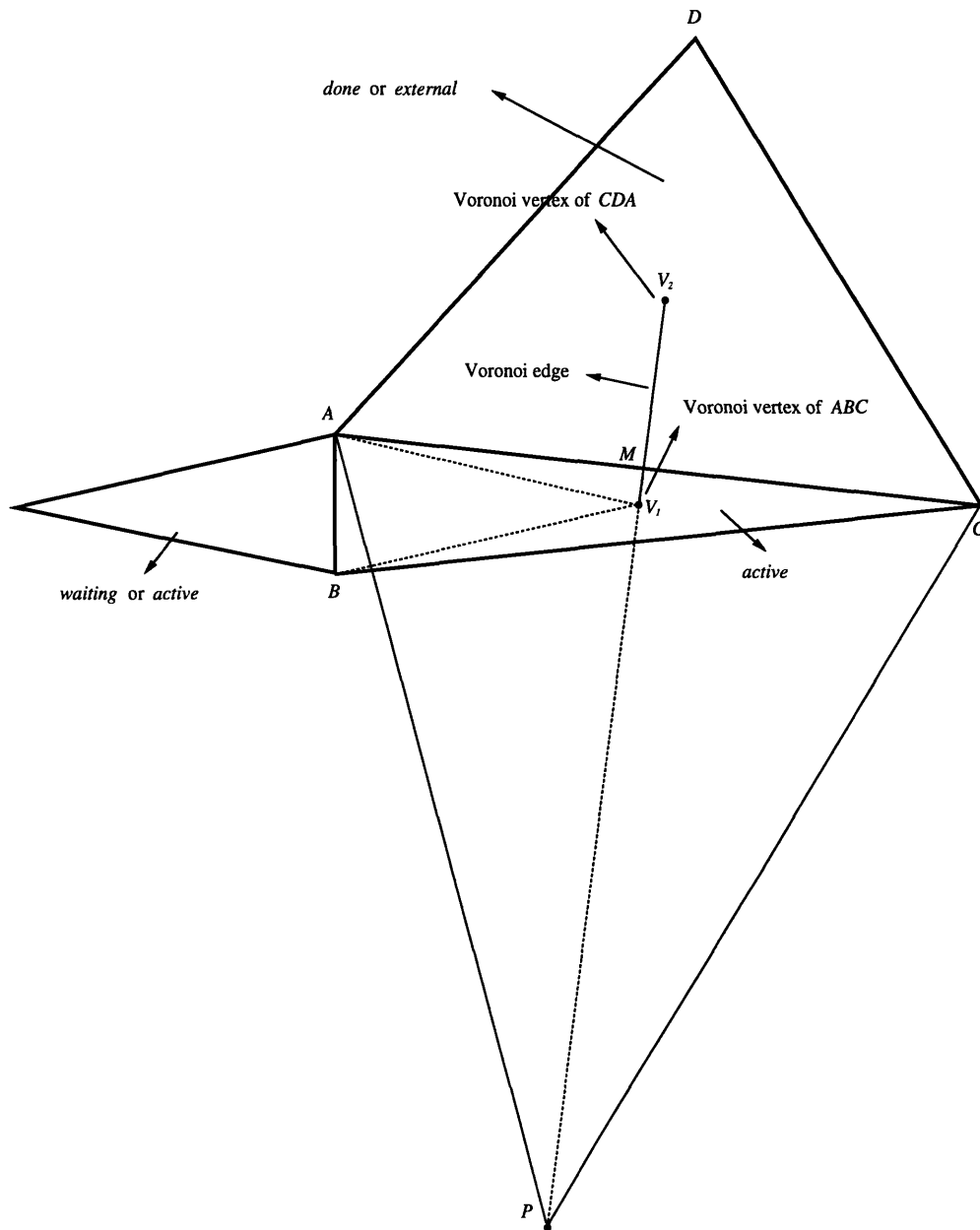


Figure 4-32: A new node P not on the Voronoi edge V_1V_2 : case 2

happens when the above treatment for **case 2** is performed for an *optuse* triangle whose *longest* edge is a *boundary loop segment*. In other words, this is a case that a Voronoi vertex of the boundary *active* triangle is inserted in the triangulation instead of a point computed by Eq. (4.66) and moreover, the Voronoi vertex is within the circumcircle of an *external* triangle. In such a case, we apply the same treatment as described in Figures 4-24 and 4-25 of Section 4.7.3. Notice here that a triangle having *better* AR, is usually generated by inserting a node at the midpoint of the *longest* edge of an *optuse* triangle.

Now, we need to consider the *trade-off* between the δ -improving and AR-improving triangulations. In order to make the algorithm as optimal as possible in terms of the approximation tolerance and the number of approximating elements, we have to achieve a surface triangulation satisfying the approximation tolerance ϵ with as small a number of approximating elements as possible. However, regardless of ϵ , the AR-improving procedure can generate a number of triangles in addition to the nearly optimal number of triangles obtained through the δ -improving triangulation. This drawback is further aggravated as τ_{AR} becomes more ideal. To remedy this problem, it is desirable to introduce a threshold τ_r which bounds the *smallest size* of a triangle on the triangulation domain in AR-improving procedure. The most appropriate quantity which measures the size of a triangle is the square r^2 of a circumcircle radius of the triangle. Apart from its close geometric relation to the size of a triangle, it is introduced for the efficiency of the algorithm – r^2 is already computed in Bowyer-Watson algorithm and hence, no additional computations are required to determine r^2 . We may input the lower bound τ_r arbitrarily and modify it interactively by trial and error. However, for the *fully automatic* surface meshing procedure, we need to define an appropriate formula governing τ_r . Furthermore, τ_r should be expressed in terms of the prescribed approximation tolerance ϵ to achieve a *dynamic relation* between ϵ and τ_r . In other words, we automatically assign smaller (bigger) τ_r as ϵ becomes tighter (looser) respectively, by the formula $\tau_r = \tau_r(\epsilon)$. Note here again that all the approximating triangles already satisfy the approximation tolerance ϵ when entering AR-improving triangulation, and hence the AR-improving procedure should

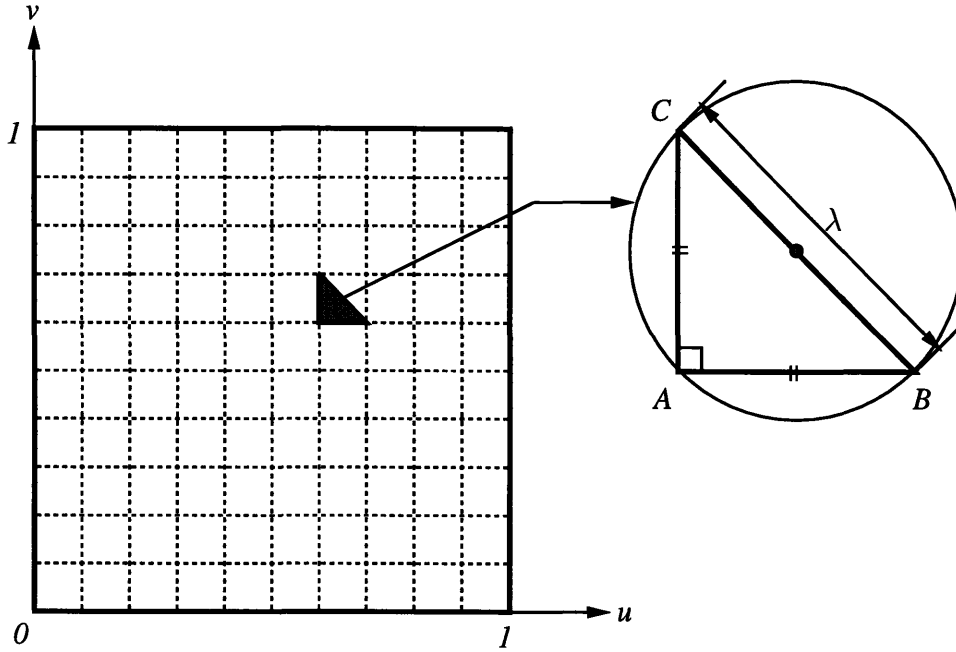


Figure 4-33: A uniform triangulation on the unit uv -parametric space

not unreasonably increase the number of approximating triangles with respect to ϵ .

In order to keep AR-improving procedure from possibly generating a number of triangles, we propose an appropriate formula for τ_r as follows.

$$\tau_r \equiv (\rho r_0)^2, \quad (4.70)$$

where

$$\rho = \frac{\epsilon}{\epsilon_0}, \quad r_0 = \frac{\lambda}{2}, \quad (4.71)$$

ϵ , ϵ_0 are the approximation tolerances on a *real* and *normalized* scale, respectively and

$$\lambda = 3\sqrt{\frac{\epsilon}{2(D_1 + 2D_2 + D_3)}}, \quad (4.72)$$

where

$$D_1 \equiv \max_{i,j} |\mathbf{r}_{i,j}^{2,0}|, \quad (4.73)$$

$$D_2 \equiv \max_{i,j} |\mathbf{r}_{i,j}^{1,1}|, \quad (4.74)$$

$$D_3 \equiv \max_{i,j} |\mathbf{r}_{i,j}^{0,2}|. \quad (4.75)$$

In Eq.'s (4.73) – (4.75), the $\mathbf{r}_{i,j}^{k,l}$ denote the *control points* of a partial derivative $\frac{\partial^{k+l}\mathbf{r}}{\partial^k u \partial^l v}$ of the exact Bézier surface patch $\mathbf{r}(u, v)$ under triangulation. Therefore, D_1 is the maximum of the norms $|\mathbf{r}_{i,j}^{2,0}|$ for all possible $0 \leq i \leq m - 2$ and $0 \leq j \leq n$, where m , n are the degrees of $\mathbf{r}(u, v)$ in the u , v parametric directions, respectively. In a similar way, D_2 and D_3 can be described. We now need to illustrate the meaning of λ in Eq. (4.72). For an untrimmed Bézier surface patch \mathbf{r} , if the longest edge (hypotenuse) size of the right-angled isosceles triangles, uniformly distributed on the unit uv -parametric space, does not exceed λ defined in Eq. (4.72), then all the corresponding approximating triangles in the three-dimensional space guarantee the approximation tolerance ϵ , as stated in [73] which utilized a theorem by Filip *et al.* [32] – see also Figure 4-33. This is a really strong condition globally bounding the allowable size of the triangles in terms of ϵ and thus, the corresponding triangulation usually results in unnecessarily large number N^* of approximating elements, as mentioned in Section 2.3. We base the lower bound τ_r of the square r^2 of circumcircle radius in our AR-improving triangulation upon the size limit λ , expecting the number of triangles *even* after the AR-improving triangulation does not exceed $\mathcal{O}(N^*)$. For an isosceles right-angled triangle ABC in Figure 4-33, its circumcircle radius is $\frac{\lambda}{2}$, which is r_0 in Eq. (4.71). Furthermore, the size of our triangulation domain is comparable to the input surface patch and hence, we have to rescale r_0 with respect to the *real* scale, which is achieved by ρ in Eq. (4.71). Based on the above descriptions, the lower bound τ_r is defined as Eq. (4.70).

In AR-improving triangulation, if the currently chosen worst active triangle T has $r^2 > \tau_r$, then T is regarded as *not improvable* and is marked as *done* and we choose the second worst active triangle T' and repeat the AR-improving procedure. This

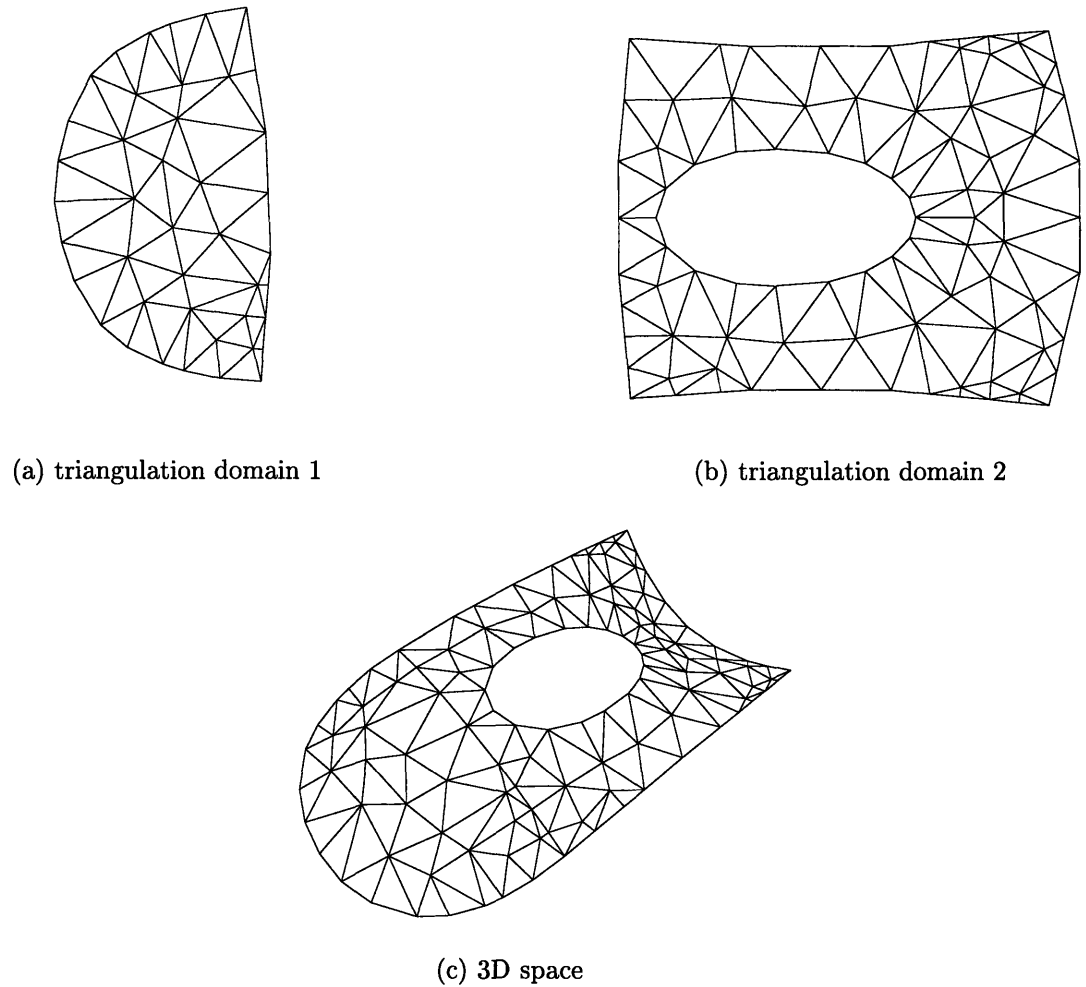


Figure 4-34: AR-improving triangulation

is an appropriate strategy which balances the number and AR of the approximating triangles.

Based on the node insertion strategy described so far, we repeatedly place a node on the triangulation domain and update the triangulation by Bowyer-Watson algorithm until every internal triangle becomes *done* i.e., **condition a** or **condition b**, mentioned earlier in this Section, is satisfied.

Figure 4-34 shows the result of AR-improving procedure corresponding to the δ -improving triangulation with $\epsilon = 10^{-2}$ shown in Figures 4-26. We set τ_{AR} to be 3, which guarantees that any angle θ of a triangle in the triangulation domain is $20^\circ < \theta < 113^\circ$. Furthermore, $\tau_r \approx 0.02$ and 0.015 for the first and second surface patch, respectively. Total number of approximating triangles is 181 and the worst

and average AR is 3.64, 1.99 in the triangulation domain, and 3.60, 2.00 in the three-dimensional space, respectively. The number of triangles whose AR's are greater than $\tau_{AR} = 3$ is 10 (13) in the triangulation domain (three-dimensional-space), respectively. In other words, approximately only 6% of the total number of triangles are regarded as *not improvable* in the AR-improving triangulation and furthermore, only 7% of the approximating triangles in the three-dimensional space do not satisfy the threshold τ_{AR} .

More importantly, we can verify the excellent *average* AR of the approximating triangles in the three-dimensional space. This is due to the coupled effects of the features of Delaunay based triangulation, AR-improving procedure in the planar domain of triangulation and preservation of triangles' shape during the mapping process from the triangulation domain into the three-dimensional space.

4.8 Intersection Test and Final Triangulation

Suppose that the exact input trimmed surfaces constitute *a set of mutually non-intersecting simple composite surfaces*, the approximating triangular elements should not have any inappropriate *gap* nor have any inappropriate *intersection* in order to achieve a *homeomorphism* between the exact composite surface and its approximation [2, 10, 62, 72]. The term *simple composite surface* here denotes a composite surface without self-intersections, whose trimmed sub-patches $\mathbf{r}(u, v)$ of class C^m with $m \geq 2$ are simple (non-self-intersecting) as well as regular ($\mathbf{r}_u \times \mathbf{r}_v \neq \mathbf{0}$).

During the surface triangulation procedure based on the robust *interval* Delaunay test, the inappropriate gap is prevented by utilizing the half edge data structure coupled with interval boundary representations. In other words, whenever a node is inserted on the triangulation domain, a *conforming* triangulation is achieved with the help of the interval Delaunay test and in particular, if the newly inserted node exists on the *boundary* loop segment, its corresponding *mate boundary* can be also updated in a consistent manner. However, inappropriate intersections may exist between the approximating triangles in E^3 e.g., global distance function features of the

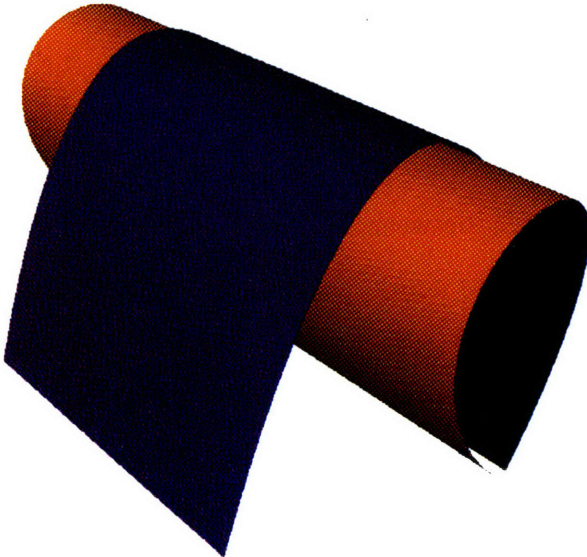
input surface such as *constrictions* as depicted in Figure 4-35. If such inappropriate intersections exist, they necessitate further local refinement of the approximation until every approximating triangle does not have any inappropriate intersection – *final triangulation*. Similar to the case of linear approximation described in Section 3.6, a bucketing technique [5, 19] is used to identify such an inappropriate intersection.

4.8.1 Homeomorphism between the Exact and Approximating Surface

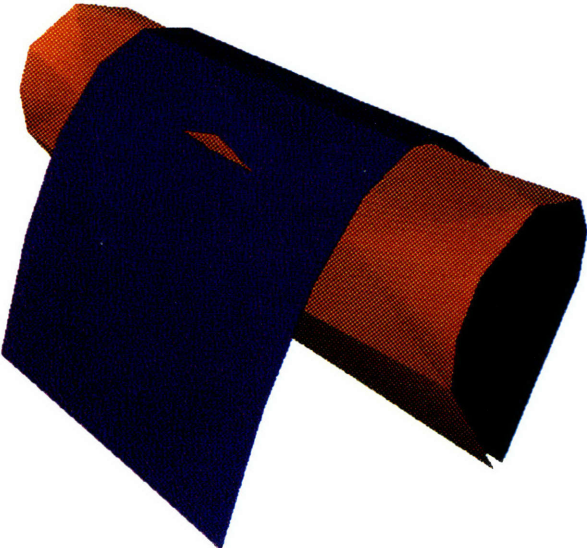
Before we proceed to the final triangulation scheme, we prove the existence of a *homeomorphism* between a set of mutually non-intersecting simple composite surfaces and the corresponding triangular approximation without any inappropriate intersection nor gap. The homeomorphism under consideration is constructed by using **Definition 3.6.1** and **Lemma 3.6.2** in Section 3.6.1. Furthermore, based on the fact that any trimmed simple parametric surface patch can be decomposed into a set of conforming triangular surface patches, we only need to prove the following **Theorem**.

Theorem 4.8.1 *A simple composite surface X comprising of n conforming triangular surface patches is homeomorphic to the corresponding heap Y of n planar approximating triangles which do not have any inappropriate intersection nor have any inappropriate gap.*

Proof. The proof is easily achieved by induction. Let the simple composite surface be X and the corresponding heap Y of approximating triangles without inappropriate intersections nor gaps be $X = \bigcup_{i=1}^n X_i$ and $Y = \bigcup_{i=1}^n Y_i$, where X_i and Y_i are the i^{th} triangular surface patch and the corresponding approximating triangle, respectively. To prove a map $f : X (= \bigcup_{i=1}^n X_i) \rightarrow Y (= \bigcup_{i=1}^n Y_i)$ is a homeomorphism, we first need to show that a triangular surface patch X_1 is homeomorphic to the corresponding approximating triangle Y_1 , namely a map $f_1 : X_1 \rightarrow Y_1$ is a homeomorphism. This is clearly true because any non-self-intersecting triangular surface is *homeomorphic to a planar triangle*. Assume now a map $f^* : X^* (= \bigcup_{i=1}^k X_i) \rightarrow Y^* (= \bigcup_{i=1}^k Y_i)$ is a homeomorphism for $1 < k < n$ and consider a map $f^{**} : X^{**} (= \bigcup_{i=1}^{k+1} X_i) \rightarrow$



(a) exact composite surface



(b) invalid approximation

Figure 4-35: A composite Bézier surface and its topologically inconsistent approximation with $\epsilon = 10^{-1}$ and $\tau_{AR} = 3$

$Y^{**} (= \bigcup_{i=1}^{k+1} Y_i)$. Since $X^{**} = X^* \cup X_{k+1}$ and $Y^{**} = Y^* \cup Y_{k+1}$, each subspace X^* , X_{k+1} are closed in X^{**} and Y^* , Y_{k+1} are also closed in Y^{**} . Furthermore, the map $f^* : X^* \rightarrow Y^*$ is homeomorphic by the assumption and so is a map $f_{k+1} : X_{k+1} \rightarrow Y_{k+1}$ as described in the homeomorphism of f_1 . Moreover, a possible $X^* \cap X_{k+1}$ is a *null set* $\mathbf{0}$, a *common vertex* V_k or a *common curved edge* C_k between X^* and X_{k+1} . Then, the corresponding possible $Y^* \cap Y_{k+1}$ is a *null set* $\mathbf{0}$, a *common vertex* V_k or a *common linear edge* L_k between Y^* and Y_{k+1} , respectively. This is because X^{**} is a *simple* composite surface composed of *conforming* triangular surface patches and the corresponding Y^{**} does not have any inappropriate intersection nor have any inappropriate gap. Therefore, a map $f_0 : X^* \cap X_{k+1} \rightarrow Y^* \cap Y_{k+1}$ is equivalent to $\mathbf{0} \rightarrow \mathbf{0}$, $V_k \rightarrow V_k$ or $C_k \rightarrow L_k$, which is obviously homeomorphic. By Lemma 3.6.2, a map $f^{**} : X^{**} \rightarrow Y^{**}$ is accordingly a homeomorphism. Therefore, a map f is a *homeomorphism* from a simple composite surface X comprising of n conforming triangular surface patches to the corresponding heap Y of n planar approximating triangles which do not have inappropriate intersections nor gaps. \square

4.8.2 Preprocessing-Putting Triangles into Buckets

To perform an intersection test based on the bucketing technique, we first construct $\mathcal{O}(n)$ uniform buckets B_{ijk} which will contain n approximating triangles. The procedure of constructing buckets is analogous to what was described in Section 3.6.2.

