Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

11-12-2010 12:00 AM

Collision Detection and Merging of Deformable B-Spline Surfaces in Virtual Reality Environment

Harish Pungotra, *The Univeristy of Western Ontario*

Supervisor: Dr. George Knopf, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Mechanical and Materials Engineering
© Harish Pungotra 2010

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Part of the Computer-Aided Engineering and Design Commons

Recommended Citation

Pungotra, Harish, "Collision Detection and Merging of Deformable B-Spline Surfaces in Virtual Reality Environment" (2010). *Electronic Thesis and Dissertation Repository*. 32.
https://ir.lib.uwo.ca/etd/32

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

COLLISION DETECTION AND MERGING OF DEFORMABLE B-SPLINE SURFACES IN VIRTUAL REALITY ENVIRONMENT

Spine title: Collision Detection and Merging of Deformable Surfaces

Thesis Format: Monograph

by

Harish Pungotra

Graduate Program in Mechanical and Materials Engineering

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

November, 2010

© Harish Pungotra 2010

THE UNIVERSITY OF WESTERN ONTARIO
School of Graduate and Postdoctoral Studies

CERTIFICATE OF EXAMINATION

Supervisor

____
Dr. George Knopf

Co-Supervisor

____
Dr. Robert Canas

Supervisory Committee

____

Examiners

____
Dr. Ralph Buchal

____
Dr. Michael Naish

____
Dr. Roy Eagleson

____
Dr. Thenkurussi Kesavadas

The thesis by

Harish Pungotra

entitled:

Collision Detection and Merging of Deformable B-spline Surfaces in Virtual Reality Environment

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Date____________________

____
Chair of the Thesis Examination Board

ABSTRACT

This thesis presents a computational framework for representing, manipulating and merging rigid and deformable freeform objects in virtual reality (VR) environment. The core algorithms for collision detection, merging, and physics-based modeling used within this framework assume that all 3D deformable objects are B-spline surfaces. The interactive design tool can be represented as a B-spline surface, an implicit surface or a point, to allow the user a variety of rigid or deformable tools. The *collision detection system* utilizes the fact that the blending matrices used to discretize the B-spline surface are independent of the position of the control points and, therefore, can be pre-calculated. Complex B-spline surfaces can be generated by merging various B-spline surface patches using the *B-spline surface patches merging algorithm* presented in this thesis. Finally, the *physics-based modeling system* uses the mass-spring representation to determine the deformation and the reaction force values provided to the user. This helps to simulate realistic material behaviour of the model and assist the user in validating the design before performing extensive product detailing or finite element analysis using commercially available CAD software. The novelty of the proposed method stems from the pre-calculated blending matrices used to generate the points for graphical rendering, collision detection, merging of B-spline patches, and nodes for the mass spring system. This approach reduces computational time by avoiding the need to solve complex equations for blending functions of B-splines and perform the inversion of large matrices. This alternative approach to the mechanical concept design will also help to do away with the need to build prototypes for conceptualization and preliminary validation of the idea thereby reducing the time and cost of concept design phase and the wastage of resources.

Keywords: collision detection; merging B-spline surfaces; virtual reality; interactive design; modeling and simulation; deformable object; B-spline surface.

ACKNOWLEDGEMENTS

The completion of this PhD thesis and the work accomplished over the past four years would not have been the possible without the wisdom, help, and support of many whom I do not have enough words to thank.

First and foremost, I would like to thank Prof. George Knopf for giving me the opportunity to be part of his research group, for inspiring and supporting me, and for sharing his research insight and passion with me throughout this long process. I owe him many thanks for his constructive criticism, for his support and encouragement. He has been a source of inspiration in all respects and has provided me a unique prospective to face the challenges in research and life. This thesis is a direct consequence of his endless patience and support.

I would also like to thank Dr. Roberto Canas of the National Research Council of Canada - Institute for Research in Construction (NRC-IRC, London) for his remarkable guidance, encouragement, and constructive criticism throughout my research. I owe him many thanks for helping me during the implementation of algorithms and constantly reminding me to incorporate different scenarios. I am also thankful to researchers and staff of NRC-London. Special thanks to Ms. Percy Gail for editing research papers for grammatical mistakes.

Valuable guidance has been provided by the members of my advisory committee, Prof. Michael D. Naish and Prof. Samuel F. Asokanthan, whose insightful suggestions, criticisms, and encouragements helped me to focus and gave me the motivation to learn and overcome some of the challenges. On the same note, I would like to thank members of my comprehensive examination committee, Prof. Steve Feng and Prof. Jagath Samrabandu for their advice and constructive criticism and for helping me focus on critical aspects of my research.

I owe many thanks to department graduate secretaries Belle Smaill, Chris Seres, Susan Bock, and Stephanie Laurence for their assistance. It was a great pleasure working with my former colleagues Philip and Matthew as well as my colleagues Khaled and Kuldeep.

My sincere gratitude goes to Beant College of Engineering and Technology, Gurdaspur (India) for sanctioning study leave and providing financial assistance during my PhD. I owe many thanks to Dr. Satish Bansal, Dr. Nirmal Singh, Dr. Dial Chand, Mr. Inderpal Singh, Mr. Anil Kumar and Mr. Ashok Kumar for their unwavering support in sanctioning leave and providing timely financial assistance.

I do not have enough words to thank my parents and family for their unconditional love and support over the years. I also feel highly indebted to my brother and sister for their moral and intellectual support and prayers. To my wife Ritu, for always being there for me throughout this journey. To my kids, Anjali and Anuj for their love and understanding. You all have always been there for me. Thank you for everything and for encouraging me to follow my dreams.

Finally, I would like to acknowledge the financial assistance from Faculty of Graduate studies and Natural Sciences and Engineering Research Council of Canada (NSERC).

NOMENCLATURE

$\mathbf{A}_u$ = Blending matrix for the B-spline surface in u direction
$\mathbf{A}_v$ = Blending matrix for the B-spline surface in v direction
C^n = Continuity of order n for the B-spline surface
c_{ij} = Material stiffness assigned to spring between nodes i and j
D = Damping constant or damping coefficient
$[D]$ = Damping matrix
$dt = \Delta t$ = Time step
E = Young's Modulus
e = Strain
e_{ij} = Strain in spring between nodes i and j after application of force
F = Force
$f_{i,j,k}$ = Force acting on node i, j, k
f_d = Damping force
f_{ext} = External force
f_k = Force due to spring stiffness
K = Spring constant or spring coefficient
$[K]$ = Stiffness matrix
k = Degree of the B-spline surface in u direction
l = Degree of the B-spline surface in v direction
l_{ij} = New length of spring between nodes i and j after application of force
L_{ij} = Natural or rest length of spring between nodes i and j
$\mathbf{M}$ = Matrix of discrete points on the B-spline surface
m = Number of rows of matrix $\mathbf{M}$
N = Number of B-spline surfaces being merged simultaneously
n = Number of columns of matrix $\mathbf{M}$
n_{ei} = Immediate neighborhood around node i
$\mathbf{P}_{ij}$ = Matrix of control points of B-spline surface
r = Number of control points of the B-spline surface in u direction
s = Number of control points of the B-spline surface in v direction

S = B-spline surface

U = Knot vector of B-spline surface in *u* direction

u, v = Parametric directions of B-spline surface

V = Knot vector of B-spline surface in *v* direction

x = Node position of location, representing VSOFM weight points

$\dot{x}$ = Node or weight point velocity

$\ddot{x}$ = Node or weight point acceleration

v = Poisson's ratio

σ = Stress

ρ_i = Point mass of node *i*

ζ = Damping factor

TABLE OF CONTENTS

CERTIFICATE OF EXAMINATION ii

ABSTRACT iii

ACKNOWLEDGEMENTS iv

NOMENCLATURE vi

TABLE OF CONTENTS viii

LIST OF TABLES xiv

LIST OF FIGURES xvi

CHAPTER 1 INTRODUCTION 1

1.1 Problem Statement 1

1.2 Basic Terminology 4

1.3 Virtual Reality (VR) Environment 4

1.4 Haptics-based Interactive Design 5

1.5 Outline of the Thesis 7

CHAPTER 2 LITERATURE SURVEY 9

2.1 Introduction 9

2.2 Virtual Reality in Concept Design 9

2.3 Haptic Interaction with Virtual Model 12

2.4 Virtual Sculpting 15

2.5 Surface Representations 16

2.5.1 Implicit surfaces 16

2.5.2 Tessellated surfaces 18

2.5.3 Parametric surfaces 19

2.6 Collision Detection 20

2.6.1 Collision detection of rigid bodies 21

2.6.2 Collision detection of deformable bodies 22

2.7 Force and Deformation Modeling Techniques 25

2.7.1 Geometric deformation techniques 25

2.7.2 Physics-based deformation techniques 28

2.7.2.1 Mass spring method 28

2.7.2.2 Finite element method 30

2.7.2.3 Continuum method 31

2.8 Concluding Remarks 32

CHAPTER 3 COLLISION DETECTION ALGORITHM 34

3.1 Introduction 34

3.2 Background 34

3.2.1 B-spline surface 34

3.2.2 Generation of blending matrices 38

3.2.3 Bounding volume for B-spline surfaces 39

3.3 Collision Detection Algorithm 39

3.3.1 Overview 39

3.3.2 Pre-processing phase 41

3.3.2.1 Generation of blending matrices 41

3.3.2.2 Generation of convex hull 42

3.3.3 Run-time phase 42

3.3.3.1 Intersection test of convex hulls 44

3.3.3.2 Calculating minimum and maximum values of u and v 45

3.3.3.3 Surface discretization 45

3.3.3.4 Generation of spheres 47

3.3.3.5 Tessellation of surfaces... 48

3.3.3.5.1 Generation of points for tessellation.. 48

3.3.3.5.2 Triangulation of surface... 49

3.3.3.6 Triangle-triangle intersection test 50

3.3.3.7 Updating bounding volume... 51

3.4 Special Cases... 52

3.5 Analytical Comparison and Performance 52

3.5.1 Comparison with a tessellated model 52

3.5.2 Comparison with parametric surface models 57

3.6 Concluding Remarks... 58

CHAPTER 4 MERGING MULTIPLE B-SPLINE SURFACE PATCHES 60

4.1 Introduction... 60

4.2 Related Work... 61

4.3 Merging Multiple B-spline Surface Patches 63

4.3.1 Discretization of B-spline surfaces.. 64

4.3.2 Determining revised number of control points.................................... 65

4.3.3 Determining the new knot vector 66

4.3.4 Revised blending matrices... 67

4.4 Special Cases... 69

4.4.1 Intersecting and trimmed surfaces.. 70

4.4.2 Surfaces having different degrees.. 71

4.4.3 Multiple surfaces 73

4.5 Concluding Remarks... 75

CHAPTER 5 INTEGRATION OF MASS-SPRING SYSTEM WITH COLLISION DETECTION SYSTEM 76

5.1 Introduction 76

5.2 Mass Spring Damper System 77

5.3 Integration of Mass Spring System with Collision Detection 79

5.3.1 Mass spring system node generation 80

5.3.2 Deformable mesh generation 81

5.3.3 Mapping of forces to nodes of mass spring system 83

5.4 Model Deformation and Force Response 85

5.5 Concluding Remarks 89

CHAPTER 6 SIMULATION STUDIES AND PERFORMANCE EVALUATION 91

6.1 Introduction 91

6.2 Computational Efficiency of Collision Detection Algorithm 91

6.2.1 Effect of the number of control points 93

6.2.2 Effect of the maximum number of points that can be generated 97

6.2.3 Effect of the area of contact 101

6.3 Performance of B-spline Surface Patches Merging Algorithm 103

6.3.1 Robustness and accuracy of the algorithm 104

6.3.1.1 Case 1: Similar curvatures and knot vectors 105

6.3.1.2 Case 2: Different curvatures but similar knot vectors 107

6.3.1.3 Case 3: Similar curvature but different knot vectors 109

6.3.1.4 Case 4: Similar curvatures but different knot vectors and dimensions.... 110

6.3.1.5 Case 5: Intersecting and trimmed surfaces 111

6.3.1.6 Case 6: Surfaces having different degrees 112

6.3.1.7 Case 7: Multiple surfaces 114
6.3.2 Computational efficiency of the merging algorithm 114
6.4 Computational Efficiency of Physics-based Deformation Algorithm 116
6.5 Computational Efficiency of the Interactive Design Framework 117
6.5.1 Pre-processing phase 118
6.5.2 Run-time phase 120
6.6 Concluding Remarks 121
CHAPTER 7 DEFORMABLE MODELS FOR INTERACTIVE DESIGN AND USER TRAINING 123
7.1 Introduction 123
7.2 Conceptual Product Design 124
7.3 Interactive Design of an Ergonomic Spoon 126
7.3.1.1 Spoon for eating food 129
7.3.1.2 Spoon for serving food 136
7.4 Evaluation of Model and User Training 137
7.4.1 Evaluation of model 138
7.4.2 User training 142
7.5 Concluding Remarks 145
CHAPTER 8 CONCLUSIONS AND RECOMMENDATIONS 146
8.1 Review of Methodologies Developed for Deformable Modelling 146
8.2 Novel Features of the Proposed Method 147
8.2.1 Blending matrices 147
8.2.2 Collision detection algorithm 148
8.2.3 B-spline surface patches merging algorithm 150

8.2.4 Integration of mass spring system 151
8.3 Other applications 151
8.4 Recommendations to Resolve Limitations.. 152
8.5 Future Work 153
8.6 Final Remarks 154
BIBLIOGRAPHY 156
APPENDIX A: ERROR ANALYSIS OF THE MERGED B-SPLINE SURFACE 168
A.1 Introduction 168
A.2 Case 1: Similar Curvatures and Knot Vectors 168
A.3 Case 2: Different Curvatures but Similar Knot Vectors 169
A.4 Case 3: Similar Curvature but Different Knot Vectors 171
A.5 Case 4: Similar Curvatures but Different Knot Vectors and Dimensions 172
A.6 Case 5: Intersecting and Trimmed Surfaces... 173
A.7 Case 6: Surfaces having Different Degrees.. 174
A.7 Case 7: Multiple Surfaces 176
A.8 Concluding Remarks 176
CURRICULUM VITAE 178

LIST OF TABLES

Table 3.1 Comparison of computational cost of collision detection for tessellated model with the proposed algorithm. 54

Table 6.1 Computational time for collision detection of B-spline surface having different number of control points with a point, a sphere, and a plane. 94

Table 6.2 Computational time for collision detection of B-spline surface having different number of the maximum points generated at lowest level of detail with a point, sphere, and plane. 98

Table 6.3 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm. 106

Table 6.4 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm. 112

Table 6.5 Computational time of the algorithm for merging B-spline surface patches. 115

Table 6.6 Update time for single iteration of mass spring system for different node sizes. 117

Table 6.7 Pre-processing time for generation of mass spring system for different node sizes. 119

Table A.1 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for surface having similar curvatures and knot vectors. 169

Table A.2 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for surface having different curvatures but similar knot vectors. 170

Table A.3 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for surface having similar curvature but different knot vectors. 171

Table A.4 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for surface having similar curvatures but different knot vectors and dimensions. 173

Table A.5 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for intersecting and trimmed surfaces. 174

Table A.6 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for surface having different degrees. 175

Table A.7 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for multiple surfaces. 176

LIST OF FIGURES

Figure 1.1 Basic architecture of a physics-based haptic system. 2

Figure 1.2 Schematic representation of a haptic interaction with a virtual object. 6

Figure 2.1 Haptic interaction with a virtual model through a haptic device (PHANTOM® Omni of SensAble Technologies, 2008) located at the University of Western Ontario. 13

Figure 3.1 A typical blending matrix, A_u. 37

Figure 3.2 Discretized B-spline surface...38

Figure 3.3 Flowchart of the proposed algorithm. 40

Figure 3.4 Illustrations showing the major steps in the proposed algorithm. 43

Figure 3.5 Selecting intermittent blending matrix for generation of points within the selected range of parameter u. 46

Figure 3.6 Generation of spheres using the points generated on the B-spline surface. 47

Figure 3.7 Triangulation using the points inside intersecting spheres........................50

Figure 3.8 Comparison of steps for collision detection for tessellated model with proposed algorithm...53

Figure 3.9 Comparison of total computations required in the worst case scenario for collision detection of tessellated model versus the B-spline model using proposed algorithm. 55

Figure 3.10 Maintaining lower number of computations required for collision detection by changing resolution of the surface. 56

Figure 4.1 (a) Two B-spline surface patches matched with C^1 continuity (b) Surfaces after one surface patch is deformed. 62

Figure 4.2 Flow chart of the proposed algorithm for merging two B-spline surface patches. 64

Figure 4.3 (a) B-spline surface patches that are to be merged together (b) Discretized points on the surface, as generated by using the blending matrices. 65

Figure 4.4 The effect of multiple knots at the common edge of the merging surfaces. 67

Figure 4.5 (a) Intersecting surfaces to be merged (b) Region of the surface selected for discretization (The points shown in red are discarded). 70

Figure 4.6 (a) Initial B-spline surface patches of order 4×4 and 5×6 respectively (b) Merged surface of order 5×6 with C^4 connectivity in v direction, generated by proposed algorithm. 72

Figure 4.7 (a) Initial multiple B-spline surface patches (b) Merged surface with C^2 connectivity generated by proposed algorithm. 74

Figure 5.1 Voigt model of mass spring damper system. 77

Figure 5.2 Flow chart for mass spring mesh generation and its integration with collision detection algorithm. 80

Figure 5.3 (a) B-spline surface deformable model (b) Discrete points as nodes for the mass spring system. 81

Figure 5.4 Generation of mass spring system. 83

Figure 5.5 A tool colliding with a deformable B-spline surface model. 84

Figure 5.6 Mapping of the sculpting force acting on B-spline surface to nodes of mass spring system. 84

Figure 5.7 Hexahedral mesh and the node (mass) spring representation. 85

Figure 5.8 Illustration of interaction of the tool with the model. 88

Figure 5.9 The deformation of the model having properties of plasticine (spring stiffness = 13 MN/m^2, density = 2500 kg/m^3) while applying different forces (F). 88

Figure 5.10 The deformation of the model with different spring stiffness, and constant force (30 N) while pulling out the nose of the artifact. 89

Figure 6.1 Primitives (plane and sphere) and B-spline surfaces (a donut and a distorted donut) for calculating time of collision detection. 92

Figure 6.2 Variation of the computational time of collision detection for a point, sphere, and plane with respect to the number of control points used. 95

Figure 6.3 Variation of the computational time (ms) of collision detection for a B-spline surface with another deformable B-spline surface for different number of control points. 97

Figure 6.4 A sphere colliding with (a) A tear shaped B-spline surface (b) A deformed tear shaped B-spline surface. 98

Figure 6.5 Variation of the computational cost of collision detection for a point, sphere, and plane with respect to the maximum number of points generated. 99

Figure 6.6 Variation of the computational time(ms) of collision detection for a B-spline surface with another deformable B-spline surface with respect to the maximum number of points generated. 100

Figure 6.7 Variation of the computational time (ms) of collision detection for a B-spline surface with a sphere, plane, and another deformable B-spline surface for different percentage area of contact. 102

Figure 6.8 Variation of the computational time (ms) of collision detection for a B-spline surface with a sphere, a plane, and another deformable B-spline surface for different percentage area of contact, when interrupting the algorithm from going to lower levels of detail. 103

Figure 6.9 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by Rhino® (c) Merged surface with C^2 connectivity generated by proposed algorithm (d) Merged surface with C^0 connectivity generated by proposed algorithm. 105

Figure 6.10 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by Rhino® after matching the surfaces (c) Merged surface with C^2 connectivity generated by proposed algorithm (d) Merged surface with C^0 connectivity generated by proposed algorithm (e) Point set deviation for the C^2 surface generated by Rhino® after matching the surfaces (f) Point set deviation for the C^2 surface generated by proposed algorithm. 107

Figure 6.11 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by Rhino® with increased knots (c) Merged surface with C^2 connectivity generated by proposed algorithm (d) Merged surface with C^0 connectivity generated by proposed algorithm. 110

Figure 6.12 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by the proposed algorithm. 111

Figure 6.13 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by proposed algorithm. 111

Figure 6.14 (a) Initial B-spline surface patches (b) Merged surface of order 5×6, with C^4 connectivity in v direction, generated by Rhino® with increased knots (c) Merged surface of order 5×6 with C^4 connectivity in v direction, generated by proposed algorithm (d) Merged surface of order 4×4 with C^2 connectivity generated by proposed algorithm...113

Figure 6.15 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by proposed algorithm. 114

Figure 6.16 Pre-processing time for calculating the mass spring mesh of different sizes. 120

Figure 6.17 (a) Deformable B-spline model (b) Points for collision detection (c) Mass spring mesh. 121

Figure 7.1 Integration of various modeling techniques proposed in this thesis to develop B-spline surface based interactive design module for various applications in virtual reality environment. 124

Figure 7.2 Photograph of typical commercially available spoons. Product designs mainly focus on contemporary style, ease of use, and comfort of the user. 126

Figure 7.3 Generally accepted motions (a) Holding a spoon and (b) Rotating wrist while eating (c) Rheumatoid arthritis restricts movements of fingers and wrist, modified picture from [Joint, Pain Solutions, 2010] 127

Figure 7.4 Different methods to generate initial design represented as a B-spline surface. 128

Figure 7.5 Parts of a spoon. 129

Figure 7.6 (a) Handle of spoon from the initial design (b) A plane is used to fix nodes on one side of the plane (opposite to the direction of normal vector) and a sphere is used to apply force. 130

Figure 7.7 (a) Mass spring mesh for the spoon handle (b) Fixed nodes (green) and the nodes colliding with the sphere tool (red) (c) Side view the B-spline model showing fixed and colliding nodes. 131

Figure 7.8 Deformation of handle under the application of external force with one end fixed. 132

Figure 7.9 Modified shape of spoon to accommodate lack of wrist movement. 133

Figure 7.10 Modifying the shape of spoon handle to grip it with all the fingers of hand. 134

Figure 7.11 Merging bowl and modified spoon handle to generate the B-spline model of a of spoon. 135

Figure 7.12 Investigation of different shapes of a spoon for eating food to accommodate impaired wrist movements or weak grip of fingers. The original design is shown in (a) and design modifications are presented from (b) to (d). 136

Figure 7.13 Investigation of different shapes of a serving spoon to accommodate impaired shoulder movements. The original design is shown in (a) and design modifications are presented in (b) and (c). 137

Figure 7.14 (a) B-spline model representing food (jelly) (b-c) Mass spring mesh to incorporate material properties to the model. 138

Figure 7.15 Simulation of food (jelly) in a spoon without tilting it. 139

Figure 7.16 Simulation of food (jelly) in a spoon when the bowl is tilted by ten degree. 140

Figure 7.17 Simulation of food (jelly) in a spoon when the bowl is tilted by twenty degree. 141

Figure 7.18 Simulation of food (jelly) in a spoon when the spoon is tilted by different degrees. 143

Figure 7.19 Simulation of food (jelly) in a spoon when the spoon is tilted by different degrees and the hand of the user is shaking. 144

CHAPTER 1 INTRODUCTION

1.1 Problem Statement

During the normal design phase, the designer has the freedom to generate and explore ideas without being constrained by parameters that exist at later design stages. At the conceptual stage, if a large number of ideas can be created, modified or analyzed; there will be more chances of finding the best possible design.

Modern computer-aided design (CAD) systems and software tools have played a significant role in improving the efficiency of the overall product design process, ensuring geometric accuracy and the exchange of product model data. However, the impact of these technologies is largely restricted to the detailed modeling and engineering analysis that occurs during the embodiment design phase. Conceptual design has not benefited from these sophisticated and highly precise software tools to the same degree because the creative activities associated with developing and communicating potential solutions with minimal detail is far less formulaic in its implementation. At the early stages of product design the specifications and constraints have not been fully established. The industrial designers and engineers need the freedom to change and modify the product configuration and mechanical behavior to investigate a wide range of alternative solutions. Any CAD system that seeks to support and enhance interactive 3D free form modeling must, therefore, enable natural and haptic modes of human-computer interaction. Therefore, industrial designers and engineers continue to seek new tools that provide them with the freedom to artistically modify product concepts. The need for a viable VR-based conceptual design tool comes from several case studies [Cheshire, 2001; Sener, 2002; Ye, 2006a; Ye, 2006b]. One key conclusion derived is that the human-computer interface and related software tools for interacting with the virtual models must be intuitive to the user, provide sensory feedback during design, and mimic the natural way that the consumer would interact with the product concepts that are being created. Recent advancements in high-speed, multi-core computer hardware and virtual reality (VR) technology provide opportunities to link the more fluid processes of creative conceptual design with the rigidly defined tasks of product detailing and engineering analysis. The objective of this thesis is to develop tools for a framework for an interactive design module in virtual reality environment. This will enable a designer to have natural and haptic modes of human-computer interaction for modeling and validating the conceptual designs. The proposed technique provides rapid verifications of the early design, before exchanging the information with commercially available CAD/CAM software to carry out detailed analysis and generate the final design.

Figure 1.1 shows the basic architecture of a typical physics-based haptic system. The haptic device works as an interface between the real and the virtual worlds. A collision detection algorithm then provides the contact information of the virtual tool with the object. The haptic device takes the force exerted by the user, converts it into appropriate signals and transmits the signal to the virtual object so as to make it deform (deformable bodies) or show resistance (rigid bodies). The force exerted by the user and position of interaction with the object are used by a physics-based model to calculate the deformation of the object and reactive forces to be sent back to the user.

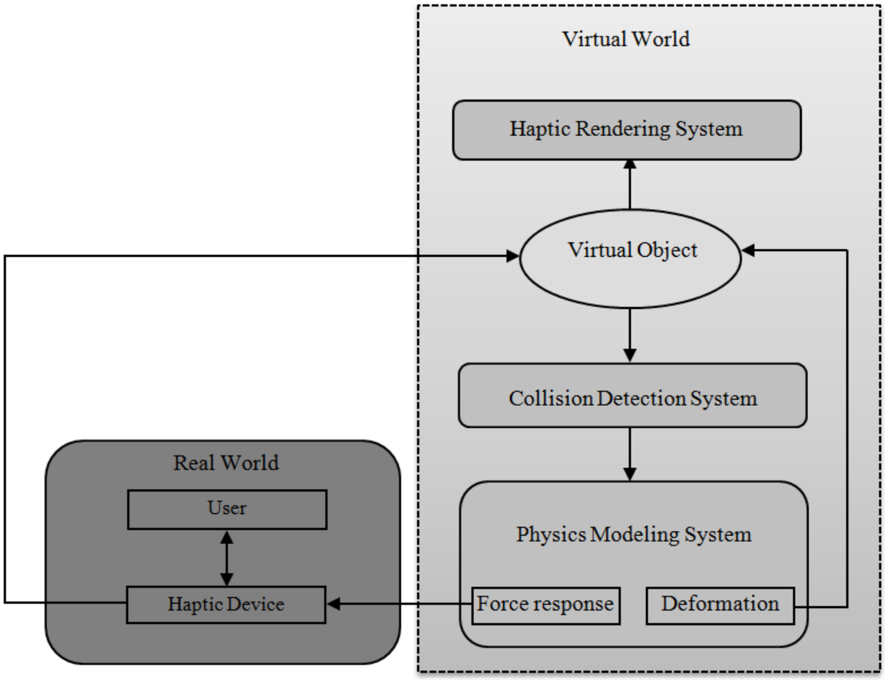


Figure 1.1 Basic architecture of a physics-based haptic system.

Collision detection is an active research topic in engineering, computer graphics and virtual reality (VR) [Jimenez, 2001]. Collision detection is the necessary step before haptic interaction can be achieved. An efficient collision detection algorithm plays a very significant role towards achieving real-time haptic interaction. Most of the collision detection algorithms, available in literature, are mostly for rigid bodies. These rigid bodies are mostly represented as lower order implicit surfaces (spheres, cylinders, cones etc.) or tessellated surfaces. For applications in haptic-based modeling, mostly tessellated bodies are used, and many constraints are imposed on how the virtual model can deform.

Shape modification of a virtual object can be simulated using either geometric- or physics-based algorithms [Basdogan, 1998]. Geometric techniques only adjust the vertices of the underlying mesh model in response to external forces. The reaction force is typically determined using Hooke's law where the depth of haptic tool penetration is calculated based on the current and home positions of the node that is nearest to the contact point. The concept was originally suggested by [Sederberg, 1986] and was further extended by [Basdogan, 1998; Basdogan, 2004] for applications in medical simulation. This technique, though computationally efficient, cannot determine real behavior of a multiple-material or non-homogeneous material virtual model. Physics-based models, on the other hand, are able to both estimate the direction and magnitude of nodal movement based on realistic material properties and the external forces introduced to the model through the haptic tool.

In the past decade, B-spline/NURBS representation has become the standard for CAD/CAM systems. Thus it is imperative that any haptic interactive design module should utilize B-spline surfaces to represent the virtual model in order to streamline the exchange of information with existing CAD/CAM systems. A major obstacle in using B-spline surfaces to represent a deformable model is the absence of an efficient algorithm to detect collision between two or more B-spline surfaces having complex surface. For applications in interactive 3D free form modeling using haptics based virtual reality environment, a collision detection algorithm must be capable of tackling complex surfaces, large areas of contact, multiple contacts, and high deformations.

1.2 Basic Terminology

Conceptual Design is the early identification and generation of ideas for a design solution. *Virtual Reality (*VR) is a technology that allows a user to interact with a computer simulated environment representing a real or imaginary word. *Haptics* means pertaining to touch. Haptic technology allows the user to interact with a virtual model via a *Haptic Device* which works as an interface between real and virtual world. The user feels the sense of touch through the application of force, vibrations, motion and resistance via a haptic device. Haptics has applications in the field of medicine (training for laparoscopic surgery), games, art, shape design and user training.