Once the buckets are constructed, we associate each approximating triangle with the buckets. For each approximating triangle s_p in xyz -coordinate system, we perform a transformation with respect to the bucket coordinates ijk i.e.,

$$s_p \equiv u\mathbf{v}_p^1 + v\mathbf{v}_p^2 + w\mathbf{v}_p^3 \xrightarrow{T} u\tilde{\mathbf{v}}_p^1 + v\tilde{\mathbf{v}}_p^2 + w\tilde{\mathbf{v}}_p^3 \equiv \tilde{s}_p, \quad p = 1, 2, \dots, n, \quad (4.76)$$

where n is the number of approximating triangles and $0 \leq u, v, w \leq 1$, $u + v + w = 1$ and \mathbf{v}_p^1 , \mathbf{v}_p^2 , \mathbf{v}_p^3 are vertices of an approximating triangle s_p and the tilde $\tilde{}$ represents the corresponding transformed version. Each ijk -component of the transformed vertices $\tilde{\mathbf{v}}_p^1$, $\tilde{\mathbf{v}}_p^2$ and $\tilde{\mathbf{v}}_p^3$ is determined in an analogous way to Eq.'s (3.58) – (3.63).

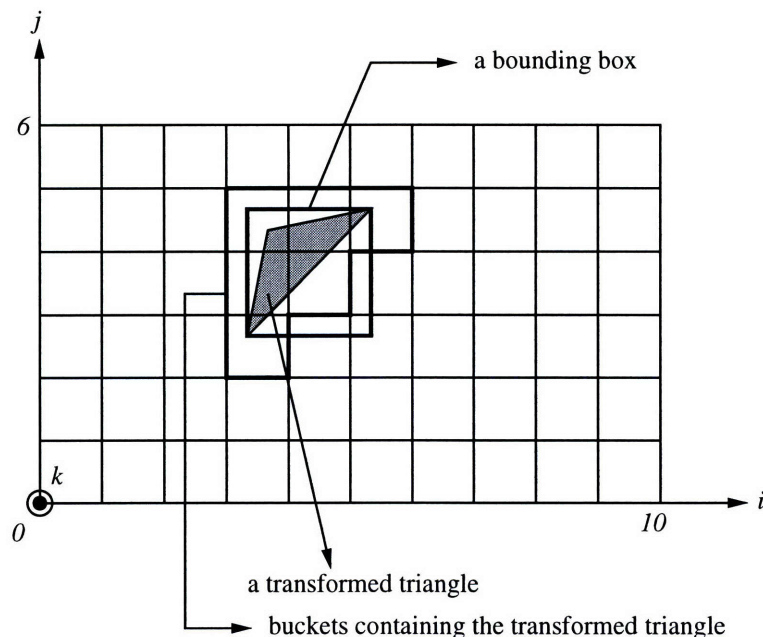


Figure 4-36: A $10 \times 6 \times 1$ bucket set (front view)

We now illustrate how to put each transformed triangle \tilde{s}_p into buckets B_{ijk} . As an example, we consider a $10 \times 6 \times 1$ bucket set together with an approximating triangle \tilde{s}_p , whose *front view* is shown in Figure 4-36. We first determine buckets which contain a *bounding box* of a triangle \tilde{s}_p . This is quickly achieved by just taking the *integer part* of each coordinate of the transformed vertices $\tilde{v}_p^1, \tilde{v}_p^2, \tilde{v}_p^3$ and finding min/max coordinate value for each ijk -direction. For a transformed triangle \tilde{s}_p and the bucket set in Figure 4-36, those nine *candidate* buckets B_{ijk} , ($i = 3, 4, 5, j = 2, 3, 4, k = 0$) are thus determined. In case the bounding box is completely within a bucket, we skip the following further considerations. We next remove the *false* candidate buckets which are not really associated with \tilde{s}_p e.g., B_{420}, B_{520} and B_{530} in Figure 4-36. An efficient and robust method to test for intersection between a plane and a bucket is given by Ratschek and Rokne [77]. Based on their method, our current problem can be stated and *partly* solved as follows.

Given : three vertices $\tilde{v}_p^1(v_{1i}, v_{1j}, v_{1k}), \tilde{v}_p^2(v_{2i}, v_{2j}, v_{2k}), \tilde{v}_p^3(v_{3i}, v_{3j}, v_{3k})$ of a transformed triangle \tilde{s}_p and a bucket $B_{ijk}([i, i + 1], [j, j + 1], [k, k + 1])$.

Compute :

$$\mathbf{P} = \tilde{\mathbf{v}}_p^1 - \tilde{\mathbf{v}}_p^3, \quad \mathbf{Q} = \tilde{\mathbf{v}}_p^2 - \tilde{\mathbf{v}}_p^3, \quad \mathbf{R} = B_{ijk} - \tilde{\mathbf{v}}_p^3, \quad (4.77)$$

$$\det(\mathbf{P}, \mathbf{Q}, \mathbf{R}) = \begin{vmatrix} P_i & P_j & P_k \\ Q_i & Q_j & Q_k \\ R_i & R_j & R_k \end{vmatrix}, \quad (4.78)$$

where notice that each ijk -component R_i, R_j, R_k of \mathbf{R} is an *interval number* and hence, so is $\det(\mathbf{P}, \mathbf{Q}, \mathbf{R})$.

Solution : if $\det(\mathbf{P}, \mathbf{Q}, \mathbf{R})$ in Eq. (4.78) does not contain *zero* in its *interval*, then we decide the bucket B_{ijk} and the transformed triangle $\tilde{\mathbf{s}}_p$ do not intersect each other i.e., $B_{ijk} \cap \tilde{\mathbf{s}}_p = \mathbf{0}$ and thus, we do not associate the corresponding candidate bucket B_{ijk} with $\tilde{\mathbf{s}}_p$.

However, the above procedure does not completely remove the false candidate buckets. This happens if a point in a false candidate bucket B_{ijk} exists on the *infinite* plane defined by three vertices of $\tilde{\mathbf{s}}_p$ e.g., in case of the transformed triangle $\tilde{\mathbf{s}}_p$ in Figure 4-36 being *parallel* to ij -plane. Such a false candidate bucket B_{ijk} can be further detected by *naively* checking if B_{ijk} and $\tilde{\mathbf{s}}_p$ intersect each other i.e., checking if at least one of 3 edges of $\tilde{\mathbf{s}}_p$ intersects at least one of 6 faces of B_{ijk} or if at least one of 12 edges of B_{ijk} intersects $\tilde{\mathbf{s}}_p$. In other words, to find and remove a false bucket out of currently remaining candidate buckets, we need to solve the problem of *bounded face to line segment intersection* 30 times ($= 3 \text{ edges} \times 6 \text{ faces} + 12 \text{ edges} \times 1 \text{ face}$) in the *worst* case – the case of a false candidate bucket, otherwise the intersection is detected in a small number of tests. In most practical examples, the number of buckets associated with a triangle is really $\mathcal{O}(1)$ and also, *very few* false candidate buckets exist in general.

By repeating the procedure described so far for each transformed triangle $\tilde{\mathbf{s}}_p$ ($p = 1, 2, \dots, n$), we can associate every $\tilde{\mathbf{s}}_p$ with the corresponding buckets B_{ijk} .

4.8.3 Intersection Check and Final Triangulation

For each bucket B_{ijk} , an intersection test is performed to identify possible inappropriate intersections between the approximating triangles. The intersection test begins with inquiring if at least two triangles are associated with B_{ijk} . If it is true, an intersection check between each pair of transformed triangles in B_{ijk} is performed. If a pair of transformed approximating triangles $\tilde{\mathbf{s}}_p$ and $\tilde{\mathbf{s}}_q$ inappropriately intersect each other, further local refinement of the approximation is necessitated. Such an intersection is detected by checking if at least one edge $\tilde{\mathbf{l}}_p^i(t)$ of $\tilde{\mathbf{s}}_p$ intersects $\tilde{\mathbf{s}}_q(u, v)$ or if at least one edge $\tilde{\mathbf{l}}_q^i(t)$ of $\tilde{\mathbf{s}}_q$ intersects $\tilde{\mathbf{s}}_p(u, v)$ at $t = t_0$, $(u, v) = (u_0, v_0)$ i.e.,

$$\tilde{\mathbf{l}}_p^i(t_0) \equiv (1 - t_0)\tilde{\mathbf{b}}_p^i + t_0\tilde{\mathbf{e}}_p^i = u_0\tilde{\mathbf{v}}_q^1 + v_0\tilde{\mathbf{v}}_q^2 + (1 - u_0 - v_0)\tilde{\mathbf{v}}_q^3 \equiv \tilde{\mathbf{s}}_q(u_0, v_0), \quad (4.79)$$

or

$$\tilde{\mathbf{l}}_q^i(t_0) \equiv (1 - t_0)\tilde{\mathbf{b}}_q^i + t_0\tilde{\mathbf{e}}_q^i = u_0\tilde{\mathbf{v}}_p^1 + v_0\tilde{\mathbf{v}}_p^2 + (1 - u_0 - v_0)\tilde{\mathbf{v}}_p^3 \equiv \tilde{\mathbf{s}}_p(u_0, v_0), \quad (4.80)$$

where $0 \leq t_0, u_0, v_0 \leq 1$, $i = 1, 2$, or 3 and furthermore, $\tilde{\mathbf{b}}_p^i$, $\tilde{\mathbf{e}}_p^i$, $\tilde{\mathbf{v}}_p^1$, $\tilde{\mathbf{v}}_p^2$ and $\tilde{\mathbf{v}}_p^3$ are the beginning and ending points of the edge $\tilde{\mathbf{l}}_p^i$ and three vertices of the triangle $\tilde{\mathbf{s}}_p$, respectively. Similarly, $\tilde{\mathbf{b}}_q^i$, $\tilde{\mathbf{e}}_q^i$, $\tilde{\mathbf{v}}_q^1$, $\tilde{\mathbf{v}}_q^2$ and $\tilde{\mathbf{v}}_q^3$ are defined.

If a pair of intersecting triangles $\tilde{\mathbf{s}}_p$ and $\tilde{\mathbf{s}}_q$ are found, we pick up the corresponding triangles T_p and T_q in the triangulation domains. Additional nodes are inserted on the midpoint of the *longest* edge of each T_p , T_q and a local refinement of the approximation is performed by bisecting T_p and T_q together with their neighboring triangles. Note again that if the longest edge is a boundary loop segment, we update the triangulation as described in Figure 4-25.

The bucketing procedure described in Section 4.8.2 and the intersection test are then repeated until every approximating triangle does not inappropriately intersect each other. This procedure, called *final triangulation*, provides a *topologically consistent* approximation to the exact input composite surface, as shown in Figure 4-37 corresponding to the example shown in Figure 4-35.

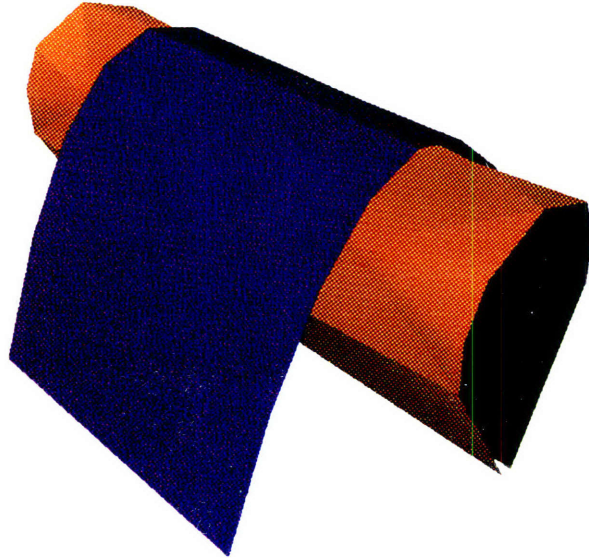


Figure 4-37: A topologically consistent approximation to the surface of Figure 4-35-(a)

4.9 Approximation of Interval Bézier Surfaces

A solid object bordered by free-form surface patches is generally described with a boundary representation (B-rep). The topological information defining how the various patches are glued together is described in an adjacency graph. Assembling a collection of surface patches of limited precision frequently yields *gaps* in the boundary parts. These gaps will yield ambiguities in the definition of point sets contained in the interior of the solid. Therefore, we may get vaguely defined solids which produce more vaguely defined objects when they are used in Boolean operations, involving intersections of solid volumes.

To achieve robustness in B-rep solid modelers, *interval* geometric entities (points, curves, surfaces, etc) computed in *rounded interval arithmetic* (RIA) should be introduced, as detailed in Hu *et al.* [45, 46, 47, 48, 49]. The interval B-rep prevents the inappropriate gaps mentioned above by the robust interval geometric definitions and computations together with consistent incidence criteria of points on the neighboring

geometric entities, [47, 48].

Interval Bézier surfaces differ from classical Bézier surfaces in that the real numbers representing control point coordinates are replaced by *intervals*. This implies that, in three-dimensional space, interval Bézier surfaces represent *thin shells*, if the intervals are chosen sufficiently small. Approximating triangles to such interval Bézier surfaces consequently have the form of *thin triangular plates*. The interval approximating triangles can be still described by the conventional vector formula,

$$\mathbf{s}(u, v, w) \equiv u\mathbf{v}_1 + v\mathbf{v}_2 + w\mathbf{v}_3, \quad (4.81)$$

where $u + v + w = 1$, $0 \leq u, v, w \leq 1$ and \mathbf{v}_1 , \mathbf{v}_2 and \mathbf{v}_3 are the vertices of $\mathbf{s}(u, v, w)$ represented by *interval* points in E^3 , i.e., $\mathbf{v}_i \equiv ([x_{il}, x_{iu}], [y_{il}, y_{iu}], [z_{il}, z_{iu}])$, $i = 1, 2, 3$.

Given a set of trimmed interval Bézier surfaces, the interval boundary loop approximation discussed in Section 3.7 together with identification of κ_{rms} stationary points using the interval projected polyhedron algorithm is performed to obtain an *initial triangulation* based on the interval Delaunay test. Although there is no special motivation to minimize the isometric mapping error function in RIA, the intersection test between each edge of the developed surface net should be robustly performed in RIA to guarantee the existence of a homeomorphism between the uv -parametric space and the corresponding triangulation domain. We also need to note that the mapping of mesh from the triangulation domain into the three-dimensional space results in a set of interval triangular elements, $\mathbf{s}(u, v, w)$ described in Eq. (4.81).

For each initial approximating triangle, an approximation error $\delta \equiv [\delta_l, \delta_u]$ defined in Eq. (4.55) is computed. In order to *conservatively* satisfy the prescribed tolerance ϵ , we choose δ_u as an approximation error. Repeated node insertion followed by Bowyer-Watson algorithm is performed until every interval approximating triangle satisfies the prescribed tolerance ϵ .

In AR-improving triangulation, RIA is not necessarily required to compute the aspect ratio (AR) of triangles in the triangulation domain since AR defined in Eq. (4.56) does not cause any numerical nor conceptual failure unless a triangle is degen-

erate. As discussed in Section 4.6.4, the proposed interval Delaunay test guarantees non-degenerate triangulation in the course of Bowyer-Watson algorithm. Furthermore, a conservative measure of AR does not provide any special meaningfulness for a triangle in the triangulation domain since it is not a real AR of the corresponding triangle in the three-dimensional space.

A robust intersection test operating in RIA coupled with an efficient bucketing technique is next performed to identify inappropriate intersections between interval approximating triangles. As discussed in Section 3.7.3, an *interval* intersection test should be employed to solve the plane to plane intersection problem defined in Eq.'s (4.79) and (4.80) robustly. If such intersections exist, as described in Section 4.8.3, further local refinement of the approximation is performed until every interval approximating triangle does not involve such an inappropriate intersection.

Finally, we note again that the mapping of each approximating triangle from a triangulation domain into the three-dimensional space should be performed in RIA to prevent gaps between the approximating triangles whose common edges correspond to the boundary loop segments of different input surface patches.

4.10 Robustness Issues

In this Section, we summarize *robustness issues* of our surface tessellation algorithm.

- Operations in FPA may :
 - not find *all* the significant points,
 - incorrectly decide $\delta < \epsilon$,
 - result in incorrect Delaunay triangulation,
 - not correctly map a node from the triangulation domain onto the parametric space,
 - fail to identify inappropriate intersections of the approximating elements or developed net edges.

- Operations in RIA will :
 - not miss significant points due to the use of the *Interval Projected Polyhedron Algorithm*,
 - guarantee $\delta < \epsilon \longrightarrow \delta_u < \epsilon$ where $\delta \equiv [\delta_l, \delta_u]$,
 - guarantee correct Delaunay triangulation,
 - not fail in the mapping process,
 - not miss inappropriate intersections of the approximating elements or developed net edges.

4.11 Complexity Analysis

In this Section, we perform complexity analysis for each surface approximation procedure, i.e., initial, δ -improving, AR-improving and final triangulation.

4.11.1 Initial Triangulation

In the initial triangulation procedures, time cost is dominated by the three steps, namely topologically reliable boundary loop approximation, determination of triangulation domains and the computation of κ_{rms} stationary points. For simplicity, *we assume that each input surface patch is of the same degrees bi- m -ic in the uv -parametric directions.*

Complexity analysis for the boundary loop approximation has already been discussed in Section 3.8. The only thing we need to note is that a trimming loop segment has a degree $2mn$ if it is defined with a polynomial curve of degree n on the uv -parametric space of a bi- m -ic input surface patch. Therefore, we substitute $2mn$, as the degree of an input curve, into each complexity formula in Section 3.8.

We next analyze the time complexity of the determination of triangulation domains i.e., the minimization of the mapping error function E defined in Eq. (4.44). *We further assume here that each surface net has the same number $p \times p$ of distributed*

points in the uv -parametric directions, respectively. The number p is arbitrary, but we usually set $p = m$. The minimization of E requires N_{var} times evaluation of E for each iteration, where $N_{var} (\equiv 2p^2)$ is the number of independent variables of E . Furthermore, one evaluation of E takes $\mathcal{O}(N_{var})$ i.e., $\mathcal{O}(p^2)$ cost. Suppose that N_s is the total number of input surfaces and N_{iter}^{iso} is the average number of iterations to achieve the minimum of E for each input surface, then the construction of triangulation domains costs $\mathcal{O}(N_s N_{iter}^{iso} p^4)$ on the average.

We now consider the time cost required in the computation of κ_{rms} stationary points. Given a 2×2 system of nonlinear polynomial equations of degree l in each variable, the projected polyhedron algorithm takes $\mathcal{O}(l^3)$ for each iteration [85] and therefore, solving the system of polynomial equations (4.38) and (4.39) costs $\mathcal{O}(N_{iter}^{poly} m^3)$, where N_{iter}^{poly} is the number of iterations to get the solution and m is the degree of the bi- m -ic surface patch. Assume that N_{iter}^{poly} is the average number of iterations for all input surfaces, then the computation of κ_{rms} stationary points takes $\mathcal{O}(N_s N_{iter}^{poly} m^3)$ on the average.

Based on the analysis described so far, the average time complexity of the initial triangulation is described by $\mathcal{O}(C_{bound} + N_s (N_{iter}^{iso} p^4 + N_{iter}^{poly} m^3))$, where C_{bound} is the time cost spent in the boundary loop approximation, given in Section 3.8.

4.11.2 δ -Improving Triangulation

For each input tensor product Bézier surface patch, in order to achieve N_T^ϵ approximating triangles satisfying the prescribed tolerance ϵ , we need to perform extraction of a triangular sub-Bézier patch $N_T^\delta (\geq N_T^\epsilon)$ times to measure the approximation error δ . Notice that N_T^δ includes not only N_T^ϵ but also the number of all the deleted triangles during each application of Bowyer-Watson algorithm. By Eq. (4.32), extraction of one triangular sub-Bézier patch from a bi- m -ic tensor product Bézier surface patch runs in $\mathcal{O}(m^6)$ and the approximation error δ is then computed in just $\mathcal{O}(m^2)$. If we assume the above N_T^δ to be the average number of all input surface patches \mathbf{r}_i ($i = 1, 2, \dots, N_s$), then total time cost spent in measuring δ is described by $\mathcal{O}(N_s N_T^\delta m^6)$ on the average.

The δ -improving algorithm chooses the position of the $(N+1)^{th}$ mesh node on the basis of the mesh connecting the already existing N nodes. The position is chosen to be the Voronoi vertex of a triangle with the largest approximation error δ and therefore, an efficient *heap list* [19] is used to store pointers to the approximating triangles ordered according to the value of δ . Any removal or insertion of a new element in the heap list can be achieved in $\mathcal{O}(\log N)$ operations and furthermore, all other triangles to be deleted at each node insertion can be found by means of $\mathcal{O}(1)$ search looking at only neighboring triangles. Therefore, the incremental node insertion algorithm can be achieved within $\mathcal{O}(N_P^\epsilon \log N_P^\epsilon)$, where N_P^ϵ is the total number of nodes on a triangulation domain after δ -improving triangulation. If we assume each triangulation domain has the number of final nodes N_P^ϵ on the average, then the total time cost associated with the current consideration is $\mathcal{O}(N_s N_P^\epsilon \log N_P^\epsilon)$ on the average, where N_s is the total number of input surface patches. By the above two kinds of complexity analyses, we can conclude that the δ -improving triangulation runs in $\mathcal{O}(N_s(N_T^\delta m^6 + N_P^\epsilon \log N_P^\epsilon))$ on the average. In fact, N_P^ϵ depends on the approximation tolerance ϵ and its relation is described by $N_P^\epsilon \sim \frac{1}{\epsilon}$ [32].

4.11.3 AR-Improving Triangulation

Time cost of AR-improving triangulation is dominated by the repetition of choosing a triangle with the largest AR in the heap list followed by a Bowyer-Watson algorithm. As we discussed in the previous Section, this procedure runs in $\mathcal{O}(N_s(N_P^{AR} \log N_P^{AR}))$ on the average, where N_s is the total number of input surface patches and N_P^{AR} is the average number of nodes inserted in AR-improving triangulation for all input surface patches.

4.11.4 Intersection Test and Final Triangulation

Suppose that AR-improving triangulation results in N_T^{AR} number of approximating triangles, then we construct corresponding buckets in $\mathcal{O}(N_T^{AR})$. Since $\mathcal{O}(1)$ approximating triangles are associated with each bucket, time complexity for the intersec-

tion test is also $\mathcal{O}(N_T^{AR})$. In case that an inappropriate intersection between a pair of approximating triangles is detected, we bisect such triangles together with their neighboring triangles (if they exist) - the number of triangles is increased by 4 at most - and immediately update the buckets for the further intersection test.

Therefore, the total complexity is described as $\mathcal{O}(f)$ where

$$f = \sum_{i=0}^{N_{inter}} (N_T^{AR} + 4i) = (N_{inter} + 1)N_T^{AR} + 2(N_{inter} + 1)(N_{inter} + 2), \quad (4.82)$$

where N_{inter} is the total number of pairs of intersecting triangles detected during the test. By taking the leading terms in Eq. (4.82), the total complexity will be $\mathcal{O}(N_{inter}N_T^{AR} + N_{inter}^2)$ for $N_{inter} > 0$ and $\mathcal{O}(N_T^{AR})$ in case $N_{inter} = 0$.

4.12 Examples

We have implemented the proposed surface tessellation method with a bi-cubic *wave-like* Bézier surface patch, a *high degree* Bézier surface patch and four *realistic composite* Bézier surfaces on a graphics workstation running at 150 MHz. Notations used in Tables 4.2 – 4.20 are summarized as follows.

- ϵ : User specified approximation tolerance (normalized by the patch length scale defined as the cubic root of the volume of the rectangular bounding box of the patch in the given coordinate system and using the control vertices).
- *Init*- Δ : Initial triangulation including construction of triangulation domains, boundary loop approximation and computation of κ_{rms} stationary points – see also Sections 4.7.1 and 4.7.2.
- δ - Δ : δ -improving triangulation described in Section 4.7.3.
- AR- Δ : AR-improving triangulation described in Section 4.7.4.
- *Fin*- Δ : Intersection test of the approximating triangles described in Section 4.8.

- **Total** : Total procedure i.e., *Init*- Δ , δ - Δ , AR- Δ and *Fin*- Δ .
- N : Total number of approximating triangles.
- τ_{AR} : User specified AR-threshold.
- N_{fail} : Number of the approximating triangles with $\text{AR} > \tau_{\text{AR}}$.
- AR_{avg} : Average AR of N approximating triangles.
- AR_{max} : The maximum AR of N approximating triangles.
- N_{δ} : Number of triangles generated up to δ - Δ .
- N_{AR} : Number of triangles generated in AR- Δ process.

4.12.1 Wave-Like Surface

Our first example is a bi-cubic Bézier surface patch [57]. The boundary 12 control points are coplanar so that the boundary curves form a square. The remaining four interior control points make the surface patch wave-like. The control points are given as follows:

$$\begin{pmatrix} \mathbf{P}_{00} & \mathbf{P}_{01} & \mathbf{P}_{02} & \mathbf{P}_{03} \\ \mathbf{P}_{10} & \mathbf{P}_{11} & \mathbf{P}_{12} & \mathbf{P}_{13} \\ \mathbf{P}_{20} & \mathbf{P}_{21} & \mathbf{P}_{22} & \mathbf{P}_{23} \\ \mathbf{P}_{30} & \mathbf{P}_{31} & \mathbf{P}_{32} & \mathbf{P}_{33} \end{pmatrix} = \begin{pmatrix} (0, 0, 0) & (0, \frac{1}{3}, 0) & (0, \frac{2}{3}, 0) & (0, 1, 0) \\ (\frac{1}{3}, 0, 0) & (\frac{1}{3}, \frac{1}{3}, 1) & (\frac{1}{3}, \frac{2}{3}, 1) & (\frac{1}{3}, 1, 0) \\ (\frac{2}{3}, 0, 0) & (\frac{2}{3}, \frac{1}{3}, -1) & (\frac{2}{3}, \frac{2}{3}, -1) & (\frac{2}{3}, 1, 0) \\ (1, 0, 0) & (1, \frac{1}{3}, 0) & (1, \frac{2}{3}, 0) & (1, 1, 0) \end{pmatrix}.$$

The surface is anti-symmetric with respect to $u = 0.5$. Although the surface looks simple, it is rich in its variety of differential geometric properties [57]. Tables 4.2 – 4.5 together with Figures 4-38 – 4-48 illustrate the performance for the wave-like surface.

There are 11 stationary points of the root mean square curvature κ_{rms} – 2 local maxima at $(u, v) = (0.805, 0.5)$, $(0.195, 0.5)$, 5 saddle points at $(u, v) = (0.667, 0.838)$, $(0.667, 0.162)$, $(0.333, 0.838)$, $(0.5, 0.5)$, $(0.333, 0.162)$ and 4 local minima at $(u, v) = (0.893, 0.822)$, $(0.893, 0.178)$, $(0.107, 0.822)$, $(0.107, 0.178)$. Our algorithm – *Case 1* in Table 4.2 – uses those κ_{rms} stationary points as well as nodes on the boundary loops as initial nodes. We also implemented the *initial* and δ -*improving* triangulations

without those κ_{rms} stationary points (*Case 2* in Table 4.2), and furthermore, we used only points of κ_{rms} *local maximum* as initial interior nodes (*Case 3* in Table 4.2). Obviously, *Case 1* and *Case 3* are more expensive than *Case 2* in the *initial* triangulation process, since they require solving a system of high degree nonlinear polynomial equations. The difference in CPU-time of initial triangulation between *Case 1* and *Case 3* mainly results from the time cost spent in classifying the *types* of κ_{rms} stationary points i.e., local maxima, saddle points or local minima. As far as space cost or conciseness of the approximation i.e., number of triangles is concerned, *Case 3* shows the most optimal result for a *loose* approximation tolerance such as 10^{-1} , which is not a surprising result, since *Case 3* starts δ -improving triangulation with those nodes at points of κ_{rms} local maxima. For tighter approximation tolerances such as 10^{-2} , 10^{-3} , our algorithm (*Case 1*) shows only slightly better results in terms of the space cost.

Table 4.3 and Figures 4-39, 4-42 – 4-46 show the result of meshing corresponding to the approximation tolerance $\epsilon = 10^{-1}, 10^{-2}, 10^{-3}$ and given AR-threshold $\tau_{AR} = 4$ including computation of *all* the stationary points of κ_{rms} . If the approximation tolerance ϵ is not sufficiently tight, most of time cost is spent on the initial triangulation which includes construction of triangulation domains, boundary loop approximation and computation of κ_{rms} stationary points. Time cost C_{init} of the initial triangulation is not sensitive to the change of ϵ since the dominant procedures of C_{init} are the construction of triangulation domains and computation of κ_{rms} stationary points, which are not affected by ϵ . In general, as ϵ becomes tighter, the time cost of δ -improving triangulation becomes comparable and dominant with respect to C_{init} . The other procedures are relatively quick compared with the initial and δ -improving triangulations.

Table 4.3 also shows the AR of the approximating triangles after AR-improving triangulation. For a given AR-threshold $\tau_{AR} = 4$, the number N_{fail} of approximating triangles with $AR > \tau_{AR}$ is not more than than 1% of total number N of the approximating triangles for each ϵ . The N_{fail} triangles are regarded as not improvable, in other words, those triangles meet a certain condition described in Eq. (4.70) which

terminates the AR-improving process for such triangles. It is worthwhile to mention again that the satisfactory *average* AR results from the coupled effects of the features of Delaunay based Voronoi vertex point insertion algorithm, AR-improving procedure in the planar domain of triangulation and preservation of triangles' shape during the mapping process from the triangulation domain into the three-dimensional space.

Table 4.4 shows the dependency of AR-improving procedure upon the AR-threshold τ_{AR} for a fixed approximation tolerance $\epsilon = 10^{-2}$. As τ_{AR} becomes more ideal, N_{fail} increases because, for a fixed ϵ , the value of τ_r described in Eq. (4.70) is invariant with respect to τ_{AR} . Even if a smaller τ_{AR} slightly reduces the average aspect ratio AR_{avg} , it does not tend to reduce AR_{max} of the approximating triangles. Other examples also have the similar tendency and for some examples, as τ_{AR} becomes more ideal, AR_{max} even increases. From this empirical result, we can deduce that an AR-improving procedure with too ideal a τ_{AR} possibly causes a triangle of very unsatisfactory AR_{max} , if the termination criterion value τ_r is fixed. Furthermore, if an input surface patch does not involve a high curvature region, then τ_r evaluated by Eq. (4.70) has quite a large value. This means that the termination condition becomes so strong that the algorithm easily stops improving AR of badly-shaped triangles with $r^2 > \tau_r$, where r is a circumcircle radius of such badly-shaped triangles. A lot of surface patches of our realistic engineering examples have such a curvature characteristic. *Further research is required on the termination condition of our AR-improving algorithm e.g., intelligently weakening the termination condition by introducing a weighting factor to Eq. (4.70) as a function of AR of currently chosen worst triangle and number of triangles, in order to balance the quality of triangles and the space cost.* Table 4.4 also shows how many triangles N_{AR} are generated in AR-improving procedures for $\tau_{AR} = 3, 4$ and 5 .

We have performed the triangulation in rounded interval arithmetic (RIA) as well as floating point arithmetic (FPA) – see Table 4.5. As far as the time cost is concerned, RIA is $\mathcal{O}(10)$ times more expensive than FPA. However, we need to keep in mind that extremely important computations such as *intersection test between the edges of an approximately developed surface net, mappings between the triangulation domain and*

the parametric space, Delaunay test and self-intersection test of the approximating triangles may result in system failure if the algorithm is operating purely in FPA, even if the input surface patches are *precisely* defined. Such failures have not been encountered in this example; however, we have experienced system failures in FPA execution during the algorithm development for example, when mapping a node from a triangulation domain onto the corresponding parametric space – this process is executed extremely many times e.g., when computing the approximation errors. Due to the imprecision of FPA, a node on the triangulation domain is mapped outside the corresponding parametric space, if the system of equations (4.48), (4.49) is ill-conditioned.

We also applied a uniform grid generation scheme based on the computation of the maximum allowable edge length of a triangle on the parametric space, using the maximum norms of the second partial derivatives of an input surface patch [32, 73, 84] – see also Eq. (4.72). Since D_1 , D_2 and D_3 in Eq.'s (4.72) – (4.75) are 18, 18 and 6, respectively, space cost for each $\epsilon = 10^{-1}$, 10^{-2} and 10^{-3} is approximately 20 times higher than our method – see also Table 4.3.

4.12.2 High Degree Surface

Our second example involves a Bézier surface patch of very *high degrees* – 9 by 9 in the u , v parametric directions. The testing surface patch was supplied by Prof. J. Hoschek of Technische Hochschule Darmstadt. The control points are given as follows:

$$\begin{pmatrix} \mathbf{M}_{00} & \mathbf{M}_{01} \\ \mathbf{M}_{10} & \mathbf{M}_{11} \end{pmatrix}$$

where

$$\mathbf{M}_{00} = \begin{pmatrix} (-0.1, 0, 0) & (-0.1, 3, 0) & (-0.1, 5.5, 0) & (-0.1, 7.5, 0) & (-0.1, 8.5, 0) \\ (-0.05, 0, 2) & (-0.05, 3, 2) & (-0.05, 5.5, 2) & (-0.05, 7.5, 2) & (-0.05, 8.5, 2) \\ (0, 0, 3.5) & (0, 3, 3.5) & (0, 5.5, 3.5) & (0, 7.5, 3.5) & (0, 8.5, 3.5) \\ (1, 0, 4.5) & (1, 3, 4.5) & (1, 5.5, 4.5) & (1, 7.5, 4.5) & (1, 8.5, 4.5) \\ (2.5, 0, 4.5) & (2.5, 3, 4.5) & (2.5, 5.5, 4.5) & (2.5, 7.5, 4.5) & (2.5, 8.5, 4.5) \end{pmatrix}$$

$$\mathbf{M}_{01} = \begin{pmatrix} (-0.1, 10, 0) & (-0.1, 11, 0) & (-0.1, 12, 0) & (-0.1, 13, 0) & (-0.1, 14, 0) \\ (-0.05, 10, 1.8) & (-0.05, 11, 1.5) & (-0.05, 13, 0.8) & (-0.05, 13, 0.4) & (-0.05, 15, 0) \\ (0, 10, 3) & (0, 11, 2.5) & (0, 13, 1.4) & (0, 14, 0.7) & (0, 15, 0) \\ (1, 10, 4) & (1, 11, 3) & (1, 13, 1.6) & (1, 14, 0.8) & (1, 15, 0) \\ (2.5, 10, 4) & (2.5, 11, 3) & (2.5, 13, 1.6) & (2.5, 14, 0.8) & (2.5, 15, 0) \end{pmatrix}$$

$$\mathbf{M}_{10} = \begin{pmatrix} (7.5, 0, 4.5) & (7.5, 3, 4.5) & (7.5, 5.5, 4.5) & (7.5, 7.5, 4.5) & (7.5, 8.5, 4.5) \\ (9, 0, 4.5) & (9, 3, 4.5) & (9, 5.5, 4.5) & (9, 7.5, 4.5) & (9, 8.5, 4.5) \\ (10, 0, 3.5) & (10, 3, 3.5) & (10, 5.5, 3.5) & (10, 7.5, 3.5) & (10, 8.5, 3.5) \\ (10, 0, 2) & (10, 3, 2) & (10, 5.5, 2) & (10, 7.5, 2) & (10, 8.5, 2) \\ (10, 0, 0) & (10, 3, 0) & (10, 5.5, 0) & (10, 7.5, 0) & (10, 8.5, 0) \end{pmatrix}$$

$$\mathbf{M}_{11} = \begin{pmatrix} (7.5, 10, 4) & (7.5, 11, 3) & (7.5, 13, 1.6) & (7.5, 14, 0.8) & (7.5, 15, 0) \\ (9, 10, 4) & (9, 11, 3) & (9, 13, 1.6) & (9, 14, 0.8) & (9, 15, 0) \\ (10, 10, 3) & (10, 11, 2.5) & (10, 13, 1.4) & (10, 14, 0.7) & (10, 15, 0) \\ (10, 10, 1.8) & (10, 11, 1.5) & (10, 13, 0.8) & (10, 13, 0.4) & (10, 15, 0) \\ (10, 10, 0) & (10, 11, 0) & (10, 12, 0) & (10, 13, 0) & (10, 14, 0) \end{pmatrix}$$

Tables 4.6 – 4.8 together with Figures 4-49 – 4-56 illustrate the performance for the algorithm in this example. The present example as well as all subsequent examples include the use of all the stationary points of κ_{rms} . Its global behavior is consistent with the previous example except that the time cost spent in the *initial* and δ -*improving* triangulations is relatively *high*. This is a natural result because the time cost of the initial and δ -improving procedures highly depends on the *degrees* of

input surface patches, as described in Sections 4.11.1 and 4.11.2 in detail.

We need to notice here that, for a degree 9 by 9 Bézier surface patch, the approximation error measure based on the polar forms is $2^{18}(= 262144)$ times more expensive than our method as described in Section 4.3.3. Therefore, the time cost of δ -improving procedure is increased by about 2^{18} times if the polar form based method is used. In other words, the δ -improving triangulation operating in floating point arithmetic (FPA) takes CPU-time on the order of *years* and *tens of years* for $\epsilon = 10^{-1}$, 10^{-2} and 10^{-3} while our method requires CPU-time on the order of *minutes* and *hours*, respectively. Time cost of the other procedures does not depend on the degrees of input surface patches and hence, it is very low compared with the initial and δ -improving triangulations.

4.12.3 Ship Hull

The next example is bottom and side panels of a ship hull which consists of 12 bi-quartic ($m = n = 4$) Bézier surface patches. Figure 4-57 shows the input composite surface. This example was provided by Mr. D. Danmeier, a graduate student of the Computational Hydrodynamics Group at the MIT Ocean Engineering Department.

We note that each input surface patch is noticeably stretched in the u -parametric direction. This geometric feature motivates constructing a $p \times 1$ surface net with $p = 7$ for *each* surface patch to achieve our triangulation domain. If a surface patch of degrees m, n has a length ratio in the uv -parametric directions not very different from $m : n$, then we usually construct an $m \times n$ surface net to determine its triangulation domain. This means we inevitably insert $2(m + n)$ nodes on the boundary of the input surface patch, if it is *not trimmed*. Therefore, to construct a $p \times 1$ surface net of a noticeably stretched surface patch, an appropriate value for p is $m + n - 1$, which is 7 for a bi-quartic surface patch.

Tables 4.9 – 4.11 together with Figures 4-57 – 4-67 illustrate the performance for the ship hull example. Global behavior of the results is quite similar to the previous examples.

Figures 4-61 and 4-62 illustrate the local adaptivity of the algorithm for a tight ϵ .

Figure 4-61 shows the *absolute* curvature distribution over the ship hull produced using the *Praxiteles* system [1]. Relatively very high curvature region is marked by a red color (dark color in this thesis printout). We can clearly see the dense triangulation near the high curvature region in Figures 4-60 and 4-62.

4.12.4 Airfoil

We next consider an airfoil composed of 220 bi-cubic Bézier surface patches – 200 sub-patches forming the upper and lower planform surfaces and 20 sub-patches near the trailing edge which connect the upper and lower surface patches. Prof. J. Peraire of MIT provided the airfoil surface patches. Figure 4-68 shows the input composite surface.

The ratio of length scales in the uv -parametric directions of each sub-patch forming the main upper and lower surfaces is similar to the ratio of the span and cord length of the foil. On the other hand, the sub-patches near the trailing edge are highly stretched in the u -parametric direction. We applied the same technique as in the case of ship hull example in the construction of triangulation domains for such noticeably stretched sub-patches.

Tables 4.12 – 4.14 and Figures 4-68 – 4-80 show the results for the performance of our algorithm. Comparing Tables 4.12 – 4.14 with Tables 4.9 – 4.11 respectively, we can see a similar behavior in this example to the previous ones, except that N is less sensitive to the approximation tolerance ϵ , if it does not become sufficiently tight e.g., $\epsilon = 10^{-3}$. This is essentially caused by the fact that the number of input sub-patches is large compared with the whole dimension of the foil and furthermore, most sub-patches are very smooth in terms of curvature distribution.

Figures 4-74 and 4-75 show the meshes near the trailing edge for $\epsilon = 10^{-2}$, 10^{-3} respectively. Those small triangles adjacent to the trailing edge in Figure 4-74 illustrates the procedure of node repositioning to the mid-point of the longest edge of a currently chosen worst boundary triangle in δ -improving (AR-improving) procedure, as described in Section 4.7.3 (4.7.4), respectively. When the node repositioning procedure reaches at some level, the algorithm can find an appropriate node inside the

triangulation domain and perform the usual δ -improving (AR-improving) procedure, as shown in Figure 4-75.

4.12.5 Utah Teapot

We have also implemented the method with a well-known example, the Utah teapot, which consists of 32 bi-cubic Bézier surface patches. This example was originally produced by Prof. M. Newell of University of Utah.

There are 4 main groups;

Group 1: body (12 sub-patches including 3 trimmed patches) and lip (4 sub-patches)

Group 2: knob (4 sub-patches) and lid (4 sub-patches)

Group 3: handle (4 sub-patches)

Group 4: spout (4 sub-patches)

The input geometry is shown in Figure 4-81.

We note that each group is completely defined in terms of the topological relation between each sub-patch in a group; however, **Group 1** and two other groups **Group 3**, **Group 4** do not form a complete solid model because the trimming curves of **Group 3** and **Group 4** corresponding to the three trimmed patches of **Group 1** are not explicitly available with the data – see also Figure 4-85. Furthermore, the lip and lid touch each other and they are not connected, which is natural, considering the utility of the lid; however, gaps between the lip and lid are unavoidable if the approximation tolerance is loose, as shown in Figure 4-86. For those reasons explained so far, we perform the final triangulation i.e., the test of possible inappropriate intersections for each group, separately.

Tables 4.15 – 4.17 together with Figures 4-81 – 4-90 show the results for the performance of our algorithm. Its global behavior is consistent with the other examples, described in the previous Sections.

4.12.6 Stem with a Bulbous Bow

Figure 4-91 shows a part of a ship hull with a bulbous bow, which is composed of 294 bi-cubic Bézier surface patches, originally produced by Dr. Steffan Schalck of Technical University of Denmark. The example has two main groups i.e., the part of a bulbous bow (196 sub-patches) and the neighboring hull (98 sub-patches).