There are three distinct features that any haptics-based interactive design module must have. The *Haptic Rendering System* is used for the visualization of the model and the tool. *Haptic Rendering* includes graphics animation and haptic feedback. Through haptic rendering, a user can feel and manipulate a virtual object using a haptic device. A *Collision Detection* algorithm calculates the position and extent of the interference of the virtual model(s) and the tool. A *Physics-Based System* uses the information provided by the collision detection algorithm to determine the deformation of the model(s) and the resultant reactive forces to be fed back to the user. *Force Feedback* is provided through haptic devices. The nature and amount of force feedback depends upon the haptic device used, and the underlying algorithm used to calculate the force feedback.

1.3 Virtual Reality (VR) Environment

Virtual reality provides real-time interaction with virtual world through several communication methodologies including visual (computer screen or stereoscopic display), tactile (force feedback) and audio (stereo sound) feedback. The overall goal of the virtual reality is to provide a far more natural environment to the user than that is possible by workstations. This can help to enhance the creativity of the user and increase productivity. This environment is especially suitable for free form shape design, wherein an industrial designer or engineer can explore all conceivable options without the constraints imposed by commercial CAD/CAM environments.

The goal of efficiently integrating sensory-motor functionalities and skills within a VR system poses extraordinary challenges for researchers and engineers in the field. There are many bottlenecks in efficient use of the virtual reality environment. Firstly, the real-time rendering of the complex word during simulation is not advanced enough for industrial applications. In the real word, people can use vision to estimate the distance between objects, and if, these are far or closer to them. The quality of the graphics has improved, of late, but still it has not reached the point of maturity.

The second bottleneck is in the implementation of collision detection algorithms. At present most of the algorithms are limited to particular applications. Most of these algorithms are for rigid bodies. When applied to deformable objects having complex surface, it may not be possible to get real-time interaction.

The third bottleneck is in providing physical properties to the virtual models. The sense of touch and force feed mechanisms allow the user, a rich experience in virtual world. However, to simulate the behavior of a real object, the simulation must include object rigidity/strength, mass, friction, surface texture, and heat transfer. Adding these physical characteristics to virtual objects require powerful computing hardware and efficient algorithms.

However, virtual reality development is a fast growing area in computer graphics and engineering. Already, it is being used for training for laparoscopic surgery and games. The interactive design through virtual reality promises to be very intuitive, creative, and cost effective method. With better integration with existing CAD/CAM software, it can provide a very effective way for the industrial designers and engineers to exploit their creativity. One way to achieve it is by using B-spline surfaces to represent virtual model and tool. The development of efficient collision detection and mass spring system to incorporate physical properties is described later in this thesis.

1.4 Haptics-based Interactive Design

Various studies [Sener, 2002; Ye, 2006a] suggest that a haptics-based interactive design system is well suited to exploit the creativity of an industrial designer or engineer. It would make it possible to artistically modify and quickly evaluate different concepts. The user gets real-time visual and force feedback, while interacting with the virtual model. Haptics allows the user to benefit from the natural way of working with the model, without the constraints imposed by commercially available CAD/CAM software.

During traditional concept design phase, prototypes are sometimes built for conceptualization and preliminary validation of the idea which increases the time and the cost of the concept design phase and leads to the wastage of natural resources. The 3D visualization capabilities and the ability to directly interact with physics-based models using haptic tools suggest that Virtual Reality (VR) environments can provide new opportunities to assist the creative design process. Interactive modeling techniques, using the virtual reality, have rapidly evolved in recent times. This provides the designer, an insight to the physical characteristics of the concept, which can be evaluated before going in for a detailed design.

Figure 1.2 shows the schematic representation of a general framework for a haptic interaction with a virtual object.

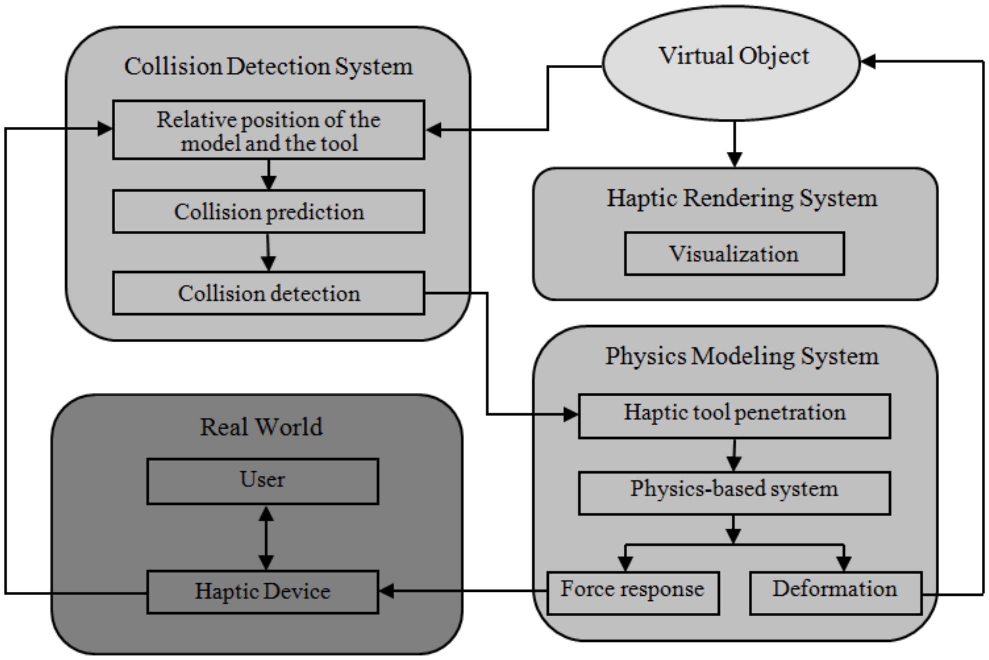


Figure 1.2 Schematic representation of a haptic interaction with a virtual object.

There are three distinct features to enable any haptics interaction with a virtual object. As the user interacts with virtual object, the relative positions of the virtual tool and the object are calculated. The *collision detection system* provides the information regarding the details of the contact of the tool with the model and the tool penetration. The *physics-based system* uses the information provided by the collision detection algorithm to determine the deformation of the model(s) and the resultant reactive forces to be fed back to the user. The *haptic rendering system* is used for the visualization of the model and the tool.

1.5 Outline of the Thesis

The literature review, on virtual reality-based concept design, haptic interaction, virtual sculpting, surface representation techniques, collision detection and basic geometric- and physics-based modeling approaches, is discussed in Chapter 2. The chapter also summarises the approaches for haptics based shape design and development of deformable surface and solid models. The techniques discussed in the chapter includes collision detection of rigid and deformable bodies; implicit, tessellated and NURBS surface representation of virtual models; mass spring damper systems and finite element methods. The chapter also presents various types of surface representations of virtual models along with their relative merits and demerits and reasons of choosing B-spline surface representation for the haptic-based interactive design framework.

Chapter 3 introduces the collision detection algorithm for B-spline surface patches. The algorithm is evaluated for its efficiency using the Big ***O*** notation for worst case scenario. The algorithm is further developed for merging multiple B-spline surface patches in Chapter 4. The development of initial mass spring damper system to introduce material properties to the virtual model is discussed in Chapter 5. It also presents the integration of the mass spring system with the collision detection system so as to achieve denser high computational efficiency. Chapter 6 takes a closer look at the efficacy of the algorithms developed for collision detection through simulation tests and performance. It verifies the efficiency and robustness of the collision detection, B-spline surface patches merging, and physically-based deformation model algorithms. The collision detection algorithm, merging of B-spline surface patches and deformation and force response of the mass spring system are combined for the haptics-based interactive design framework. The performance of this integrated module is also evaluated in this chapter. Chapter 7 presents deformable shape medications and user training in VR space. Chapter 8 summarizes the performance of collision detection, B-spline merging algorithm, and mass spring system. It also provides the limitations of the proposed system, and recommendations for the improvement and future work.

CHAPTER 2 LITERATURE SURVEY

2.1 Introduction

In addition to realistic three dimensional graphics, a virtual reality environment based interactive design must support visual object collision detection, physics-based modeling, and haptic manipulation. The *collision detection* sub-system provides detailed information about when and how multiple virtual objects make contact and interact within the VR space. The *physics-based* sub-system uses the information provided by the collision detection algorithm to determine the reactive forces to be fed back to the user and degree of deformation of any non-rigid elastic and plastic objects. Finally, the *haptic rendering* sub-system is used for tactile and visual interaction with virtual objects and tools used during the creative design exercise.

This chapter discusses the related research work involved in various aspects of a interactive design module. To better understand the idea behind the development of techniques for VR based interactive design and specific contribution of this thesis, it is necessary to review previous techniques used and their relative merits and demerits. Much of the work in rendering, sculpting, and collision detection has been in the field of computer graphics with applications in games and medical field in mind. Most of these techniques use tessellated surfaces for representing rigid bodies or deformable bodies with several constraints. The algorithms for rendering, collision detection, haptic interaction, and physics based system depend upon the type of surface representation of the models and tools in virtual reality environment.

2.2 Virtual Reality in Concept Design

The rapid advancement of virtual reality (VR) technology has led to the development of a variety of applications in computer graphics, gaming and entertainment, surgical training, engineering analysis, and industrial design. Specifically, virtual reality promises to be a very intuitive, creative and cost effective method for concept design [Cheshire, 2001; Sener, 2002; Ye, 2006a; Ye, 2006b]. The most prominent characteristic of virtual reality systems concerns multimodal (real-time) sensory-motor interaction between the human operator (user) and the virtual environment. Such a natural and intuitive human/computer interaction should involve all the sensory channels of the human being.

During conceptual design, product specifications are not fully established and designers have significant freedom to change and modify the product configuration so as to meet the design requirements. Industrial designers tend to make extensive use of physical models created with their hands as it is natural and intuitive process. Unfortunately, current CAD and geometric modeling systems lack a natural and intuitive human-computer interface.

Initially, VR-enhanced 3D visualization and analysis systems were used, such as Virtual Design II [Astheimer, 1995], and ISAAC [Mine, 1997]. In these systems, the product models are initially created in existing 3D CAD systems and then appropriately translated into a VR environment. Such systems only permit designers to visualize and analyze CAD objects in a 3D virtual environment. Designers cannot directly create or modify pre-existing CAD models and so when any change or modification is required, they must go back to the conventional CAD systems.

A VR-based concept design system can provide industrial designers with more familiar interactive capabilities for creating and representing their design intent easily, flexibly and efficiently on computers [Ye, 2002]. Compared to conventional CAD systems, a VR-based CAD system allows more tools for the designers to perform various design activities. The COnceptual VIRtual Design System (COVIRDS) is a VR-based CAD modeling environment that allows rapid shape creation by using a bi-modal, voice, and hand tracking interface [Chu, 1997; Dani, 1997]. It provides parametric and free form design modes. Mouse/keyboard interface is replaced with voice recognition and 3D interaction devices. A voice command interface has several advantages including its simple input device (a microphone) and freedom to use hands for other operations. However, it also suffers from fundamental weaknesses including limited recognition capability and difficulty in specifying continuous and complex commands [Ye, 2005]. Many VR-based design system are reported in the literature such as 3-Draw [Sachs, 1991], 3DM [Butterworth, 1992], DesignSpace [Chapin, 1994], CDS [Bowman, 1996], and CUP [Anthony, 2001]. The 3DM allows the designer a better feel for the object's appearance in VR environment through a head mounded display. 3DM includes several grid and snap functions. It however lacks many other aids and constraints that are necessary to accomplish precise work. All of these techniques provide the designer with real-time interaction with the virtual object. However, each of these VR-based interaction techniques for CAD applications has its own potential and limitations. There is a limitation on the size of the model. When geometries get complex, a time lag sets into the system. Fully immersive design systems can create a more realistic environment but these often tend to make the system infrastructure more complex, cause uncomfortable intrusive viewing problems and make the system computationally expensive.

The Loughborough University Conceptual Interactive Design (LUCID) system [Ye, 2005; Ye, 2006a] was developed to integrate VR-based Human-Computer Interfaces (HCIs) into the design process in order to maximize its interactivity and efficiency so as to provide better support to conceptual design. It used a six degree of freedom (DOF) SpaceMouse Classic® input device from 3Dconnexion Corp. to create a two-handed operation mechanism. A 3D Phantom Desktop® haptic device from SensAble Technologies Inc. was used to implement haptic interaction. A NuVision GX60® stereoscopic display toolkit from MacNaughton Inc. was used to offer a stereoscopic display interface. A universal computer-supported speaker-based auditory system was employed to provide a sound feedback interface. This system can allow users to experience 3D haptic force feedback from the 3D Phantom Desktop® haptic device and navigate the virtual model through six DOF SpaceMouse Classic® input device.

Weidlich *et al.* [Weidlich, 2007] focused on integrating VR as a user interface into the process of geometric modeling and detailing. It presents three paths towards a solution: VRAx®, Navigation Interface for Modeling (NavIMode), and Construct|Tool.

Most of the preliminary work focused on the user interfaces and the modeling tools for the designers. Sener *et al.* [Sener, 2002] conducted several case studies that pointed out the expectations of the designers and industrial engineers. The designers expected that any interactive design framework must mimic the natural way designers interact with the physical world and provide direct sensory feedback during the interaction. Current

CAD systems do not fulfill these expectations. Robinson *et al.* [Robinson, 2007] evaluated Co-Star, an immersive stereoscopic system for cable harness design. Overall, the results obtained and the positive experience of the participants indicated that 3D immersive design and direct body motion tracked interfaces did provide a very intuitive, easy to use, and useful addition to the technologies available to design engineers. Bourdot *et al.* [Bourdot, 2010], presented an approach for the integration of Virtual Reality (VR) and Computer-Aided Design (CAD) by developing a VR-CAD framework. The framework allowed intuitive and direct 3D edition on CAD objects within virtual reality environments by combining the VR-CAD framework with multimodal immersive interaction (using 6 DoF tracking, speech, and gesture recognition systems) to gain direct and intuitive deformation of the objects' shapes within a VR environment. There are more research groups [Duriez, 2006; Gironimo, 2006] using virtual reality for concept design. Overall there is high demand for a virtual concept design tools in industry.

2.3 Haptic Interaction with Virtual Model

In the context of virtual reality applications, haptics is a force feedback technology which allows a user to use his/her sense of touch while interacting with a virtual model. By using haptics devices, the user can interact with a virtual model by feeding and receiving information through tactile/kinaesthetic sensation.

Figure 2.1 shows the user interacting with a virtual model through a haptic device. The haptic sense is usually divided into two main distinct sensory modalities. The first sense is the kinaesthetic sense (motion and force sensing), which includes perception of muscular effort. The second sense is the tactile sense, which provides cutaneous information, related to contact between the skin of the human body and the external environment (pressure, vibration, temperature etc.). These sensory interactions enable the user to perceive physical properties such as rigidity of the model and the surface characteristics of model (roughness etc.).

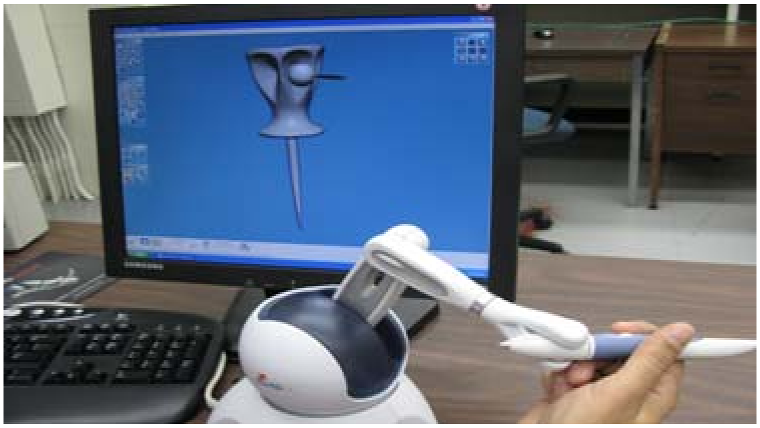


Figure 2.1 Haptic interaction with a virtual model through a haptic device (PHANTOM® Omni of SensAble Technologies, 2008) located at the University of Western Ontario.

Bloomenthal and Shoemaker [Bloomenthal, 1991] used implicit geometry techniques to represent clay-like objects proposed. A convolution surface was proposed as a natural and powerful extension to point-based field surfaces which was obtained by convolving a skeleton. In principle, this can comprise points, line segments, curves, polygons, or other geometrical primitives. This approach overcomes the drawback of bulges and curvature discontinuity in distance surfaces. Convolution surfaces offer many desirable advantages, such as intuitive shape design, well-behaved blending and fluid topology changes with the underlying skeleton. Computer vision research has shown that any 3D object can be defined entirely from a geometric skeleton [Attali, 1997], which implies that skeletons are natural abstractions for 3D objects. Convolution surfaces provide us with a means to control the shape of an underlying modeling object by controlling its skeleton, just as controlling a parametric surface by manipulating its control vertices. A major drawback of this approach is that the cost of field evaluation grows with the number of primitives. During sculpting, if the actions of the user results in creation of a new primitive, the field evaluation would quickly become prohibitive and interaction will become sluggish. The mathematical formulations of convolution surfaces also pose some open problems because there are limited choices of kernel functions and skeletal primitives that can be convolved together analytically. By using the superposition property of convolution surfaces and the separable property of Gaussian filters, Bloomenthal and Shoemaker [Bloomenthal, 1991] calculated field functions numerically based on a point-sampling method, which unfortunately implies potential under-sampling artifacts and storage problems. McCormack and Sherstyuk [McCormack, 1998] addressed this weakness by employing a new kernel function, called Cauchy function, and were able to deduce analytical solutions for several useful primitives, namely, points, line segments, polygons, arcs, and planes.

Field-based implicit surfaces have become an increasingly popular modeling approach [Bloomenthal, 1997a; Cani-Gascuel, 1997]. Their implicit representations, which have smooth-blending properties, make them convenient for modeling and animating smooth objects of complex topology that may change over time. Examples of such objects are liquids, clouds, plants, sea-life forms, and other organic shapes. In addition to object modeling, implicit surfaces have gained acceptance in other applications, namely, shape morphing [Turk, 1999b], surface reconstruction [Savchenko, 1995], natural phenomena simulation [Dobashi, 2000; Nishita, 1997], and space deformation [Jin, 2000]. Since implicit surface can produce visually striking special effects, they have become a powerful tool for animators.

Witkin *et al.* [Witkin, 1998] used a physically based particle approach to sample and control implicit surfaces. On the other hand, Raviv and Elber [Raviv, 2000] presented an interactive sculpting algorithm that used a set of uniform trivariate B-spline functions as the underlying representations. Martin *et al.* [Martin, 2001] used a trivariate spline based mathematical framework to represent and extract volumetric attributes. Park and Kunwoo [Park, 1997] used high dimensional NURBS representation for analyzing and visualizing fluid flow data. Schmitt *et al.* [Schmitt, 2001] presented an approach to constructive modeling of FRep solids defined by real-values functions using 4D uniform rational cubic B-spline volumes as primitives. The first three coordinates are used to represent the spatial component of the volume to be sculpted and the fourth coordinate corresponds to volume density. Knopf and Sangole [Knopf, 2002] investigated the *Self Organization Feature Map* (SOFM) as the starting point for haptic interaction. This technique has also been extended for use in surface fitting [Knopf, 2004], geometric parameterization [Knopf, 2003], and visual exploration of numerical data [Knopf, 2007].

Zhong *et al.* [Zhong, 2005] presented a methodology for solid modelling in a virtual reality (VR) environment in an intuitive manner through constraint-based manipulations. The data model integrates a high-level constraint-based model for precise object definition, a mid-level Constructive Solid Geometry/Boundary representation (CSG/Brep) hybrid solid model for hierarchical geometry abstractions and object creation, and a low-level polygon model for real-time visualization and interaction in the VR environment.

2.4 Virtual Sculpting

Mathematically, virtual sculpting refers to the dynamic manipulation of virtual object by the user to generate different shapes. Galyean and Hughes [Galyean, 1991] presented interactive modeling technique based on the notion of sculpting a solid material. A sculpting tool is controlled by a 3D input device and the material is represented by voxel data. The tool acts by modifying the values in the voxel array and particular attention was made to prevent aliasing when tool was re-sampled into the field grid. The tool was able to remove material as well as smoothen the surface through convolution. Wang and Kaufman [Wang, 1995] further extended haptic interaction to carving and sawing. The affected regions are indicated directly on the 2D projected image of the 3D model. The carving tools are pre-generated using a volume sampling technique and stored in a volume raster of $20 \times 20 \times 20$ resolution. Avila and Sobierajski [Avila, 1996] used a force feedback articulated arm to command a tool in a similar context. However, the tool size was limited to 3-5 voxels because in order to meet the requirements of the system, the contents of each voxel must contain a large number of physical properties. This includes a scalar value for density, values for material classification and shading properties, as well as values for mechanical properties such as stiffness, and viscosity.

Alternative data structure to represent virtual sculpting has also be proposed such as, voxel-based system [McNeely, 1999], a voxel-based system with iso-surface extraction [Ferley, 2000], B-spline surfaces [Dachille, 1999], and sub-division surfaces [Gregory, 2000b]. These are suitable for low to moderately complex virtual models. Research is underway to improve computational efficiency and flexibility of haptic sculpting. Multi-resolution surfaces have also been used to reduce computational cost of haptic interaction, particularly of collision detection. Baerentzen [Baerentzen, 1998] proposed an octree-based representation to accelerate ray-casting. The algorithm can be extended by allowing voxels to be inserted at different levels in the octree. This way, a sparsely represented voxel raster with *dynamic resolution*, can be obtained. Raviv and Elber [Raviv, 2000] proposed a hierarchical approach based on the combination of trivariate B-spline volumes to represent the field. This allows sculpting at different levels of details and arbitrary orientations. The presented approach provides the user with intuitive sculpting abilities in an interactive speed for modeling arbitrary geometries and/or topologies. Ferley *et al.* [Ferley, 2001] proposed a sculpture metaphor based on a multi-resolution volumetric representation. It allows the user to model both precise and coarse features while maintaining interactive updates and display rates. The modeled surface is an iso-surface of a scalar field, which is sampled on an adaptive hierarchical grid that dynamically subdivides or un-divides itself. Gao and Gibson [Gao, 2006] used multi-resolution B-spline surfaces to reduce the computational cost of haptic interaction. Though the multi-resolution techniques reduce the computation cost of haptic interaction, the haptic force to be fed back to the user is not realistic.

2.5 Surface Representations

A variety of techniques is available for representing the surface of a virtual object representing a model or tool. Each technique has its advantages and disadvantages with respect to other techniques. Before selecting any particular type of representation, it is necessary to compare the characteristics of these techniques and how these would fit in the overall picture of concept generation. Some of the prominent techniques are discussed in detail in the following subsections.

2.5.1 Implicit surfaces

Implicit surfaces are two-dimensional geometric shapes that exist in three-dimensional space and are defined in a particular mathematical form [Liu, 2006]. This type of representation scheme is also called volumetric method. Implicit surfaces can be generated using analytical, variational and multi-level partition of unity (MPU) approach. Implicit surfaces use 3D volumetric methods, 3D implicit primitives (e.g. blobby), level sets and radial basis functions to represent surfaces by superposition of weighted basis function. A consistent, smooth and water-tight surface can be represented using this approach.

An implicit surface consists of those points in three dimensional space that satisfy a certain requirement represented mathematically by a function 'f' whose argument is a point 'p'. By definition, a point 'p' would lie on the surface if, $f(p) = 0$. The function 'f' may contain any mathematical expression. These expressions can be polynomials or may include transcendental expressions such as trigonometric or exponential expressions. For a continuous function 'f', its value at any point 'p' is often a measure of proximity between the point 'p' and the surface [Bloomenthal, 1997b]. This property is unique to the implicit surfaces and can be very useful for collision detection. However the intersection test for implicit surfaces of degree higher than 3 is computationally very intensive.

An implicit surface naturally describes an object's interior. The ability to enclose volume and to represent blends of volumes provides a straightforward implicit alternative to fillets, rounds, and other '*free-form*' parametric surfaces that require care in joining so that geometric continuity is established along the seams [Charrot, 1984]. Consequently, animations of organic shapes commonly employ implicit surfaces. Several types of implicit surfaces have been described in the literature. They include metaballs, [Wyvill, 1989], distance surfaces [Bloomenthal, 1990], convolution surfaces [Bloomenthal, 1991], R-functions [Pasko, 1995], variational surfaces [Savchenko, 1995], and blob trees [Wyvill, 1999].

Both implicit surfaces and parametric surfaces are well developed in computer graphics. Implicit surfaces with a signed distance function are the generalization of implicit surfaces and have been extensively discussed in the computer graphics literature [Benko, 2001; Bloomenthal, 1997b; Liu, 2006]. However, parametric surfaces are preferred over implicit surfaces because parametric surfaces are simpler to render and more convenient for certain geometric operations, such as computation of curvature and the control of position and tangency. In particular, parametric surfaces are generally easier to draw, tessellate or to perform any operation that requires a knowledge of '*where*' of the surface [Rockwood, 1989].

Manipulation of an implicit surface is complex because properties like surface point and its normal are not easily specified. Turk and O'Brien [Turk, 1999a] first presented the concept of variational implicit surfaces to handle this problem. Variational implicit surface facilitate direct specification of both the location of points on the surface and surface normals and are defined with additional interior and exterior constraints. They demonstrated the use of variational implicit surfaces for shaped transformation [Turk, 1999b]. This approach can successfully generate interpolating implicit surfaces [Turk, 2002]. Bloomenthal [Bloomenthal, 1999] used skeltonization as effective way for storing and manipulation and projected this technique for applications involving animation. Shen *et al.* [Shen, 2004] developed a generalized approach of generating interpolating or approximating implicit surface from polygonal data with *moving least squares* (MLS) approach and iteratively fitted normal constraints. Implicit surfaces with sharp features are modeled by kernel approximation [Reuter, 2004] and dual-primal mesh optimization [Ohtake, 2002]. Reuter *et al.* [Reuter, 2004] used MLS methods and integrated sharp edges by specifying the implicit equation of half space that cuts the sharp edge with discontinuous implicit surface. Implicit surfaces are extensively used for character animations [Cani-Gascuel, 1997]. Implicit surfaces are generally used as an extra layer coating for any kind of structure that moves and deforms over time.

2.5.2 Tessellated surfaces

Tessellated surface modeling schemes attempt to interpolate the point clouds by mapping a flat 3D triangulated domain to generate 3D triangulated surface model. Delaunay and its derived methods are frequently used for generating triangulated surfaces. In Delaunay triangulation, each triangle is generated from three points in the point set '*P*' in such a way that no other point from the data set '*P*' lies within the circum-circle of the corresponding triangle. This condition minimizes sharp triangles. A detailed description of Delaunay triangulation, related issues, theory and algorithms have been described in literature [Cazals, 2004; Chew, 1989; Dey, 2007]. Tessellated surfaces are widely used because of inherent simplicity of triangle-triangle collision detection and advantage of hardware compatibility with tessellated models for rendering and graphical representations. However, there are many disadvantages of using tessellated surfaces while representing deformable surfaces. The most significant disadvantage of using tessellated model is that graphical rendering, collision detection and the number of nodes for mass spring system are all dependent upon the number of triangles used to represent a model.

The resolution of the surface cannot be changed during haptic interaction. A low resolution model can be used to decrease computation time. Conversely, a high resolution model can be used to increase accuracy of collision detection and better graphical representation. However, this trade off cannot be used during haptic interaction. Furthermore, as a tessellated surface model deforms during interaction with the tool or other model, the quality of the triangles deteriorate. This makes collision detection computationally extensive and inefficient. Other drawbacks of this approach include its incapability of modeling very smooth surfaces, and inability to represent smooth boundary edges.

2.5.3 Parametric surfaces

A parametric surface is a surface in the Euclidean space R^3 which is defined by an equation with two parameters. In other words a parametric surface is a function with domain R^2 and range R^3. Typically variables u and v are used for the domain and x, y, and z for the range. These notations have been used in this thesis. These surfaces require fewer geometric parameters for their definitions. However, geometrically and topologically complex freeform surfaces cannot be represented with simple analytical surfaces (planes, quadrics, spheres, cones and cylinders etc.).

Parametric surface representation is a well established geometric modeling approach of modern day geometric modeling software, because it facilitates compact storage, complex editing and modification tools and reliable data exchange for model sharing. Parametric modeling is quite effective modeling scheme for generating accurate, concise, and affine invariant approximate surface of relatively simple size. It requires less storage for surface representation. However, this modeling scheme has not emerged as an effecting modeling tool for B-spline based shape modifications mainly because of the computationally expensive collision detection and complexity involved in estimating controls points from for the deformed surface.

The general form of the parametric representation of a surface is $p = p(u, v) = [x(u, v)\ y(u, v)\ z(u, v)]^T$. There are many types of synthetic parametric surfaces such as, Bilinear surfaces, Coons surface patches, Bicubic patches, Bezier surfaces, B-spline surfaces and NURBS surfaces [Farin, 1997; Piegl, 1997]. Of all these types of the surfaces, B-spline surfaces are the most widely used.

Over the past decade, B-spline representation has become the standard for CAD/CAM systems. Thus it is imperative that any haptic interactive design module should utilize a B-spline surface to represent the virtual model in order to streamline the exchange of the information with existing CAD/CAM systems. A major obstacle in using the B-spline surface to represent a deformable model is the absence of an efficient algorithm to detect collision between two or more B-spline surfaces having a complex surface. A collision detection algorithm must be capable of tackling complex surfaces, a large area of contact, multiple contacts, and high deformation.

2.6 Collision Detection

Collision detection enables simulation-based interactive design, engineering analysis, assembly, motion planning, medical training and animation. Collision detection is considered a major computational bottleneck in these applications. The goal of collision detection is to automatically report a geometric contact when it is about to occur or has actually occurred. There are many collision detection algorithms proposed by various researchers. Lin and Gottschalk [Lin, 1998] presented a survey on the state of the art in collision detection between geometric models represented by smooth surfaces. These surfaces were sub-divided into four groups; constructive solid geometry, implicit surfaces, parametric surfaces and polygonal models. The collision detection algorithms can be further sub-divided in three groups; pair versus n-body, static versus dynamic, and rigid bodies versus deformable models. Another survey [Jimenez, 2001] focused more on how the model representation leads to different collision detection algorithms. These algorithms were grouped into four approaches: space–time volume intersection, swept volume interference, multiple interference detection and trajectory parameterization.