Tables 4.18 – 4.20 and the corresponding Figures 4-91 – 4-99 illustrate the performance of our algorithm for this example. Since the number of input sub-patches is sufficiently large – especially, near the relatively high curvature region – with respect to the whole dimension of the input geometry, the number of approximating triangles N is not so sensitive to the approximation tolerance ϵ . Similar behavior was found when approximating the airfoil example. Other than that, the global behavior of the triangulation according to the changing of parameters follows what is explained in Section 4.12.1 in detail.

| ϵ | CPU (<i>sec</i>) | | | | | | N | | |
|------------|------------------------|---------------|---------------|---------------------|---------------|---------------|---------------|---------------|---------------|
| | <i>Init</i> - Δ | | | δ - Δ | | | | | |
| | <i>Case 1</i> | <i>Case 2</i> | <i>Case 3</i> | <i>Case 1</i> | <i>Case 2</i> | <i>Case 3</i> | <i>Case 1</i> | <i>Case 2</i> | <i>Case 3</i> |
| 10^{-1} | 20 | 2 | 26 | 0.7 | 1.1 | 1.4 | 32 | 26 | 21 |
| 10^{-2} | 34 | 8 | 41 | 11 | 14 | 15 | 229 | 254 | 254 |
| 10^{-3} | 38 | 12 | 48 | 204 | 222 | 223 | 2325 | 2364 | 2364 |

Table 4.2: Wave-like surface; dependency of time and space cost upon κ_{rms} stationary points and ϵ (see also Figures 4-38, 4-39 – 4-41)

| ϵ | CPU (<i>sec</i>) | | | | | N | $\frac{N_{fail}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|------------|------------------------|---------------------|--------------|-----------------------|--------------|------|-------------------------------------|-------------------|-------------------|
| | <i>Init</i> - Δ | δ - Δ | AR- Δ | <i>Fin</i> - Δ | Total | | | | |
| 10^{-1} | 20 | 0.7 | 0 | 0.05 | 20.75 | 32 | 0 | 1.8 | 2.4 |
| 10^{-2} | 34 | 11 | 0.33 | 0.73 | 46.06 | 239 | 1 | 2.0 | 4.8 |
| 10^{-3} | 38 | 204 | 7 | 10 | 259 | 2356 | 0.1 | 1.8 | 4.4 |

Table 4.3: Wave-like surface; $\tau_{AR} = 4$ (see also Figures 4-38, 4-39, 4-42 – 4-46)

| τ_{AR} | CPU (<i>sec</i>) | N_{AR} | $\frac{N_{fail}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|-------------|--------------------|----------|-------------------------------------|-------------------|-------------------|
| 3 | 0.58 | 21 | 2 | 1.9 | 4.9 |
| 4 | 0.33 | 10 | 1 | 2.0 | 4.8 |
| 5 | 0.14 | 4 | 0 | 2.1 | 4.8 |

Table 4.4: Wave-like surface; AR- Δ , $\epsilon = 10^{-2}$, $N_{\delta} = 229$ (see also Figures 4-42, 4-47, 4-48)

| CPU (<i>sec</i>) | | | | | | | | | | N | |
|------------------------|------|---------------------|-----|--------------|------|-----------------------|------|--------------|---------|-----|-----|
| <i>Init</i> - Δ | | δ - Δ | | AR- Δ | | <i>Fin</i> - Δ | | Total | | | |
| FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA |
| 34 | 1076 | 11 | 251 | 0.33 | 3.11 | 0.73 | 14.2 | 46.06 | 1344.31 | 239 | 242 |

Table 4.5: Wave-like surface; $\epsilon = 10^{-2}$, $\tau_{AR} = 4$

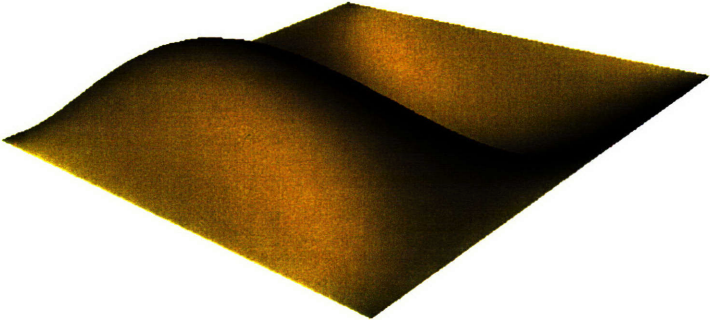


Figure 4-38: Wave-like surface; exact

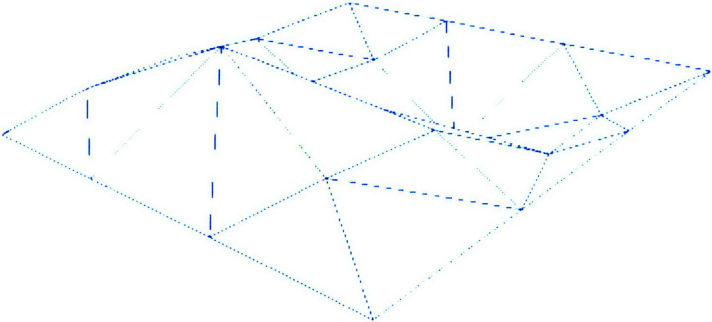


Figure 4-39: Wave-like surface; mesh with κ_{rms} stationary points, $\epsilon = 10^{-1}$ (see also Tables 4.2, 4.3)

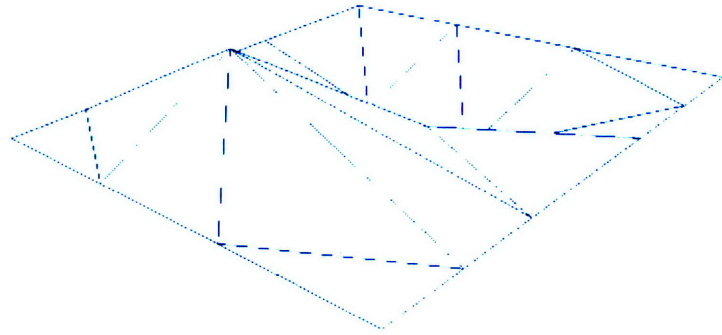


Figure 4-40: Wave-like surface; mesh with κ_{rms} local maximum points, $\epsilon = 10^{-1}$ (see also Table 4.2)

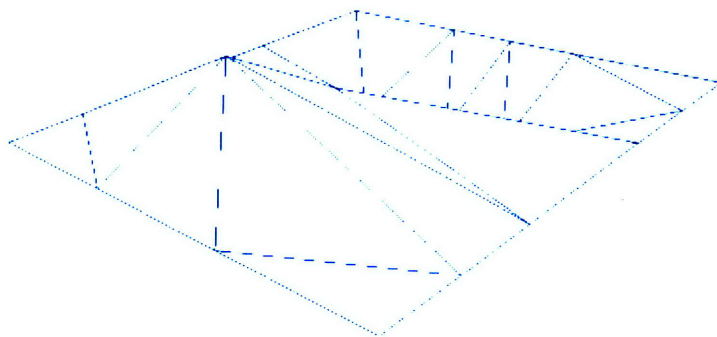


Figure 4-41: Wave-like surface; mesh without κ_{rms} stationary points, $\epsilon = 10^{-1}$ (see also Table 4.2)

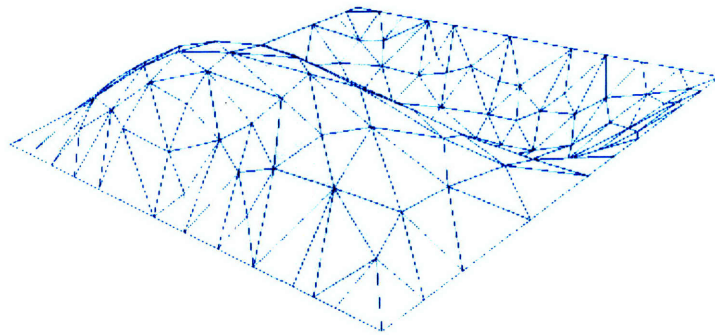


Figure 4-42: Wave-like surface; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.3, 4.4)

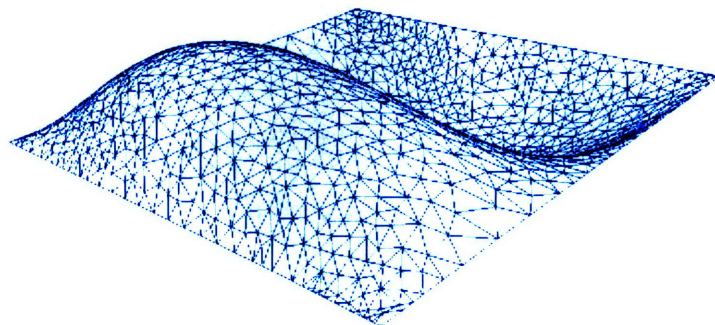


Figure 4-43: Wave-like surface; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.3)

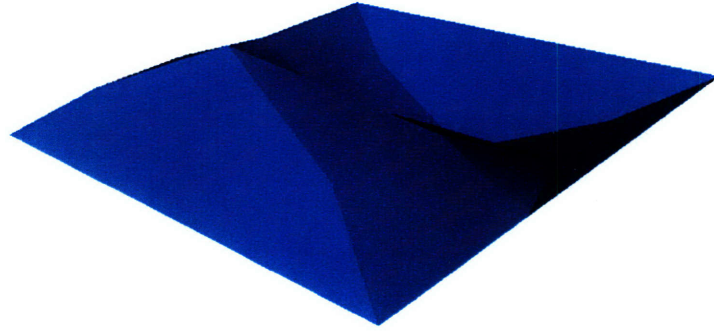


Figure 4-44: Wave-like surface; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.3)

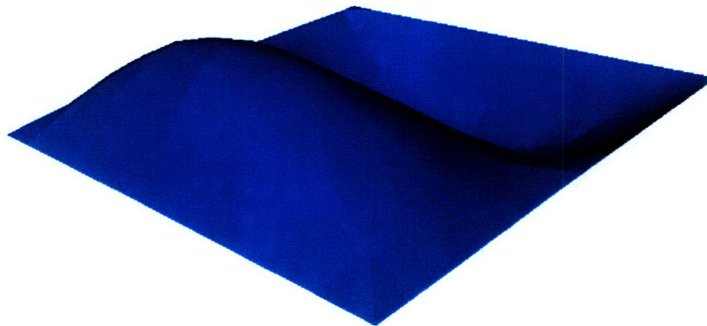


Figure 4-45: Wave-like surface; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.3)

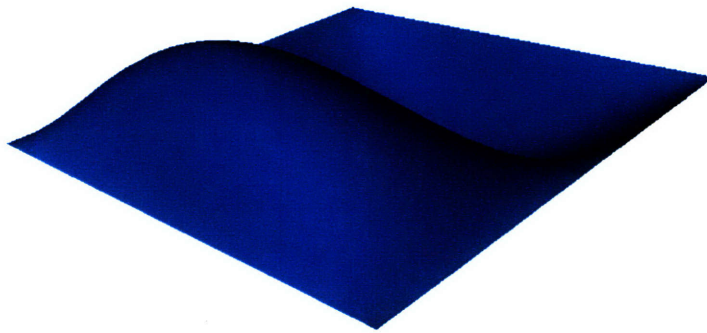


Figure 4-46: Wave-like surface; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.3)

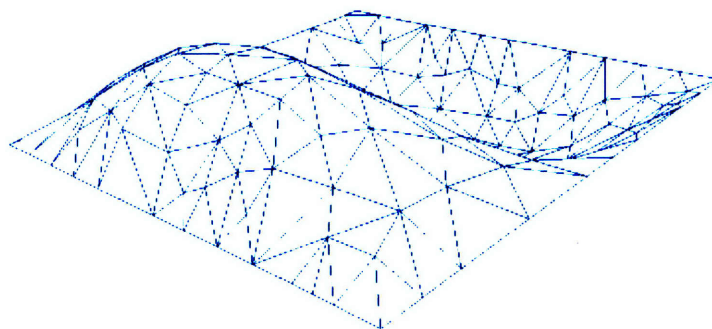


Figure 4-47: Wave-like surface; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.4)

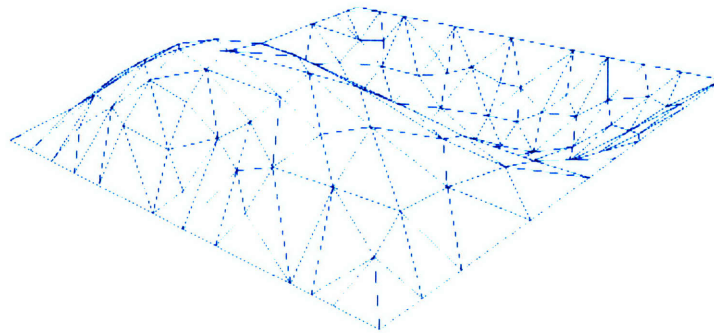


Figure 4-48: Wave-like surface; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 5$ (see also Table 4.4)

| ϵ | CPU (<i>sec</i>) | | | | | N | $\frac{N_{\text{fail}}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|------------|------------------------|---------------------|--------------|-----------------------|--------------|-----|--|-------------------|-------------------|
| | <i>Init</i> - Δ | δ - Δ | AR- Δ | <i>Fin</i> - Δ | Total | | | | |
| 10^{-1} | 1719 | 266 | 0.3 | 0.15 | 1985.45 | 69 | 1 | 2.3 | 4.1 |
| 10^{-2} | 1720 | 679 | 0.1 | 0.33 | 2399.43 | 132 | 0 | 1.9 | 3.5 |
| 10^{-3} | 1724 | 3188 | 1.3 | 1.61 | 4914.91 | 594 | 0 | 1.8 | 3.7 |

Table 4.6: High degree surface; $\tau_{\text{AR}} = 4$ (see also Figures 4-49 – 4-55)

| τ_{AR} | CPU (<i>sec</i>) | N_{AR} | $\frac{N_{\text{fail}}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|--------------------|--------------------|-----------------|--|-------------------|-------------------|
| 3 | 0.2 | 6 | 1 | 1.8 | 3.1 |
| 4 | 0.1 | 0 | 0 | 1.9 | 3.5 |

Table 4.7: High degree surface; AR- Δ , $\epsilon = 10^{-2}$, $N_{\delta} = 132$ (see also Figures 4-51, 4-56)

| CPU (<i>sec</i>) | | | | | | | | | | | |
|------------------------|-------|---------------------|------|--------------|-----|-----------------------|------|--------------|----------|-----|-----|
| <i>Init</i> - Δ | | δ - Δ | | AR- Δ | | <i>Fin</i> - Δ | | Total | | N | |
| FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA |
| 1720 | 53761 | 679 | 8689 | 0.1 | 1.2 | 0.33 | 5.48 | 2399.43 | 62456.68 | 132 | 133 |

Table 4.8: High degree surface; $\epsilon = 10^{-2}$, $\tau_{\text{AR}} = 4$

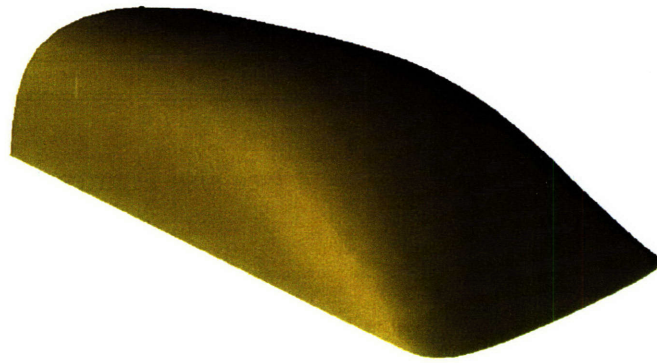


Figure 4-49: High degree surface; exact

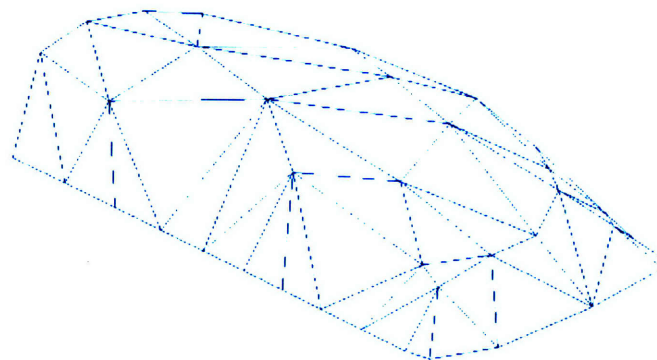


Figure 4-50: High degree surface; mesh with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.6)

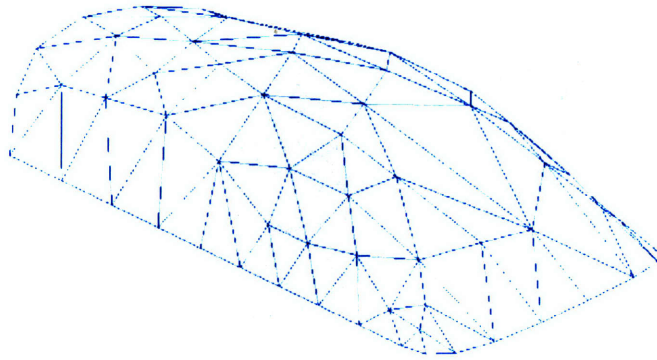


Figure 4-51: High degree surface; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.6, 4.7)

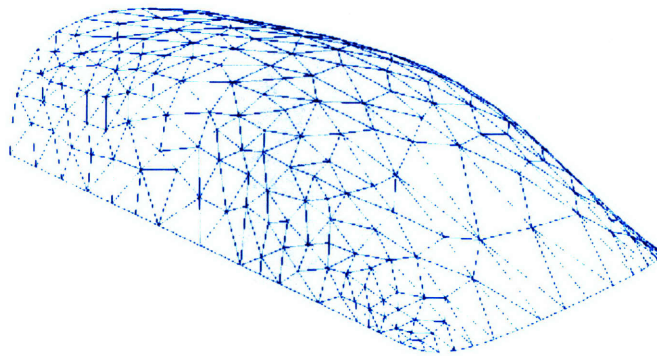


Figure 4-52: High degree surface; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.6)

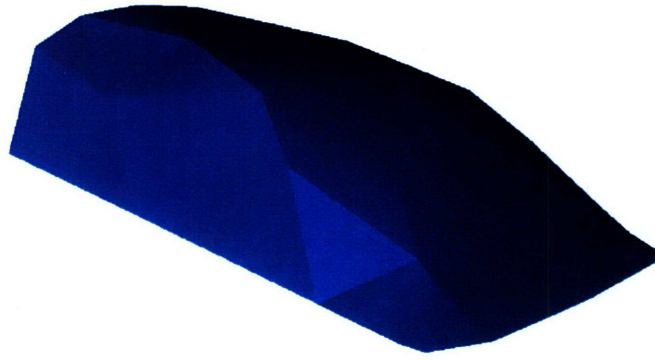


Figure 4-53: High degree surface; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.6)

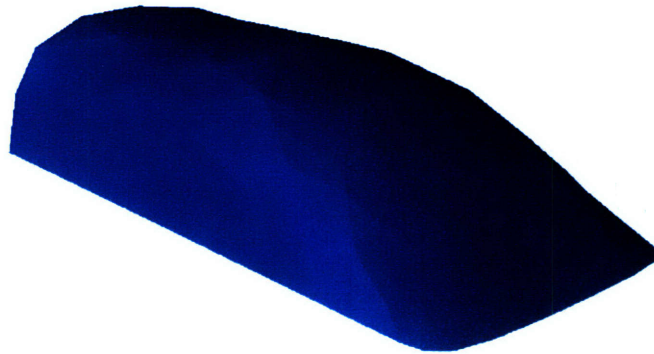


Figure 4-54: High degree surface; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.6)

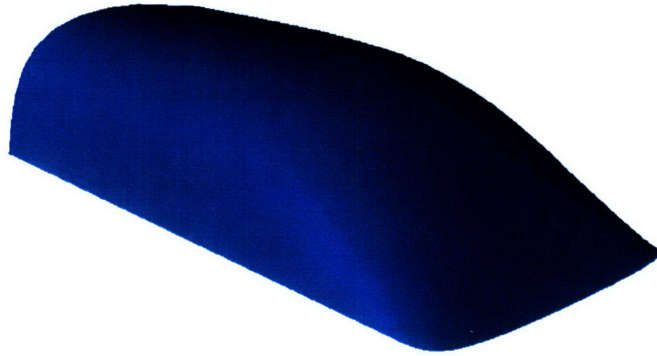


Figure 4-55: High degree surface; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.6)

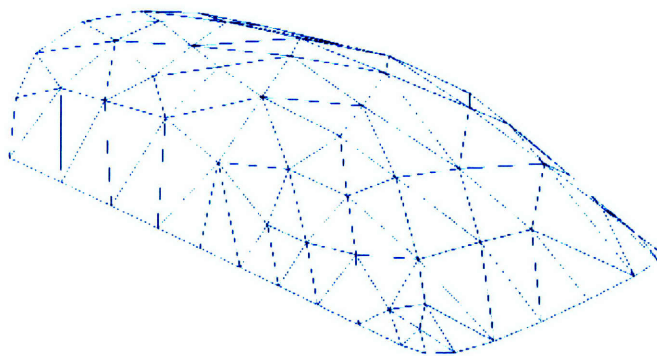


Figure 4-56: High degree surface; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.7)

| ϵ | CPU (sec) | | | | | N | $\frac{N_{\text{fail}}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|------------|----------------|---------------------|--------------|---------------|--------|------|--|-------------------|-------------------|
| | Init- Δ | δ - Δ | AR- Δ | Fin- Δ | Total | | | | |
| 10^{-1} | 210 | 31 | 0.13 | 0.23 | 241.36 | 188 | 1 | 2.2 | 4.4 |
| 10^{-2} | 211 | 52 | 0.39 | 0.87 | 264.26 | 433 | 1 | 2.0 | 6.8 |
| 10^{-3} | 216 | 518 | 38 | 11 | 783 | 4109 | 1 | 2.0 | 11.6 |

Table 4.9: Ship hull; $\tau_{\text{AR}} = 4$ (see also Figures 4-57 – 4-65)

| τ_{AR} | CPU (sec) | N_{AR} | $\frac{N_{\text{fail}}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|--------------------|-----------|-----------------|--|-------------------|-------------------|
| 3 | 2.65 | 250 | 6 | 1.9 | 7.1 |
| 4 | 0.39 | 43 | 1 | 2.0 | 6.8 |
| 5 | 0.13 | 10 | 0 | 2.1 | 5.0 |

Table 4.10: Ship hull; AR- Δ , $\epsilon = 10^{-2}$, $N_{\delta} = 390$ (see also Figures 4-59, 4-66, 4-67)

| CPU (sec) | | | | | | | | | | N | |
|----------------|------|---------------------|------|--------------|------|---------------|-----|--------|------|-----|-----|
| Init- Δ | | δ - Δ | | AR- Δ | | Fin- Δ | | Total | | | |
| FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA |
| 211 | 6595 | 52 | 1506 | 0.39 | 4.31 | 0.87 | 13 | 264.26 | 8118 | 433 | 431 |

Table 4.11: Ship hull; $\epsilon = 10^{-2}$, $\tau_{\text{AR}} = 4$

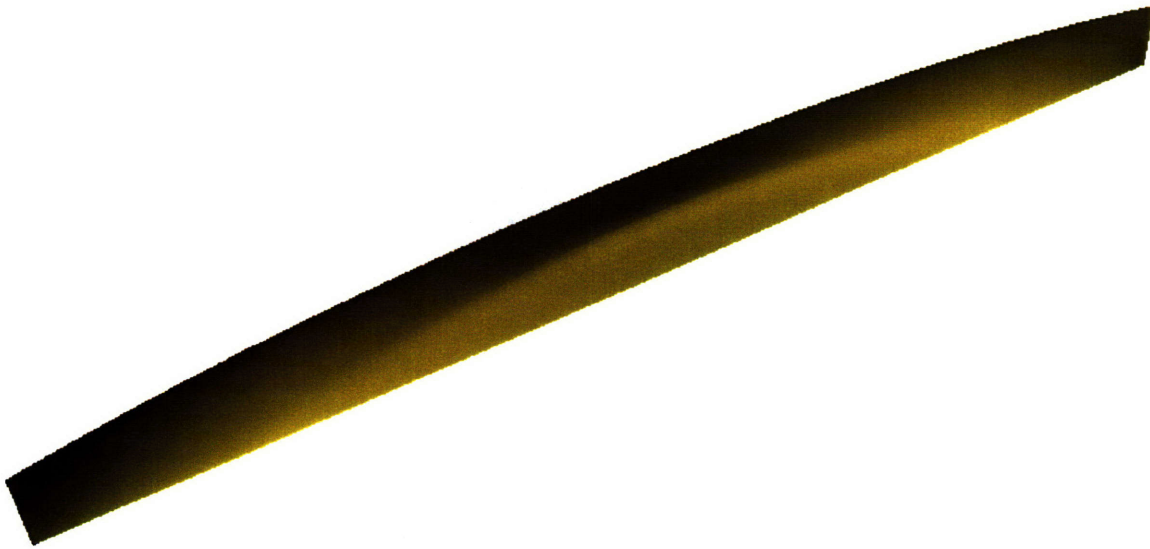


Figure 4-57: Ship hull; exact

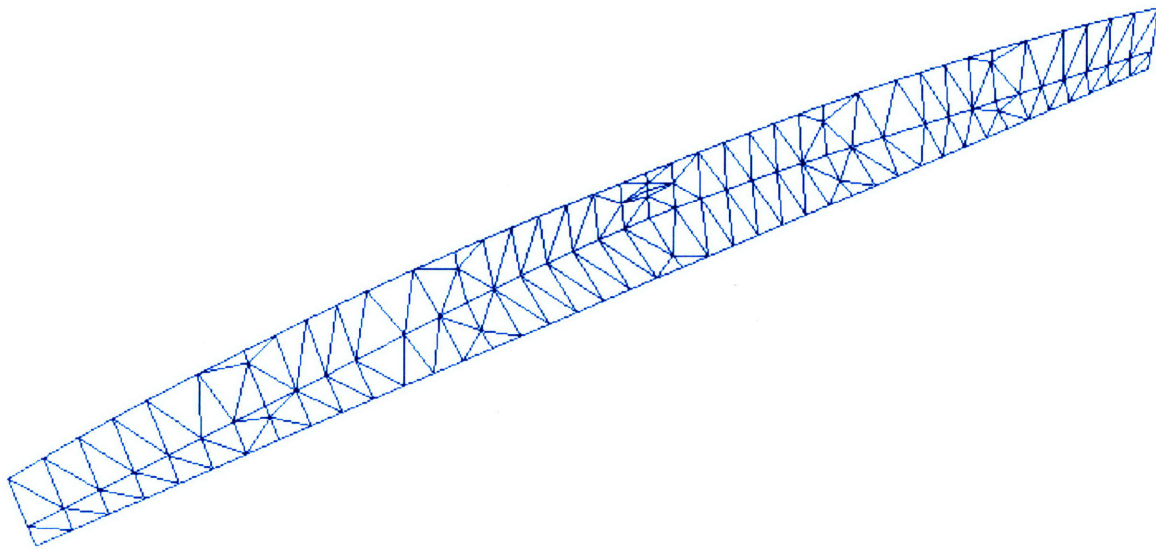


Figure 4-58: Ship hull; mesh with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.9)

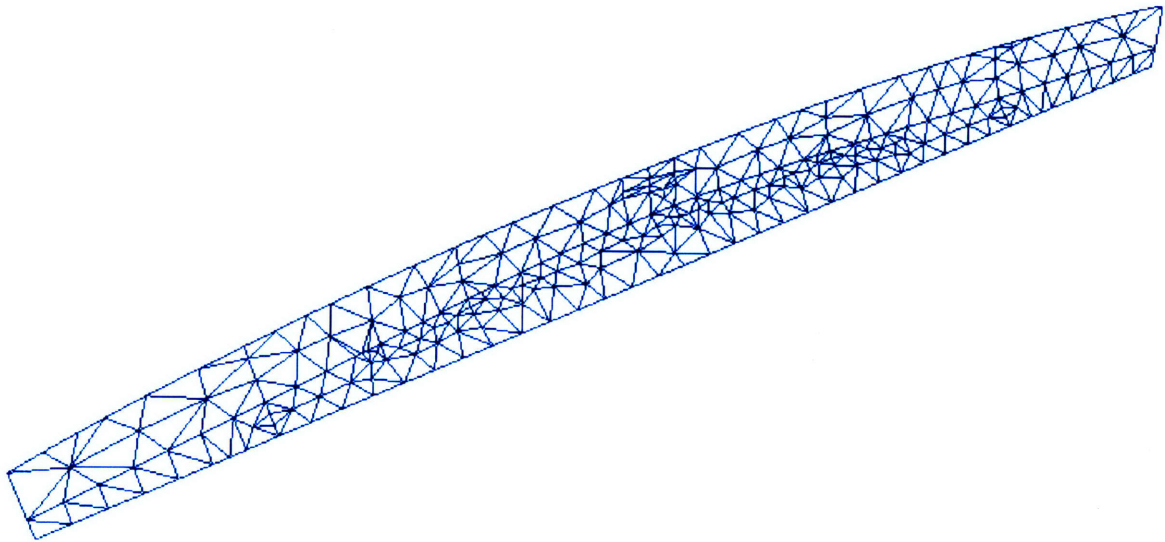


Figure 4-59: Ship hull; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.9, 4.10)

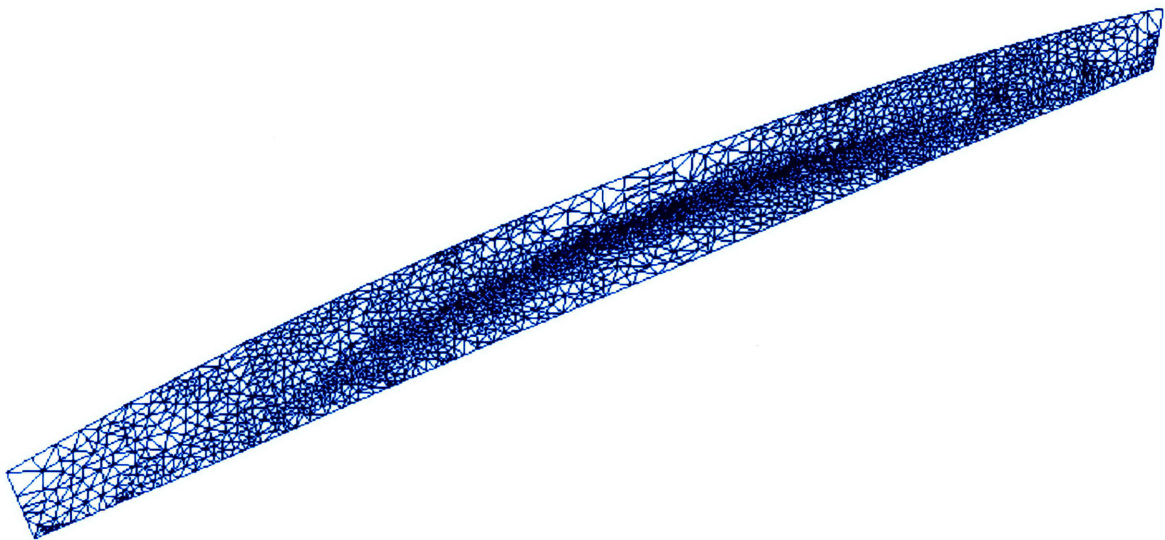


Figure 4-60: Ship hull; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Figure 4-62 and Table 4.9)

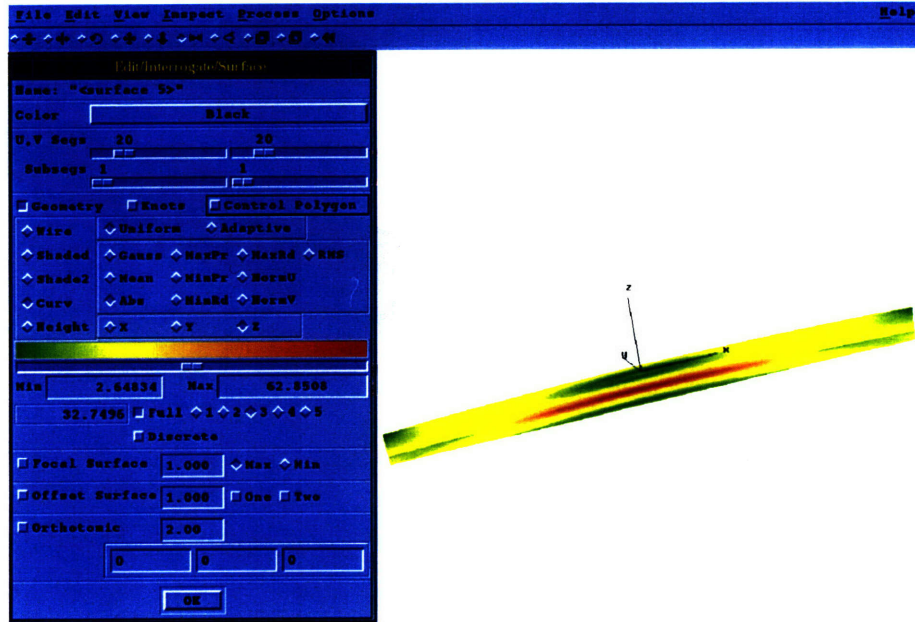


Figure 4-61: Ship hull; absolute curvature map

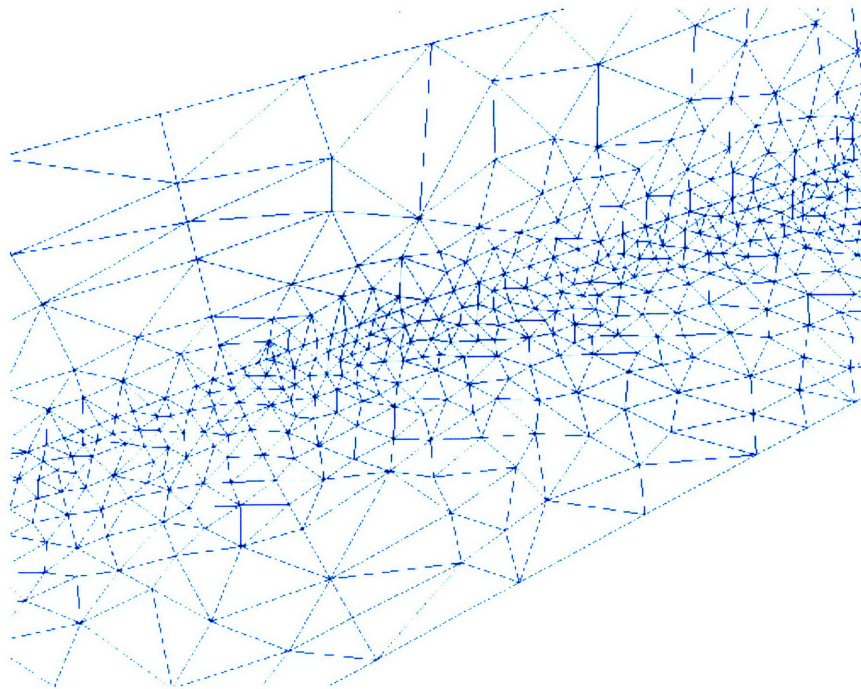
Figure 4-62: Ship hull; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (magnified)



Figure 4-63: Ship hull; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.9)



Figure 4-64: Ship hull; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.9)



Figure 4-65: Ship hull; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.9)

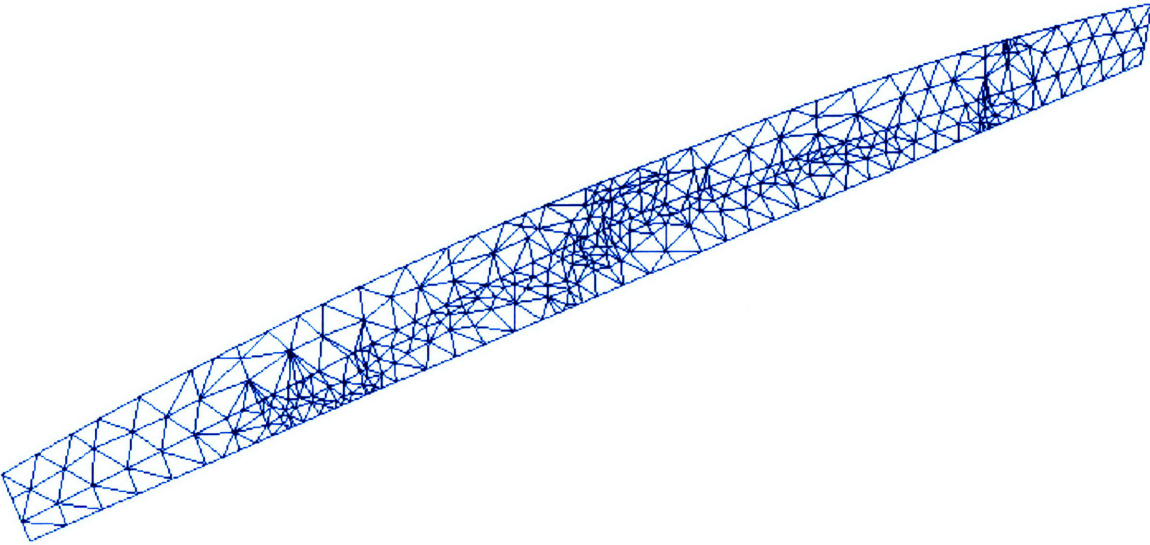


Figure 4-66: Ship hull; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.10)

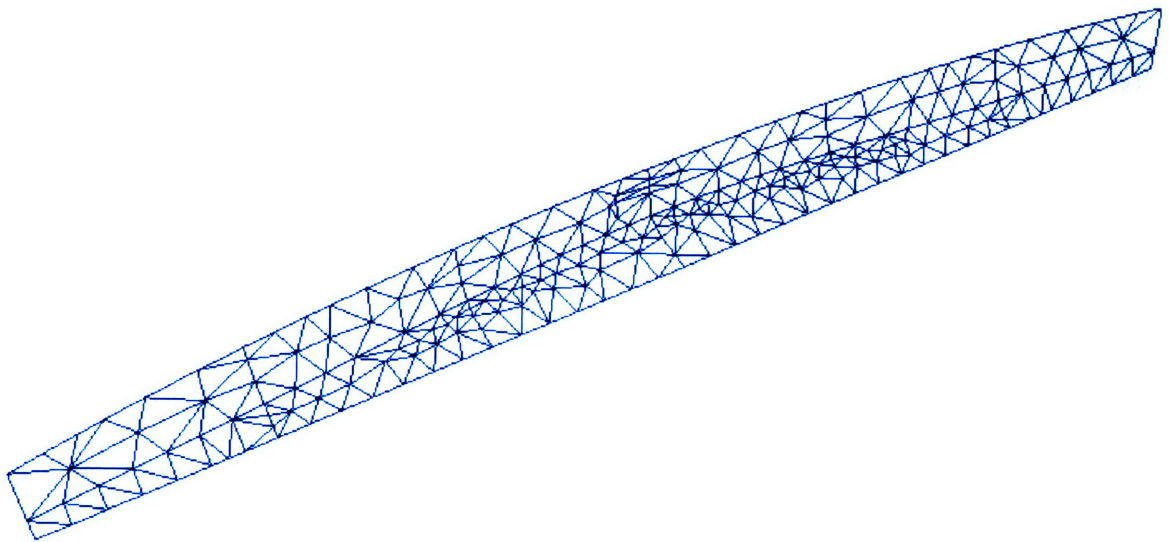


Figure 4-67: Ship hull; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 5$ (see also Table 4.10)

| ϵ | CPU (sec) | | | | | N | $\frac{N_{\text{fail}}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|------------|----------------|---------------------|--------------|---------------|-------|------|--|-------------------|-------------------|
| | Init- Δ | δ - Δ | AR- Δ | Fin- Δ | Total | | | | |
| 10^{-1} | 1508 | 78 | 4 | 21 | 1611 | 2574 | 3 | 2.6 | 5.7 |
| 10^{-2} | 1510 | 80 | 5 | 22 | 1617 | 2760 | 2 | 2.6 | 8.3 |
| 10^{-3} | 1515 | 387 | 33 | 78 | 2013 | 6800 | 2 | 2.1 | 7.7 |

Table 4.12: Airfoil; $\tau_{AR} = 4$ (see also Figures 4-68 – 4-78)

| τ_{AR} | CPU (sec) | N_{AR} | $\frac{N_{\text{fail}}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|-------------|-----------|----------|--|-------------------|-------------------|
| 3 | 9 | 1449 | 7 | 2.1 | 11.6 |
| 4 | 5 | 542 | 2 | 2.6 | 8.3 |
| 5 | 2 | 114 | 0.1 | 2.8 | 5.3 |

Table 4.13: Airfoil; AR- Δ , $\epsilon = 10^{-2}$, $N_{\delta} = 2218$ (see also Figures 4-70, 4-79, 4-80)

| CPU (sec) | | | | | | | | | | N | |
|----------------|-------|---------------------|------|--------------|-----|---------------|-----|-------|-------|------|------|
| Init- Δ | | δ - Δ | | AR- Δ | | Fin- Δ | | Total | | | |
| FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA |
| 1510 | 38835 | 80 | 2059 | 5 | 49 | 22 | 349 | 1617 | 41292 | 2760 | 2761 |

Table 4.14: Airfoil; $\epsilon = 10^{-2}$, $\tau_{AR} = 4$

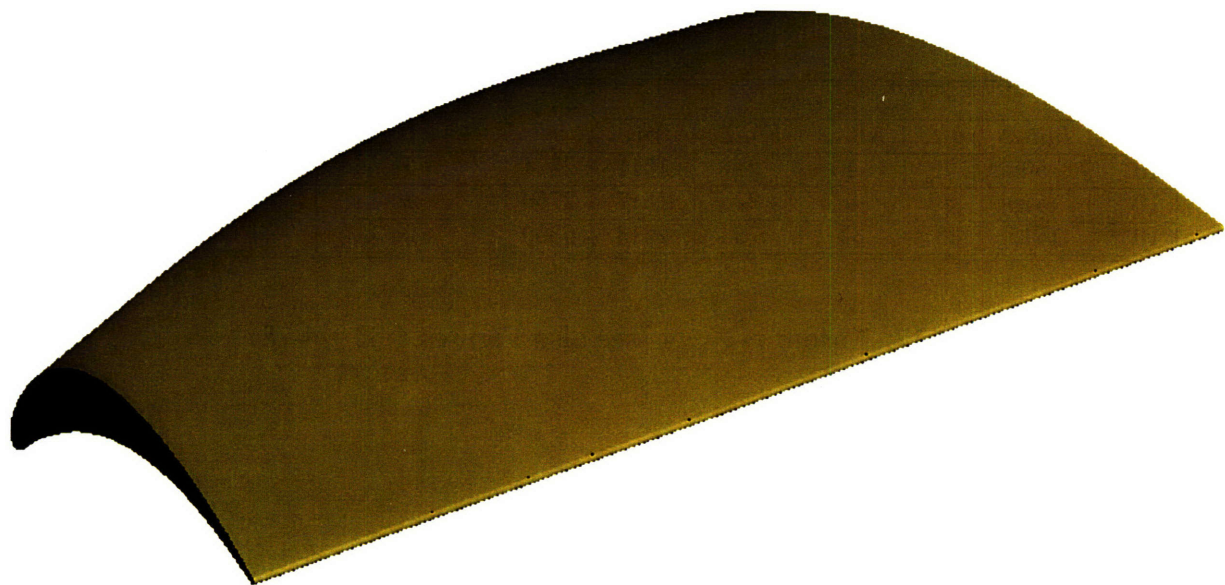


Figure 4-68: Airfoil; exact

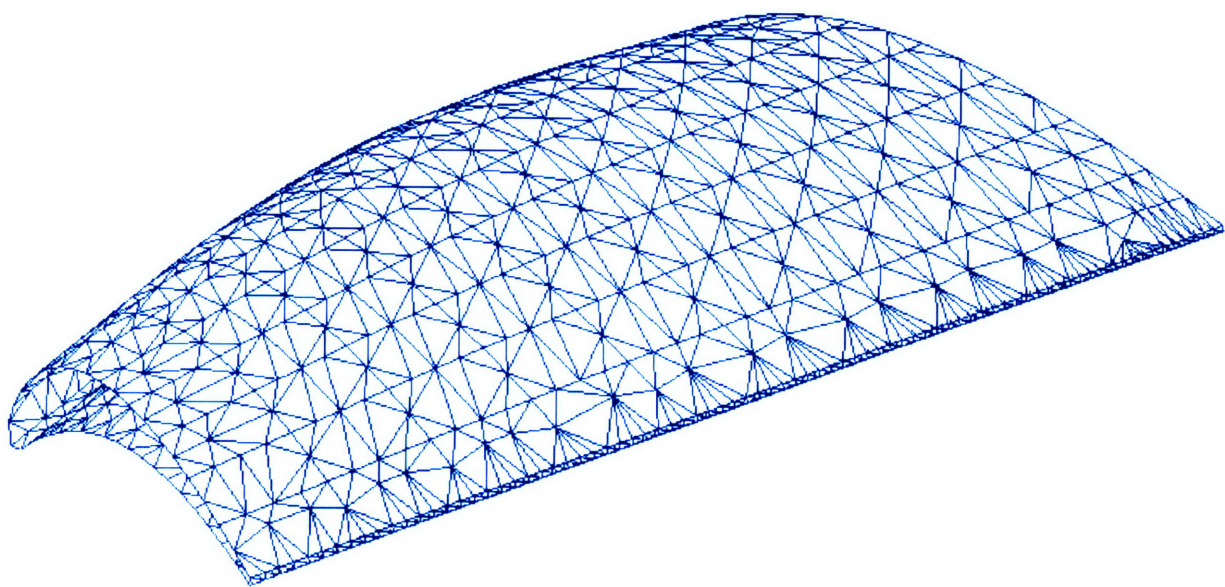


Figure 4-69: Airfoil; mesh with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.12)

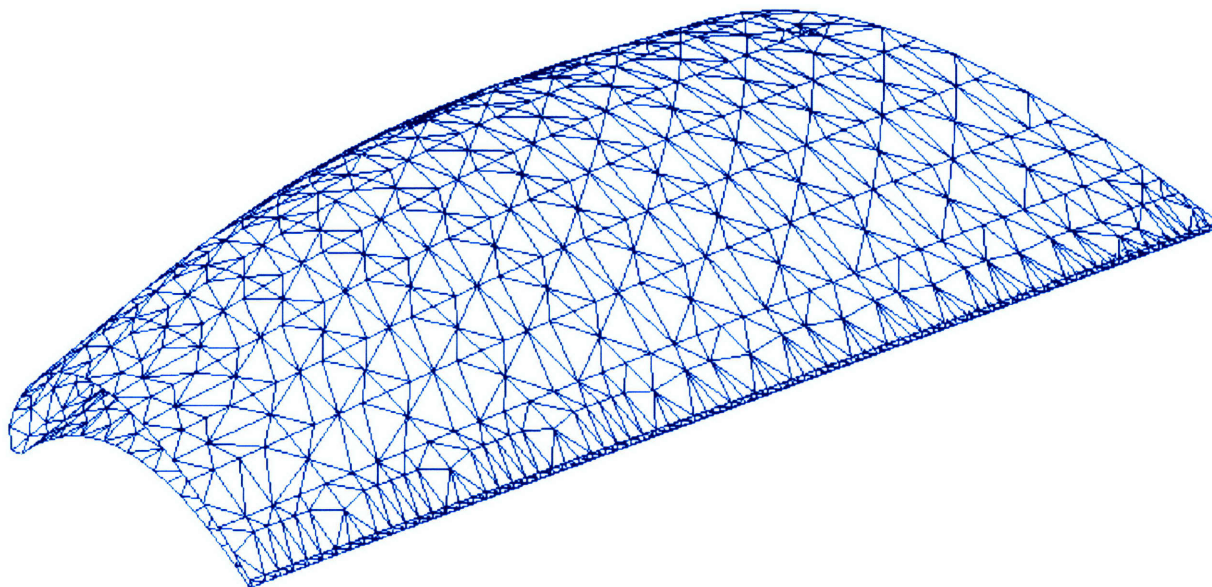


Figure 4-70: Airfoil; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.12, 4.13)