In this chapter, the collision detection algorithms have been grouped mainly as rigid body algorithms and deformable model algorithms. Most of these algorithms fall in the former category and there are not many efficient collision detection algorithms for deformable bodies having complex surface. This section briefly discusses some of the important algorithms.

2.6.1 Collision detection of rigid bodies

Ho *et al.* [Ho, 1999] proposed a *Neighbourhood Watch* algorithm that was capable of handling rigid bodies with both convex and concave surfaces. The *Neighbourhood Watch* algorithm took advantage of pre-computed connectivity information for detecting collisions between the end effector of a force-reflecting robot and polyhedral objects in VR environment. Using this method and a haptic interface device, the users can manually explore and feel the shape and surface details of virtual objects. Mirtich [Mirtich, 1997] employed two phased approach to detect collision of rigid, polyhedral geometries. For the broad phase, an algorithm using axes-aligned bounding boxes (AABB) and a hierarchical spatial hash table was described. For the narrow-phase, primarily, the Lin-Canny algorithm [Lin, 1991] was used. Various bounding geometries and spatial geometries help in performing a *rejection test* when two virtual objects are apart. Some of the well-known examples include trees of sphere [Bradshaw, 2002; Hubbard, 1995; Quinlan, 1994], Axis Aligned Bounding Boxes (AABB) [van-den-Bergen, 1997] OBBTrees [Barequet, 1996; Gottschalk, 1996; Gregory, 2000a], k-DOPs [Klosowski, 1998; Krishnan, 1998] and SSVs [Larsen, 2000]. Many other types of volume have been suggested as bounding volume namely cones, cylinders, spherical shells, ellipsoids and zonotopes [Ericson, 2005]. Spatial partitioning decomposes the workspace into uniform grids or cells, implemented as a hash table to efficiently deal with large storage requirements. At runtime, the algorithm can quickly find the cells containing the path swept out by the hand-held probe. Once the objects are in close proximity, spatial decomposition is used to perform a *rejection test* for decomposed/subdivided parts of the objects. Gregory *et al.* [Gregory, 2000a] used a pre-computed hybrid hierarchical representation, consisting of uniform grids and trees of tight-fitting Oriented Bounding Box Trees (OBB Trees). At run time, these hybrid hierarchical representations exploit frame-to-frame coherence for fast proximity queries. Ehmann and Lin [Ehmann, 2001] presented a unified approach to perform a set of proximity queries for general, rigid polyhedral objects. Hubbard [Hubbard, 1995] used simple four-dimensional geometry to approximate motion, and hierarchies of spheres to approximate three-dimensional surfaces at multiple resolutions. For time-critical algorithms, such as interruptible collision detection, there are distinct advantages in using hierarchies of spheres, known as sphere-trees. Quinlan [Quinlan, 1994] described an efficient algorithm for computing the distance between non-convex objects. Objects were modeled as the union of a set of convex components. From this model a hierarchical bounding representation based on spheres was constructed. Bradshaw and O'Sullian [Bradshaw, 2002] also presented work in Sphere-Tree construction and medial axis approximation using the Veronoï diagrams [Okabe, 2000]. Hoffmann and Hopcroft [Hoffmann, 1987] approximated the object as the union of several cuboids enclosed by a single cuboid. The bounding geometry techniques work very well with rigid bodies but when applied to deformable bodies, the cost of updating these geometries, as the object deforms, slows down the collision detection response. If these bounding geometries are not updated in each frame to reduce the computational cost, the overlapping of bounding volumes make the *rejection test* inefficient.

2.6.2 Collision detection of deformable bodies

In general, rigid body collision detection methods pre-compute some geometric information of each object, such as bounding boxes, to be used for run-time collision detection. However, if the object deforms, the pre-computed information may not be valid anymore and hence needs to be recomputed in every frame while the object is deforming. Instead of rebuilding, most often, the bounding boxes are refitted as the object deforms. While this simplifies the bounding box updating, it often increases the overlapping areas of the bounding boxes.

Larsson and Möller [Larsson, 2001] proposed and evaluated suitable bounding volume trees for deforming bodies that can be pre-built and then updated during simulation. The technique was used to address the collision detection problem in applications where deformable bodies are used, which change their overall shape at every time step of the simulation. Several heuristics for updating the trees due to deformations were compared to each other. Deformable objects are very challenging for BVH (Bounding Volume Hierarchies) because hierarchy structures (trees) have to be updated when an object deforms itself. Teschner *et al.* [Teschner, 2005] presented various approaches based on bounding volume hierarchies, distance fields, and spatial partitioning for deformable bodies and used the techniques for surgical and cloth simulations. Various bounding volumes were considered, in particular, OBBs and k-DOP's, and investigated for their efficiencies.

For parametric surfaces, Herzen *et al.* [Herzen, 1990] developed an algorithm to detect geometric collisions between pairs of time-dependent parametric surfaces. It used the Lipschitz condition on a surface to create sets of bounding volumes that are guaranteed to bound the parametric surface. A surface-surface intersection test for a Bezier and B-spline surface by subdividing the surface patch till the sub-patches are sufficiently plane has also been proposed [Hughes, 1996]. The method constructed an AABB tree for each surface and used the pseudo-normal patch and Gauss map to detect self-collisions and the sweep-and-prune method to detect other collisions. However, the method needed to update the Axis Aligned Bounding Box (AABB) tree for each sub-patch and to find the solution of many algebraic equations for testing self collision. This made the algorithm computationally expensive. For complex surfaces, the number of levels needs to be high, further escalating the cost. As the model is deformable, the sub-patches that were plane initially may deform into curved surfaces and may no longer remain plane.

Thompson *et al.* [Thompson, 1997] used a tracing algorithm, supporting the rendering of NURBS surface, which traced the closest point on a NURBS patch to the tool point. The system linked an advanced CAD modeling system with a Sarcos force-reflecting exo-skeleton arm. Initially a rough check for surface proximity was done using bounding boxes around each surface. Later on *nodal mapping* was used to find a first order approximation to the closest point on the surface. Dachille *et al.* [Dachille, 2001] used a polyhedral representation which makes it easier to search for the nearest point on the surface, unlike the complicated NURBS surface intersection task proposed by Thompson [Thompson, 1997]. In this case the nearest point on the surface did not need to be updated too frequently, so the system could compute the distance from the cursor to all the vertices of the polyhedral representation. In both of the techniques, the interaction with the virtual model was only at a point through a point based tool. This limits the utility of the approaches, particularly in the area of 3D free form shape design where the sculpting activity is required to be done by hands or other surface based tools. Another shortcoming of point-surface interaction is the difficulty in anticipating results in advance. The user often does not know exactly if the deformation is going to be highly curved or flatter. Gao and Gibson [Gao, 2005; 2006] used a B-spline patch to represent the surface of a virtual model and an implicit surface to represent the virtual tool. Nodes were generated on the B-spline surface by discretization. These were used to incorporate a mass spring system and to detect collision by inputting these nodes in the equation of the implicit surface of the tool. This technique cannot be used when two or more B-spline surfaces are present in the virtual environment and are interacting with each other. Only rigid and lower degree (up to 2) implicit surface tools had been used and the collision detection technique did not allow a point based tool which further limited the scope of sculpting.

Galoppo *et al.* [Galoppo, 2007] used a two-level layered model representation. Low resolution layer was used for the collision detection and the force response while the high resolution tetrahedral mesh was used to show deformation.

Recently specialized graphics processing units (GPUs) have been used for processing bounding volume generation. Lauterbach *et al*. [Lauterbach, 2009] presented two parallel algorithms for rapidly constructing bounding volume hierarchies on multi-core GPUs. This Linear Bounding Volume Hierarchy (LBVH) algorithm is focused on minimizing the cost of construction, while still producing BVHs of good quality. The algorithm was implemented in CUDA on an NVIDIA GeForce 280 GTX GPU to compute axis-aligned bounding boxes (AABBs). The algorithm was used for ray tracing and can be implemented for collision detection. Avril *et al*. [Avril, 2010] has discussed various approaches which use General-Purpose Processing on Graphics Processing Unit (GPGPU) and more recently using multi-core computers.

In general if the model is represented as a tessellated surface, the cost of updating the bounding box during deformation is very high. In case of a parametric representation of a deformable model, the collision detection techniques for B-spline intersection require to calculate computationally intensive blending functions. Hence the intersection test of two parametric surfaces is complex as compared to the simplicity and efficiency of triangle-triangle intersection test. The method proposed in this thesis uses the best of parametric representation of surface and efficiency of triangle-triangle intersection test.

2.7 Force and Deformation Modeling Techniques

Various techniques have been developed to generate fairly smooth surfaces and volumes that satisfy multiple constraints in deformable modeling. Shape modification of a virtual object can be simulated using either geometric- or physics-based algorithms. Sederberg and Parry [Sederberg, 1986] introduced free-form deformation by allowing object to change shape independent of its structure by embedding it in easily-parameterized domains. The scheme was based on trivariate Bernstein polynomials, and provided the designer with an intuitive appreciation for its effects. Terzopoulos *et al.* [Terzopoulos, 1987] pioneered the work on physically-based deformable models in computer graphics. It applied the Lagrangian equations of motion using finite difference scheme to simulate elastic objects with regular parameterizations. This framework was further extended to include inelastic behaviours [Terzopoulos, 1988a] and to handle stiff rotating bodies using linearized equations [Terzopoulos, 1988b]. Physics-based deformation models give the designer, more opportunities to try different types of materials during the interactive design phase and validate product models in real-time. Major techniques and relevant research is presented in the following sub-sections.

2.7.1 Geometric deformation techniques

Geometric techniques are computationally efficient. These techniques only simulate the deformation and do not use the physical principles to calculate the deformation. In

Computer Aided Geometric Design (CAGD), mostly parametric curves and surface patches such as Bezier, B-spline, Rational B-spline, and non-uniform rational B-spline (NURBS) are used to represent a model. The model can be deformed by moving the control points, adding or deleting the control points or changing weights of the control points. The model can also be deformed by changing the knot vector. However, this is a cumbersome process and a perceptually simple change may require significant and simultaneous adjustments of several control points.

Free-from deformation is a general geometric-based deformation method to deform objects. This method deforms the space in which the virtual model lies, thereby, deforming the model. Barr [Barr, 1984] presented this hierarchical solid modeling operations, which could simulate twisting, bending, tapering, or similar transformations of geometric objects. He used geometric mappings of three-dimensional space to examine model deformation. This mapping causes the objects to twist about z-axis. More complex deformations can be constructed by composing mappings. These operations extend the conventional operations of rotation, translation, Boolean union, intersection and difference. This technique can be applied to different graphical representations such as, points, polygons, splines, parametric patches, and implicit surface. Barr's method provides a powerful design tool, but the possible regions and types of deformation are limited. The user cannot control the deformations intuitively. Sederberg and Parry [Sederberg, 1986] introduced the term Free-Form Deformation (FFD). They generalized Barr's approach by embedding an object in a lattice of grid points of some standard geometry, such as a cube or cylinder. Manipulating nodes of the grid induces deformation on the space inside the grid, and these deformations transform the underlying primitives that for the object. FFD involves a mapping from R^3 to R^3 through a trivariate tensor product Bernstein polynomial. It can be applied to CSG based solid models as well as those using Euler operators. It can sculpt solids bounded by any analytic surface: planes, quadrics, parametric surfaces patches, or implicit surfaces. However, this technique restricted the parametric solid to a regular parallelepiped with uniform divisions and a Bernstein polynomial basis. The degree of parameterization of the solid in each parametric coordinate was set directly by the number of uniform divisions within the control lattice-two for a quadratic parameterization, three for a cubic, and so on.

Griessmair and Purgathofer [Griessmair, 1989] presented an FFD based on a trivariate B-spline. Their technique, however, focused on an adaptive triangulation technique for tessellating the deformed surfaces. Coquillart [Coquillart, 1990] further extended this method by providing a toolkit of lattices with different sizes, resolutions and geometries. These geometries can be positioned over the object for selective control of sub-regions of the surface. Free-form deformation (FFD) is a powerful modeling tool, but controlling the shape of an object under complex deformations is often difficult. The difficulty in controlling the shape arises because the deformed object does not follow the control points exactly. Davis and Burton [Davis, 1991] subsequently demonstrated techniques for deforming the lattice in an interactive system that incorporated rational Bernstein bases. Bernstein-based formulations, however, yield an unfortunate relation between lattice divisions and the degree of the enclosing parametric solid. Lattices requiring more than two or three divisions for flexibility are undesirable, since evaluating the basis becomes computationally more expensive as the degrees of the polynomials increase. Hsu *et al.* [Hsu, 1992] allowed direct manipulation of surface or curve points by converting the desired movement of the point to an equivalent grid point movement. The under-constrained problem is solved by choosing the grid point movement with minimum least-squares energy that produces the desired object manipulation. Lamousin and Waggenspack Jr. [Lamousin, 1994] further extended current FFD techniques by basing them on non-uniform rational B-splines (NURBS). The resulting NURBS-based FFDs (NFFDs) offer more flexibility and control. McDonnel and Qin [McDonnell, 2007] presented an interactive, point-based technique for performing free-form deformation of polygonal meshes. In this technique, a volumetric deformation space is defined as the linear combination of overlapping, ellipsoidal radial basis functions (EBFs) of compact support. Mesh vertices are then parameterized with respect to local coordinate frames centered over the origins of the EBFs. Other important developments in FFD techniques include dynamic free-form deformations [Faloutsos, 1997], volume-preserving FFD [Hirota, 1999], sketch-driven FFD [Hua, 2003], and discontinuity-introducing deformations [Schein, 2004].

2.7.2 Physics-based deformation techniques

Unlike geometric techniques, computationally intensive physics-based techniques can yield real material behavior of a multiple-material/non-homogeneous virtual model [Knopf, 2005]. Physics-based deformation models give the designer, more opportunities to try different types of materials during the interactive design phase and validate product models in real-time. Important physics-based techniques are discussed in the following sub-sections.

2.7.2.1 Mass spring method

Mass spring systems are simple physical model with well understood dynamics. In physically-based techniques on mass-spring-damper models, an elastic object is constructed by applying a mass at each point of a mesh and using springs to link the points as edges and diagonals [Cotin, 2000; Lin, 2002; Nedel, 1998]. Elastic forces and damping forces act on mass points as internal forces, and gravity and the user induced forces act on them as external forces. These are easy to construct and computational cost is moderate. It is possible to achieve interactive real-time interaction with the virtual model using mass spring system, on the ubiquitous desktop computers. Mass spring systems are also suitable for parallel processing and hence can benefit from multi-core processors being used in desktop computers. The pioneering work of Terzopoulos *et al.* [Terzopoulos, 1987], Waters [Waters, 1992], and Platt and Barr [Platt, 1988] has shown the advantages of physically based models over geometric-based computer animation techniques.

However, mass-spring-damper models have drawbacks. The model is a significant approximation of the true physics that occurs in a continuous body. The lattice is tuned through its spring constant, and proper values for these constants are not always easy to derive from measured material properties. The physical accuracy of modeling is often not sufficient and cannot realize the global deformation. The above models are linear, and to simulate nonlinear force responses, it is necessary to use a precise integration mechanism such as the finite element method (FEM). However, such a method generally cannot provide update rates that are sufficient for haptic interactions [Luo, 2007]. In addition, certain constraints are not naturally expressed in model such as incompressible volumetric objects, or thin surfaces resistant to bending. Mass spring systems also sometimes exhibit *stiffness* problem which can occur when very large spring constants are used. Stiff systems are very problematic because they have poor stability even with longer time steps.

Terzopoulos *et al.* [Terzopoulos, 1991] described a mass spring model for deformable bodies that experience state transition from solid to liquid. The deformable model features non-rigid dynamics governed by Lagrangian equations of motion and conductive heat transfer governed by the heat equation for non-homogeneous, non-isotropic media. In its solid state, the discretized model is an assembly of hexahedral finite elements in which thermo-elastic units interconnect particles situated in a lattice. A discretized form of the heat equation is used to compute the diffusion of heat through the material. At melting point the stiffness reaches zero thereby severing the bond. The molten state of the model involves a molecular dynamics simulation in which *fluid* particles that have broken free from the lattice interact through long-range attraction forces and short-range repulsion forces. Tu and Terzopoulos [Tu, 1994] used mass spring system to generate *artificial fish* with internal contractile muscles that are activated to produce the desired motions. The fish model comprised of 23 mass points and 91 springs. The spring-mass system is simulated using implicit Euler method which maintains the stability of the simulation over the large dynamic range of forces. Christensen *et al.* [Christensen, 1997] embedded objects in a cubic eight nodes lattice connected by 28 damped linear springs and used dynamic simulation and FFD's to animate the embedded objects. The simple Coulomb friction model is used, though friction is permitted to be stronger in one preferred direction if the animator so specifies. The resulting equations of motion are solved numerically by a variable time-step, fifth-order Runge-Kutta integration procedure. Actuation of the mass-spring lattices is achieved by varying the rest lengths of the springs.

Mass-spring system has been extensively used for facial and modeling and animation. Terzopoulos and Waters [Terzopoulos, 1990] incorporated physically based approximation to facial tissue and muscle actuators. Different spring constants were used to model different layers based on tissue properties. Lee *et al.* [Lee, 1995] presented a physics-based model of the face which consisted of a biological tissue layer with nonlinear deformation properties, a muscle layer knit together under the skin, and an impenetrable skull structure beneath the muscle layer. For improved realism, the model used a constraint which prevented muscles and facial nodes form penetrating the scull. Koch *et al.* [Koch, 1996] used a mass spring model to predict the postoperative appearance of patients whose underlying bone structure for surgical planning and prediction of human facial shape after craniofacial and maxillofacial surgery for patients with facial deformities.

Mass-spring models have also been extensively used in surgical simulations due to their simplicity of implementation and their relatively low computational complexity [Baumann, 1996; Kuehnapfel, 1993]. Kuehnapfel and Neisius [Kuehnapfel, 1993] presented a simulation of endoscopic surgery based on a surface spring-mass model. Cotin *et al.* [Cotin, 2000] presented a hybrid model consisting of mass spring system and tensor-mass model based on continuum mechanics and linear elasticity theory. The tensor-mass model is used for the simulation of tearing and cutting.

Mass-spring system has also been used for concept design validation. Igwe *et al.* [Igwe, 2008b] proposed to generate hexahedral mesh of mass spring system by using *volumetric self-organizing feature map* (VSOFM). The VSOFM exploits the adaptive and self-organizing ability of Kohonen’s original algorithm [Kohonen, 2001] to develop a 3D mesh where the position the exterior nodes represent surface points of the underlying object. Material removal and tearing are achieved by eliminating selected mass points and spring coefficients in the evolving mesh. Pungotra *et al.* [Pungotra, 2009a] used the mass spring system based on VSOFM for validating the concept design.

The underlying geometry of mass spring models can easily be modified to represent topology changes. However, spring-mass models are discrete representations of a continuum, and the update of stiffness and mass values is hard to handle. This becomes a problem when complex models are to be deformed in real time. To avoid this problem, iterative method was used to solve the deformation at any localized region.

2.7.2.2 Finite element method

In the finite element method, the model's solution is subject to the constraints at the node points and the element boundaries so as to achieve continuity between the elements.

Unfortunately, finite element calculations are notoriously slow, making them not very appealing for real-time applications. Finite element methods are often considered to be less efficient than spring-mass models. In FEM, the applied forces must be converted to their equivalent force vectors. This requires numerically integrating distributed forces over the volume at each time step. This can lead to a significant pre-processing time for finite element methods. If the topology of the object changes during the simulation, mass and stiffness matrices must be re-evaluated during the simulation. Traditional FEM is more accurate in modeling materials such as metals, where the amount of deformation is limited. As the model deforms, the volume over which equivalent force, mass, and stiffness matrix integrations are performed will change. Real-time finite element modeling requires high computational power to achieve visual realism [Berkley, 2004]. FEM has been used for fairly simple physical systems to simulate tissue deformation [Cotin, 1996; Sagar, 1994].

However, the finite element method (FEM) is a very common and accurate way to solve continuum-mechanical boundary-value problems [Bathe, 1996; Zienkiewicz, 2005]. Finite element methods provide a more physically realistic simulation than mass-spring system with fewer nodes. There is a growing trend in using finite element soft tissue models for real-time computation, as shown for instance by Székely *et al.* [Székely, 2000] who simulated the deformation of a nonlinearly elastic material using a parallel processing architecture.

2.7.2.3 Continuum method

Instead of considering the model as a discrete object model, it can be considered as a continuum that is the solid bodies with mass and energies distributed throughout. Models can be discrete or continuous but the method used to solve in computer simulation is always discrete. The numerical integration techniques used to solve the model approximate the system at discrete time steps. However, unlike the mass-spring model, continuum models are derived from equations of continuum mechanics. The continuum model of a deformable object considers the equilibrium of a general body acted upon by external forces. The object deformation system is a function of these acting forces and object's material properties. The object reaches equilibrium when its potential energy is at a minimum.

Several authors have based their soft tissue models on continuum mechanics theory, and the use of elastic solids is widely described in the literature [Bainville, 1995; Speeter, 1992]. Bayville *et al.* [Bainville, 1995] define the evolution of a set of rigid and deformable solids under the influence of various forces. In this case, the deformation law is represented by a hyper-elastic, quasi-static model, associated with a finite element method for the numerical resolution. Unfortunately, the computation time makes this approach impractical for real-time simulations.

2.8 Concluding Remarks

The designers may expect that any interactive design framework must mimic the natural way designers interact with the physical world and provide direct sensory feedback during the interaction. Current CAD systems do not fulfill these expectations. The literature survey on the role that Virtual Reality (VR) can play in the field of interactive design shows that VR can play a vital role in fulfilling the expectations of the engineers and industrial designers.

This chapter provided a detailed discussion on the techniques for developing a virtual reality based interactive free form modeling framework. Firstly the virtual model representation needs to be decided. Some of the surface representation techniques that were considered include implicit surfaces, tessellated surfaces, and parametric surfaces. Implicit surfaces generated using analytical, variational, and multi-level partition of unity (MPU) approach, were considered. Implicit surfaces can represent water tight surfaces and solids. These surfaces are very efficient for collision detection for lower order surface. However, when the degree of implicit surface is large or when a large number of primitives are used to represent an implicit surface, the computational cost of collision detection is enormous. Tessellated surfaces are preferred to represent rigid bodies because of its compatibility with graphics hardware and efficiency of triangle-triangle collision detection. However, this is not efficient for deformable surface. Parametric surfaces and particularly B-splines have become the standard for CAD. Thus, it is imperative that any interactive design module should use parametric representation for the virtual models so as to have easy exchange between CAD software and haptic model. For these reasons, B-spline surfaces were chosen to represent virtual model.

The collision detection algorithms described in literature are not efficient enough for B-spline to B-spline collision detection. Thus, there is a need to have efficient collision detection algorithm which can efficiently tackle collision between two or more B-spline models and a B-spline and a point based or an implicit surface based tool. The collision detection should be compatible with the model deformation technique and should not be restricted to the use of a particular model.

This chapter also provided discussion on various techniques for deformation of model. Physically based models can predict the deformation of a model more accurately, particularly, when the model consists of multiple or non-homogenous materials. A detailed study of the physics-based models provided the insight to choose mass-spring system. Although this is not the best model to accurately model the deformation of the model, it can provide reasonable accuracy and speed.

The next chapter introduces an efficient collision detection algorithm for a deformable model represented as a B-spline surface. The tool can be represented as a B-spline, a point, or an implicit surface.

CHAPTER 3 COLLISION DETECTION ALGORITHM

3.1 Introduction

Collision detection is an active research topic in engineering, computer graphics and virtual reality (VR) [Jimenez, 2001]. Collision detection is the necessary step before haptic interaction can be achieved. An efficient collision detection algorithm plays a very significant role towards achieving real-time haptic interaction. A general collision detection algorithm would be highly complex with high computational cost. Due to this reason, most of the collision detection algorithms are efficient only in a given domain. Most of the research has been carried out for applications in the video game industry and simulations for medical applications. The algorithms developed are mostly for the rigid bodies which can be represented as lower order implicit surfaces (spheres, cylinders, cones etc.) or tessellated surfaces. For applications in haptic-based interaction with virtual objects, mostly tessellated bodies are used, and many constraints imposed on model deformation under the influence of external force.

In this thesis both the model and tool are represented as B-spline surfaces. A rigid tool can also be represented as a point, an implicit surface, or a tessellated surface. No constraint with regard to the shape and rigidity of the tool and model has been considered. Both the model as well as the tool can have complex shape, elastic or plastic properties, and multiple contacts. This chapter discusses the step by step implementation of the collision detection algorithm for deformable models. The comparison of the computations required for a tessellated surface with that for the B-Spline surface using the worst case scenario (Big ***O*** notation) is also presented.

3.2 Background

3.2.1 B-spline surface

A B-spline surface with r and s number of control points in u and v directions is mathematically represented by the equation,

$$S(u,v)=\sum_{i=0}^{r-1}\sum_{j=0}^{s-1}B_{ik}(u)\,B_{jl}(v)\mathbf{P}_{ij} \tag{3.1}$$

where $\mathbf{P}_{ij}$ is the control points vector, $B_{ik}(u)$ and $B_{jl}(v)$ are the B-spline basis functions of the surface with degrees k and l in u and v directions, respectively; defined over non-periodic knot vectors,

$U=\{\underbrace{0,...,0}_{k+1},u_{k+1},...,u_{r-1},\underbrace{1,...,1}_{k+1}\}$ and $V=\{\underbrace{0,...,0}_{l+1},v_{l+1},...,v_{s-1},\underbrace{1,...,1}_{l+1}\}$.

The blending functions depend upon periodicity of the surface (in u or v or both directions), the knot vector, the degrees of the surface in u and v directions and the number of control points in u and v direction. However, the blending functions are independent of the position of the control points. A B-spline blending function has the property of recursion which is defined as:

$$B_{i,k}(u)=(u-u_i)\frac{B_{i,k-1}(u)}{u_{i+k}-u_i}+(u_{i+k+1}-u)\frac{B_{i+1,k-1}(u)}{u_{i+k+1}-u_{i+1}} \tag{3.2}$$

where $B_{i,1}(u)=\begin{cases}1, & u_i\le u\le u_{i+1}\\ 0, & otherwise\end{cases}$

If the B-spline surface is periodic in u direction, the blending function and the knot vector (U) are modified as:

$$B_{i,k}=B_{0,k}\left((u-i+r)\bmod(r)\right)\,;\; U=\{u_0,u_1,...u_r\} \tag{3.3}$$

The function, mod (r) in Eqn. (3.3), is the modulo function and is defined as:

$$A\bmod(r)=\begin{cases}A, & A<r\\ 0, & A=r\\ remainder\ of\ \dfrac{A}{r} & A>r\end{cases}$$

A B-spline and NURBS based surface can be expressed as a tensor product. A point on the B-spline surface, within the knot span u_i, u_{i+1}; v_j, v_{j+1}, can be determined by using the tensor product as:

$$S(u,v)=[B_{a,k}(u)][\mathbf{P}_{a,b}][B_{b,l}(v)]^T\,; i-k\le a\le i\,,\; j-l\le b\le j \tag{3.4}$$

In Eqn. (3.4), $[B_{a,k}(u)]$ is a 1 × (*k*+1) row vector of scalars, $[\mathbf{P}_{a,b}]$ is a (*k*+1) × (*l*+1) matrix of control points, and $[B_{b,l}(v)]^T$ is a (*l*+1) x 1 column vector of scalars. This equation can also be written as:

$$S(u,v)=[u][B_u][\mathbf{P}_{ij}][B_v][v]^T \tag{3.5}$$

where $[u]=[1\ u\ u^2 \ldots\ldots.. u^k]$ and $[v]=[1\ v\ v^2 \ldots\ldots.. v^l]$

To generate a large number of points on the whole B-spline surface, Eqns. (3.4 -3.5) need to be computed many times. Instead of using a discrete value for *u* and *v*, a set of *u* and *v* parametric values can be used simultaneously. Equations (3.4-3.5) are modified to generate the matrix of a large number of points to discretize the B-spline surface patch into a set of a parametrically uniform grid of discrete points in *u* and *v* directions. The matrix of these discrete points M, is then given as:

$$\mathbf{M} = \mathbf{A}_u \mathbf{P}_{ij} \mathbf{A}_v;\ 0 \le i \le r-1,\ 0 \le j \le s-1 \tag{3.6}$$

where $\mathbf{A}_u$ and $\mathbf{A}_v$ are blending matrices (also known as B-spline Blending Transformation Matrices or simply Transformation Matrices).

The magnitudes of the entries of the blending matrices depend upon the blending functions and *u* and *v* parametric values. In Eqn. (3.6); **M** is an $m \times n$ vector of scalars, $\mathbf{A}_u$ is an $m \times (r+1)$ vector of scalars, $\mathbf{P}_{ij}$ is an $r \times s$ matrix of control points, and $\mathbf{A}_v$ is an $s \times n$ vector of scalars.

Equation (3.6) can be re-written as:

$$\mathbf{P}_{ij} = [\mathbf{A}_u^{\mathrm{T}} \mathbf{A}_u]^{-1} \mathbf{A}_u^{\mathrm{T}} \mathbf{M}\, \mathbf{A}_v^{\mathrm{T}} [\mathbf{A}_v \mathbf{A}_v^{\mathrm{T}}]^{-1}, \tag{3.7}$$

to determine the position of the control points that can represent a B-spline surface approximating the parametrically uniform discrete points of matrix **M**. As the blending matrices are independent of the position of the control points, these matrices ($\mathbf{A}_u$, $\mathbf{A}_v$) and their inverses ($[\mathbf{A}_u^{\mathrm{T}} \mathbf{A}_u]^{-1}$, $[\mathbf{A}_v \mathbf{A}_v^{\mathrm{T}}]^{-1}$) can be pre-calculated.