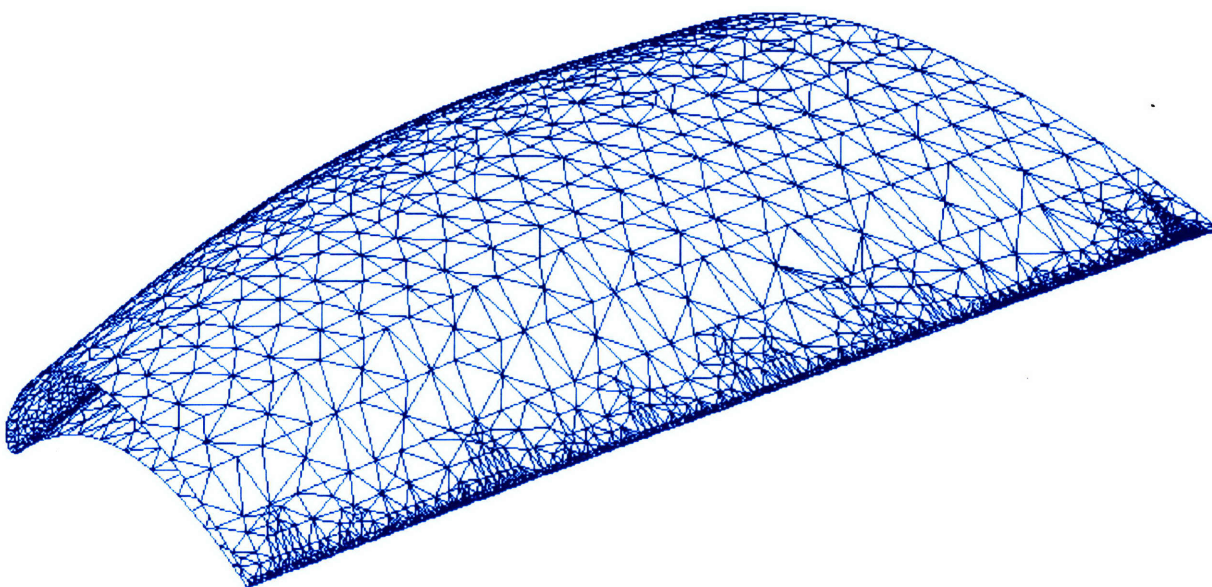


Figure 4-71: Airfoil; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.12)

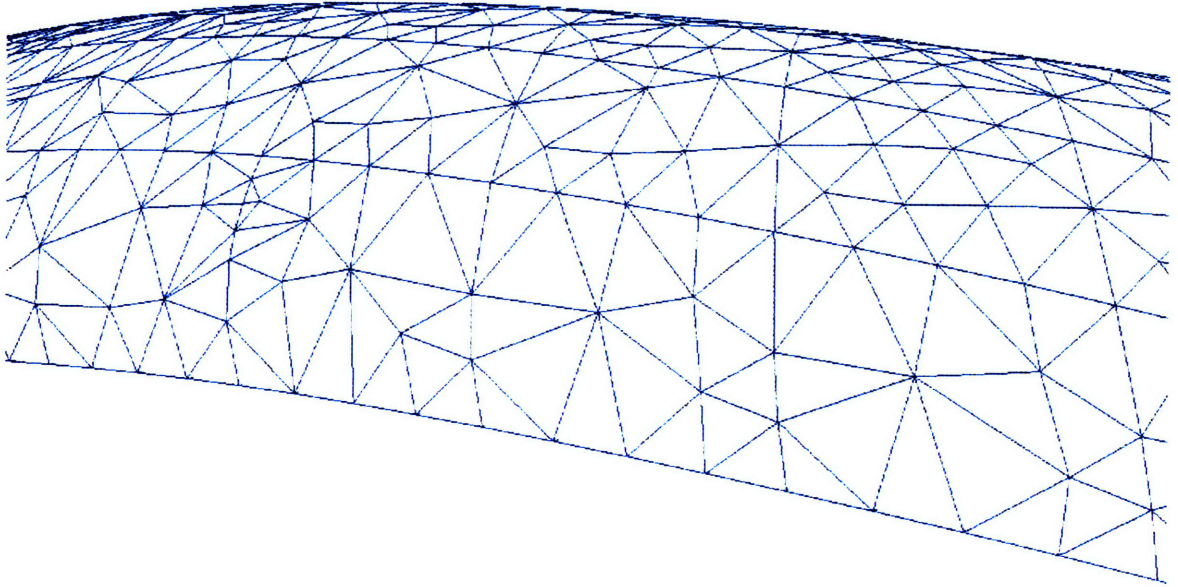


Figure 4-72: Airfoil; mesh near leading edge with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$

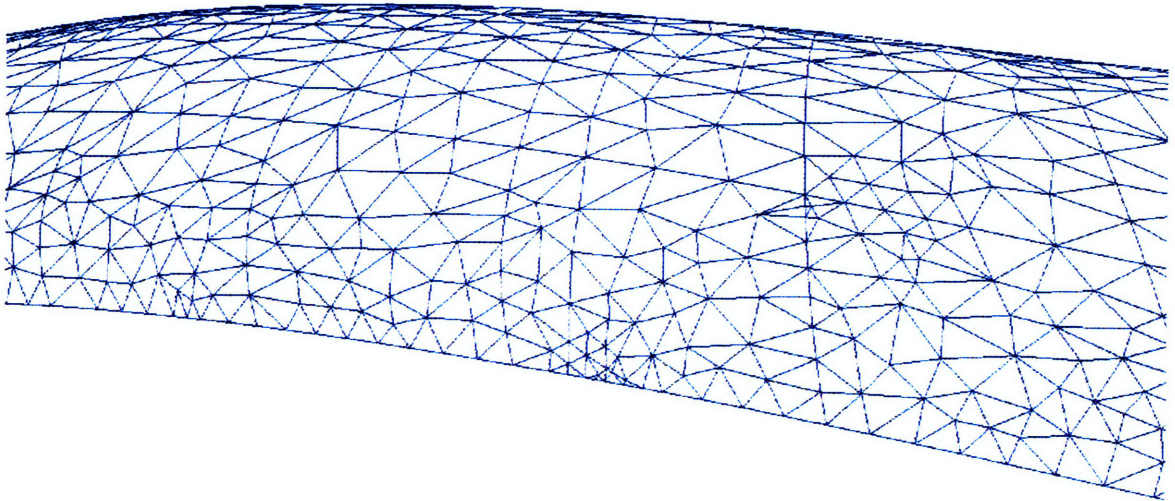


Figure 4-73: Airfoil; mesh near leading edge with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$

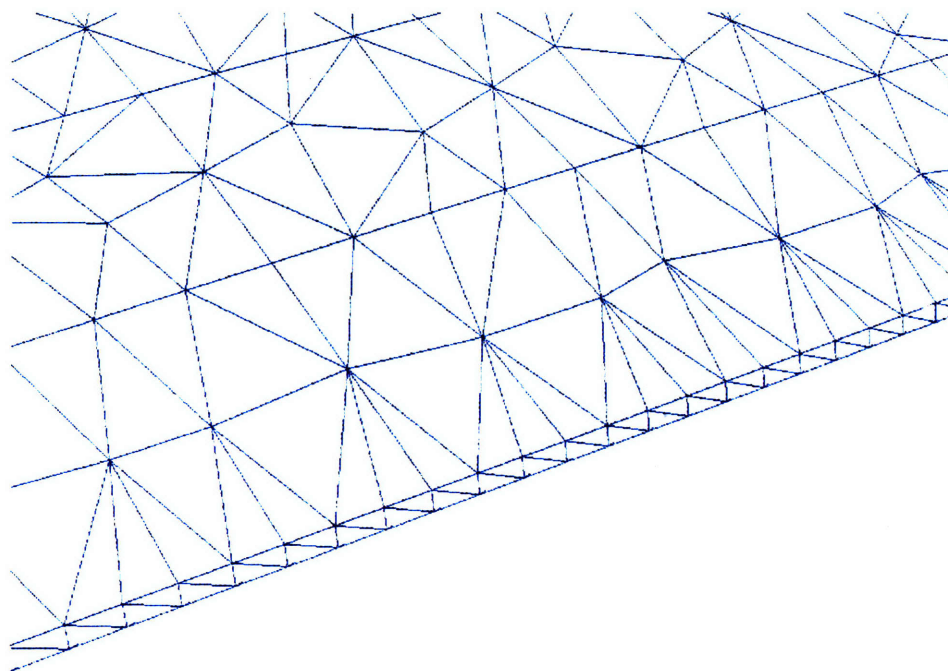


Figure 4-74: Airfoil; mesh near trailing edge with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$

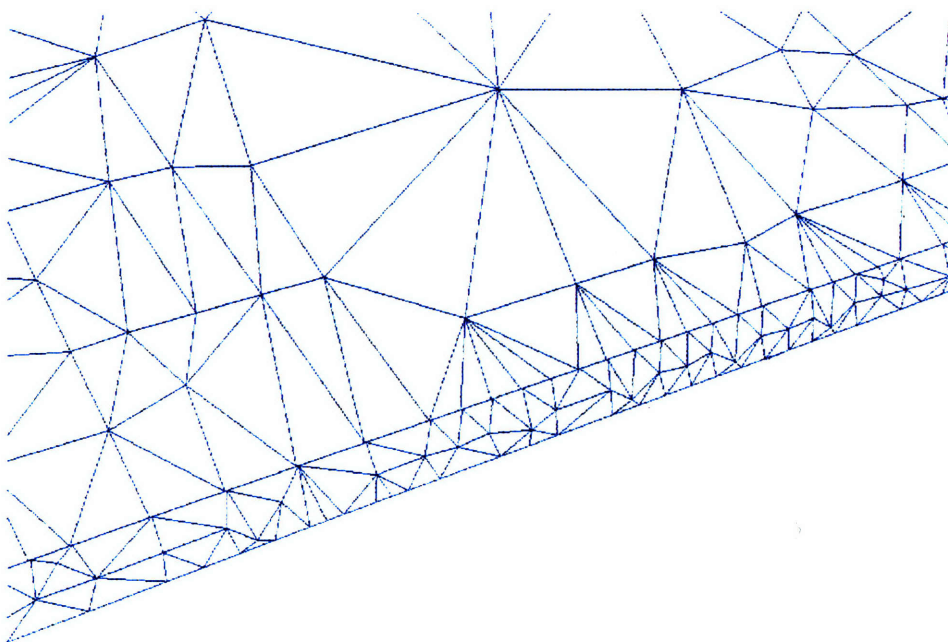


Figure 4-75: Airfoil; mesh near trailing edge with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$

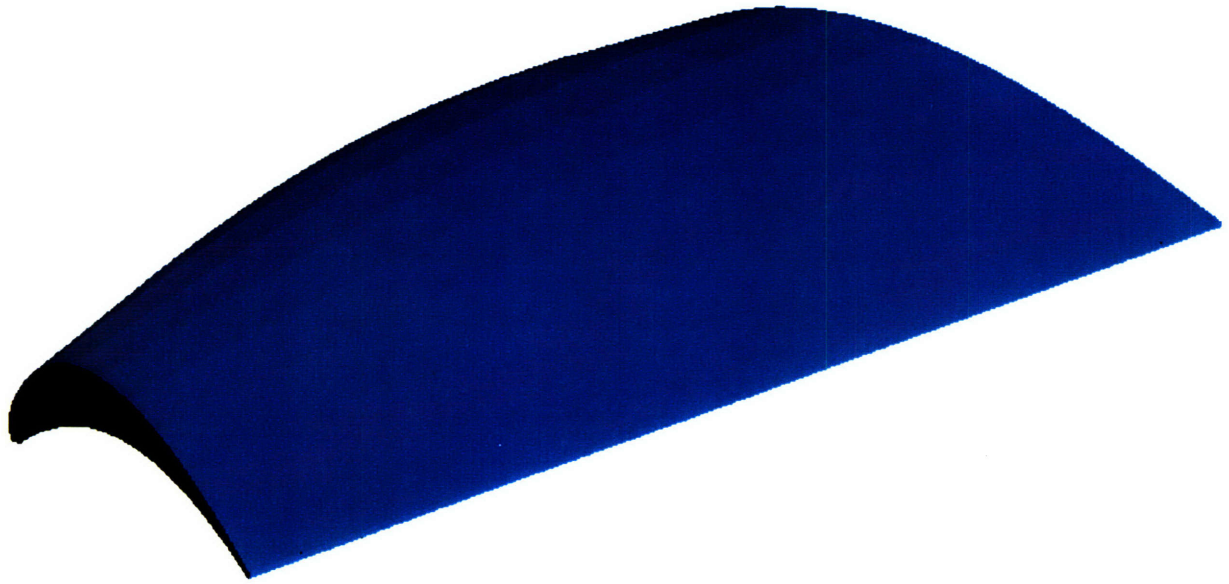


Figure 4-76: Airfoil; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.12)

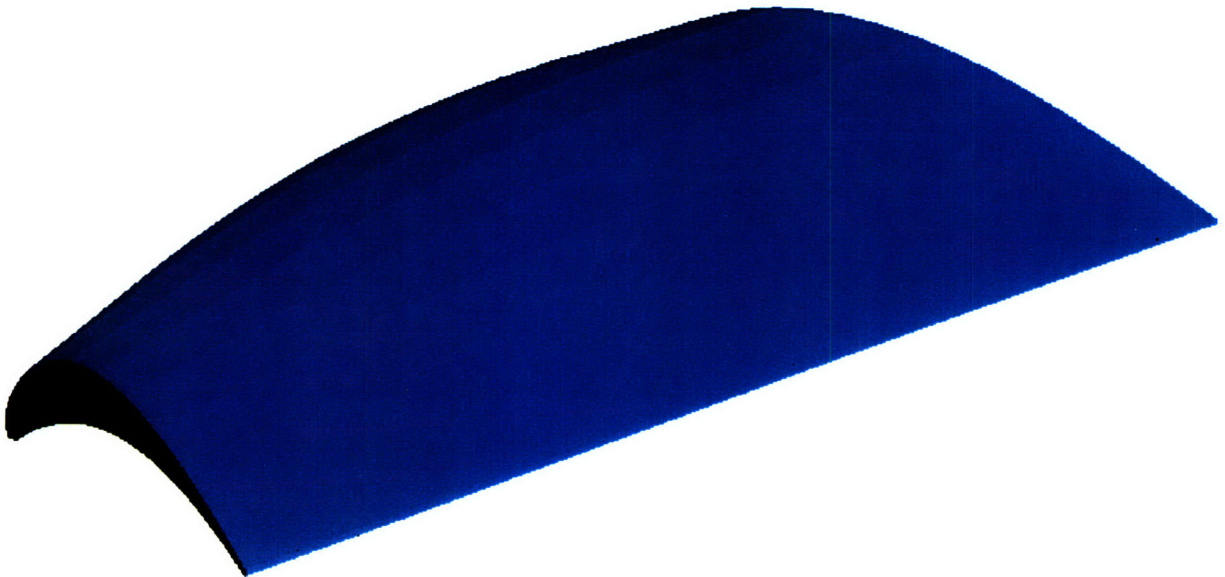


Figure 4-77: Airfoil; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.12)

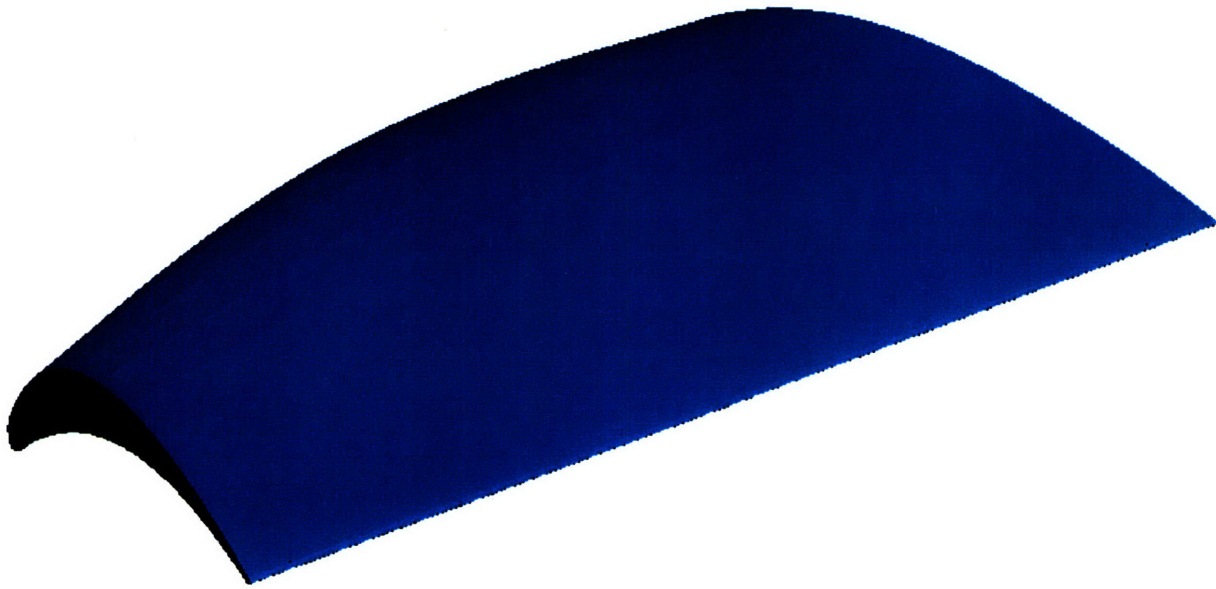


Figure 4-78: Airfoil; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.12)

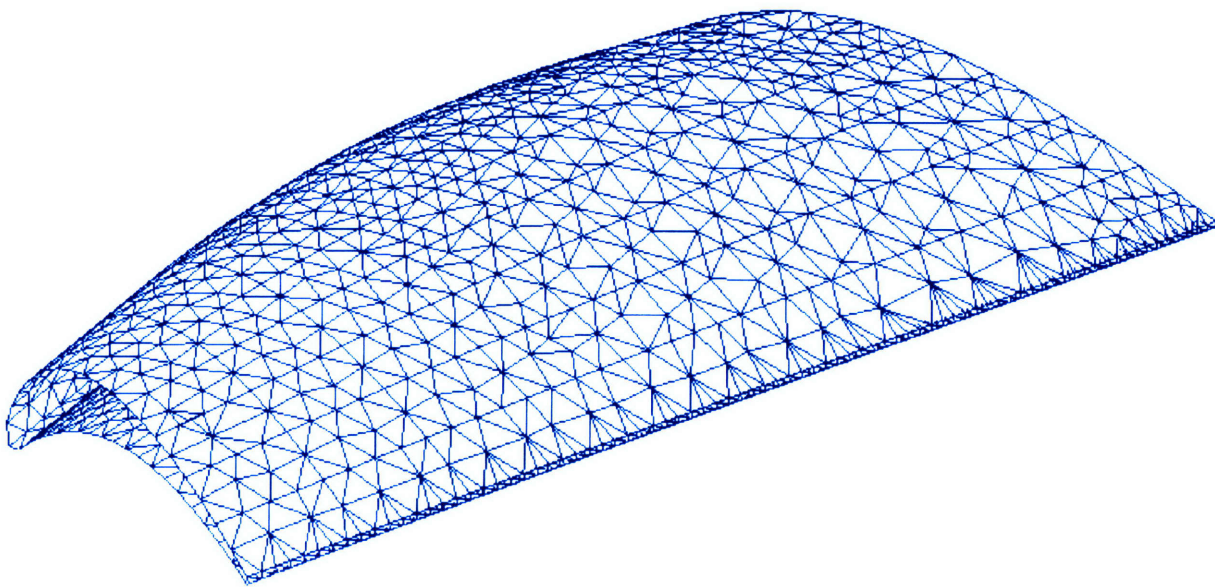


Figure 4-79: Airfoil; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.13)

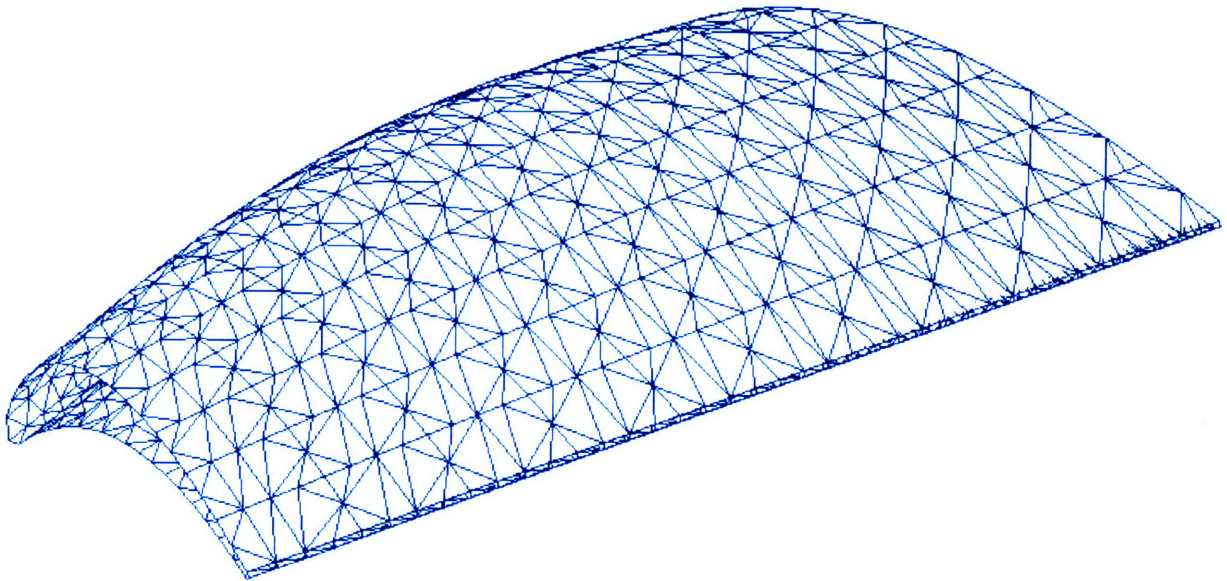


Figure 4-80: Airfoil; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 5$ (see also Table 4.13)

| ϵ | CPU (<i>sec</i>) | | | | | N | $\frac{N_{\text{fail}}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|------------|------------------------|---------------------|--------------|-----------------------|--------------|-------|--|-------------------|-------------------|
| | <i>Init</i> - Δ | δ - Δ | AR- Δ | <i>Fin</i> - Δ | Total | | | | |
| 10^{-1} | 453 | 13 | 2 | 4 | 472 | 636 | 4 | 2.7 | 10.9 |
| 10^{-2} | 457 | 57 | 4 | 15 | 533 | 1824 | 4 | 2.4 | 7.6 |
| 10^{-3} | 464 | 1613 | 320 | 139 | 2536 | 16109 | 4 | 2.2 | 11.8 |

Table 4.15: Teapot; $\tau_{\text{AR}} = 4$ (see also Figures 4-81 – 4-88)

| τ_{AR} | CPU (<i>sec</i>) | N_{AR} | $\frac{N_{\text{fail}}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|--------------------|--------------------|-----------------|--|-------------------|-------------------|
| 3 | 13 | 850 | 6 | 2.1 | 9.2 |
| 4 | 4 | 304 | 4 | 2.4 | 7.6 |
| 5 | 1 | 86 | 1 | 2.6 | 6.7 |