Figure 3.1 shows a typical blending matrix, $\mathbf{A}_u$. It is a band matrix and any row of this matrix can have a maximum of *k*+1 number of non-zero terms.

$$\mathbf{A}_u = \left[\begin{array}{ccccccc}
1.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\
0.512 & 0.434 & 0.053 & 0.001 & 0.000 & 0.000 & 0.000 \\
0.216 & 0.592 & 0.181 & 0.011 & 0.000 & 0.000 & 0.000 \\
0.064 & 0.558 & 0.342 & 0.036 & 0.000 & 0.000 & 0.000 \\
0.008 & 0.416 & 0.491 & 0.085 & 0.000 & 0.000 & 0.000 \\
0.000 & 0.250 & 0.583 & 0.167 & 0.000 & 0.000 & 0.000 \\
0.000 & 0.128 & 0.588 & 0.283 & 0.001 & 0.000 & 0.000 \\
0.000 & 0.054 & 0.521 & 0.415 & 0.011 & 0.000 & 0.000 \\
0.000 & 0.016 & 0.409 & 0.539 & 0.036 & 0.000 & 0.000 \\
0.000 & 0.002 & 0.282 & 0.631 & 0.085 & 0.000 & 0.000 \\
0.000 & 0.000 & 0.167 & 0.667 & 0.167 & 0.000 & 0.000 \\
0.000 & 0.000 & 0.085 & 0.631 & 0.282 & 0.002 & 0.000 \\
0.000 & 0.000 & 0.036 & 0.539 & 0.409 & 0.016 & 0.000 \\
0.000 & 0.000 & 0.011 & 0.415 & 0.521 & 0.054 & 0.000 \\
0.000 & 0.000 & 0.001 & 0.283 & 0.588 & 0.128 & 0.000 \\
0.000 & 0.000 & 0.000 & 0.167 & 0.583 & 0.250 & 0.000 \\
0.000 & 0.000 & 0.000 & 0.085 & 0.491 & 0.416 & 0.008 \\
0.000 & 0.000 & 0.000 & 0.036 & 0.342 & 0.558 & 0.064 \\
0.000 & 0.000 & 0.000 & 0.011 & 0.181 & 0.592 & 0.216 \\
0.000 & 0.000 & 0.000 & 0.001 & 0.053 & 0.434 & 0.512 \\
0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 1.000
\end{array}\right]
\begin{array}{l}
u = 0 \\ \\ \\ \\ u_{min} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ u_{max} \\ \\ \\ \\ \\ \\ u = 1
\end{array}$$

Figure 3.1 A typical blending matrix, $\mathbf{A}_u$.

As shown in Figure 3.1, a part of the blending matrix can be used to discretize a part of the B-spline patch bounded by the minimum and maximum values of the parameters u and v ($u_{\min}$, $v_{\min}$; $u_{\max}$, $v_{\max}$). Thus, if a part of the B-spline model is colliding with the tool, only that part is discretized to detect collision.

Equations (3.1), (3.6), and (3.7) will determine the correlation between control points and the discrete points generated on the B-spline surface. These equations can be used to discretize a B-spline surface and to calculate the control points which can represent a deformed B-spline surface.

Figure 3.2 shows a discretized B-spline surface along with the set of control points $\mathbf{P}_{ij}$.

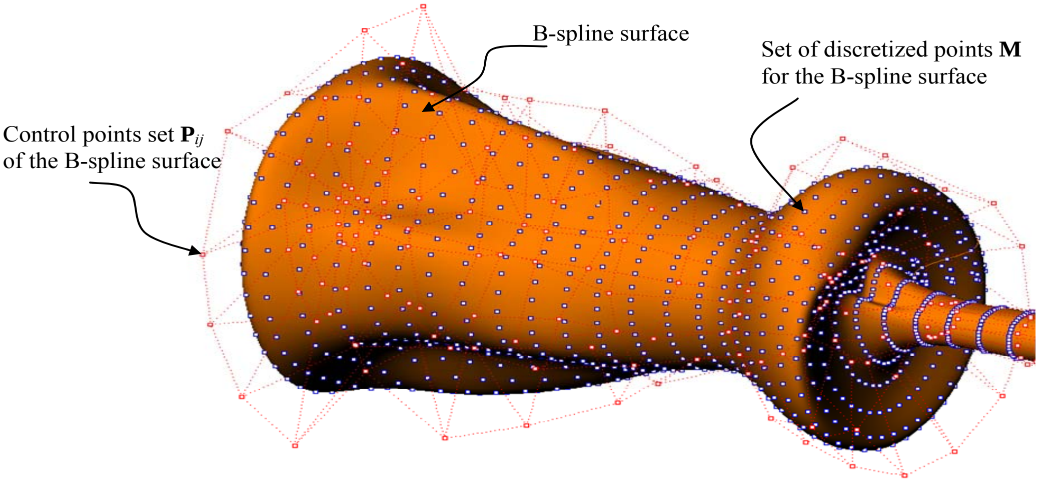

B-spline surface
Set of discretized points **M** for the B-spline surface
Control points set $\mathbf{P}_{ij}$ of the B-spline surface

Figure 3.2 Discretized B-spline surface.

3.2.2 Generation of blending matrices

Once various parameters of a B-spline surface are known, a blending matrix can be generated. The size of this matrix will depend upon the number of control points and the maximum number of points to be generated in u and v direction. The maximum number of points to be generated depends upon the accuracy required for collision detection. The size of a blending matrix for generating m points in u direction with n control points will be $m \times n$. For the same number of control points in the v direction, the matrix is transpose of the matrix in the u direction and its size will be $n \times m$. This will generate a total of m^2 points on the B-spline surface. Similarly the blending matrices for finding tangents at each point in the u and v directions can be pre-computed. The cross multiplication of the tangents in the u and v directions yields the resultant surface normal at the point.

As per Eqn. (3.7), the inverse of the blending matrix, to find new positions of control points, needs to be computed so that the deformed surface can be represented as a B-spline surface. Finding the inverse of the matrix during real time haptic interaction will increase the computational cost and sometimes may be problematic or even impossible. Since the blending matrices are pre-computed and are independent of the position of the control points, their inverse can also be pre-computed. This reduces the computational cost of rendering the deformation of the B-spline surface. The algorithm can also use a property of B-spline surfaces called local support, which indicates that each segment of the B-spline surface is influenced by only '$k+1$' control points where 'k' is the degree of the surface. If only a small segment of the B-spline surface undergoes deformation due to interaction with another surface, only some of the control points need to be updated. This reduces the cost of updating convex hull and the nodes on the B-spline surface. However if the deformation of the model, based on real material properties is to be calculated, all control points need to be updated.

3.2.3 Bounding volume for B-spline surfaces

The entire B-spline surface lies in the convex hull of the control points P_{ij} [Bloomenthal, 1997b]. This is due to the property of *positivity* which ensures that a B-spline curve or surface always remains within its convex hull. This makes the convex hull, a good choice as a bounding volume. As the position of the control points is always known and their numbers limited, the cost of updating the bounding volume (convex hull) is minimal. Thus, the intersection test algorithm between two B-spline surfaces checks whether two convex hulls are overlapping. If convex hulls are not intersecting each other, the B-spline surfaces are not colliding.

3.3 Collision Detection Algorithm

The collision detection algorithm for deformable bodies assumes that all the deformable models and tools are represented as a single B-spline patch. The algorithm also allows the rigid tools to have point-, implicit surface-, or tessellated surface-based representation.

3.3.1 Overview

Figure 3.3 shows the flow chart of the proposed collision detection algorithm. The interaction of collision detection with the force response model is also shown in the flow chart. The algorithm has two phases: the pre-processing phase and the run-time phase.

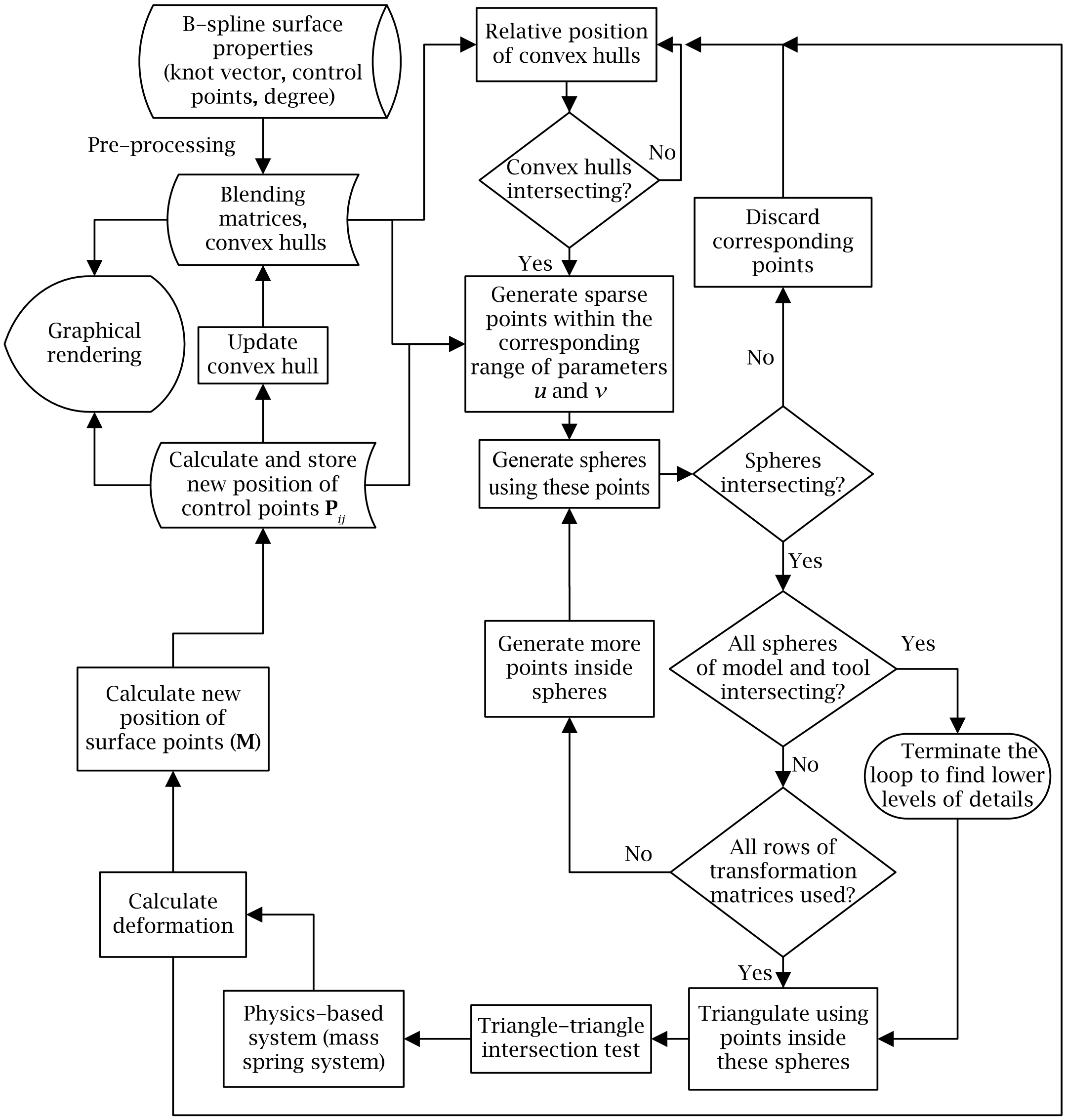

B-spline surface properties (knot vector, control points, degree)
Pre-processing
Blending matrices, convex hulls
Graphical rendering
Update convex hull
Calculate and store new position of control points $\mathbf{P}_{ij}$
Calculate new position of surface points (M)
Calculate deformation
Physics-based system (mass spring system)
Relative position of convex hulls
Convex hulls intersecting?
No
Yes
Generate sparse points within the corresponding range of parameters u and v
Generate spheres using these points
Generate more points inside spheres
Spheres intersecting?
No
Discard corresponding points
Yes
All spheres of model and tool intersecting?
Yes
Terminate the loop to find lower levels of details
No
All rows of transformation matrices used?
No
Yes
Triangulate using points inside these spheres
Triangle-triangle intersection test

Figure 3.3 Flowchart of the proposed algorithm.

During the pre-processing phase the blending matrices ($\mathbf{A}_u$ and $\mathbf{A}_v$) of the given B-spline surfaces, along with their inverses ($[\mathbf{A}_u{}^T \mathbf{A}_u]^{-1}$ and $[\mathbf{A}_v \mathbf{A}_v{}^T]^{-1}$), are calculated and stored. A convex hull is generated from the control point set.

The run-time phase starts by checking for the intersection of convex hulls (bounding volume) of B-spline surfaces. If the convex hulls are intersecting, the corresponding minimum and maximum values of u and v parameters associated with the control points of these surfaces are determined. Within this range of u and v, sparse points are generated on surfaces of the model and the tool, by using intermittent rows and columns of blending matrices. The points are used to generate spheres on both the model and the tool and these spheres are then checked for intersection. More points are generated within these intersecting spheres. This process of generating spheres at the lower levels of detail continues until all the rows and columns for the blending matrices have been used. The process of generating lower levels of detail is terminated if, at a particular level of detail, all the spheres on the model and the tool are intersecting. This is possible if the curvature of model and tool is similar in the region of probable collision.

The points within the intersecting spheres at the lowest level of detail are subsequently used to generate a triangulated mesh. The triangle-triangle intersection test is then carried out to find out the parts of the surfaces which are intersecting. This information is used to map the forces, applied by the user, to the nodes of the mass spring system. The physics-based model then determines the resultant deformation and the reactive forces to be sent back to the user. Using the inverse blending matrices, new position of control points is determined and the convex hull for the B-spline surface is updated. Following sub-sections describe the algorithm in detail.

3.3.2 Pre-processing phase

3.3.2.1 Generation of blending matrices

During the pre-processing stage, depending upon the parameters of the B-spline surface (periodic/non-periodic, number of control points in the *u* and *v* directions, the maximum number of points to be generated) blending matrices are generated and stored along with their inverse.

A maximum of two blending matrices for generating points are needed per B-spline surface. Similarly their inverse is also calculated and stored. This inverse is calculated by the Gaussian Elimination Method using '*complete pivoting'*. The complete pivot method enhances the robustness of the algorithm and, as this is done during the preprocessing phase, the time required to calculate inverse of the matrix does not increase the computational cost during the run-time phase. The algorithm not only calculates surface normal at the points generated but also calculates tangents in the *u* and *v* direction. This is useful if tangential properties, such as friction, are also to be considered to realistically model material properties. The surface normals also help in correctly mapping the forces to the surface by calculating the forces normal to the surface. The blending matrices for finding tangents are also generated and stored during the preprocessing phase.

3.3.2.2 Generation of convex hull

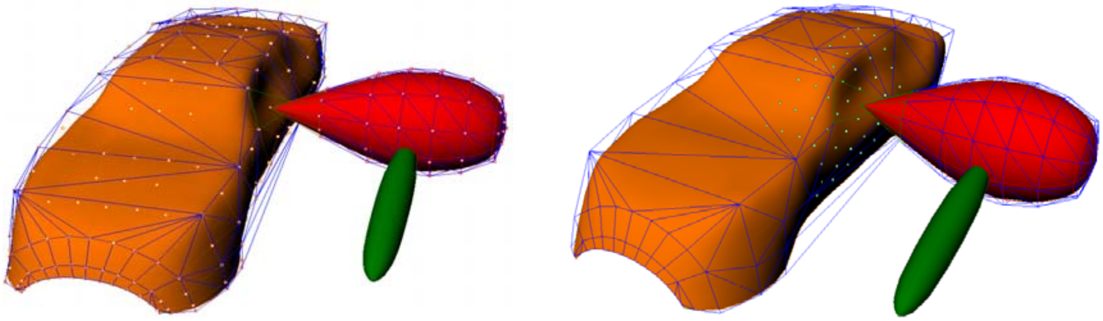
Many convex hull algorithms are available in literature. Avis [Avis, 1995] has considered most of the known classes of algorithms for generating convex hulls. Most of the algorithms deal with the general case of '*d*' dimensions. In the present application, the control points will always be in three dimensional space, which simplifies the algorithm. The algorithm used for the generation of a convex hull is similar to the *Quickhull Algorithm* [Barber, 1996], which is a variation of the randomized incremental algorithm. It is a recursive algorithm and partitions the control points data into several sets. Instead of selecting a random point, it selects the point which is farthest from the existing plane of the partial convex hull. In the worst case scenario the computational cost of this algorithm is $\boldsymbol{O}$ ($n \log n$), where n is the number of control points. As control points used for B-spline surfaces are usually small in number, the computational cost is quite small.

3.3.3 Run-time phase

During the pre-processing phase the control point matrices and blending matrices of the given B-spline surfaces and the inverse of these blending matrices are calculated and stored.

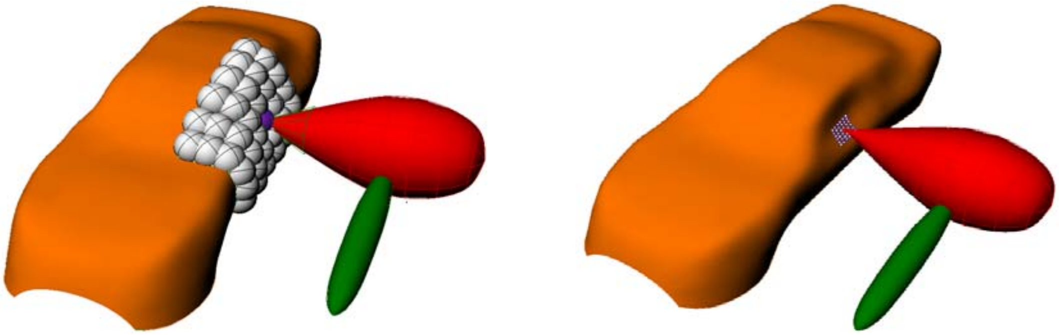
The run-time phase starts by checking for the intersection of convex hulls of B-spline surfaces initially calculated during pre-processing phase. This is done to determine if there is a need to carry out complex intersection test between B-spline surfaces. If the convex hulls are not intersecting, the collision detection process exits. However, if the convex hulls are intersecting, the algorithm goes to lower levels of detail to determine exact regions of intersection.

Figure 3.4 illustrates major steps during the run-time phase of the proposed algorithm.



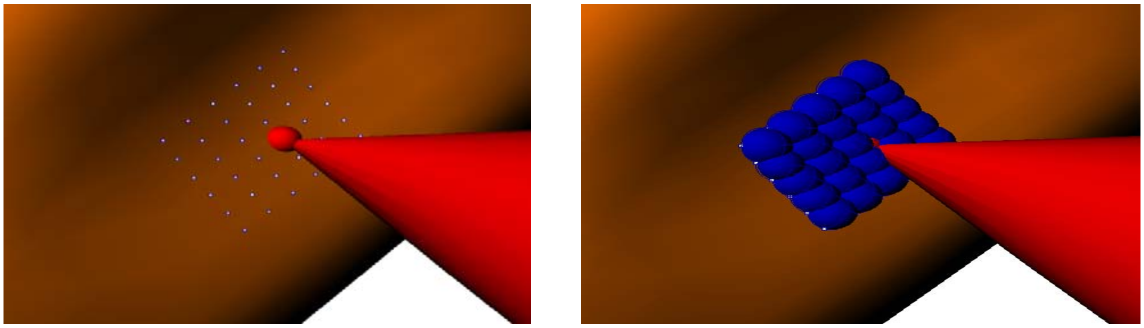
(a) Tool intersecting the plane (triangle) on model

(b) Points generated within maximum and minimum *u* and *v* values of intersecting triangle



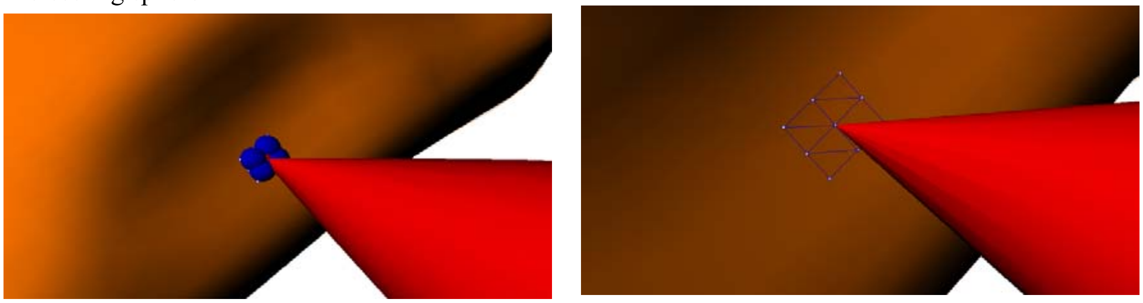
(c) Spheres generated from the points

(d) Points generated within intersecting sphere



(e) Expanded view of points generated within intersecting sphere

(f) Spheres generated from the points



(g) Intersecting spheres

(h) Triangulation using the points of intersecting spheres

Figure 3.4 Illustrations showing the major steps in the proposed algorithm.

Once the surfaces of the convex hulls intersect, the algorithm determines the corresponding minimum and maximum values of *u* and *v* parameters associated with the control points of these surfaces. It then generates sparse points within these limits on the surfaces of the model and tool. Spheres are generated using these sparse points in such a way that they cover the whole surface area. Spheres generated on the model surface are then checked for intersection with those on the tool or another surface model. If some of the spheres are intersecting, the algorithm generates more points (by using more values of *u* and *v* from the blending matrix) within the *u* and *v* parameter limits of the intersecting spheres and discards the points generated in non-intersecting spheres. This process of generating lower levels of details continues until all the rows of the blending matrix are used. The points generated at the end of this loop are then used for triangulation of the surface.

If all the spheres generated on the model and the tool are intersecting, then the algorithm does not proceed to lower levels of details and rather stops at the last level of detail. It then compares the normals of new points being generated within the *u* and *v* parameters. If the difference between unit normal of the point generated and that of the previous point is less that a small number δ, the new point is discarded. The rest of the points are then used for tessellation of the surface. The triangle-triangle intersection test is then carried out to find out the parts of the surfaces which are intersecting. Once the region(s) undergoing collision are known, the forces on these regions can be used by a physics-based model to determine the deformation of the model and the tool and the force to be fed back to the user through a haptic device. Using the inverse blending matrices, new position of control points is determined and the convex hull for the B-spline surface is updated. Each step of the algorithm is elaborated in the following subsections.

3.3.3.1 Intersection test of convex hulls

Collision detection between convex polytopes has been extensively researched in computational geometry and robotics [Baraff, 1990; Gilbert, 1988]. These algorithms can be easily applied to a convex hull generated from the control points set. The algorithm presented by Gilbert uses information from previous time steps for fast initialization and can also be used successfully in this case. The algorithm presented by Baraff for the intersection test of convex hulls, using linear programming and coherence, is more suitable. This algorithm uses the argument that if two convex polytopes are not intersecting, there exists a separating plane with the face or edge of one of the polytopes. This face is called the '*witness*' and is used in computing the separating plane. A given plane can be verified as the separating plane between two convex polyhedra in $\boldsymbol{O}(n)$ time where n is the number of vertices of two convex hulls. The surfaces will not be intersecting with each other until a separating plane exists.

3.3.3.2 Calculating minimum and maximum values of u and v

Each control point of a B-spline surface has an influence within a particular range of knot vectors in u and v directions. Depending upon the degree of the surface in u and v direction and the knot vector in u and v direction, the minimum and maximum value of u and v associated with a control point can be calculated [Piegl, 1997; Zeid, 1991]. When two convex hull surfaces (or edges or an edge and a surface) intersect with each other such that there does not exist any separate plane, the vertices of the surface (edge) are noted. These vertices are the subset of the control points set of the B-spline surface. The corresponding u and v parametric values associated with these vertices (control points) of the intersecting surfaces are calculated. These values are then used for generating minimum and maximum values of u and v (u_{min}, v_{min}; u_{max}, v_{max}).

3.3.3.3 Surface discretization

When the convex hulls intersect, there is a strong possibility that the surface close to the intersecting surface will experience collision. The limits for discretization of the surface are set by minimum and maximum values of parameters u and v (u_{min}, v_{min}; u_{max},v_{max}), as calculated in Section 3.3.3.2. The setting of limits on the surface for point generation and subsequent tessellation reduces the computational cost of point generation as compared to the algorithm proposed by Gao and Gibson [Gao, 2005; 2006].

Figure 3.5 shows a typical blending matrix for generation of points in the u direction and the selection of intermittent blending matrix for generating points on the surface within the selected maximum and minimum range of parameter u.

$$
\begin{array}{r|cccccccc|l}
u=0 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.167 & 0.667 & 0.167 & \\
 & 0.012 & 0.000 & 0.000 & 0.000 & 0.000 & 0.032 & 0.527 & 0.429 & \\
 & 0.100 & 0.000 & 0.000 & 0.000 & 0.000 & 0.001 & 0.256 & 0.644 & \\
 & 0.324 & 0.003 & 0.000 & 0.000 & 0.000 & 0.000 & 0.067 & 0.607 & \\
\hline
 & \mathbf{0.583} & \mathbf{0.053} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.005} & \mathbf{0.359} & \leftarrow \quad u_{min} \\
 & 0.656 & 0.224 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.119 & \\
 & 0.463 & 0.495 & 0.024 & 0.000 & 0.000 & 0.000 & 0.000 & 0.018 & \\
 & \mathbf{0.194} & \mathbf{0.664} & \mathbf{0.142} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \leftarrow \\
 & 0.042 & 0.556 & 0.394 & 0.008 & 0.000 & 0.000 & 0.000 & 0.000 & \\
 & 0.002 & 0.289 & 0.627 & 0.082 & 0.000 & 0.000 & 0.000 & 0.000 & \\
\mathbf{A}_u = & \mathbf{0.000} & \mathbf{0.082} & \mathbf{0.627} & \mathbf{0.289} & \mathbf{0.002} & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} & \leftarrow \\
 & 0.000 & 0.008 & 0.394 & 0.556 & 0.042 & 0.000 & 0.000 & 0.000 & \\
 & 0.000 & 0.000 & 0.142 & 0.664 & 0.194 & 0.000 & 0.000 & 0.000 & \\
 & \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.024} & \mathbf{0.495} & \mathbf{0.463} & \mathbf{0.018} & \mathbf{0.000} & \mathbf{0.000} & \leftarrow \quad u_{max} \\
\hline
 & 0.000 & 0.000 & 0.000 & 0.224 & 0.656 & 0.119 & 0.000 & 0.000 & \\
 & 0.000 & 0.000 & 0.000 & 0.053 & 0.583 & 0.359 & 0.005 & 0.000 & \\
 & 0.000 & 0.000 & 0.000 & 0.003 & 0.324 & 0.607 & 0.067 & 0.000 & \\
 & 0.000 & 0.000 & 0.000 & 0.000 & 0.100 & 0.644 & 0.256 & 0.001 & \\
 & 0.000 & 0.000 & 0.000 & 0.000 & 0.012 & 0.429 & 0.527 & 0.032 & \\
u=1 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.167 & 0.667 & 0.167 &
\end{array}
$$

Selection of intermittent rows for sparse points grid

Figure 3.5 Selecting intermittent blending matrix for generation of points within the selected range of parameter *u*.

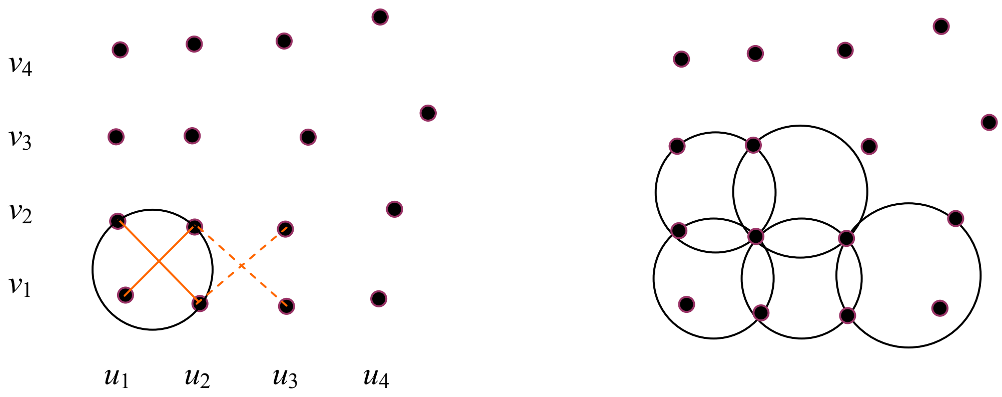
For generating points in the selected portion of the B-spline surface, a portion of the blending matrix is selected within minimum and maximum values of parameters *u* and *v*. Initially the B-spline surface is coarsely discretized. This is done by using intermittent rows and columns of the blending matrix for *u* and *v* directions, respectively. The matrix of the points generated is given by Eqn. (3.6), $\mathbf{M} = \mathbf{A}_{iu}\,\mathbf{P}_{ij}\,\mathbf{A}_{iv}$, where subscript '*i*' stands for intermittent while *u* and *v* stand for blending matrix in *u* and *v* directions respectively. Using these blending matrices points will be generated on the B-spline surface. Similarly tangents for all these points in *u* ($\mathbf{T}_{iu}$) and *v* direction ($\mathbf{T}_{iv}$) can be calculated by using the equations: $\mathbf{T}_{iu} = \mathbf{AT}_{iu}\,\mathbf{P}_{ij}\,\mathbf{A}_{iv}$ and $\mathbf{T}_{iv} = \mathbf{A}_{iu}\,\mathbf{P}_{ij}\,\mathbf{AT}_{iv}$

These tangents can then be used to impart tangential properties such as friction on the surface and for accurate calculation of the net force acting on the surface when the applied force is not normal to the surface. The surface normal at any point (*iu*, *iv*) is calculated by cross multiplying $\mathbf{T}_{iu}$ and $\mathbf{T}_{iv}$: $\mathbf{N} = \mathbf{T}_{iu} \times \mathbf{T}_{iv}$

3.3.3.4 Generation of spheres

The sparse points generated on the surface are then used to generate spheres. As discussed in Section 3.3.3.3, the points are generated within minimum and maximum limits for parameter u and v. The algorithm to generate spheres starts from point at $u_{\min},v_{\min}$. It compares the diagonal distance between points at u_1,v_1; u_2,v_2; and u_2,v_1 ; u_1,v_2 and selects two points of longer diagonal and one of the two remaining points. These three points in space can generate a unique circle of definite radius and center points. If a sphere is generated with the same center point and radius as of the circle, the fourth point will lie on or within the boundary of this sphere. This way of creating sphere ensures that there is no degenerative case, the computational cost is less and size of the sphere is optimal. This is better than creating a unique sphere from all the four points. This process continues with the next values of u and v until all points has been used.