Table 4.16: Teapot; AR- Δ , $\epsilon = 10^{-2}$, $N_{\delta} = 1520$ (see also Figures 4-83, 4-89, 4-90)

| CPU (<i>sec</i>) | | | | | | | | | | N | |
|------------------------|-------|---------------------|-----|--------------|-----|-----------------------|-----|--------------|-------|------|------|
| <i>Init</i> - Δ | | δ - Δ | | AR- Δ | | <i>Fin</i> - Δ | | Total | | | |
| FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA |
| 457 | 12187 | 57 | 721 | 4 | 53 | 15 | 249 | 533 | 13210 | 1824 | 1819 |

Table 4.17: Teapot; $\epsilon = 10^{-2}$, $\tau_{\text{AR}} = 4$



Figure 4-81: Teapot; exact

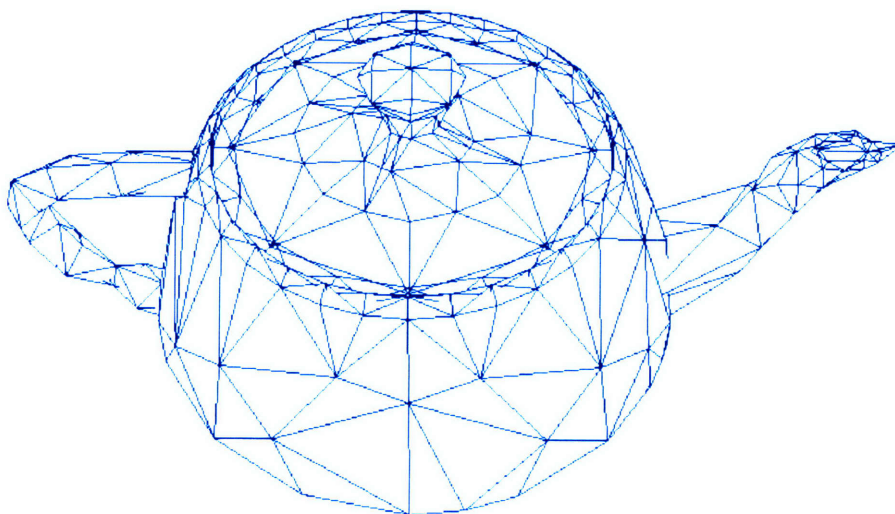


Figure 4-82: Teapot; mesh with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.15)

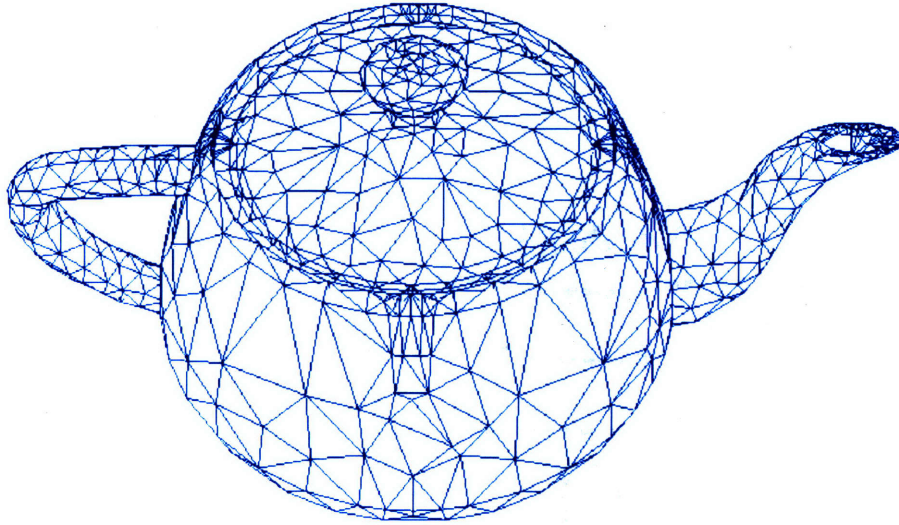


Figure 4-83: Teapot; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.15, 4.16)

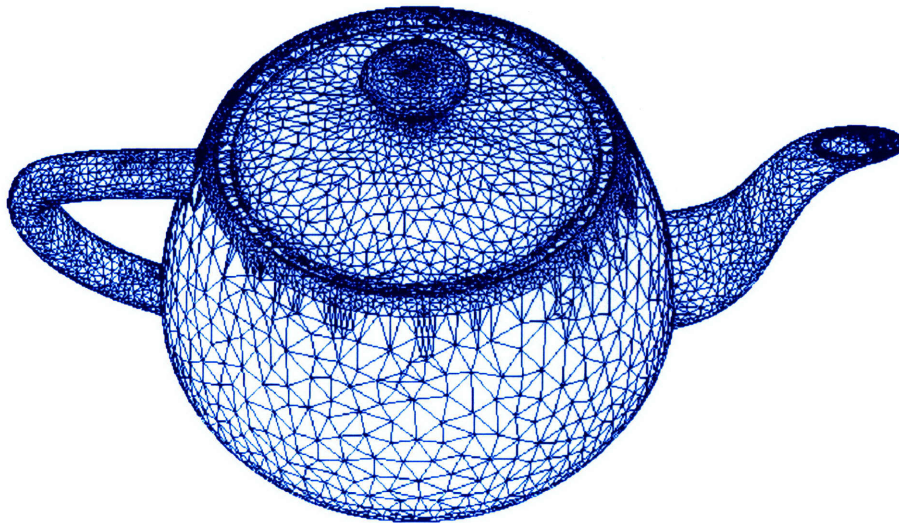


Figure 4-84: Teapot; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.15)



Figure 4-85: Teapot; approximation near handle with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$



Figure 4-86: Teapot; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.15)



Figure 4-87: Teapot; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.15)



Figure 4-88: Teapot; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.15)

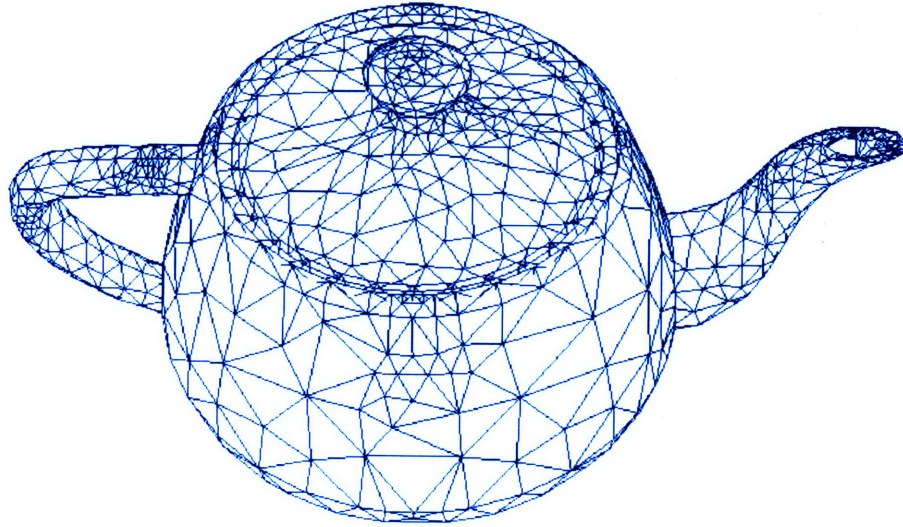


Figure 4-89: Teapot; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.16)

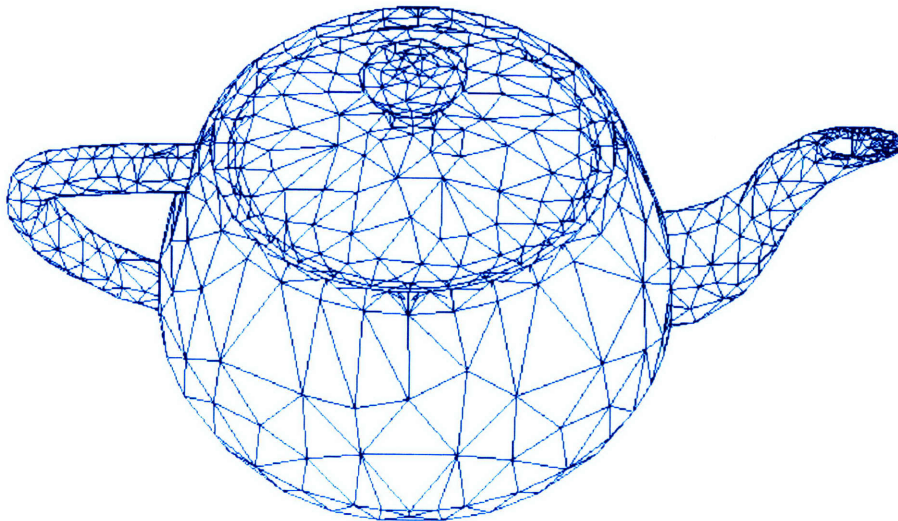


Figure 4-90: Teapot; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 5$ (see also Table 4.16)

| ϵ | CPU (sec) | | | | | N | $\frac{N_{\text{fail}}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|------------|----------------|---------------------|--------------|---------------|-------|------|--|-------------------|-------------------|
| | Init- Δ | δ - Δ | AR- Δ | Fin- Δ | Total | | | | |
| 10^{-1} | 609 | 42 | 4 | 12 | 667 | 1358 | 9 | 2.5 | 11.2 |
| 10^{-2} | 637 | 43 | 8 | 20 | 708 | 1827 | 3 | 2.4 | 6.9 |
| 10^{-3} | 640 | 124 | 15 | 49 | 828 | 4386 | 2 | 2.2 | 9.7 |

Table 4.18: Stem; $\tau_{AR} = 4$ (see also Figures 4-91 – 4-97)

| τ_{AR} | CPU (sec) | N_{AR} | $\frac{N_{\text{fail}}}{N} \times 100$ (%) | AR _{avg} | AR _{max} |
|-------------|-----------|----------|--|-------------------|-------------------|
| 3 | 14 | 1246 | 4 | 2.0 | 18.6 |
| 4 | 8 | 860 | 3 | 2.4 | 6.9 |
| 5 | 5 | 498 | 1 | 2.6 | 5.7 |

Table 4.19: Stem; AR- Δ , $\epsilon = 10^{-2}$, $N_{\delta} = 967$ (see also Figures 4-93, 4-98, 4-99)

| CPU (sec) | | | | | | | | | | N | |
|----------------|-------|---------------------|------|--------------|-----|---------------|-----|-------|-------|------|------|
| Init- Δ | | δ - Δ | | AR- Δ | | Fin- Δ | | Total | | | |
| FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA | FPA | RIA |
| 637 | 15487 | 43 | 1281 | 8 | 114 | 20 | 387 | 708 | 17269 | 1827 | 1831 |

Table 4.20: Stem; $\epsilon = 10^{-2}$, $\tau_{AR} = 4$

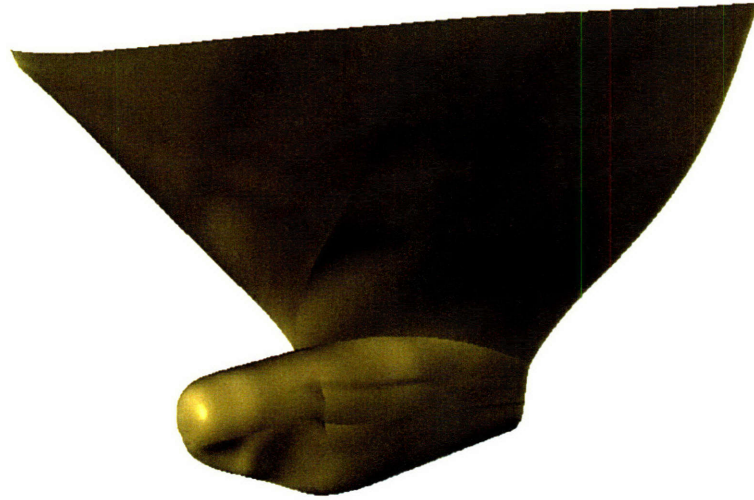


Figure 4-91: Stem; exact

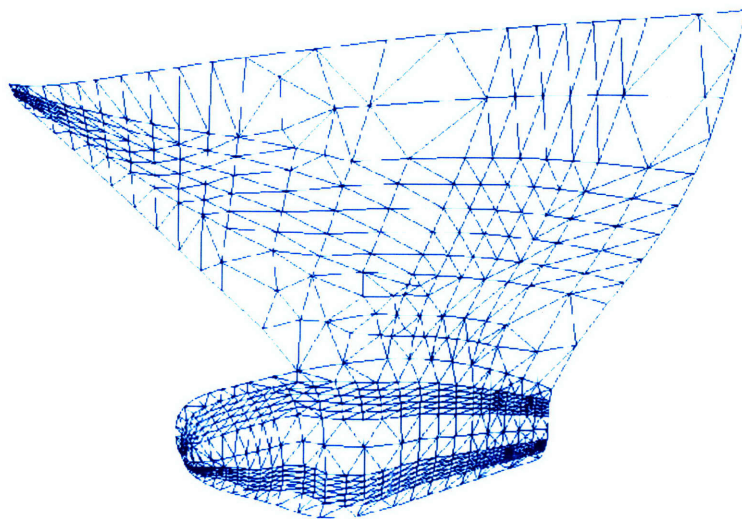


Figure 4-92: Stem; mesh with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.18)

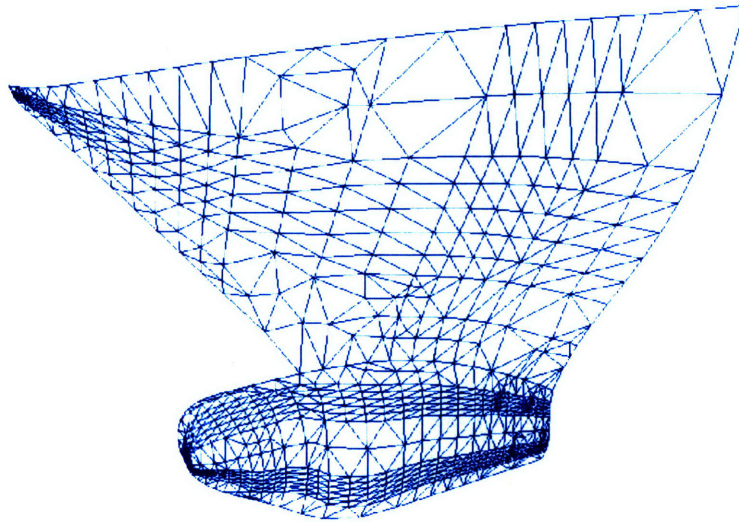


Figure 4-93: Stem; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Tables 4.18, 4.19)

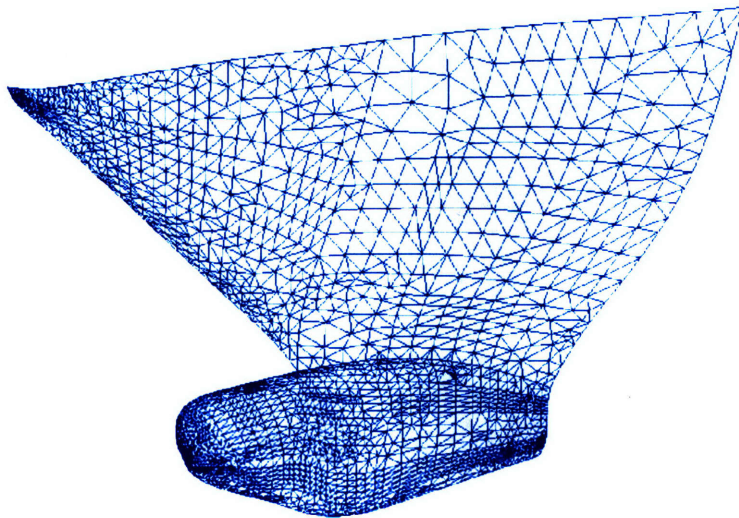


Figure 4-94: Stem; mesh with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.18)

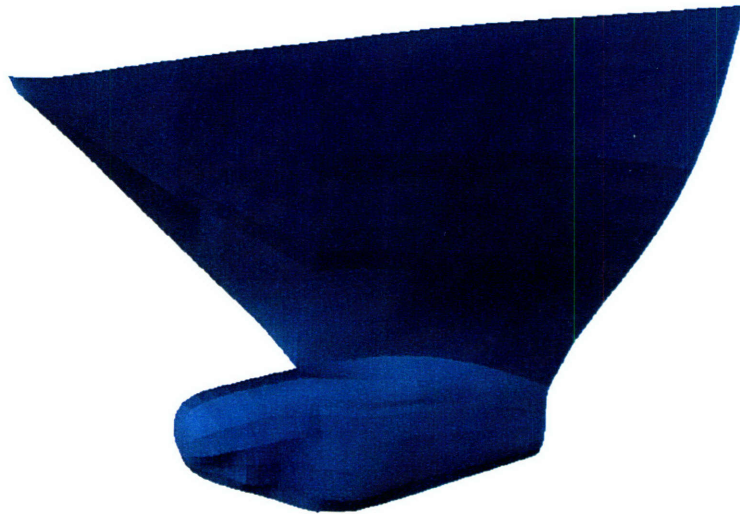


Figure 4-95: Stem; approximating surface with $\epsilon = 10^{-1}$, $\tau_{AR} = 4$ (see also Table 4.18)



Figure 4-96: Stem; approximating surface with $\epsilon = 10^{-2}$, $\tau_{AR} = 4$ (see also Table 4.18)

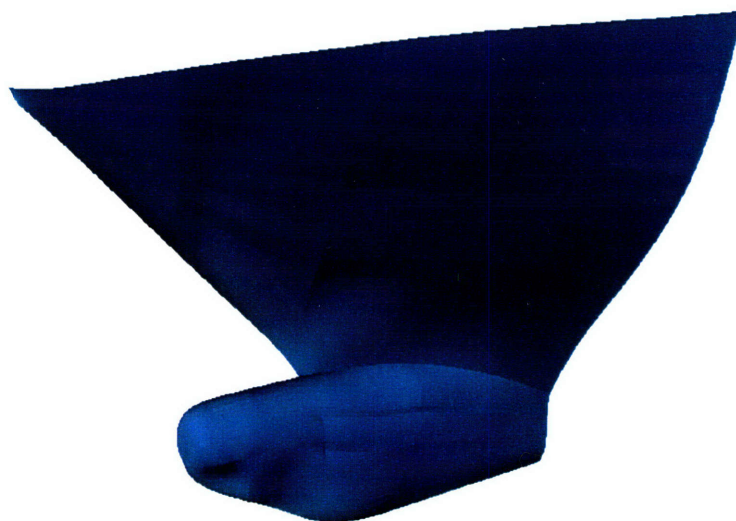


Figure 4-97: Stem; approximating surface with $\epsilon = 10^{-3}$, $\tau_{AR} = 4$ (see also Table 4.18)

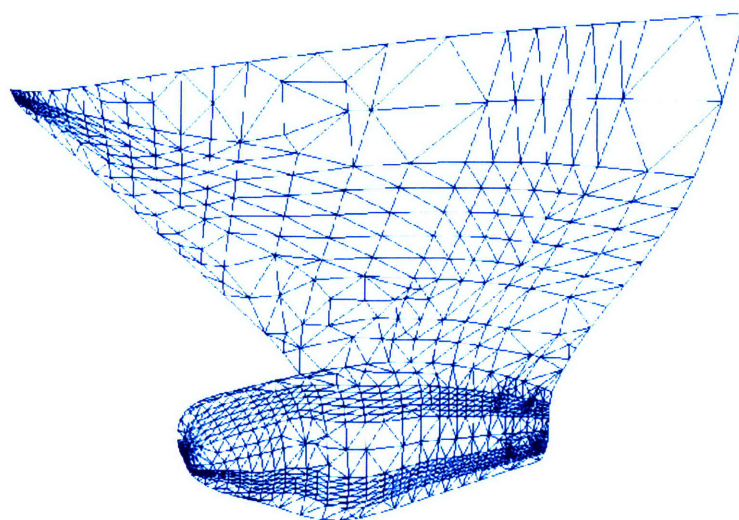


Figure 4-98: Stem; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 3$ (see also Table 4.19)

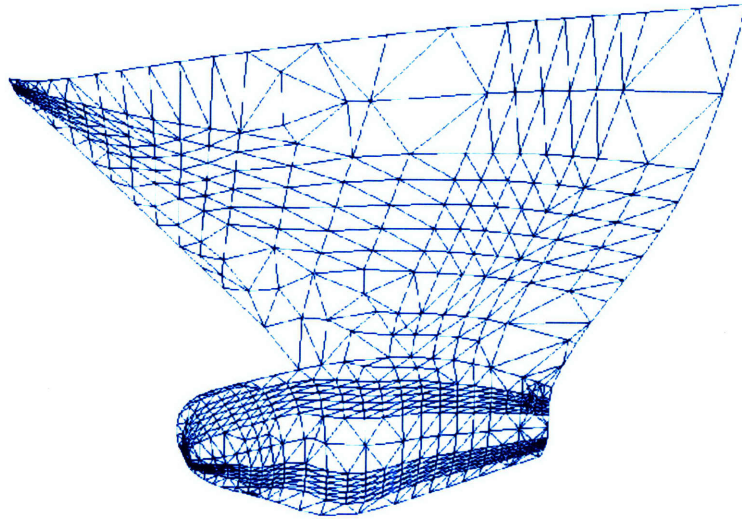


Figure 4-99: Stem; mesh with $\epsilon = 10^{-2}$, $\tau_{AR} = 5$ (see also Table 4.19)

Chapter 5

Conclusions and Recommendations

We finally summarize the major results and contributions of this thesis and present recommendations for future research.

5.1 Summary and Contributions

Development of piecewise linear approximation methods of high order polynomial composite curves and trimmed composite surfaces, which include :

- Use of *numerically robust* interval geometric definitions and computations.
 - The use of rounded interval arithmetic (RIA) prevents a possible system failure which happens in the extremely important computations of the approximation algorithm such as intersection test between the edges of an approximately developed surface net, mappings between the triangulation domain and the parametric space, Delaunay test and self-intersection test of the approximating triangles. Especially, robust decision criteria of Delaunay test relying on RIA, are presented to remove the possibility of degenerate or non-conforming triangulations.
- Identification of various kinds of *geometrically significant* points.

- Equations and solution methodology are described to identify geometrically significant points of polynomial curves and surfaces, which reflect differential geometric features of the exact geometry and provide meaningful initial approximations. For a loose approximation tolerance, a triangulation with only points of κ_{rms} local maxima as initial nodes shows the most optimal result in terms of a space cost compared with the result without using the stationary points of κ_{rms} or the result by including all the stationary points of κ_{rms} . For a tight approximation tolerance, slightly less number of triangles are generated in case of all the stationary points of κ_{rms} being included as initial nodes; however, the difference in space cost is not so sensitive to the existence of stationary points of κ_{rms} as initial nodes.
- An *efficient* and *adaptive* approximation method to satisfy the prescribed geometric tolerance.
 - An adaptive unstructured meshing algorithm, which employs a convex hull method to compute a tight upper bound of the approximation error, is developed. An efficient method is proposed to compute such an upper bound of the approximation error between an arbitrarily oriented triangular sub-Bézier patch on the exact tensor product Bézier surface and the corresponding approximating triangle. Compared with an existing polar form based method, our method is 2^p times more efficient in terms of time cost, where p is the sum of degrees of a given tensor product Bézier surface in each parametric direction.
- A surface meshing algorithm which generates triangles with an *acceptable aspect ratio* (AR) in three-dimensional space.
 - A method is presented to construct a two dimensional domain of triangulation which sufficiently preserves the shape of triangular elements when mapped into the three-dimensional space.