Figure 3.6(a) shows the selection of points to generate a sphere and Figure 3.6(b) shows the progress of the generation of spheres.



(a) Selection of points to generate the sphere

(b) Progress of generation of spheres

Figure 3.6 Generation of spheres using the points generated on the B-spline surface.

Spheres are generated on both the intersecting B-spline surfaces, and an intersection test is carried out between these spheres. The minimum and maximum u and v values for the intersecting spheres are noted. More points are generated within these intersecting spheres by selecting more lines and columns of the blending matrices in both *u* and *v* directions. Again spheres are generated and intersection tests are carried out to generate lower levels of detail.

The process is carried out alternatively for the model and the tool. This reduces the computational cost as compared to the case when the lower levels of details are generated simultaneously for the model and the tool. Initially more points are generated within the intersecting sphere(s) on the model, and the intersection test is carried out between these spheres and the intersecting sphere of the tool. At the next stage, more points and consequently spheres are generated within the intersecting sphere on the tool surface prior to the intersection test being performed. This loop, for generating lower levels of detail, works until all the values of *u* and *v* parameters have been used from the blending matrices. An exception occurs when the number of spheres generated at a particular level is equal to the number of intersecting spheres on both the tool and the model, in which case the loop is terminated. This is possible when two surfaces have a similar curvature in the region of collision. An interesting and computationally expensive situation will emerge when two flat surfaces come in contact. When both the model and the tool are deformable, there is strong possibility that both surfaces will deform during the interaction to yield a flatter surface of interaction. Similarly when a deformable model rests on a rigid plane (e.g. table) or is sculpted using a rigid plane tool, the region at collision will yield a flatter surface. During these interactions the region of contact is large and most of the collision detection algorithms for deformable models fail when the area of contact is so large. In this scenario all the spheres generated on the surfaces will be intersecting and lower levels of details will only add to the computational cost. The proposed algorithm for generating spheres stops generating more points and spheres in such a case and straightaway goes to the next step of tessellation of surface.

3.3.3.5 Tessellation of surfaces

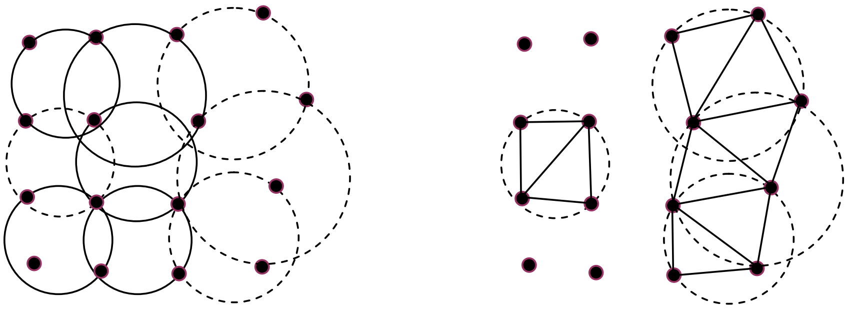
3.3.3.5.1 Generation of points for tessellation

There are two scenarios in which the algorithm for sphere generation ends the loop and moves to tessellation of the surface of probable collision. In the first scenario if all the values of *u* and *v* parameters have been used from the blending matrices, more points inside the intersecting spheres cannot be generated. In such a scenario, tessellation of the surface starts using the points within the u and v parametric values of intersecting spheres. As the number of intersecting spheres is small, only a small number of points will be generated on the surface for tessellation. At the same time the tangents as well as normals at these points are generated using the tangent blending matrices as discussed in Section 3.3.3.3.

In the second scenario the number of intersecting spheres is equal to the number of spheres generated for both the model and the tool. The number of points, and consequently triangles generated, will be large. This will not only increase the computational cost of the generation of triangles but also that of the intersection test of these triangles. To avoid the increase in computational cost, the proposed algorithm checks the normal of the points being generated in the region. If the normal of a point is within the limits of that of the previous point by a small number δ, this point is not generated. This way the points generated on the surface are significantly reduced, without sacrificing accuracy. This reduces the number of triangles to be generated.

3.3.3.5.2 Triangulation of surface

Tessellation of surfaces is a well researched topic. A large number of algorithms are available with varying cost of computation. Most of the algorithms, in the worst case scenario, achieve a computational cost of $\boldsymbol{O}(n \log n)$. The most commonly used triangulation algorithms are Delaunay based. A major drawback of these algorithms is that while being efficient for a convex object, these tend to triangulate cavities in the case of non-convex objects. Thus, the tessellated surface does not show the cavities. The algorithm presented in the thesis does not generate too many points for triangulation to reduce computational cost. As the deformable model can have steep curvature and sharp cavities, a general Delaunay triangulation method will be unable to efficiently generate accurate triangulated surface. Fortunately, the method of generating spheres used in this algorithm helps us generate accurate triangulated surface. As per one of the definitions of the Delaunay triangle, “All interior edges of a triangulation Δ of a point set are locally optimal if and only if no point from this point set is interior to any circumcircle of the triangle in Δ” [Øyvind, 2006]. As per the algorithm of generating spheres, there is a fourth point inside the sphere. This means that there cannot be an optimal triangle by joining the three points that are on the surface of sphere. Thus the fourth point is joined with all the points on the sphere surface. The distance between the three points on the surface is calculated. The largest side of the possible triangle between these three points is ignored and the rest of the lines are generated. This yields optimal triangles in an intersecting sphere. Figure 3.7(a) shows the spheres generated for the given points on the surface.



(a) Four intersecting spheres shown in dotted lines (b) Triangulation of surface using points in the spheres

Figure 3.7 Triangulation using the points inside intersecting spheres.

Similarly if there are many intersecting spheres with common points as shown in Figure 3.7(b), the same process will result in triangulated surface. As two points will be common, a regular mesh within the region is generated. Another advantage of the process is that all the intersecting spheres can be triangulated simultaneously rather than by adding one point at a time as in the case of most of the algorithms.

3.3.3.6 Triangle-triangle intersection test

Once the regions of probable collision of the surfaces of the virtual model and tool are tessellated, a triangle-triangle intersection test is carried out. The simplest approach of testing all the triangles of the model against all the triangles of the tool requires an immense number of triangle-triangle intersection tests. Thus, many algorithms have been devised to reduce the computational cost. In this respect the algorithm, presented in this thesis first limits the number of triangles of the model which can intersect with that of the tool. Only those triangles are checked for intersection whose bounding spheres are intersecting at the lowest level of detail. As the computational cost of a sphere-sphere intersection test is far less than that of a triangle-triangle test, the overall computational cost is reduced.

Once it is determined as to which triangles have the probability of intersection, an actual triangle-triangle test is carried out. A simple method for determining whether two triangles in a three dimensional space intersect, requires the solution of six sets of linear equations, each corresponding to an intersection of one triangle's edge with the surface of the other triangle. Two algorithms from the literature were considered that use less computation than this simplistic approach [Guigue, 2003; Möller, 1997]. The algorithm developed by Guigue [Guigue, 2003] is an improvement of that by Möller [Möller, 1997] and is computationally more efficient. The major limitation is that it cannot tackle degenerate cases. As the points are dynamically generated in our algorithm and the conditions are such that chances of degeneracy are minimal, this algorithm is very suitable. Depending upon the penetration of the tool in the model at various points, as computed by a triangle-triangle intersection test, the physics based model will calculate the deformation of the model and tool and the force feedback to the user.

3.3.3.7 Updating bounding volume

Once the physics based force response system determines the deformation and the new position of the points on the surface of the B-spline model, the control points are updated using the inverse matrices stored during pre-processing phase. Equation (3.7),

$$\mathbf{P}_{ij} = [\mathbf{A}_u{}^{\mathrm{T}}\mathbf{A}_u]^{-1}\,\mathbf{A}_u{}^{\mathrm{T}}\,\mathbf{M}\,\mathbf{A}_v{}^{\mathrm{T}}\,[\mathbf{A}_v\mathbf{A}_v{}^{\mathrm{T}}]^{-1},$$

will determine the new set of control points. As matrices are already stored and the program does not need to do the inverse of a matrix, the computational cost is substantially decreased. The change in the position of the control points warrants the updating of a new convex hull. The same function, as described in Section 3.3.2.2 is called to update it. All the steps described in Section 3.3.3 are then repeated as per the requirement until the desired shape of the model is achieved and/or the model is validated.

3.4 Special Cases

This algorithm is for collision detection between two or more B-spline surface model(s) and tool. In special cases a tool can be represented as a rigid implicit surface tool or a point based tool. This algorithm is capable of handling these special cases efficiently. In the case of a point based tool, the process discussed in Section 3.3.3 is followed for the model only. More points are generated in a sphere with which point-based tool is intersecting and the process continues till all the rows of the blending matrices have been used. Finally, two triangles will be generated within the intersecting sphere at the lowest level of detail. The intersection test of the point with these two triangles then determines the point where the tool is colliding with the model.

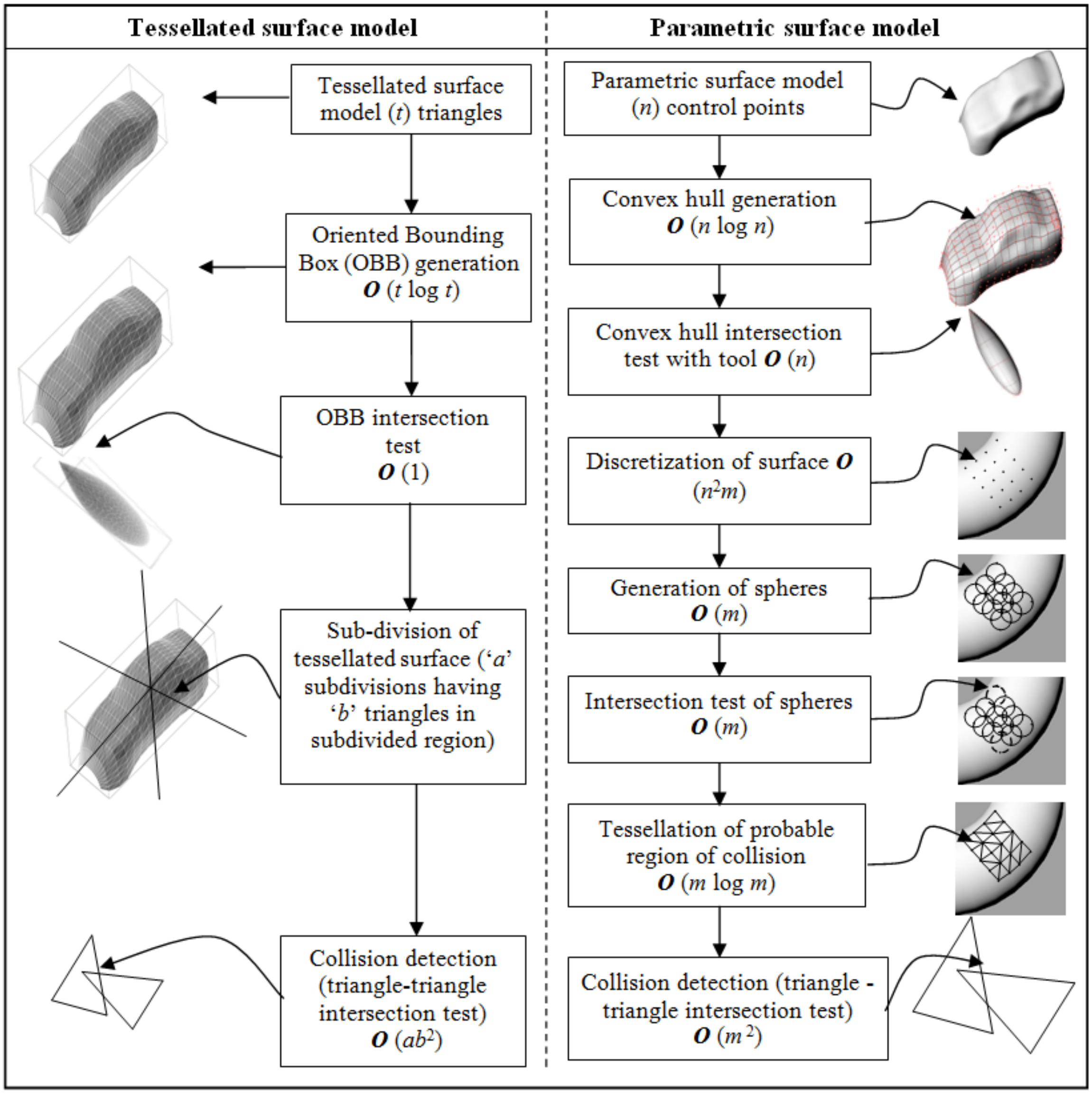
In the special case of an implicit surface tool, only part of the algorithm is used. In the algorithm, first the tool is checked for intersection with the convex hull of the B-spline. Once the implicit surface intersects the convex hull, the algorithm finds out the corresponding values of u and v parameters within which the intersecting surfaces of the convex hull lies. It then generates points on the surface within these limits using the blending matrices. These points are then input into the implicit surface to detect collision.

3.5 Analytical Comparison and Performance

The approach, presented in this thesis for collision detection, is to use the advantages of the B-spline surface representation and the computational efficiency of a triangle-triangle intersection test. Most of the existing algorithms use either of these (tessellated surface representation or B-spline/Bezier/NURBS surface representation) approaches. Hence we compare our algorithm with the tessellated surface representation and the B-spline/Bezier/NURBS surface representation.

3.5.1 Comparison with a tessellated model

Figure 3.8 shows the various steps required for complete collision detection for the tessellated model and of the proposed model, along with the computational cost considering the worst case scenario.


Tessellated surface model
Parametric surface model
Tessellated surface model (t) triangles
Oriented Bounding Box (OBB) generation O (t log t)
OBB intersection test O (1)
Sub-division of tessellated surface ('a' subdivisions having 'b' triangles in subdivided region)
Collision detection (triangle-triangle intersection test) O (ab²)
Parametric surface model (n) control points
Convex hull generation O (n log n)
Convex hull intersection test with tool O (n)
Discretization of surface O (n²m)
Generation of spheres O (m)
Intersection test of spheres O (m)
Tessellation of probable region of collision O (m log m)
Collision detection (triangle - triangle intersection test) O (m²)

***t* = number of triangles; *n* = number of control points; *m* = maximum number of points to be generated on B-spline surface**

Figure 3.8 Comparison of steps for collision detection for tessellated model with proposed algorithm.

Most of the collision detection algorithms, based on tessellated geometry, are for rigid bodies. Some of these algorithms have been used for deformable objects or proposed for use in deformable objects. Bordegoni [Bordegoni, 2006] relied on the algorithm of Barraff [Baraff, 1990] for rigid bodies and Gottschalk [Gottschalk, 1996] has proposed that, in future, the algorithm can be adopted for deformable bodies. In the absence of any benchmark for comparison of different algorithms, the worst case scenario represented as Big ***O*** notation, can be used as one of the tools to compare the results of proposed algorithm. Though constants and lower order terms are not used in the conventional representation of Big ***O*** notation, these terms have been used to provide a better comparison. For comparison, oriented bounding boxes (OBBs) are used as the bounding box [Gottschalk, 1996] for a deformable model represented as a tessellated surface. The total cost for each step is tabulated in Table 3.1.

Table 3.1 Comparison of computational cost of collision detection for tessellated model with the proposed algorithm.

Operations	Computational Cost	
	Tessellated Model	Parametric Model
Bounding Box	$O(t \log t)$	$O(n \log n)$
Collision Prediction	$O(1)$	$O(n)$
Discretization of B-Spline surface	Nil	$O(n^2m)$
Generation of Spheres	Nil	$O(\mathrm{m})$
Intersection Test of Spheres	Nil	$O(\mathrm{m})$
Tessellation of region of probable collision	Nil	$O(m \log m)$
Collision Detection	$O(ab^2)$	$O(m^2)$
Total computation cost	$O(t \log t) + O(1) + O(ab^2)$	$O(n \log n) + O(n) + O(n^2m) + O(m) + O(m) + O(m \log m) + O(m)^2$

The number of triangles that can be generated from a given set of points depends upon the location of the points, the total number of points and the topology of the surface being discretized. In general, the number of triangles generated is given by equation, $t = 2m - 2 - e$, where m is the number of points and e is the number of points on the convex hull of the triangulated surface. Considering that the number of triangles that can be generated is 1.5 times the number of points, the computational cost required for tessellated model with an equivalent number of points generated, can be compared. Different percentages of *area of contact* are considered, assuming an even distribution of triangles on the surface of the model.

Figure 3.9 shows the graphs for different areas of contact and the computational costs of the tessellated model and that of the B-spline surface model for various control points.

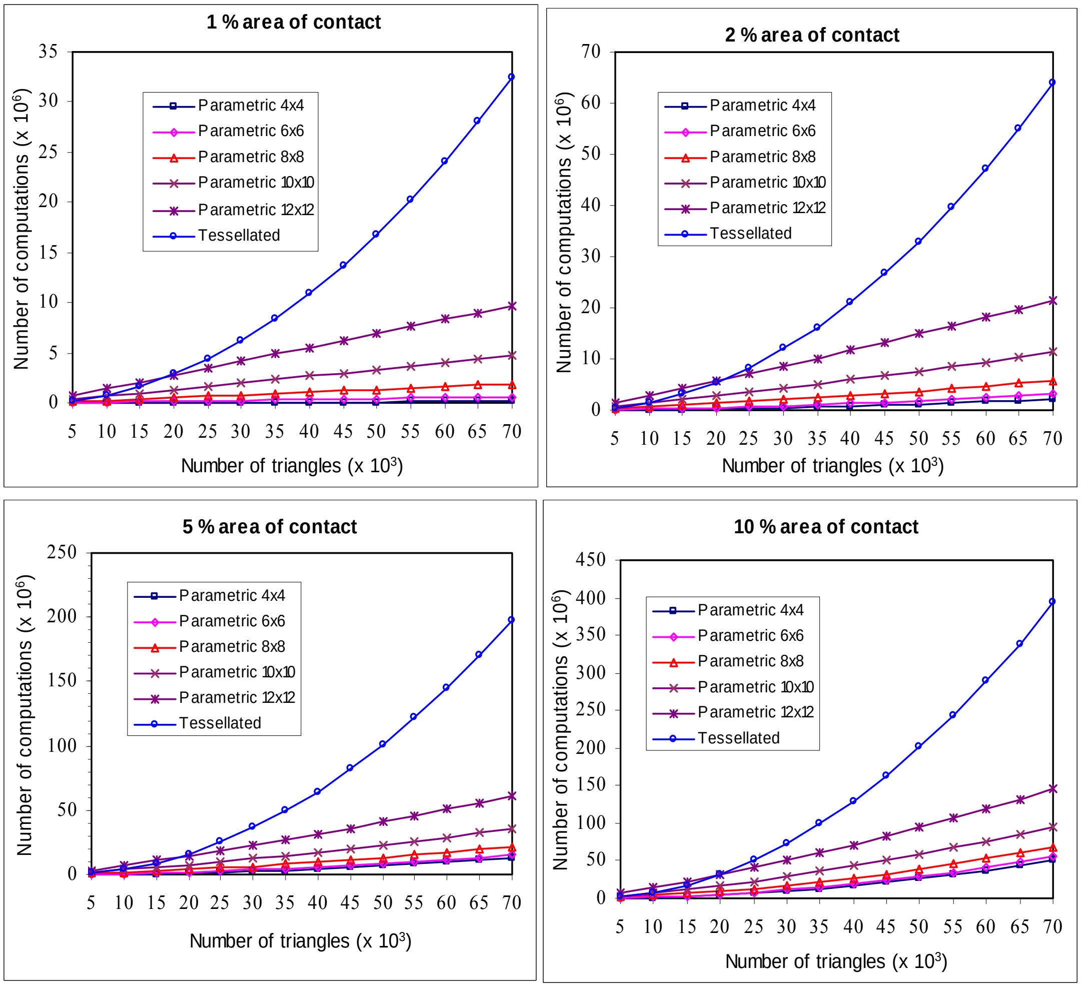


Figure 3.9 **Comparison of total computations required in the worst case scenario for collision detection of tessellated model versus the B-spline model using proposed algorithm.**

The "X" axis shows the number of triangles for the tessellated model and the equivalent number of triangles for the B-spline surface model. The "Y" axis shows the number of computations required for the worst case scenario. It is clear from the graphs that the computational cost of the proposed algorithm is much less than that of a tessellated model. This reduction in computational cost is more pronounced for a higher resolution. During the sculpting and evaluation of the model, multiple contacts and a large contact area are very common. The proposed algorithm is very suitable for this application, though it can be extended to other applications as well.

Another aspect of sculpting is that during the early interaction with the objects, the contours are not very sharp but the area of contact is very large. As the sculpting process progresses further, the details get finer and the area of contact is reduced. The proposed algorithm can aptly take advantage of this fact. Figure 3.10 shows the total computation cost of collision detection for different areas of contact for a 12 × 12 control point B-spline surface model.

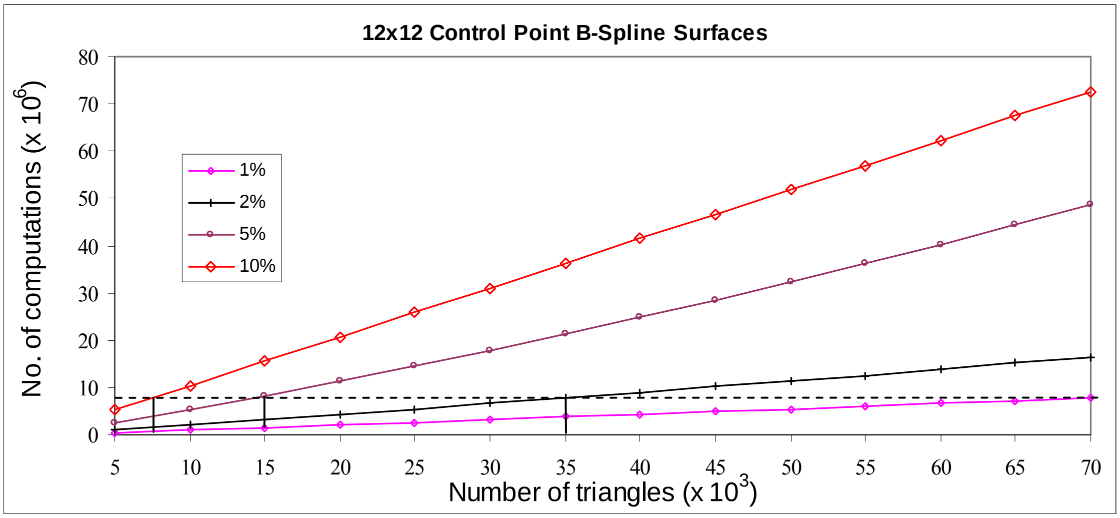


Figure 3.10 Maintaining lower number of computations required for collision detection by changing resolution of the surface.

The number of points generated on the surface can be decreased by using fewer rows and columns from the blending matrices as discussed in previous sections. Thus in the beginning when the surface to be sculpted does not have finer details but a large area of contact, the user can specify a lower resolution. This will reduce the number of points generated and therefore the computational cost will be lower. It is shown by the dotted horizontal line in Figure 3.10. Thus, by changing the resolution of the model (using fewer number of equivalent triangles), the number of computations can be maintained to less than 8×10^6, even during the worst case scenario while maintaining high resolution and accuracy of collision detection during the sculpting of the finer details of the model. The number of computations in an average case will be much lower. As the sculpting process progresses to the finer details, the number of points generated and hence the resolution and accuracy of collision detection can be increased without increasing the computational cost, as the area of contact is also being reduced.

3.5.2 Comparison with parametric surface models

A number of collision detection algorithms for parametric surface models representing deformable objects have been proposed. Most of these algorithms are based on subdividing the surface into patches and sub-patches until these are sufficiently planer. This is done during the pre-processing phase or the run-time phase. These sub-patches are then bounded by bounding boxes, mostly AABB's [Hughes, 1996]. If this process is carried out during the run-time, it is time consuming and computationally intensive. If it is done during the pre-processing phase, then the subsequent deformations of the model make the subdivision prone to large errors and inefficient collision detection. Most of these algorithms use point contact and are suitable for game engines and not for applications in interactive free form modeling. These algorithms also tend to fail when large surface areas are in contact. At the same time, these algorithms tend to deform a small region and change the position of a small number of control points. In physics based haptic interaction, collision at any point will change the geometry of a large part of the surface. Hence, a whole set of control points will need to be updated. As the proposed algorithm provides a different approach, it is not possible to compare this algorithm with these methods. Nonetheless, the proposed algorithm is capable of efficiently detecting the collision for any type of B-spline surface and any subsequent deformation does not affect the accuracy of collision detection. The user is given freedom to control the accuracy of collision detection and this increases the robustness of the system.

Gao and Gibson [Gao, 2005; 2006] used an implicit surface to represent the tool and used points generated to discretize the B-spline surface to detect collision. At any time, all the points are input in an implicit surface equation of the tool to determine if the surface is colliding with the tool. This limits the applications of the technique and it cannot handle a B-spline surface or point-based tool. The advantage of the proposed algorithm over the collision detection algorithm presented by Gao and Gibson, is that it uses the points only in the region of probable collision and not on the entire surface of the model. This reduces the cost of collision detection. Furthermore, if the tool and model are apart (convex hull of B-spline model not intersecting with implicit surface), there is no need to input a large number of points in the equation of implicit surface to detect collision. The proposed algorithm is also capable of detecting collision between a B-spline surface and a point based tool unlike the algorithm used by Gao and Gibson.

3.6 Concluding Remarks

In this chapter, the collision detection algorithm for single B-spline surface patch was introduced. No limitation has been imposed on the shape, complexity, degree or the number of control points of the B-spline surface representing the tool or model. Both the model and the tool can have complex shape, elastic or plastic properties, and multiple contacts.

The algorithm is also capable of detecting the collision of a B-spline surface model with a tessellated surface, implicit surface or a point based tool. This will allow the user to use rigid or deformable tools with complex shapes and enhance the ease and productivity during the sculpting or concept validation in virtual reality environment. The ‘on the fly’ generation of points and triangles helps to maintain the quality of triangles. At the same time resolution of the model can be varied during haptic interaction, as needed, thereby reducing the overall computational cost. Although the algorithm uses a triangle-triangle intersection test for collision detection, it is far more efficient than a tessellated surface deformable model. The novel method of generating spheres at different levels of detail to find out the regions of the surface likely to collide, allows multiple contact collision detection. It is more efficient than an octree subdivision as instead of subdividing whole space only the spheres which are intersecting are subdivided. The sphere-sphere intersection test being more efficient as compared to a triangle-triangle intersection test, reduces the cost of determining the regions of probable collision and helps reduce the number and hence the cost of generation and collision detection of triangles on the surface.

The novel technique of comparing the normals of the points generated on a flat surface reduces the computational cost of collision detection between two flat B-spline surfaces. This makes the algorithm robust as it can handle a potentially computationally expensive situation at a much lower computational cost. In general, calculation of inverse of matrices is required to determine new set of control points representing the B-spline surface after deformation. A matrix inverse calculation can be problematic and sometimes impossible. The calculation and storage of blending matrices and their inverse during the preprocessing stage makes sure that no inverse need to be computed during run-time phase of the algorithm. This makes the algorithm robust and efficient.

CHAPTER 4 MERGING MULTIPLE B-SPLINE SURFACE PATCHES

4.1 Introduction

Computationally efficient bi-parametric functions, such as B-spline and NURBS (non-uniform rational B-spline), are commonly used to model organic and freeform shapes and, therefore, provide a viable mathematical representation for describing the geometry of realistic objects in VR space. CAD systems are able to create shapes by stitching or joining together numerous low-order bi-parametric patches. Unfortunately, many collision detection algorithms used in VR environments [Jimenez, 2001; Lin, 2004] do not permit more than a single B-spline or NURBS surface patch to be considered at any instant in time. Even when a collision detection algorithm is designed to tackle multiple B-spline surface patches [Gao, 2006; Hughes, 1996; Pungotra, 2008], the physics-based system cannot determine the deformation of these multiple joined B-spline patches unless these are combined into a single, more complicated integral surface. This restriction exists because the physics engine will represent the solid contained within the closed surface as a mass-spring system. Consequently, stitching the surface patches does not automatically connect the underlying dissimilar mass-spring networks. Furthermore, if an object is represented by multiple B-spline surface patches stitched together with a pre-defined continuity (C^0, C^1 or better), then any subsequent deformation of the object during haptic interaction would separate the patches and result in undesired surface representation.

Generating a complex shape from a single B-spline patch is both tedious and limits the scope of a virtual reality based interactive shape design or other applications. This problem can be solved, if the virtual reality engine allows multiple B-spline patches to be easily merged into a single integral B-spline surface. This chapter discusses the step by step implementation of the computationally fast algorithm for combining two or more dissimilar B-spline surface patches in virtual reality environment. The proposed method extends the collision detection algorithm described in Chapter 3 and utilizes the blending matrices for efficiently merging two or more B-spline surface patches. No assumption regarding the complexity, degree, curvature at the edges, or the number of control points representing the surface patches is imposed on the solution. The user would have the option to select the connectivity of the merged surfaces along the common edge. The developed algorithm performs more robustly than the common NURBS based modeling software tool, Rhino®. To illustrate the capabilities of the algorithm and verify its performance, several case studies are presented and an error analysis, based on standard deviation between the single merged surface and the original constituent surface patches is provided in this chapter.

4.2 Related Work

The early work on combining parametric functions was done on Bézier, B-spline and NURBS curves. The degree reduction and merging of Bézier/B-spline curves into a single representation have been addressed in the literature by a variety of different analytical approaches [Cheng, 2008; Hu, 2001; Piegl, 1994; 1995; Taia, 2003; Yong, 2001]. Taia *et al.* [Taia, 2003] introduced an approximate solution for merging two adjacent B-spline curves by adjusting the control points through constrained optimization. The algorithm combines the two curves by letting both B-spline curves share a common derivative. Hu *et al.* [Hu, 2001] also proposed a Bézier curve merging technique using the constrained optimization method. The basic idea is to find conditions for the precise merging of Bézier curves first, and then compute the constrained optimization solution by moving all the control points. In contrast, Cheng and Wang [Cheng, 2008] proposed an alternative technique that creates a unified matrix representation from multiple adjacent Bézier curves that have different degrees of curvature. Continuity at the endpoints is achieved by using partitioned matrices so that the last point of a Bézier curve matches the first control point of the next curve during the merging process.

The conditions of geometric continuity between two adjacent bi-parametric surface patches have been described in the literature [Du, 1990]. Shi [Shi, 2004] presented the algorithm to obtain G^1 continuity for bicubic B-spline surfaces with single interior knots over an arbitrary quad partition of a polygonal model. Che [Che, 2005] presented an improved algorithm for G^1 continuity conditions for two adjacent NURBS surfaces with arbitrary degrees and generally structured knots. The continuity conditions allow the user to represent a model having multiple B-spline surface patches. However, if one or more of these patches are deformed then the continuity no longer exists. Hence, these techniques cannot be used in a VR environment when the user intends to deform the surface to obtain the desired shape. Figure 4.1(a) shows the two surfaces, stitched together with C^1 continuity. As shown in Figure 4.1(b), when one of the surfaces is deformed, the continuity no longer exists.

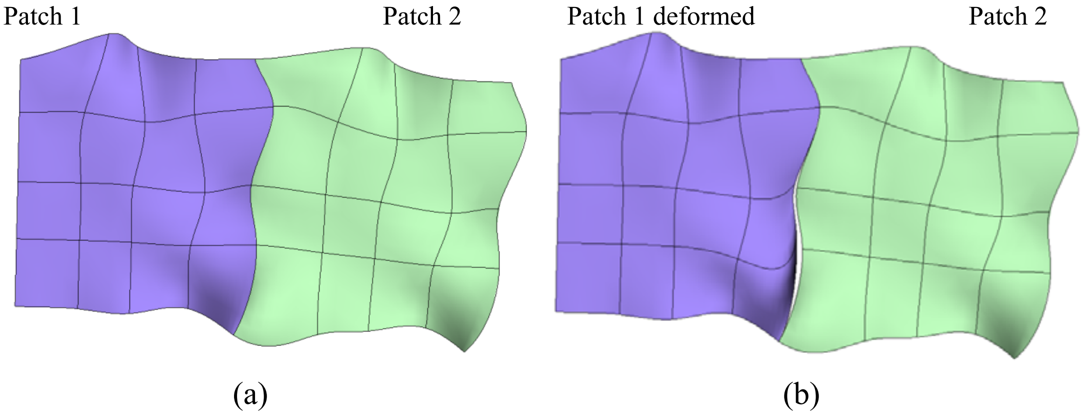


Figure 4.1 (a) Two B-spline surface patches matched with C^1 continuity (b) Surfaces after one surface patch is deformed.

To maintain continuity, even while the object is being deformed, the constituent patches need to be merged into a single B-spline surface. Unfortunately, the problem of merging B-spline surface patches is more complicated than curves because the individual control points have influence on the shape in both *u* and *v* direction. In addition, computational time and algorithm efficiency are important constraints on any virtual reality application where real-time force feedback is required for haptic interaction. Any viable technique for merging B-spline surface must also work in tandem with the collision detection algorithms in order to optimize system performance.

4.3 Merging Multiple B-spline Surface Patches

In this section, the algorithm developed for merging two or more B-spline surface patches is discussed in detail. The algorithm uses blending matrices associated with each B-spline surface being merged. Since the algorithm uses the blending matrices that have been previously computed for collision detection [Pungotra, 2008], it is not necessary to re-calculate these matrices while manipulating the patches in VR space. However, if the blending matrices are not pre-calculated, the algorithm can also generate blending matrices.

The algorithm to merge N number of B-spline surface patches, S^1, S^2,…, S^N; having (r_1, s_1), (r_2, s_2), …, (r_N, s_N) number of control points in u and v directions respectively, starts by discretizing the surfaces to be merged. These discretized matrices $\mathbf{M}1$, $\mathbf{M}2$,…, $\mathbf{M}N$ are combined together to generate a matrix $\mathbf{M}$ of discrete points. The algorithm calculates the revised number of control points "r" and "s" for the merged surface in u and v directions respectively. The knot vector U and V are computed by combining the knot vectors (U^1, V^1), (U^2, V^2),…., (U^N, V^N) of the surfaces being merged. Once this information is known, it can be used to calculate revised blending matrices $\mathbf{A}_u$ and $\mathbf{A}_v$ for the merged surface. These revised blending matrices help to find out the matrix of the control points $\mathbf{P}_{ij}$, which can generate the merged surface approximating the discrete points of the combined matrix $\mathbf{M}$.

Figure 4.2 shows the flow chart, describing the steps needed to generate a merged surface from two B-spline surface patches. This can be further extended to merge any number of B-spline surface patches. To understand the algorithm better, it will be initially assumed that there are only two B-spline surfaces patches of similar degree and these surfaces are spatially close but do not intersect each other. The special cases of intersecting surfaces, surfaces having different degrees, and merging of multiple (four) surfaces are described separately in Section 4.5. These steps are discussed in detail in the following sub-sections.

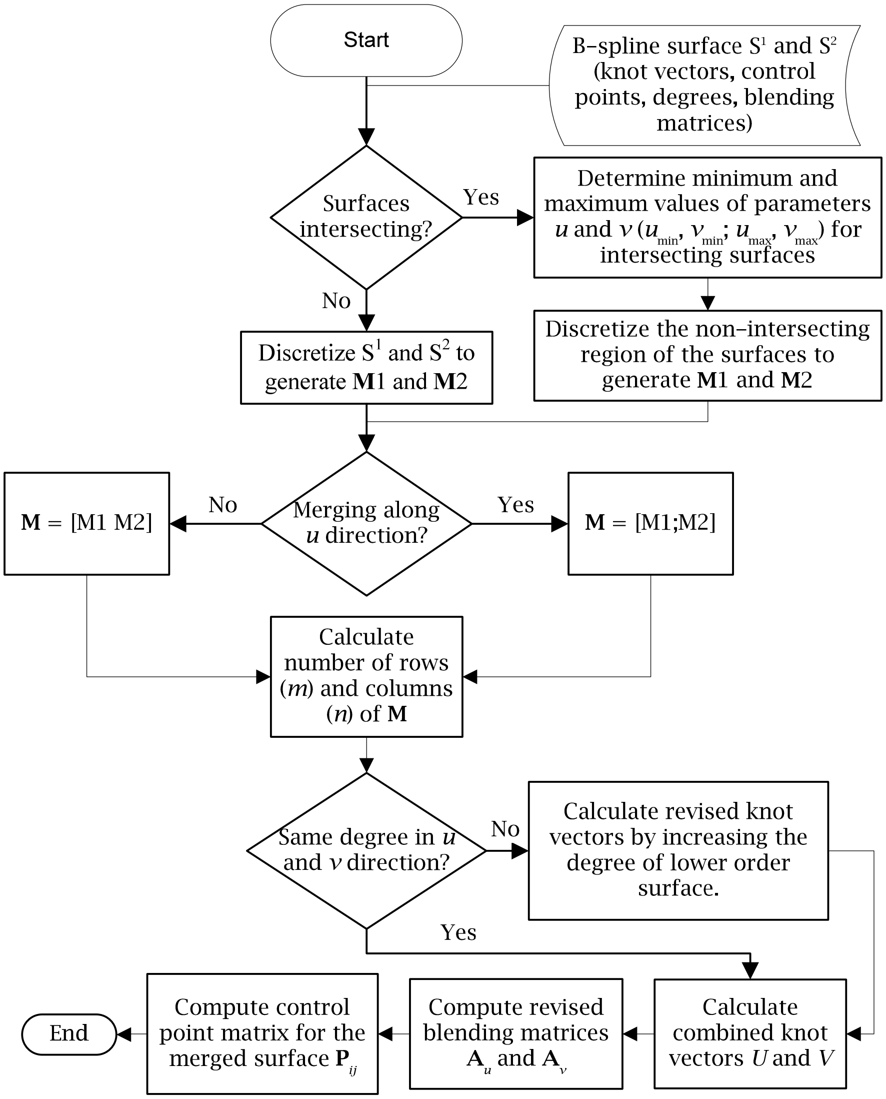
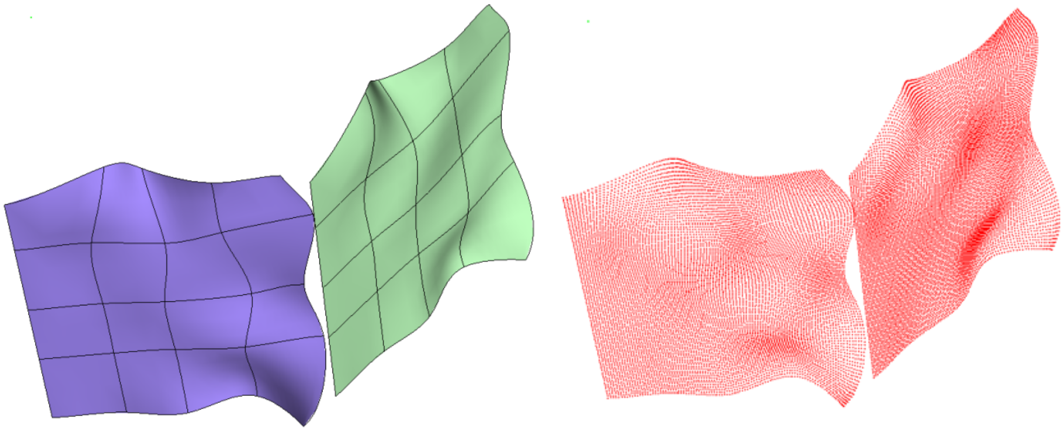

Start
B-spline surface S¹ and S² (knot vectors, control points, degrees, blending matrices)
Surfaces intersecting?
Yes
Determine minimum and maximum values of parameters u and v (umin, vmin; umax, vmax) for intersecting surfaces
No
Discretize S¹ and S² to generate M1 and M2
Discretize the non-intersecting region of the surfaces to generate M1 and M2
M = [M1 M2]
No
Merging along u direction?
Yes
M = [M1;M2]
Calculate number of rows (m) and columns (n) of M
Same degree in u and v direction?
No
Calculate revised knot vectors by increasing the degree of lower order surface.
Yes
End
Compute control point matrix for the merged surface Pij
Compute revised blending matrices Au and Av
Calculate combined knot vectors U and V

Figure 4.2 Flow chart of the proposed algorithm for merging two B-spline surface patches.

4.3.1 Discretization of B-spline surfaces

The surfaces that need to be merged are first moved closer together and allowed to touch at one or more points. The blending matrices representing the underlying surfaces are then used to discretize the individual B-spline surface patches. These discrete points are stored as matrices $\mathbf{M}1 = \mathbf{A}^1_u\,\mathbf{P}^1_{ij}\,\mathbf{A}^1_v$ and $\mathbf{M}2 = \mathbf{A}^2_u\,\mathbf{P}^2_{ij}\,\mathbf{A}^2_v$. When the merging process starts, a combined matrix, **M**, is generated by the combination of these matrices. This can be expressed as $\mathbf{M} = [\mathbf{M}1\ \mathbf{M}2]$ if the matrices are to be combined column-wise, or $\mathbf{M} =$

[**M**1;**M**2] if these are to be combined row-wise, depending upon the direction in which these surface patches are being merged. The new number of rows (m) and columns (n) of the combined matrix, **M,** are determined. Figure 4.3 shows the two constituent B-spline surface patches and the dense distribution of discretized surface points obtained from the patches.



(a) (b)

Figure 4.3 (a) B-spline surface patches that are to be merged together (b) Discretized points on the surface, as generated by using the blending matrices.

4.3.2 Determining revised number of control points

When the B-spline surface patches are combined, the number of control points in u and/or v directions will change. Consider the situation where the surfaces are being joined only in the u direction. The new number of control points in the u direction is then given by $r = r_1 + r_2 - 1$, where r is the number of control points of the merged surface in u direction, and r_1 & r_2 are the number of control points in u direction for the first and the second surface respectively. The number of control points in v direction would remain the same, provided the number of control points is the same for both the surfaces in that direction, that is, if $s_1 = s_2$. If surfaces have a different number of control points in v direction then the larger number is assumed as the new number of control points. Similarly, if the two surfaces are joined in the v direction, the number of control points in v direction will be given by $s = s_1 + s_2 - 1$.

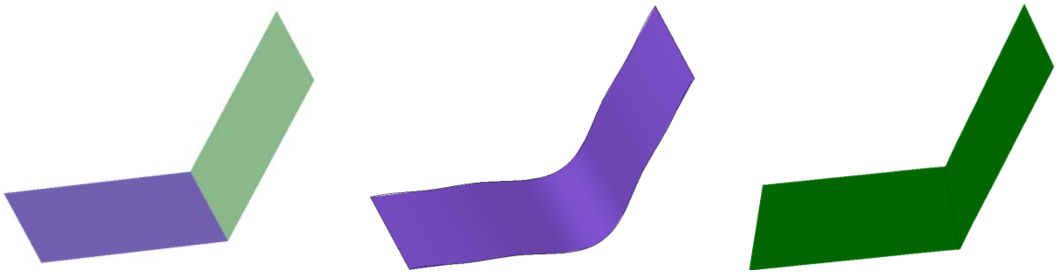
4.3.3 Determining the new knot vector

A knot vector determines the area of influence of each control point on the B-spline surface. The two surfaces can have a uniform or non-uniform knot vector. Even if both the surfaces have a uniform knot vector, these may not align with each other. For this reason, it is better to recalculate the knot vector for the merged surface. Most often a uniform knot vector is best suited because the algorithm is designed to tackle any general case.

If the degree of the initial surfaces is not being changed, the algorithm simply uses the knot vector of the first surface and then adds to it the knot vector of the second surface, in the direction of merging. As an example, if the knot vectors of the two cubic B-spline surface patches, being merged in *u* parametric direction, is given by U^1 = [0, 0, 0, 0, 0.2138, 0.4959, 0.7262, 1, 1, 1, 1] and U^2 = [0, 0, 0, 0, 0.1215, 0.2512, 0.3689, 0.5047, 0.7283, 0.8631, 1, 1, 1, 1]; then the combined knot vector, achieved by adding and normalizing these knot vectors, is given by U = [0, 0, 0, 0, 0.1069, 0.2478, 0.3631, 0.5, 0.5, 0.5, 0.5608, 0.6845, 0.7524, 0. 8642, 0.9316, 1, 1, 1, 1]. Typically the average of the last multiple knots of the first knot vector (1, 1, 1, 1) and the first knots of the second knot vector (0, 0, 0, 0) generate the multiple knots at the common edge (0.5, 0.5, 0.5, 0.5). However, only a maximum of *k* multiple knots can be retained, where *k* is the degree of the merged surface in the direction of merging. Thus in the given case, only three (*k*) multiple knots are retained at the common edges out of four obtained by combining the knot vectors, providing only C^0 connectivity at the common edge.

The multiple knots at the common edge are further reduced, depending upon the type of connectivity needed at the edge. For C^1 connectivity, the number of multiple knots would be *k*-1 and so on. For the maximum connectivity (C^2 for a degree 3 surface) only one knot is retained at the common edge. The other multiple knots are changed by averaging them with their neighboring knots. In this case, for C^2 connectivity at the common edge, the knot vector will be given by U = [0, 0, 0, 0, 0.1069, 0.2480, 0.3631, 0.4317, 0.5, 0.5304, 0.5608, 0.6845, 0.7524, 0. 8642, 0.9316, 1, 1, 1, 1]. The knot 0.4317 was obtained by averaging the knots, 0.3631 and 0.5, whereas the knot 0.5304 was obtained by averaging the knots 0.5 and 0.5608. As shown in Figure 4.4(c) and Figure

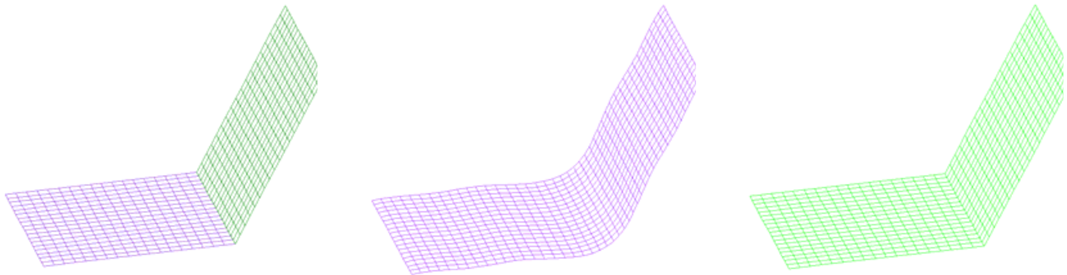
4.4(f) the common edge of the merging surface patches remains straight even after merging, in case of C^0 connectivity. Figure 4.4(b) and Figure 4.4(e) shows the merged surface with C^2 continuity at the common edge. It can be seen that to maintain C^2 connectivity at the common edge, the flat surface patches deviate in the middle as well, whereas these remained flat for C^0 connectivity.



(a) Original surfaces with degree three.

(b) Merged surface with C^2 continuity by deleting all multiple knots at the common edge.

(c) Merged surface with C^0 continuity by retaining three multiple knots at the common edge.



(d) Original surfaces, shown as wireframe, with degree three.

(e) Merged surface, shown as wireframe, with C^2 continuity by deleting all multiple knots at the common edge.

(f) Merged surface, shown as wireframe, with C^0 continuity by retaining three multiple knots at the common edge.

Figure 4.4 The effect of multiple knots at the common edge of the merging surfaces.

In the other parametric direction, the knot vector is generally made uniform. However, by using the average of the knots of two surfaces in the second direction, the tolerance in some of the cases can be reduced. In general, a uniform knot also yields good results and avoids the computation of average knot vectors for the merged surface.

4.3.4 Revised blending matrices

As discussed in the previous section, the degrees of the final merged surface (k, l), its knot vectors (U, V), the number of control points (r, s) in the u and the v directions respectively, and the total number of discrete points to be generated in the u and the v directions are calculated. Once these parameters are known, the new set of blending matrices, $\mathbf{A}_u$ and $\mathbf{A}_v$, can be generated. The number of rows of the matrix $\mathbf{A}_u$ and the number of columns of the matrix $\mathbf{A}_v$, are determined by the number of rows (m) and the number of columns (n) of the combined matrix of discrete points, **M**. The basis functions are evaluated for discrete values of parameter u and v. These parameter values are increased by a step of $1/(m-1)$ and $1/(n-1)$ in the u and the v directions respectively. As an example, for generating 101 number of points in u parametric direction, the basis functions are evaluated at $u = 0, 0.01, 0.02, \ldots, 0.99, 1$. As the degrees and the knot vectors of the B-spline surface are known, these basis functions can be calculated and stored as matrices $\mathbf{A}_u$ and $\mathbf{A}_v$. The following pseudo-code describes the process of computation of the revised blending matrix $\mathbf{A}_u$ (**A***u* in pseudo-code).

```
Algorithm:   Computation of the revised blending matrix Au
Parameters:  m, U, r, k
for (u = 0:1/(m-1):1)
    for (a = 0:1:k)
        for (i = 0:1:r+k+1-a)
            $B_{i,a}(u)=(u-U_i)\frac{B_{i,a-1}(u)}{U_{i+a}-U_i}+(U_{i+a+1}-U)\frac{B_{i+1,a-1}(u)}{U_{i+a+1}-U_{i+1}}$
            $B_{i,a}(u)=\begin{cases}1, & U_i \le u \le U_{i+1}\\ 0, & otherwise\end{cases}$
        end
    end
    x = integer of (u X (m-1))
    for (j = 0:1:r)
        $\mathbf{A}u_{x,j} = B_{i,k}$
    end
```

```
end
return Au
```

Once these blending matrices are computed, the inverse of these blending matrices ($[\mathbf{A}_u{}^{\mathrm{T}}\,\mathbf{A}_u]^{-1}$, $[\mathbf{A}_v\,\mathbf{A}_v{}^{\mathrm{T}}]^{-1}$) is also computed and stored. These newly calculated blending matrices of the merged surface ($\mathbf{A}_u$, $\mathbf{A}_v$), replace the earlier blending matrices ($\mathbf{A}^1{}_u$, $\mathbf{A}^2{}_u$; $\mathbf{A}^1{}_v$, $\mathbf{A}^2{}_v$) for the two B-spline surfaces.

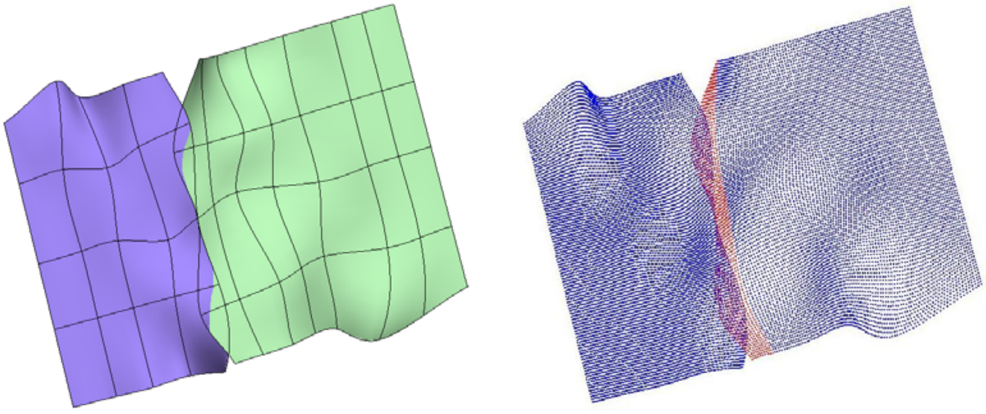
These new blending matrices are used to determine the position of the control points which can generate the matrix of discrete points (**M**). The position of the control points for the merged surface is once more calculated using Eqn. (3.6). The new blending matrices can also be used for collision detection and generation of nodes for a mass spring system as described in [Pungotra, 2009b].

4.4 Special Cases

The proposed algorithm is suitable for VR environments because the user does not need to directly manipulate the control points of the various surfaces. Furthermore, the algorithm does not impose any kind of restriction such as the degree of the surfaces, the knot vectors, or the type of connectivity. The general case, in which the merging B-spline surface patches were near to each other but were not intersecting, was considered in the discussion to make the algorithm understandable. While working in a VR environment it may not be possible to position the B-spline surface patches close enough. The surfaces may start intersecting at some regions of the surfaces, when positioned close to each other. It is also possible that the two B-spline surface patches can have different degrees of surface curvature in one or both the directions. It will require the algorithm to increase or decrease the degree of one of the surfaces so that the resultant merged surface has the same degree in any parametric direction. There is a greater chance of unchanged geometry for an increase in the degree of a surface. Conversely if the degree of a surface is decreased then the end result cannot be guaranteed to have a low tolerance unless the surface has some redundant knots. The user may also like to merge a large number of patches simultaneously. This section considers an arbitrary case in which the surfaces are intersecting before merging, an arbitrary case in which the surfaces have different degrees in both the directions, and a case for merging multiple (four) patches simultaneously.

4.4.1 Intersecting and trimmed surfaces

If the patches to be merged are intersecting, the resulting surface may turn out to be quite different than that intended by the user. The algorithm presented in this thesis uses a novel technique to merge two or more intersecting B-spline surface patches. First, the algorithm uses the collision detection algorithm [Pungotra, 2008] to determine if the two surfaces are intersecting (colliding). When the user intends to merge the surfaces, the collision detection algorithm calculates the regions which are colliding. The collision detection at the lowest level of detail determines the minimum and maximum values of the parameters u and v ($u_{\min}$, $v_{\min}$; $u_{\max}$, $v_{\max}$) for both the surfaces. These minimum and maximum values of u and v are used to set the limits on the surfaces to be discretized as **M**1 and **M**2. Figure 4.5 illustrates the intersecting surfaces. Figure 4.5(b) shows the portions of the surfaces that will be used for merging to generate matrix **M**. The discrete points generated in the intersecting region of the surfaces, shown in red, are discarded.



(a) (b)

Figure 4.5 (a) Intersecting surfaces to be merged (b) Region of the surface selected for discretization (The points shown in red are discarded).

This methodology also helps if the user would like to remove the uneven end portion of the surface which is created unintentionally while manipulating the surface. One limiting factor of this technique is that only full rows of the points can be discarded. Thus, even if some part of the surface is not intersecting, it will have to be discarded so as to maintain the rectangular matrix of discrete points.

Sometimes, a B-spline surface patch is trimmed. In this case, though the control point matrix and knot vector remain unchanged, the trimmed section of surface is not displayed to the user. For this reason, trimmed surfaces cannot be merged using commercially available CAD/CAM software. The present algorithm for intersecting surfaces can be easily extended to the merging of trimmed surfaces. If the user intends to merge a trimmed surface, the surfaces will actually be intersecting. The intersection will be occurring in the region which exists mathematically in the data base but is not displayed to the user. Hence, in case of trimmed surface, the discrete points belonging to the hidden surface (not being displayed the user) are discarded and only the points generated in the untrimmed region are used to generate matrix **M**.

4.4.2 Surfaces having different degrees

Frequently in CAD, particularly when a multiple B-spline patches surface, created from a point cloud data is imported, the degree of various surface patches may be different. Before the surfaces can be merged, the degree of the surfaces in u and v directions will have to be made uniform. Thus, the problem reduces to increasing or decreasing the degree of a surface to make it uniform with that of the other surface. There are many analytical techniques to increase or decrease the degree of a B-spline curve. Piegl and Tiller [Piegl, 1994; 1995] use a simple technique of first extracting Bézier segments from the curve. The degree of these Bézier segments is then increased/decreased by adding/deleting additional knots (control points) using constrained optimization. These Bézier segments are merged together and the multiple knots are deleted. However, for the surfaces the constrained optimization techniques are difficult to use. Degree reduction for surfaces can be achieved by first extracting iso-parametric curves from the surface and reducing their degree, before the B-spline surface is regenerated by lofting these lower order curves. In general, the process of surface approximation consists of approximating the boundaries and a number of iso-parametric curves of the original surface. These curves are then lofted to form an approximating surface [Tuohy, 1993]. Knots are added to the knot vector of the boundary curves at locations where the error exceeds a given tolerance, and new iso-parametric curves are extracted and approximated.

The proposed algorithm just needs to change the knot vector and the number of control points needed to represent the surface of higher or lower degree. If the degree is increased, the number of control points needed to represent the higher degree merged surface is increased by the same number. Thus, if the surfaces are merged in u direction then the resultant number of control points is given by $r = r_1 + r_2 - 1 + d$, where r is the number of control points of the merged surface in u direction, r_1 & r_2 are the number of control points in u direction for the first and the second surface respectively, and 'd' is the increase in degree of one of the initial B-spline surface patch to match with the degree of the other surface in the u direction. The number of control points is decreased by 'd' if the degree of the higher order surface is decreased to match that of the lower order surface. However, reducing the degree of a surface may result in reduced accuracy of the resultant surface. The number of knots are also increased/decreased by 'd' when the degree of a surface is elevated /reduced in a given parametric direction.

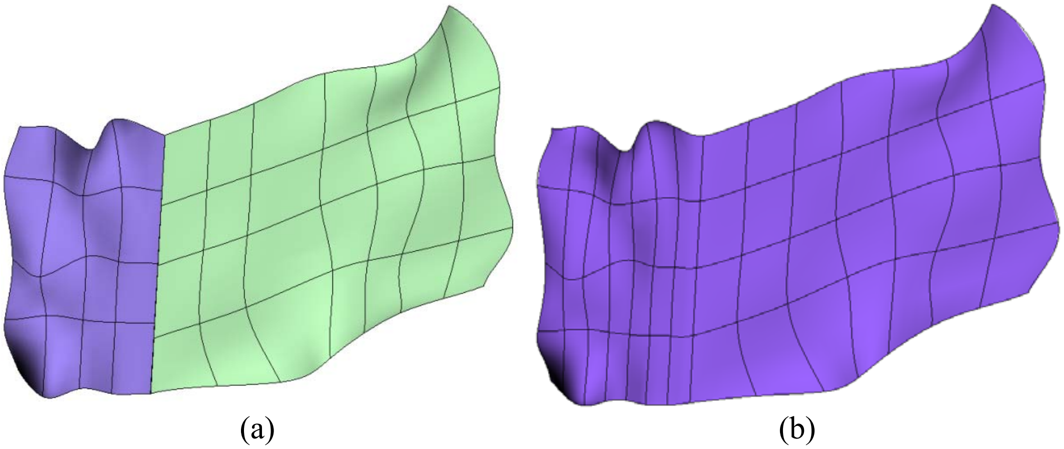

(a) (b)

Figure 4.6 (a) Initial B-spline surface patches of order 4×4 and 5×6 respectively (b) Merged surface of order 5×6 with C^4 connectivity in v direction, generated by proposed algorithm.