- An algorithm is proposed to improve the AR of badly-shaped triangles.
- An efficient method to identify and remove the possible *inappropriate intersections* of the approximating elements, ensuring the existence of a *homeomorphism* between the approximating elements and the actual nonlinear curves and surfaces.

5.2 Future Research

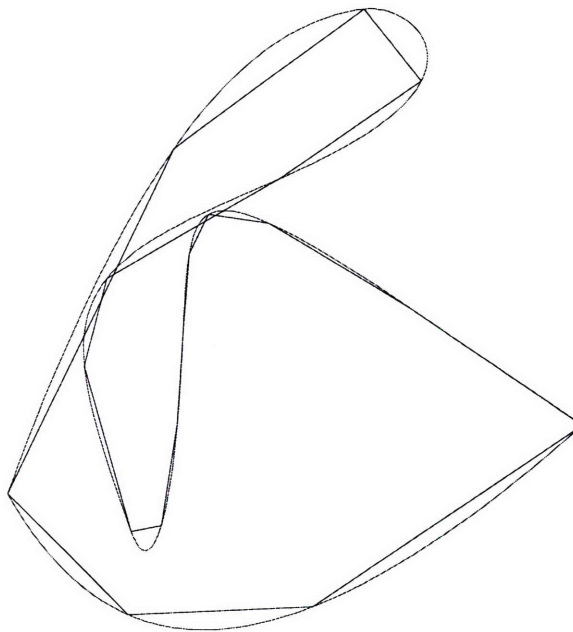
For topics of future research, we recommend:

- Development of a topologically reliable approximation algorithm for (smooth) simple space curves which guarantees the existence of a *space* homeomorphism between the curves and the corresponding approximant. More specifically, let C be a (smooth) simple space curve and \tilde{C} its piecewise linear approximant that is homeomorphic to C . Then, as Figure 5-1 shows, it may happen that knots can be created (or undone) in the curve \tilde{C} , that is, it is quite possible that the two curves C and \tilde{C} might not have the same *knot type*. The reason for this is that even though there exists a homeomorphism $\mathbf{f} : C \rightarrow \tilde{C}$, they are *not* space homeomorphic; in other words, there does not exist a homeomorphism $\mathbf{h} : \mathfrak{R}^3 \rightarrow \mathfrak{R}^3$ which carries C onto \tilde{C} , see [64].

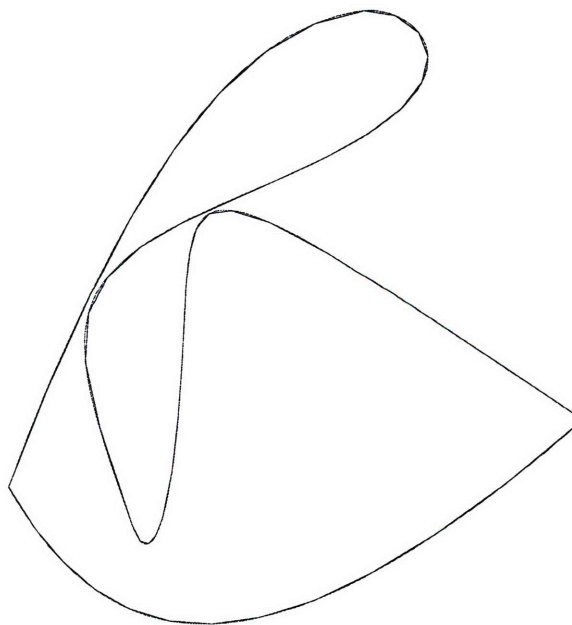
To overcome the above, one may construct a nonsingular pipe surface, (also called a *tubular neighborhood* of C), $\mathbf{P}_C(r)$, for a suitable positive number r , whose spine curve is C , see [61]. Then if the approximant curve \tilde{C} is so constructed that

- \tilde{C} lies entirely inside $\mathbf{P}_C(r)$, and
- \tilde{C} has the same orientation as C ,

then it can be shown that C and \tilde{C} are space homeomorphic via an *orientation preserving* homeomorphism \mathbf{h} , and thus they have the same knot type, (P. Sakkalis, Private Communication, 1997).



(a) exact curve and its approximant with knots



(b) exact curve and its space homeomorphic approximation

Figure 5-1: A composite *unknotted space* curve and its approximations

- Extension of the topologically reliable piecewise *linear* approximation method of high order *integral Bézier* curves to :
 - Topologically reliable piecewise linear approximation of high order *rational Bézier* or *NURBS* curves.
 - Topologically reliable piecewise *lower order non-linear* approximation of high order *integral Bézier*, *rational Bézier* or *NURBS* curves.
- Incorporation of *medial axis transform* (MAT) into the early stage of surface tessellation algorithm, which provides feature recognition such as identification of symmetries, constrictions and local length scales.
- Modification of AR-improving procedure in the surface meshing algorithm in order to achieve better *termination criteria* e.g., intelligently weakening the termination condition by introducing a weighting factor to Eq. (4.70) as a function of AR of currently chosen worst triangle and number of triangles, in order to balance the quality of triangles and the space cost.
- Extension of the topologically reliable piecewise *linear* approximation method of high order trimmed *polynomial Bézier* surface patches to :
 - Topologically reliable piecewise linear approximation of high order trimmed *rational Bézier* or *NURBS* surface patches.
 - Topologically reliable piecewise *lower order non-linear* approximation of high order trimmed *integral Bézier*, *rational Bézier* or *NURBS* surface patches.
 - Topologically reliable adaptive *tetrahedral meshes* of a B-rep solid.

Bibliography

- [1] Abrams, S. L., Bardis, L., Chrysostomidis, C., Patrikalakis, N. M., Tuohy, S. T., Wolter, F.-E. and Zhou, J. (1995), The geometric modeling and interrogation system Praxiteles, *Journal of Ship Production*, 11(2):116-131.
- [2] Agoston, M. K. (1976), *Algebraic Topology*, Marcel Dekker Inc., New York.
- [3] Allgower, E. and Gnutzmann, S. (1991), Simplicial pivoting for mesh generation of implicitly defined surfaces, *Computer Aided Geometric Design*, 8(4):305-326.
- [4] Andersson, L. E., Dorney, S. M., Peters, T. J. and Stewart, N. F. (1995), Polyhedral perturbations that preserve topological form, *Computer Aided Geometric Design*, 12:785-799.
- [5] Asano, T., Edahiro, M., Imai, H., Iri, M. and Murota, K. (1985), Practical use of bucketing technique in computational geometry, in: Toussaint, G., editor, *Computational Geometry, Machine Intelligence and Pattern Recognition*, 2:153-195, North Holland.
- [6] Aurenhammer, F. (1991), Voronoi diagrams – a survey of a fundamental geometric data structure, *ACM Computing Surveys*, 23:345-405.
- [7] Babuška, I. and Aziz. A. (1976), On the angle condition in the finite element method, *SIAM Journal of Numerical Analysis*, 13:214-227.
- [8] Baker, T. J. (1989), Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation, *Engineering with Computers*, 5:161-175.

-
- [9] Bardis, L. and Patrikalakis, N. M. (1989), Approximate conversion of rational B-spline patches, *Computer Aided Geometric Design*, 6(3):189–204.
- [10] Bardis, L. and Patrikalakis, N. M. (1994), Topological structures for generalized boundary representations, MIT Sea Grant Report 94-22, Cambridge, MA.
- [11] Boender, E., Bronsvort, W. F. and Post, F. H. (1994), Finite-element mesh generation from constructive-solid-geometry models, *Computer Aided Design*, 26(5):379–392.
- [12] Bowyer, A. (1981), Computing Dirichlet tessellations, *The Computer Journal*, 24(2):162–166.
- [13] Brown, R. (1988), *Topology : A Geometric Account of General Topology, Homotopy Types and the Fundamental Groupoid*, Ellis Horwood Ltd., Chichester, England.
- [14] Casale, M. S. (1987), Free-form solid modeling with trimmed surface patches, *IEEE Computer Graphics and Applications*, 7(1):33–43.
- [15] Chew, L. P. (1989), Guaranteed-quality triangular meshes, Department of Computer Science Tech. Report, TR 89-983, Cornell University.
- [16] Chew, L. P. (1993), Guaranteed-quality mesh generation for curved surfaces, in: *Proceedings of the Ninth Annual ACM Symposium on Computational Geometry*, 274–280. ACM Press.
- [17] Chinn, W. G. and Steenrod, N. E. (1966), *First Concepts of Topology : the Geometry of Mappings of Segments, Curves, Circles, and Disks*, Mathematical Association of America Inc., Washington D. C..
- [18] Cho, W., Maekawa, T. and Patrikalakis, N. M. (1996), Topologically reliable approximation of composite Bézier curves, *Computer Aided Geometric Design*, 13(6):497–520.

- [19] Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990), *Introduction to Algorithms*, MIT Press, Cambridge, MA.
- [20] Delaunay, B. (1934), Sur la sphere vide, *Izvestia Akademii Navk SSSR, Mathematical and Natural Sciences Division*, 6:793–800.
- [21] Dirichlet, G. L. (1850), Über die Reduction der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen, *Zeitschrift für Reine und Angewandte Mathematik.*, 40(3):209–227.
- [22] do Carmo, M. P. (1976), *Differential Geometry of Curves and Surfaces*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
- [23] Dolence, A. and Mäkelä, I. (1990), Optimized triangulation of parametric surfaces, *Mathematics of Surfaces IV*
- [24] Eck, M. (1993), Degree reduction of Bézier curves, *Computer Aided Geometric Design*, 10(3–4):237–251.
- [25] Elber, G. (1996), Error bounded piecewise linear approximation of freeform surfaces, *Computer Aided Design*, 28(1):51–57.
- [26] Fahn, C., Wang, J. and Lee, J. (1989), An adaptive reduction procedure for the piecewise linear approximation of digitized curves, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(9):967–973.
- [27] Farin, G. (1986), Triangular Bernstein-Bézier patches, *Computer Aided Geometric Design*, 3(2):83–128.
- [28] Farin, G. (1993), *Curves and Surfaces for Computer Aided Geometric Design - A Practical Guide, 3rd Edition*, Academic Press, Inc., San Diego, CA.
- [29] Farouki, R. T. (1987), Trimmed-surface algorithms for the evaluation and interrogation of solid boundary representations, *IBM Journal of Research and Development*, 31(3):314–334.

- [30] Farouki, R. T. and V. T. Rajan (1988), Algorithms for polynomials in Bernstein form, *Computer Aided Geometric Design*, 5:1-26.
- [31] Farouki, R. T. (1989), Hierarchical segmentations of algebraic curves and some applications, in: Lyche, T. and Schumaker, L. L., eds., *Mathematical Methods in Computer Aided Geometric Design*, 239–248.
- [32] Filip, D., Magedson, R. and Markot, R. (1986), Surface algorithm using bounds on derivatives, *Computer Aided Geometric Design*, 3:295-311.
- [33] Filip, D. (1986), Adaptive subdivision algorithms for a set of Bézier triangles, *Computer Aided Design*, 18(2):74-78.
- [34] Fortune, S. (1992), Voronoi diagrams and Delaunay triangulation, in: Hwang, F. K. and Du, D. Z., eds, *Computing in Euclidean Geometry*, World Scientific.
- [35] Gürsoy, H. N. and Patrikalakis, N. M. (1991), Automated interrogation and adaptive subdivision of shape using medial axis transform, *Advances in Engineering Software and Workstations*, 13(5/6):287-302.
- [36] Gürsoy, H. N. and Patrikalakis, N. M. (1992), An automated coarse and fine surface mesh generation scheme based on medial axis transform, part I: Algorithms, *Engineering with Computers*, 8(3):121-137.
- [37] Gürsoy, H. N. and Patrikalakis, N. M. (1992), An automated coarse and fine surface mesh generation scheme based on medial axis transform, part II: Implementation, *Engineering with Computers*, 8(4):179-196.
- [38] Hall, M. and Warren, J. (1990), Adaptive polygonalization of implicitly defined surfaces, *IEEE Computer Graphics and Applications*, 10(6):33–42.
- [39] Hamann, B. and Chen, J. (1994), Data point selection for piecewise linear curve approximation, *Computer Aided Geometric Design*, 11:289–301.
- [40] Hilbert, D. and Cohn-Vossen, S. (1952), *Geometry and Imagination*, Chelsea, New York.

- [41] Hoffmann, C. M. (1989), The problems of accuracy and robustness in geometric computation, *Computer*, 22(3):31–41.
- [42] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. and Stuetzle, W. (1993), Mesh optimization, Department of Computer Science and Engineering, Tech. Report, 93-01-01, University of Washington, Seattle, Washington.
- [43] Hoschek, J. and Schneider, F.-J. (1992), Approximate spline conversion for integral and rational Bézier and B-spline surfaces, in: Barnhill, R. E., editor, *Geometry Processing for Design and Manufacturing*, SIAM, Philadelphia, 45–86.
- [44] Hoschek, J. and Lasser, D. (1993), *Fundamentals of Computer Aided Geometric Design*, Peters, A. K., Wellesley, MA.
- [45] Hu, C.-Y. (1993), Robust algorithms for sculptured shape visualization, M.S. Thesis, Massachusetts Institute of Technology, Cambridge, MA.
- [46] Hu, C.-Y., Maekawa, T., Sherbrooke, E. C. and Patrikalakis, N. M. (1996), Robust interval algorithm for curve intersections, *Computer Aided Design*, 28(6/7):495–506.
- [47] Hu, C.-Y., Patrikalakis, N. M. and Ye, X. (1996), Robust interval solid modeling, Part I: Representations, *Computer Aided Design*, 28(10):807–817.
- [48] Hu, C.-Y., Patrikalakis, N. M. and Ye, X. (1996), Robust interval solid modeling, Part II: Boundary evaluation, *Computer Aided Design*, 28(10):819–830.
- [49] Hu, C.-Y., Maekawa, T., Patrikalakis, N. M. and Ye, X. (1995), Robust interval algorithm for surface intersections, *Computer Aided Design*, To appear.
- [50] Ihm, I. and Naylor, B. (1991), Piecewise linear approximations of digitized space curves with applications, in: Patrikalakis, N. M., editor, *Scientific Visualization of Physical Phenomena, Proceedings of the 9th International Conference on Computer Graphics*, CGI '91, Springer-Verlag, 545–569.

- [51] Kehtarnavaz, N. and deFigueiredo, R. J. P. (1988), A 3-D contour segmentation scheme based on curvature and torsion, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):707–713.
- [52] Klein, R. and Straßer, W. (1995), Large mesh generation from boundary models with parametric face representation, in: Hoffmann, C. and Rossignac, J., eds, *Third Symposium on Solid Modeling and Applications*, 431–440. ACM Press.
- [53] Kreyszig, E. (1959), *Differential Geometry*, University of Toronto Press.
- [54] Kumar, V. and Dutta, D. (1997), An assessment of data formats for layered manufacturing, *Advances in Engineering Software*, in press.
- [55] Löhner, R. (1987), Finite elements in CFD: what lies ahead, *International Journal for Numerical Methods in Engineering*, 24:1741–1756.
- [56] Ma, S. D. and Lin, H. (1988), Optimal texture mapping, in: Duce, D. A. and Jancene, P., eds, *Eurographics '88*, 421–428. Elsevier, Netherlands.
- [57] Maekawa, T. (1993), Robust computational methods for shape interrogation, PhD Thesis, Massachusetts Institute of Technology, Cambridge, MA.
- [58] Maekawa, T. and Patrikalakis, N. M. (1993), Computation of singularities and intersections of offsets of planar curves, *Computer Aided Geometric Design*, 10(5):407–429.
- [59] Maekawa, T. and Patrikalakis, N. M. (1994), Interrogation of differential geometry properties for design and manufacture, *The Visual Computer*, 10(4):216–237.
- [60] Maekawa, T., Wolter, F.-E. and Patrikalakis, N. M. (1996), Umbilics and lines of curvature for shape interrogation, *Computer Aided Geometric Design*, 13:133–161.
- [61] Maekawa, T., Patrikalakis, N. M., Sakkalis, T. and Yu, G. (1996), Avoidance of singularities and rational parametrization of pipe surfaces, *Design Laboratory Memorandum*, 96-10.

- [62] Mäntylä, M. (1988), *An Introduction to Solid Modeling*, Computer Science Press, Rockville, MD.
- [63] Mäntylä, M. (1989), Advanced topics in solid modeling, in: Purgathofer, W. and Schönhut, J., eds, *Advances in Computer Graphics V*, Springer-Verlag, Berlin.
- [64] Massey, W. S. (1989), *Algebraic Topology: An Introduction*, Springer-Verlag, pg. 136.
- [65] Miller, J. R. (1986), Sculptured surfaces in solid models: issues and alternative approaches, *IEEE Computer Graphics and Applications*, 6(12):37–48.
- [66] Moore, R. E. (1966), *Interval Analysis*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
- [67] O'Rourke, J. (1994), *Computational Geometry in C*, Cambridge University Press, New York.
- [68] Patrikalakis, N. M. and Gursoy, H. (1990), Shape interrogation by medial axis transform, in: Ravani, B., eds, *Proceedings of the 16th ASME Design Automation Conference: Advances in Design Automation, Computer Aided and Computational Design, Vol. I*, 77–88. Chicago, IL.
- [69] Patrikalakis, N. M. (1989), Approximate conversion of rational splines, *Computer Aided Geometric Design*, 6(2):155–165.
- [70] Peraire, J., Perio, J., Formaggia, L., Morgan, K. and Zienkiewicz, O. C. (1988), Finite element Euler computations in three dimensions, *International Journal for Numerical Methods in Engineering*, 26(10):2135–2159.
- [71] Peraire, J., Perio, J. and Morgan, K. (1992), Adaptive remeshing for three dimensional compressible flow computations, *Journal of Computational Physics*, 103(2):269–285.

- [72] Peters, T. J., Greenshields, I. R. and Dorney, S. M. (1992), Topological fidelity in surface reconstruction, in: *SPIE Vol. 1830 Curves and Surfaces in Computer Vision and Graphics, III*, 221–225, Boston, MA.
- [73] Piegl, L. A. and Richard, A. M. (1995), Tessellating trimmed NURBS surfaces, *Computer Aided Design*, 27(1):16–26.
- [74] Pratt, M. J., Goult, R. J. and Ye, L. (1993), On rational parametric curve approximation, *Computer Aided Geometric Design*, 10(3–4):363–377.
- [75] Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. (1988), *Numerical Recipes in C*, Cambridge University Press, New York.
- [76] Ramshaw, L. (1989), Blossoms are polar forms, *Computer Aided Geometric Design*, 6:323–358.
- [77] Ratschek, H. and Rokne, J. (1993), Test for intersection between plane and box, *Computer Aided Design*, 25(4):249–250.
- [78] Rebay, S. (1993), Efficient unstructured mesh generation by means of Delaunay triangulation and Bowyer-Watson algorithm, *Journal of Computational Physics*, 106:125–138.
- [79] Rockwood, A., Heaton, K. and Davis, T. (1989), Real-time rendering of trimmed surfaces, *ACM Computer Graphics*, 23(3):107–116.
- [80] Rogers, D. F. and Adams, J. A. (1990), *Mathematical Elements for Computer Graphics*, 3rd edition, McGraw-Hill, Inc., New York.
- [81] Sederberg, T. W. and Farouki, R. T. (1992), Approximation by interval Bézier curves, *IEEE Computer Graphics and Applications*, 15(2):87–95.
- [82] Sederberg, T. W. and Buehler, D. B. (1992), Offsets of polynomial Bézier curves : Hermite approximation with error bounds, in: Lyche, T. and Schumaker, L. L., eds, *Mathematical Methods in Computer Aided Geometric Design*, volume II, 549–558. Academic Press.

- [83] Seidel, H. P. (1993), An introduction to polar forms, *IEEE Computer Graphics and Applications*, 13(1):38–46.
- [84] Sheng, X. and Hirsch, B. E. (1992), Triangulation of trimmed surfaces in parametric space, *Computer Aided Design*, 24(8):437–444.
- [85] Sherbrooke, E. C. and Patrikalakis, N. M. (1993), Computation of the solutions of nonlinear polynomial systems, *Computer Aided Geometric Design*, 10(5):379–405.
- [86] Shimada, K. and Gossard, D. (1992), Computational methods for physically-based FE-mesh generation, in: Olling, G. and Kimura, F., eds, *Human Aspects in Computer Generated Manufacturing*, 411–420. Elsevier Science Publishers B. V. (North Holland).
- [87] Sloan, S. W. (1987), A fast algorithm for constructing Delaunay triangulations in the plane, *Advances in Engineering Software*, 9(1):34–55.
- [88] Strang, G. and Fix, G. J. (1973), *An Analysis of the Finite Element Method*, Prentice-Hall, Inc., New Jersey.
- [89] Struik, D. J. (1950), *Lectures on Classical Differential Geometry*, Addison-Wesley, Cambridge, MA.
- [90] Thacker, W. C. (1980), A brief review of techniques for generating irregular computational grids, *International Journal for Numerical Methods in Engineering*, 15:1335–1341.
- [91] Tuohy, S. T. and Patrikalakis, N. M. (1993), Representation of geophysical maps with uncertainty, in: Thalmann, N. M. and Thalmann, D., eds, *Communicating with Virtual Worlds, Proceedings of CG International '93*, 179–192. Springer, Tokyo.
- [92] Tuohy, S. T., Yoon, J. and Patrikalakis, N. M. (1995), Reliable interrogation of high-dimensional non-linear geophysical databases, in: Vince, J. A. and Earn-

- shaw, R. A., eds, *Computer Graphics: Developments in Virtual Environments, Proceedings of CG International '95*, 327–341. Academic Press, Leeds, UK.
- [93] Voronoi, G. (1908), Nouvelles Applications des Parametres Continus à la Théorie des Formes Quadratiques, Deuxième Mémoire: Recherches Sur les Paralleloedres Primitifs, *Zeitschrift für Reine und Angewandte Mathematik.*, 134:198–287.
- [94] Watson, D. F. (1981), Computing the n -dimensional Delaunay tessellation with application to Voronoi polytopes, *The Computer Journal*, 24(2):167–171.
- [95] Weatherill, N. P. (1992), Delaunay triangulation in computational fluid dynamics, *Computers Mathematics Application*, 24(5/6):129–150.
- [96] Wolter, F. E. and Tuohy, S. T. (1992), Approximation of high degree and procedural curves, *Engineering with Computers*, 8(2):61–80.