Figure 4.6(a) shows two surfaces of order 4×4 and 5×6 respectively. The surfaces are being merged in v direction. Thus, the first surface will have to be elevated to degree four in u direction and degree 5 in v direction (‘d’ is 1 in u direction and 2 in v direction). Figure 4.6(b) shows the merged surface of order 5×6 using the proposed algorithm. The proposed algorithm can also generate the merged surface by decreasing the degree of the higher order surface. On a standalone basis, this algorithm can be used to simply increase or decrease the degree of a surface.

4.4.3 Multiple surfaces

The algorithm is capable of merging any number of surfaces. The surfaces that are required to be merged are first moved closer together and allowed to touch at one or more points. The blending matrices representing the underlying surfaces are then used to discretize the individual B-spline surface patches. The discrete points of these N B-spline surface patches are stored as matrices $\mathbf{M}1 = \mathbf{A}^1{}_u\, \mathbf{P}^1{}_{ij}\, \mathbf{A}^1{}_v, \ldots, \mathbf{M}N = \mathbf{A}^N{}_u\, \mathbf{P}^N{}_{ij}\, \mathbf{A}^N{}_v$. When the merging process starts, a combined matrix $\mathbf{M}$ is generated by the combination of these N matrices of discrete points. Consider a case where four B-spline surface patches are to be merged. The combined matrix of discrete points can be expressed as $\mathbf{M} = [\mathbf{M}1\ \mathbf{M}2;\ \mathbf{M}3\ \mathbf{M}4]$ or any other such combination, depending upon the direction in which these surface patches are being merged. The new number of rows (m) and columns (n) of the combined matrix $\mathbf{M}$ are determined.

The algorithm adds the knot vectors of the B-spline surface patches depending upon the direction in which these surface patches are being merged. Once again, consider the case where the combined matrix of discrete points is expressed as $\mathbf{M} = [\mathbf{M}1\ \mathbf{M}2;\ \mathbf{M}3\ \mathbf{M}4]$. The algorithm first calculates the combined knot vectors (U^{12}, V^{12}) of the surfaces S^1 and S^2, and the combined knot vectors (U^{34}, V^{34}) of the surfaces S^3 and S^4. As discussed in Section 4.3.3, certain number of multiple knots is retained at the common edge depending upon the connectivity required at the common edge. In a similar fashion, these knot vectors (U^{12}, V^{12}; U^{34}, V^{34}) are further combined to get the resultant knot vectors for the merged surface (U, V).

The revised number of control points are calculated as discussed in Section 4.3.2. The combined number of control points in v direction are given by $s_{12} = s_1 + s_2 - 1$ and $s_{34} = s_3 + s_4 - 1$, where s_1, s_2, s_3 & s_4 are the number of control points in v direction for S^1, S^2, S^3 and S^4 surfaces respectively. If the number of control points in v direction, s_{12} and s_{34}, are different then the larger number is assumed as the new number of control points (s). Similarly, the combined number of control points in u direction are given by $r_{13} = r_1 + r_3 - 1$ and $r_{24} = r_2 + r_4 - 1$, where r_1, r_2, r_3 & r_4 are the number of control points in u direction for S^1, S^2, S^3 and S^4 surfaces respectively. The number of control points in u direction will then be given by the larger of r_{13}, r_{24}.

Once the revised knot vectors (U, V), the revised number of control points (r, s), and the total number of discrete points to be generated in u and v direction (m, n) are known, the new set of blending matrices $\mathbf{A}_u$ and $\mathbf{A}_v$ can be generated. These revised blending matrices are generated as discussed in Section 4.3.4. When these revised blending matrices are known, the new set of control points representing the merged surface can be determined by using Eqn. (6). Figure 4.7(a) shows four B-spline surfaces to be merged and Figure 4.7(b) shows the resultant merged surface obtained by using the proposed algorithm.

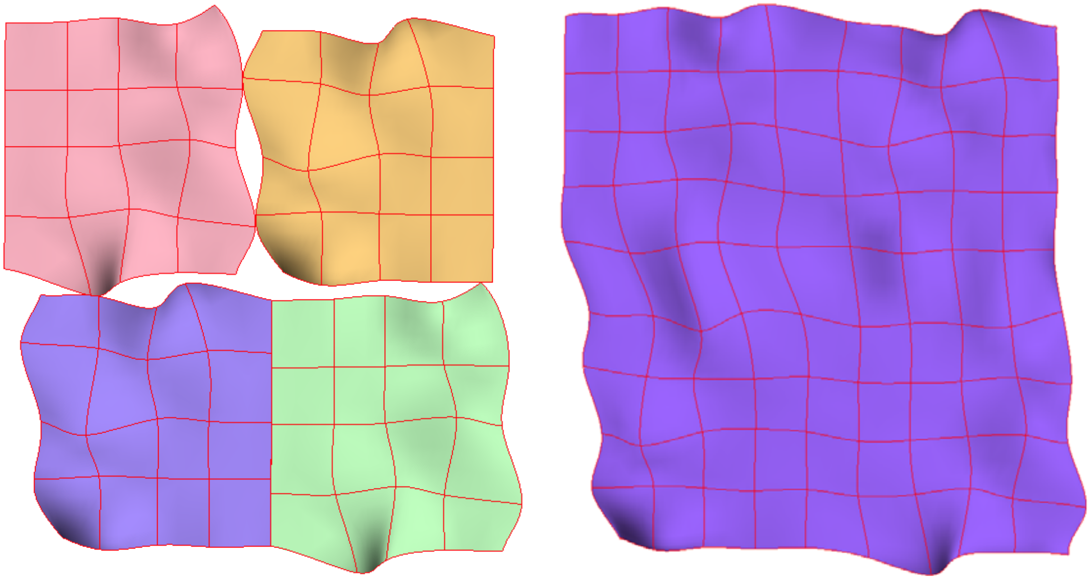

(a) (b)

Figure 4.7 (a) Initial multiple B-spline surface patches (b) Merged surface with C^2 connectivity generated by proposed algorithm.

4.5 Concluding Remarks

Product concept generation within a virtual reality environment requires a large variety of interactive tools that enable the user to enhance his or her creativity. One problem, which had not been addressed so far, is efficiently creating complex shapes by combining multiple dissimilar B-spline surface patches in a VR environment. The algorithm presented in this chapter allows the user to combine multiple B-spline surface patches in to a single B-spline surface. The algorithm proposed in this chapter is computationally efficient and exploits blending matrices of the surface patches used by the collision detection algorithm. It creates new blending matrices for the merged surface, which replace those for the original surface patches. Once created, the new blending matrices are used for object shape representation and for any further collision detection requirements. In this manner, the proposed surface merging algorithm works in tandem with the collision detection algorithm and only a small number of additional computations are performed during the merging process.

The algorithm is capable of handling all the cases that are acceptable for NURBS-based surfaces. It does not impose any restriction on the degree of the surface patches, the number of control points, the type of continuity required at the common edge, knot vector or the number of surfaces being merged simultaneously. The proposed algorithm can efficiently merge B-spline surface patches having dissimilar curvatures at the common edge, intersecting B-spline surface patches, and the B-spline surface patches having trimmed edges. These types of patches cannot be merged by using the traditional approach used by commercially available CAD software. Overall the proposed algorithm is efficient, accurate, and robust. The surface generated by merging of two or more patches has better tolerance than that is acceptable for many VR applications.

CHAPTER 5 INTEGRATION OF MASS-SPRING SYSTEM WITH COLLISION DETECTION SYSTEM

5.1 Introduction

Product designers and engineers require interactive and graphical visualization tools that enable them to quickly modify the shape, style, and functionality of a product concept. The primary role of VR technology in creative product design is to provide the designer with the ability to intuitively create and manipulate the shape of complex freeform CAD models during concept generation.

Deformable objects have been widely studied in computer graphics. The deformation of the model can be simulated by a geometric- or physics-based system. There are many geometric modeling techniques in the published literature to deform a solid model [Basdogan, 2004; Zheng, 2003]. One major limitation of the geometric models is that these cannot realistically simulate deformation of the model, particularly if the model consists of multiple materials. A physics-based technique, on the other hand, can realistically calculate the deformation and force response of the model, based on virtual material properties [Knopf, 2005]. A physics-based deformation model can provide the designer options to assign different materials properties to the virtual object and validate it in real-time for form and function.

The collision detection algorithm and the physics-based deformation model should be compatible with each other for real time haptic interaction. Some physically-based models are computationally intensive and therefore, are unsuitable for real-time interaction. Over the years different modeling techniques have been developed. However, a mass spring damper system, consisting of a set of particles (nodes) connected through a network of springs and dampers can provide reasonable accuracy and speed for real time interaction. The representation of B-spline surfaces in terms of blending matrices facilitates the integration of collision detection and merging of B-spline surfaces with the mass spring system [Pungotra, 2010b].

5.2 Mass Spring Damper System

Mass spring damper system is one of the physics based modeling technique that has been widely used for modeling deformable objects. A mass spring system can provide reasonable accuracy and speed for real-time haptic interaction. The virtual object is modeled as a collection of point masses connected by springs and dampers in a lattice structure. In general, the spring forces are assumed to be linear. However, nonlinear springs can also be used to model objects which exhibit inelastic behaviour. Such a system contains a mass ρ, a spring with spring constant K that serves to restore the mass to a neutral position, and a damping element which opposes the motion of the vibratory response with a force proportional to the velocity of the system. The constant of proportionality, also known as damping constant, is denoted by D. Different combinations of linear springs and damper can be used to model deformable objects. Voigt model is the most commonly used combination of spring and damper and has been used in this thesis. Figure 5.1 shows a mass spring damper system with two mass nodes i and j.

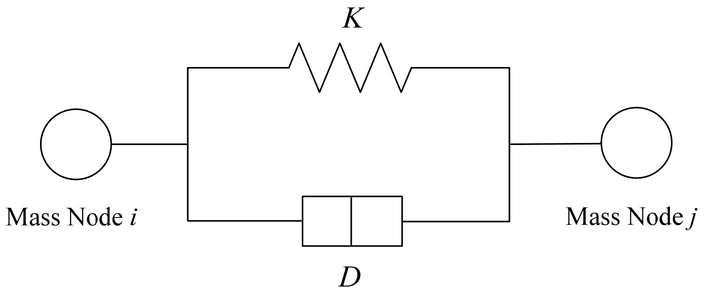


Figure 5.1 Voigt model of mass spring damper system.

The stiffness of the material primarily affects the linear and non-linear elasticity range in the deformation zone. For solids, the stiffness in proportional to the elastic modulus and also depends upon an element's dimensions. The linear model stiffness can be express as,

$$K = \frac{A \times E}{L} \tag{5.1}$$

where *A* is the cross-sectional area, *E* is the Young's modulus of elasticity, and *L* is the length of the element.

In one dimension, Young's modulus of elasticity can be considered as a measure of stiffness of material. Thus, stiffness and damping constants can be used to model a realistic behavior. The stiffness constant controls the elastic behavior and the combination of stiffness and damping constants control plastic behavior of the material.

Damping is the phenomenon by which energy is dissipated in a vibratory system. Three significant types of damping that are generally encountered in dynamic behavior of the model are; coulomb, hysteresis, and viscous damping. In this thesis, these three types of damping are approximated as a velocity dependant viscous damping. The velocity dependent damping force exerted on the node *i* from the interaction with the node *j* is given by,

$$f_d = -D(\dot{x}_i - \dot{x}_j) \tag{5.2}$$

where f_d is the damping force, *D* is the damping coefficient, $\dot{x}_i$ is the velocity of node *i*, and $\dot{x}_j$ is the velocity of node *j*.

The fundamental Lagrange equation of motion can be expressed as,

$$\rho\ddot{x} = -D\dot{x} - Kx + f_x + f_g \tag{5.3}$$

where ρ (in kg) is the mass of the node, *D* (in N.s/m) is damping coefficient, K (in N/m) is the stiffness coefficient of each spring, f_x and f_g are the external and gravitational forces acing of the node. In the absence of external and gravitational forces, Eqn. (5.3) can be written as,

$$\rho\ddot{x} + D\dot{x} + Kx = 0 \tag{5.4}$$

Rearranging Eqn. (5.4), we get

$$\ddot{x} + \frac{D}{\rho}\dot{x} + \frac{K}{\rho}x = 0 \tag{5.5}$$

The natural frequency of this model is given by,

$$\omega_0 = \sqrt{\frac{K}{m}} \tag{5.6}$$

The time step of the system *dt* must not be equal to this natural frequency. Another important parameter is called damping factor. It is given by equation,

$$\zeta = \frac{D}{2\sqrt{Km}} \tag{5.7}$$

The values of damping factor can be zero (undamped model), less than one (under-damped model), one (critically damped model), and greater than one (over-damped model).

5.3 Integration of Mass Spring System with Collision Detection

For an efficient haptic interaction with a physics-based virtual object, the collision detection and physics based system must work in tandem. The algorithm for collision detection, presented in this thesis is especially suitable for a mass spring system. The blending matrices (introduced in Chapter 3) are used to integrate collision detection algorithm with a mass spring damper model incorporating physical material properties assigned to the model. This way the interactive design module can be used for virtual sculpting and validating models before extensive product detailing is performed using commercially available CAD software [Pungotra, 2010b].

The mass spring damper mesh is generated during the pre-processing phase of the interactive design module. Once created, the mass spring system uses the information provided by the collision detection system, and the force applied by the user to determine the deformation of the model. The sculpting forces applied by the user are mapped on the mass spring nodes determined by the collision detection algorithm. The force is distributed on the nodes of the mass spring system, and subsequently used to calculate the deformation and force response. Once the mass spring system changes the model shape according to the materials properties assigned to it, new set of control points are determined to represent deformed B-spline surface. While interacting with a plane (working bench or table), the mass nodes resting on the plane are fixed. In such a case, even though these points experience external or internal forces, the nodes remain stationary. The flow chart shown in Figure 5.2 describes various steps required to generate and interact with the mass spring mesh. These steps are discussed in detail in the following sub-sections.

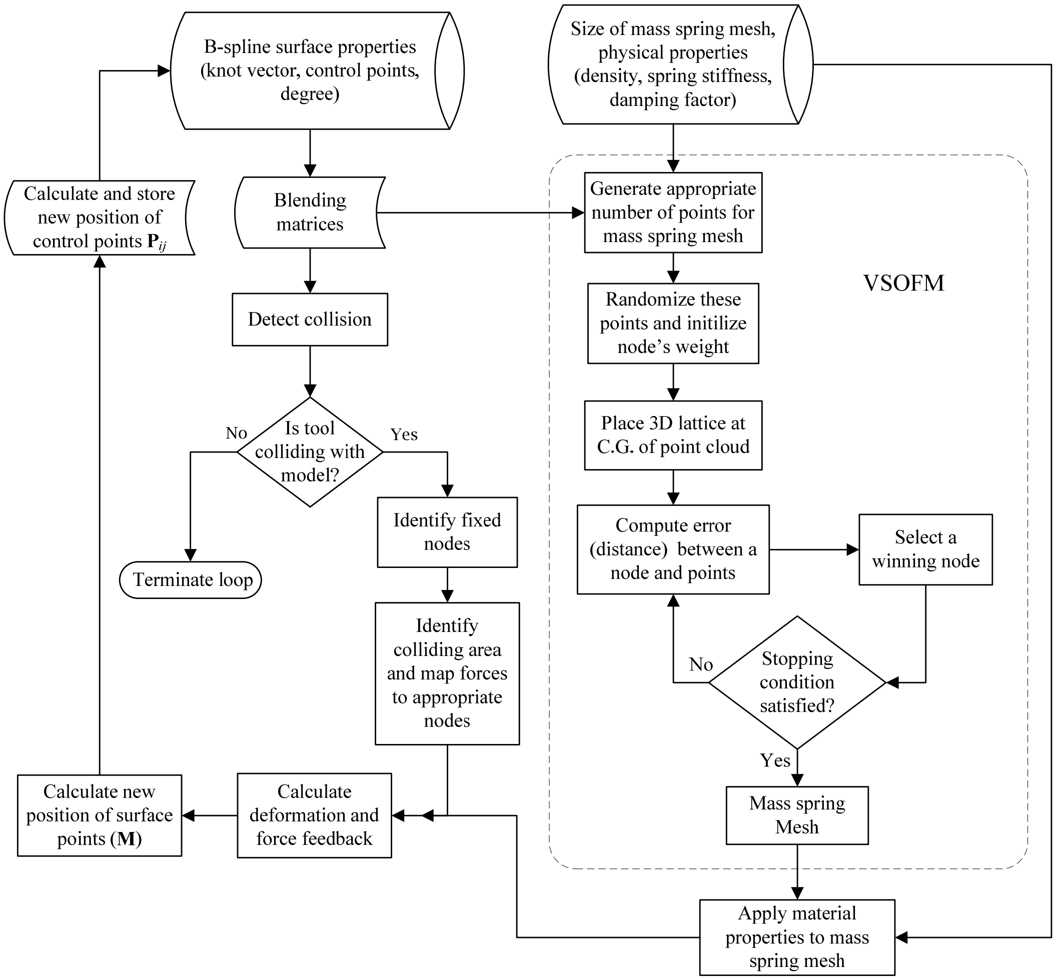
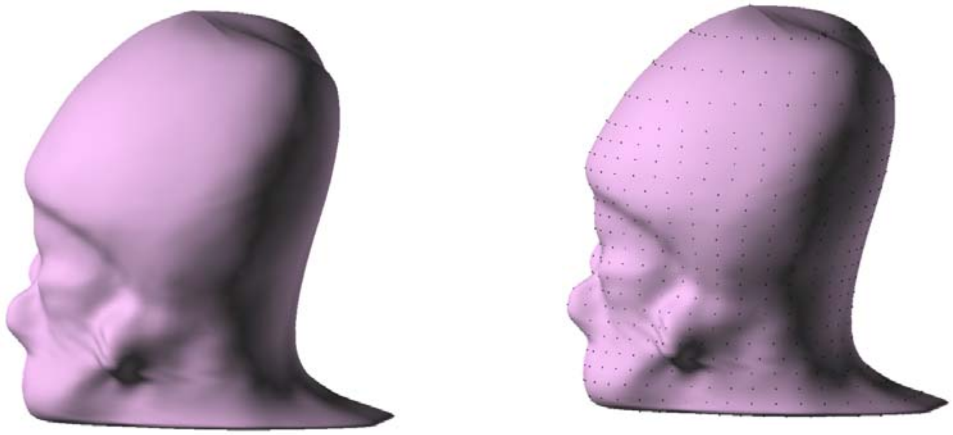

B-spline surface properties (knot vector, control points, degree)
Size of mass spring mesh, physical properties (density, spring stiffness, damping factor)
Calculate and store new position of control points $\mathbf{P}_{ij}$
Blending matrices
Generate appropriate number of points for mass spring mesh
VSOFM
Randomize these points and initilize node's weight
Detect collision
Place 3D lattice at C.G. of point cloud
No
Is tool colliding with model?
Yes
Compute error (distance) between a node and points
Select a winning node
Identify fixed nodes
Terminate loop
Identify colliding area and map forces to appropriate nodes
No
Stopping condition satisfied?
Yes
Calculate new position of surface points (M)
Calculate deformation and force feedback
Mass spring Mesh
Apply material properties to mass spring mesh

Figure 5.2 Flow chart for mass spring mesh generation and its integration with collision detection algorithm.

5.3.1 Mass spring system node generation

As discussed in Chapter 3, blending matrices and their inverses for the B-spline surface patches are calculated and stored during pre-processing phase. If the individual B-spline surface patches are merged, then the revised blending matrices are calculated for the merged B-spline surface. This process has been explained in Chapter 4.

These blending matrices can be used to discretize the corresponding B-spline surface. The density of the points generated on the B-spline surfaces can be varied by choosing intermittent rows and columns of the blending matrices. Depending upon the number of nodes required for the mass spring mesh, appropriate number of points are generated by using intermittent rows and columns of the blending matrices. Figure 5.3(a) shows the B-spline surface model and Figure 5.3(b) shows the points generated on the virtual model for calculating mass spring mesh.



(a) (b)

Figure 5.3 (a) B-spline surface deformable model (b) Discrete points as nodes for the mass spring system.

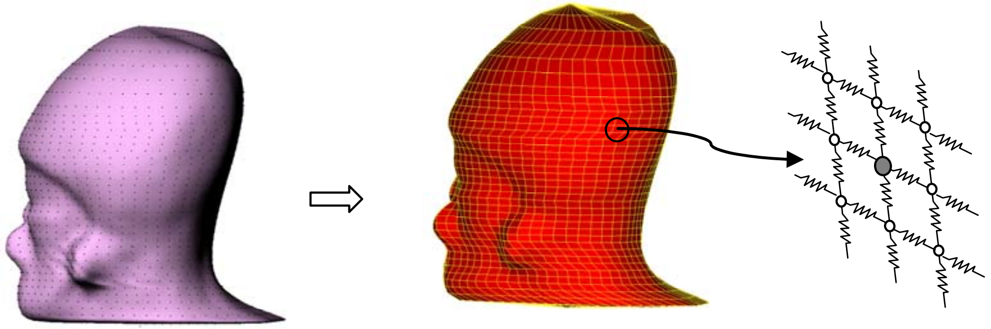
5.3.2 Deformable mesh generation

Tetrahedron and hexahedron solid elements are the most commonly used structures for creating the mass spring meshes. Although the algorithms for producing a tetrahedral mesh are easier to implement, the hexahedral meshes are computationally more efficient because fewer elements are required to represent the shape. Furthermore, a valid mesh can only be generated if the angles in the tetrahedral elements are neither too obtuse nor too acute. Consequently, a model of a simple flat plate may inadvertently result in a large number of tetrahedral elements. To correct for many of the geometric errors that occur using tetrahedrons the designer must manually insert hexahedral elements and sub-meshes in certain regions of the volume model.

A variety of mesh generation techniques have been described in the literature for creating hexahedral meshes. Multi-block topology technique [Hsu, 1992] allows the designer to first partition the object geometry into multiple hexahedral blocks or *super-elements* within which an array of solid elements is produced. Each block is then further sub-divided into an array of elements whose size depends on the desired grid density. The major advantage of this multi-block mesh generation approach is that it produces a highly structured grid. Once the block topology is determined, a variety of CAE software tools must be used to optimize the elements in the desired mesh.

The *volumetric self-organizing feature map* (VSOFM) [Igwe, 2008b] is a viable skeletal framework for modeling *realistic* objects that dynamically change shape with time. The self-organizing feature map is a lattice of nodes arranged with predefined topological connections. The deformable VSOFM provides an adaptable mesh for representing a virtual lump of clay that dynamically changes shape under external forces. The lattice structure ensures a closed geometry with no gaps or breaks in the surface. This technique is used to generate hexahedral mass spring mesh for the physically-based model. The hexahedral mesh of the underlying object is constructed using a deformable volumetric self organizing feature maps with a three-dimensional lattice of uniformly distributed nodes or weight points. The sparse point cloud generated on the B-spline surfaces encloses the 3D lattice of the deformable VSOFM, having the required number of nodes attached through springs and dampers. The lattice is allowed to expand to the point cloud. This 3D ordered lattice of the deformable VSOFM maintains the relative connection of neighboring nodes in the mesh as it geometrically transforms into the B-spline surface model shape. The surface nodes are connected to the neighboring surface nodes as well as the interior nodes that lie directly below. Connectivity and topology of the developed mesh is registered to prevent unstable dynamic behavior during modeling. In this manner, the points generated on the B-spline surface are assigned as exterior nodes of the mass spring damper mesh representation. The number of nodes in the 3D lattice limits the density of the hexahedral element in the final physics based deformable model.

Figure 5.4 shows the generation of a mass spring system. Figure 5.4(b) shows the deformable VSOFM mesh which adapted to the sparse point cloud generated on the B-spline surface model using blending matrices. The connectivity of the VSOFM nodes is shown in Figure 5.4(c).



(a) Sparse point cloud generated on the B-spline model are used as exterior nodes for the mass spring

(b) Mass spring damper mesh created using VSOFM.

(c) Hexahedral mesh of the mass spring damper

Figure 5.4 Generation of mass spring system.

Igwe *et al.* [Igwe, 2006; Igwe, 2008b] used VSOFM technique with a large point cloud data. However, the number of points generated on the B-spline surface can be correlated to the size of the mass spring mesh being created. Hence, it is possible to have one to one mapping of point cloud with the 3D lattice. The small point cloud data ensures that the computational cost of generating the mass spring damper mesh through VSOFM is small.

5.3.3 Mapping of forces to nodes of mass spring system

The collision detection algorithm, described in Chapter 3, determines the point(s) or surfaces of intersection of two or more virtual models. Figure 5.5 shows a tool colliding with a deformable B-spline surface. Once the collision detection algorithm determines the point(s) or surfaces of intersection, the forces applied by the user are mapped on the mass spring system.

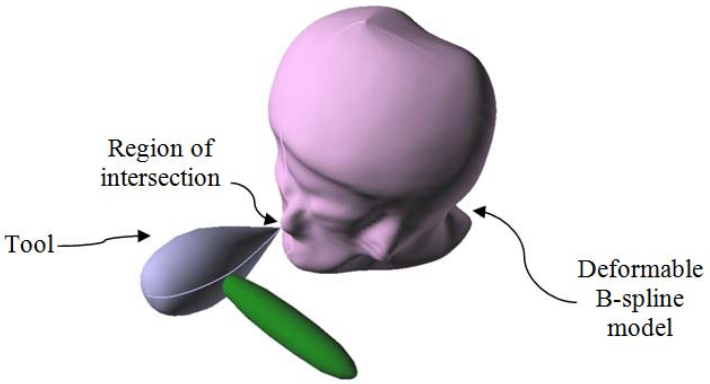


Figure 5.5 A tool colliding with a deformable B-spline surface model.

To clearly show the procedure of mapping haptic force to the nearby nodes, Figure 5.6 shows a case in which intersection is happening at a point.

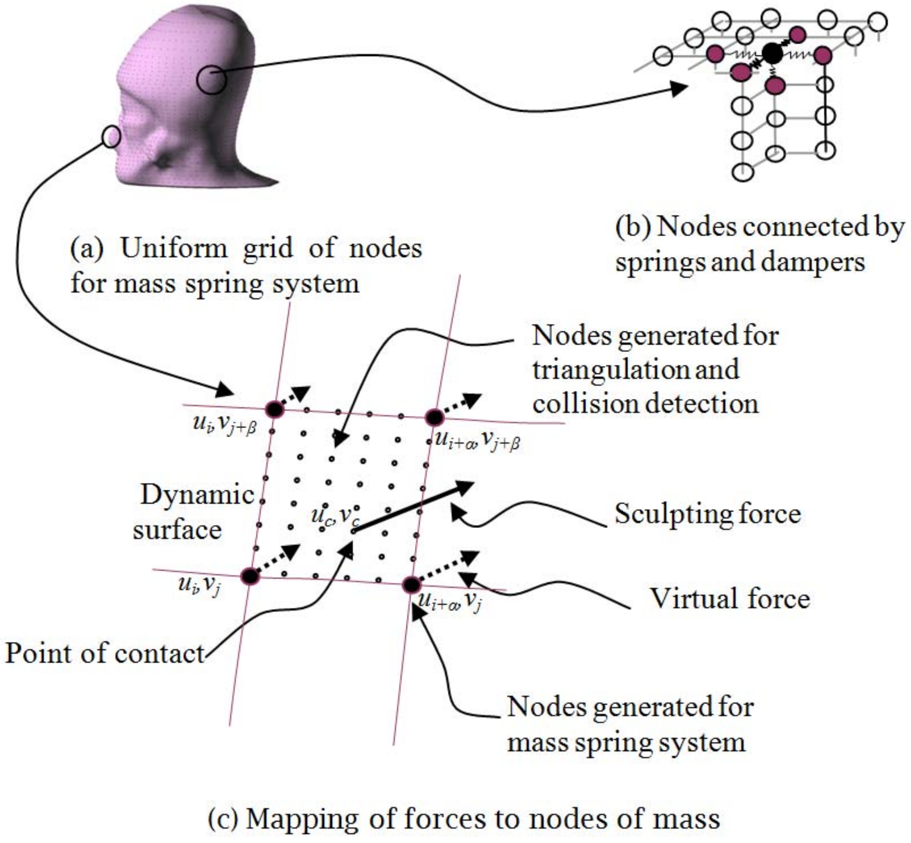

(a) Uniform grid of nodes for mass spring system
(b) Nodes connected by springs and dampers
Nodes generated for triangulation and collision detection
$u_i, v_{j+\beta}$
$u_{i+\alpha}, v_{j+\beta}$
Dynamic surface
u_c, v_c
Sculpting force
Virtual force
u_i, v_j
$u_{i+\alpha}, v_j$
Point of contact
Nodes generated for mass spring system
(c) Mapping of forces to nodes of mass

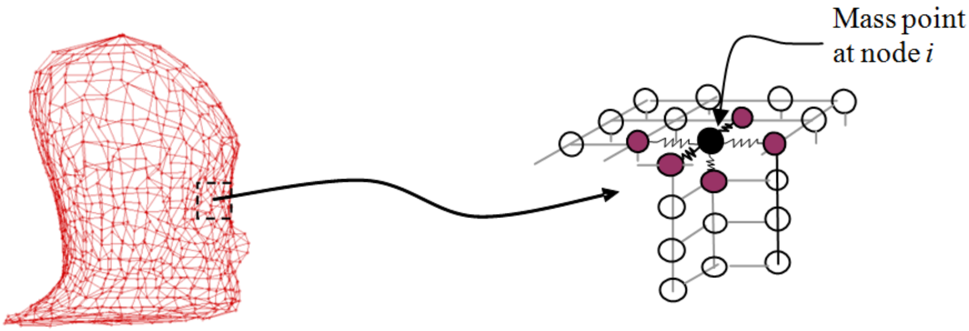
Figure 5.6 Mapping of the sculpting force acting on B-spline surface to nodes of mass spring system.

When two surfaces are in contact, there may be many points on which the sculpting forces are acting. In this case, the sculpting force acting at each point of contact is mapped to the nearest nodes of the mass spring system. The total virtual forces will then be the vector sums of the virtual forces mapped mass spring nodes from each point of contact.

5.4 Model Deformation and Force Response

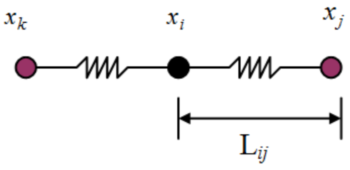
When the user interacts with the deformable model with a tool, the system must be able to compute the estimated position of the dynamic model at the next time step from the current forces being applied through the haptic tool. Collision detection algorithm determines the region where tool is interacting with the model. This information is used to map the haptic forces to the model as discussed in previous sections.

Figure 5.7 shows a deformable hexahedral mesh and the node-spring-node arrangement of mass and spring within the model. In this context, the spring located between nodes *i* and *j* has a natural or rest length of L_{ij}.



(a) Mass spring mesh representing deformable model

(b) Hexahedral mesh of mass nodes and springs



(c) Rest length of springs

Figure 5.7 Hexahedral mesh and the node (mass) spring representation.

This length will change under the application of the force and at any instant it will be given by,

$$l_{ij} = \left\| x_i - x_j \right\|, \tag{5.8}$$

and the location of a node in the three coordinate directions is denoted by the three-dimensional vector,

$$x_i = [x_1^i, x_2^i, x_3^i]. \tag{5.9}$$

During shape deformation, the initial length can be used to calculate the strain of the spring once the sculpting forces are applied. The internal and external forces cause mechanical strain, e_{ij}, in the connecting spring, given by

$$e_{ij} = \frac{(l_{ij} - L_{ij})}{L_{ij}} \tag{5.10}$$

This spring will then exert a force s_{ij} on the node *i*. If K_{ij} is the material stiffness assigned to the model, then this force will be given by

$$s_{ij} = K_{ij}\, e_{ij}\, (x_i - x_j) \tag{5.11}$$

Numerical simulation of the deformation process, while the object experiences external forces from the haptic device, is achieved using the discrete Lagrange equations of motion for a dynamic node-spring system. The system dynamics is given by the second-order differential equation,

$$\rho_i \ddot{x}_i(t) + D_i \dot{x}_i(t) + g_i(t) = f_i(t) \tag{5.12}$$

where D_i is the velocity-dependent damping coefficient which dissipates kinetic energy in the lattice through friction; ρ_i is the point mass of node *i*, and $f_i(t)$ is the external force vector applied to node *i*. If ne_i is the immediate neighbourhood around node *i*, then the total internal spring forces ($g_i(t)$) given by,

$$g_i(t) = \sum_{j \in ne_i} s_{ij} \tag{5.13}$$

The rest length of the spring L_{ij} is determined at the beginning of the deformation process. The spring is then allowed to vary in length (l_{ij}) due to the plastic deformations or other non-linear behavior that is to be modeled by the system of equations. To simulate the dynamic behavior of the adaptive mesh, it is necessary to provide the initial positions $x_i(0)$ and the initial velocities $\dot{x}_i(0)$ for each node. The initial velocities are often assumed to be zero. It is then necessary to integrate the equation of motion forward through time until the mesh stabilizes; $\dot{x}_i \approx \ddot{x}_i \approx 0$. At each time step Δt, it is necessary to evaluate the current nodal forces and accelerations, the new velocities, and the new node positions using the explicit Euler time-integration procedure [Knopf, 2005]. While the object experiences external forces from the haptic device, numerical simulation of the deformation process can be achieved using the discrete Lagrange equations of motion for a dynamic node-spring system. From Eqn. (5.12), it is possible to compute acceleration at node *i* as,

$$\ddot{x}_i(t)=(f_i(t)-D_i\dot{x}_i(t)-g_i(t))/\rho_i, \tag{5.14}$$

and the new velocity can be computed as,

$$\dot{x}(t+\Delta t)=\dot{x}(t)+\Delta t\,\ddot{x}_i(t) \tag{5.15}$$

The new position of node *i*, is then calculated using the equation:

$$x_i(t+\Delta t)=x_i(t)+\Delta t\,\dot{x}_i(t+\Delta t) \tag{5.16}$$

Once the user applies force on the model through the tool, Eqns. (5.12 - 5.16) determine the deformation of the model.

Geometric constraints are also applied to move several points within the defined neighborhood radius to the new location depending upon the magnitude of the applied force. Applying both physical and geometrical types of constraints, offer additional intuitive control over the shape during the design process. Constraining geometric and physical properties of a deformable model also facilitates feature-centered design, which can significantly improve the system simulation and performance. A neighborhood search radius is used to update the length of the neighboring springs [Knopf, 2005]. This ensures that elongation is distributed over a given area and not just regions directly involved with the deformation force.

Figure 5.8 illustrates the interaction of the tool with the hexahedron mass spring mesh of the model. A force is applied to determine its deformation in response to the applied force.

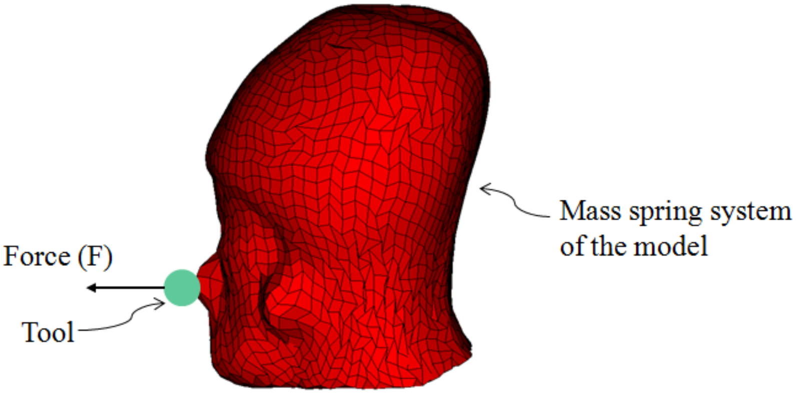


Figure 5.8 Illustration of interaction of the tool with the model.

The deformation of the model can be evaluated by applying varying amount of forces while keeping its material properties constant.

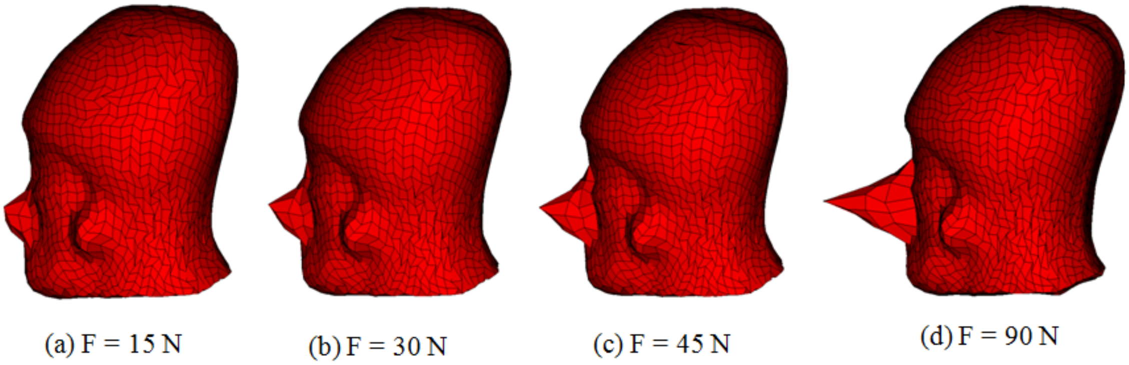
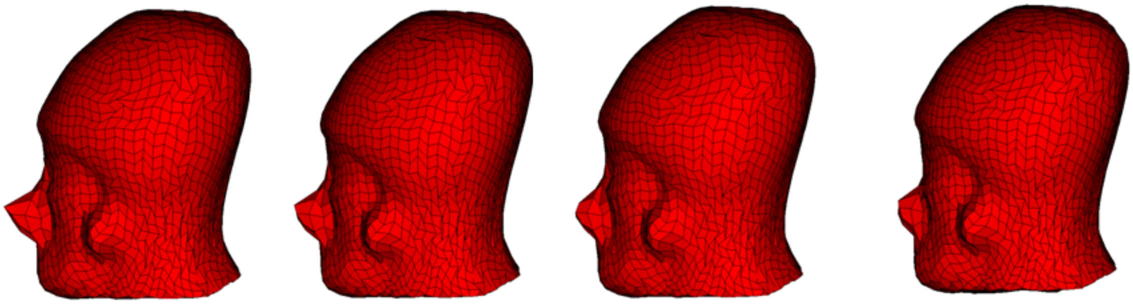


Figure 5.9 The deformation of the model having properties of plasticine (spring stiffness = 13 MN/m^2, density = 2500 kg/m^3) while applying different forces (F).

Figure 5.9 shows the deformation of the model when properties of plasticine were incorporated to the model. Varying amount of force was applied to the model. The model deforms more when a large force is applied to the model.

In a similar fashion, the force can be kept constant while the material properties are changed. This is helpful when an industrial designer plans to use different materials for the model. If the deformation of the model is within the limits, the material is selected. However, if the deformation is more than the prescribed limits, the user can modify the design or use alternate material. Figure 5.10 shows the deformation of the model having different spring stiffness, when a constant force of 30 N was applied. The model having least spring stiffness deforms the most. This process can be used for sculpting as well as to validate a range of forces that can be resisted by the model without deforming beyond the prescribed limits based on its performance criteria [Pungotra, 2009a].



(a) stiffness = 10 MN/m

(b) stiffness = 50 MN/m

(c) stiffness = 100 MN/m

(d) stiffness = 1000 MN/m

Figure 5.10 The deformation of the model with different spring stiffness, and constant force (30 N) while pulling out the nose of the artifact.

5.5 Concluding Remarks

A deformable mass spring system was introduced in this chapter as a geometric modeling tool for manipulating closed 3D shapes, using physics-based modeling. Material and dynamic properties are incorporated into the deformable mesh by treating the surface and the internal nodes as point masses connected by a network of springs. The initial mass spring mesh was created automatically using the blending matrices. This was accomplished at the pre-processing stage and hence the computational cost of mesh generation would not impact the real-time interaction with the model.

The mass spring mesh works in tandem with the collision detection algorithm. The same blending matrices, which are used for collision detection, generate exterior nodes for the model. VSOFM is used to uniformly positioning the interior nodes and linking all the nodes together with a spring and damper assembly. Once the mesh is generated, the designer will be able to reshape the virtual object by introducing external forces to the nodal mesh.

CHAPTER 6 SIMULATION STUDIES AND PERFORMANCE EVALUATION

6.1 Introduction

This chapter discusses the implementation of the algorithms presented in this thesis. The simulation study and performance evaluation of the algorithm was carried out to find out their computational efficiency. No constraint regarding the degree of the surfaces or complexity of the model was applied. However, for the majority of the cases, B-spline surface patches of degree three were used as these are the most common surfaces used in CAD applications. Overall the framework for interactive design consists of three distinct algorithms: the collision detection algorithm (Chapter 3), merging on B-spline surface patches (Chapter 4), and mass spring system for physically based deformation of the virtual model (Chapter 5).

This chapter presents the computational efficiency of these three algorithms separately. Once the efficiency of these algorithms is proved independently, it was used to present a test case which incorporated all three algorithms so as to evaluate the performance of the integrated module. The presented results proved the evidence about the validity and computational efficiency of the proposed technique.

6.2 Computational Efficiency of Collision Detection Algorithm

The focus of the research was to develop a computationally efficient collision detection algorithm for real-time interaction with deformable model, represented by a B-spline surface, during the haptic interaction with the model. As discussed in Chapter 3, following measures ensured a computationally efficient algorithm.

The algorithm uses pre-calculated blending matrices to discretize the B-spline surface. This eliminates the need to calculate blending functions during the runtime. These blending matrices are sparse having a maximum of "*degree of surface* + 1" number of non-zero entries. Thus, the cost of multiplication of these matrices with the control point matrix is small.

As the blending matrices are pre-calculated, their inverse can also be pre-calculated. Hence, no inverse of large matrices is required at runtime, further reducing the cost of collision detection.

Intermittent rows and columns of blending matrices can be used to generate sparse points on the B-spline surfaces. It allows the algorithm to generate fewer points initially to detect collision and increase the density of points at the lower levels of detail. This reduces the overall cost of collision detection.

Convex hull of B-spline surfaces is used as the bounding box. As the number of control points is always small and their position known, the cost of updating the bounding volume (convex hull) is minimal.

Most of the collision detection is carried out by checking intersection test of spheres generated from discrete points generated on B-spline surface. As the sphere-sphere intersection test is very efficient, the computational cost of collision detection is minimal.

The algorithm uses triangle-triangle intersect test at the lowest level of detail. As the number of triangles is small at the lowest level of detail, the computational cost is small. The proposed collision detection algorithm can detect the collision between two or more NURBS surfaces and/or between a NURBS surface and an implicit surface, a tessellated surface or a point. Figure 6.1 shows two NURBS surfaces, a plane and a sphere.

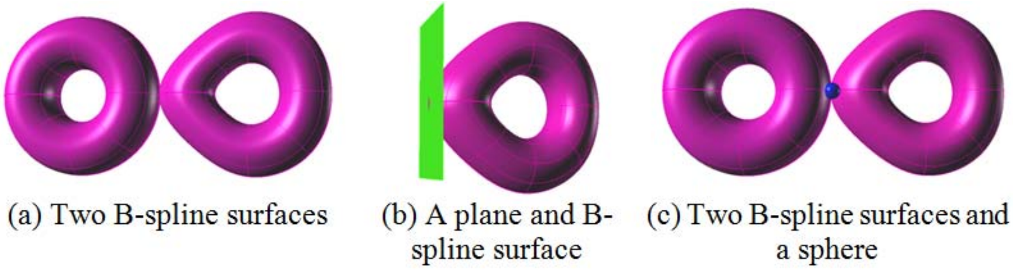

(a) Two B-spline surfaces (b) A plane and B-spline surface (c) Two B-spline surfaces and a sphere

Figure 6.1 Primitives (plane and sphere) and B-spline surfaces (a donut and a distorted donut) for calculating time of collision detection.

To check the computational efficiency of the collision detection algorithm, different types of surfaces were used. A B-spline surface was made to collide with another B-spline surface, a sphere, a plane and a point.

The computational cost of collision detection depends upon the size of the blending matrices and the area of contact. The size of the blending matrices depends upon the number of control points and the maximum number of points that can be generated on the B-spline surface. The collision detection was carried out for different number of control points, different number of the maximum points to be generated on the B-spline surface, and different overlap (thereby changing the area of contact). The collision detection was determined for B-spline surface - plane, B-spline surface - sphere, B-spline surface - point and B-spline surface - B-spline surface. The results for the computational cost for different parameters are discussed in the following sections.

6.2.1 Effect of the number of control points

A cubic B-spline surface needs a minimum of 4×4 number of control points. In fact such a surface will be a Bezier surface. For such a surface, any deformation at any point may result in change for whole geometry. However, to better represent a complex model and to allow localized deformation, more number of control points will be needed. Control points also determine the minimum number of nodes for the mass spring system that can be used to incorporate material properties. As soon as a model deforms, the new position of control points is calculated by using the new position of mass spring nodes. Hence, the number of nodes of mass spring mesh cannot be less than the number of control points used to define the B-spline surface. As an example, a minimum 16 nodes for mass spring system will have to be used for a B-spline model with 4×4 control point net. Thus, it is imperative that the effect of the number of control points on the computational efficiency be calculated.

It must be emphasized at this point that the computational time recorded in the simulation may not be the best time that the collision detection algorithm is capable of. The simulation time may be larger because of inherent inefficiencies in programming skills. A professional programming technique may be able to better utilize the resources of modern multi-core computers. In fact the algorithm is very well suited to take advantage of multithreading. Even with the programming done without utilizing multithreading, the time needed to check the collision detection was small.

The time required for collision detection also depends upon the number of points generated on the B-spline surface. In this section the number of points matrix used to determine collision is kept at 82×82 matrix. This matrix gives very good resolution. For simulation study the numbers of control points were selected in the range of 4×4 to 20×20. Most of the examples used in literature do not use more than a matrix of 12×12 for control points. Dachille [Dachille, 2001] used a maximum of 12×12 control net with only 25×25 mesh of points for collision detection compared to an 82×82 matrix of nodes used in this study. The time taken for simulation for a B-spline surface represented by a 12×12 control point net with 25×25 mesh was 780 ms with implicit solver [Dachille, 2001]. It used only a point to interact with the B-spline model. Gao and Gibson [Gao, 2006] used a resolution of 40×40 and used implicit surface rigid tools.

Table 6.1 shows the computation times for collision detection of B-spline surface having different number of control points.

Table 6.1 Computational time for collision detection of B-spline surface having different number of control points with a point, a sphere, and a plane.

Control points net	**Computational time (ms)**		
	Point based tool	**Sphere based tool**	**Plane based tool**
4×4	<1	<1	2
6×6	<1	2	3
8×8	<1	3	4
10×10	2	4	8
12×12	4	5	10
14×14	5	8	15
16×16	6	12	22
18×18	8	13	26
20×20	10	17	31

A point-based tool, a sphere-based tool, and a plane-based tool were allowed to collide with the B-spline surface. Different B-spline surfaces were considered and the time for collision detection was noted. The resolution of the model was same for all the cases and was kept at 82×82. The entries of Table 6.1 represent the average of the collision detection time. A plane can be used for two purposes. It can be used as supporting surface such as a table on which B-spline surface based model can rest. It can also be used as tool to push, pull, or cut the model. The code was written in C++ and implemented with Microsoft Visual Studio 2008 on desktop computer having 6 GB RAM with Intel(R) Core(TM) i7 CPU @ 3.06 GHz running on Windows 7 Professional.

As expected the computational cost increases with increase in the number of control points used to represent the B-spline surface. The graph shown in Figure 6.2, represents the trend of computational cost, as the numbers of control points are increased.

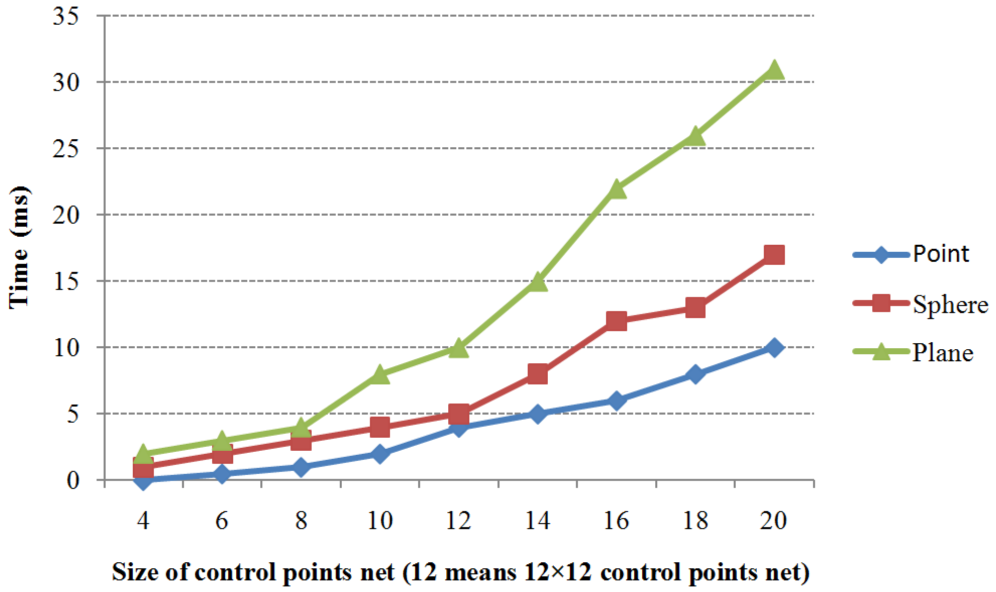


Figure 6.2 **Variation of the computational time of collision detection for a point, sphere, and plane with respect to the number of control points used.**

The computational time for collision detection of a point with the B-spline surface is the least. A plane takes more time to check collision detection with the B-spline surface. This is due to the reason that it is closer to the B-spline surface over a larger area, thereby increasing the number of spheres being generated at different levels of detail. As the number of control points increase, the computational cost of generating more points and subsequent generation of spheres increases. Since more spheres are generated for a plane, the cost of collision detection will be more. Overall the computational time for collision detection is small. Particularly for B-spline surfaces having 12×12 control points, it takes 5 ms or less for the collision detection algorithm to check intersection with a point or sphere. It takes about 10 ms to detect collision with a plane.

A unique feature of the collision detection algorithm is its ability to detect collision between a deformable B-splines and implicit surfaces as well as between two or more deformable B-spline surfaces. This would allow the user to have better control about the type of tools used for interacting with a deformable model. The ability to detect collision between two or more B-splines is also utilized by the B-spline surface patch merging algorithm discussed in Section 6.3.

Figure 6.3 shows the computational cost of collision detection between two B-spline surfaces. Again, the results are the average of the time taken by the algorithm to detect collision for different types of B-spline surfaces when these surfaces just start colliding. The control points of the two B-spline surfaces were increased independently and the computational cost was determined.

The computational cost increases with increase in the number of control points used to represent the B-spline surface. The resolution of the model was same for all the cases and was kept at 82×82. Time required to detect collision between two B-spline surfaces represented by 8×8 control points net was 15 ms. For higher number of control points, the computational time increases, reaching to about 91 ms for B-spline surfaces when both the surfaces have 20×20 control points mesh. For smaller control points mesh, the computational time is very small and reduces to 6 ms when both the surfaces have 4×4 control points net. Thus, the collision detection algorithm can be used for real time interaction with virtual models.

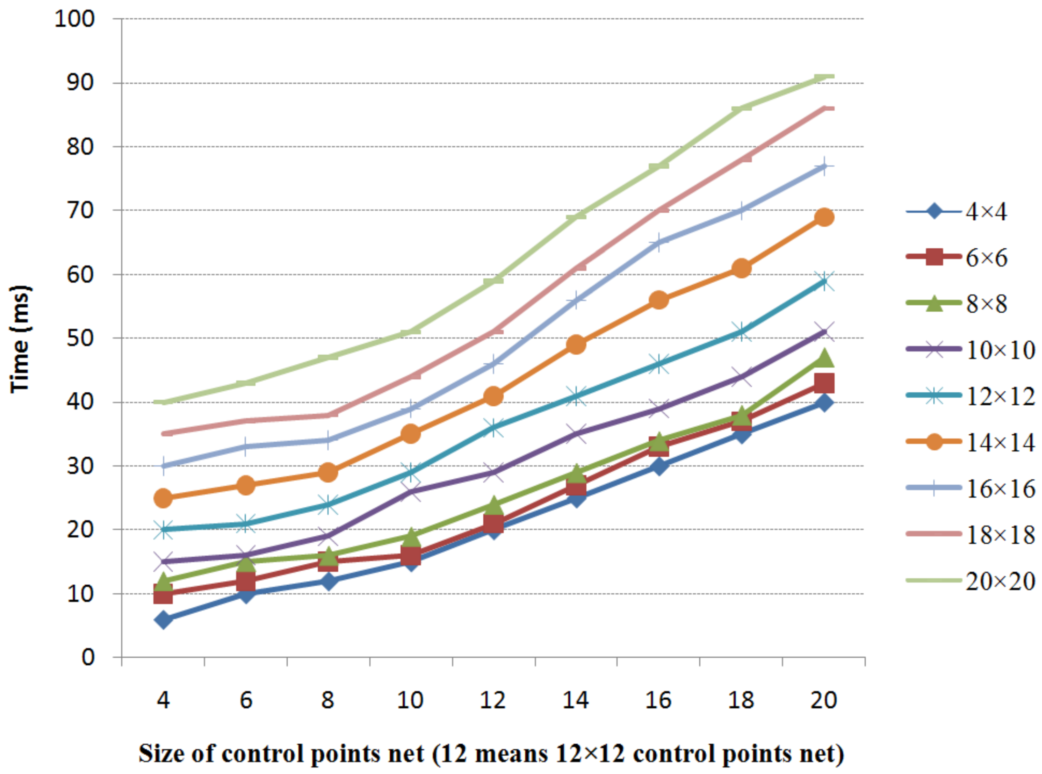
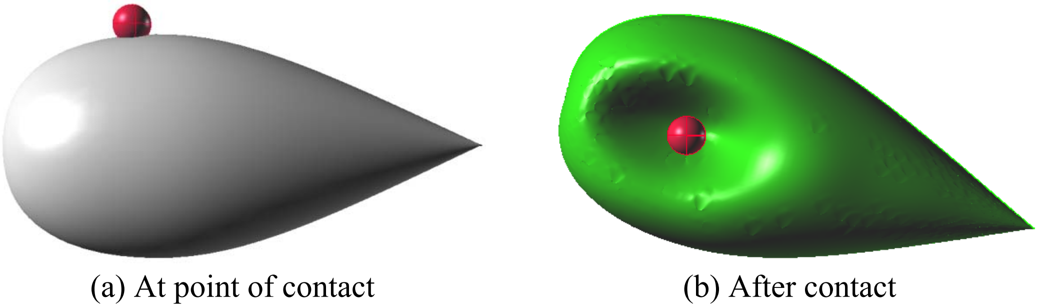


Figure 6.3 **Variation of the computational time (ms) of collision detection for a B-spline surface with another deformable B-spline surface for different number of control points.**

6.2.2 Effect of the maximum number of points that can be generated

The number of points generated on the B-spline surface defines the resolution and accuracy of the collision detection. This also determines the maximum number of nodes that can be generated for the mass spring mesh. A larger number increases the size of the blending matrices, $\mathbf{A}_u$ and $\mathbf{A}_v$, used to generate these points using Eqn. (3.6). As a consequence of the large size of the blending matrices, the size of inverse of these blending matrices will also be large.

The collision detection algorithm does not generate all the points simultaneously during collision detection process. Initially it generates sparse points in the region of probable collision. It generates denser points within the intersecting spheres, at lower levels of detail, to increase the accuracy of the collision detection as discussed in Section 3.3. To clearly determine the effect of the maximum number of points that can be generated at the lowest level of detail, a tear shaped deformable model with 8×8 number of control point net was checked for intersection with a point, a sphere, and a plane. Figure 6.4(a) shows a sphere colliding with a tear drop shape and Figure 6.4(b) shows a sphere colliding with the deformed tear drop represented as B-spline surface having 8×8 control point mesh.


(a) At point of contact

(b) After contact

Figure 6.4 **A sphere colliding with (a) A tear shaped B-spline surface (b) A deformed tear shaped B-spline surface.**

The time taken for collision detection was computed for different shapes of B-spline surfaces. Table 6.2 shows the cost of computation for different number of the maximum points that can be generated on the B-spline surface.

Table 6.2 **Computational time for collision detection of B-spline surface having different number of the maximum points generated at lowest level of detail with a point, sphere, and plane.**

Maximum points at lowest level of detail	Computational time (ms)		
	Point based tool	Sphere based tool	Plane based tool
28×28 (784 points)	<1	~1	1
28×82 (2296 points)	<1	2	2
82×82 (6724 points)	~1	3	4
82×244 (20008 points)	2	4	7
244×244 (59536 points)	3	6	15

When a large number of points are used, the accuracy of the collision detection is higher. The time shown in the table is average of the times noted for different B-spline surfaces.

Figure 6.5 shows the graph representing the trend of the computation cost with respect to the maximum number of points that can be generated on the B-spline surface.

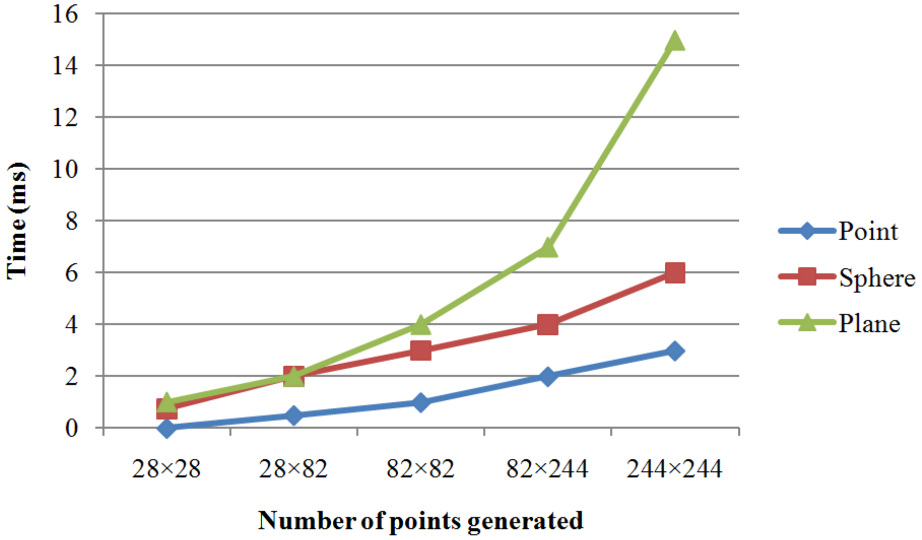


Figure 6.5 **Variation of the computational cost of collision detection for a point, sphere, and plane with respect to the maximum number of points generated.**

The number of control points of the model was the same for all the cases and were kept at 8×8. The computational cost for collision detection increases when the maximum number of points to be generated on the B-spline surface, at the lowest level of detail, is increased. When 82×82 mesh is used, the computational time is 1 ms for a point, 3 ms for a sphere and 4 ms for a plane. A mesh of size 82×82 means that a total of 6724 points can be generated on the B-spline surface. This gives high resolution for collision detection. For comparison, only 625 points (25×25 mesh of points) were used by Dachille *et al.* [Dachille, 2001] and Gao and Gibbson [Gao, 2006] used 1600 points (40×40 mesh of points). This shows that the collision detection algorithm can efficiently detect collision even when a large number of points are generated to achieve higher resolution.

The variation of the computational cost of collision detection will more pronounced if the resolution of two or more B-spline surfaces is changed simultaneously. During the simulation of collision detection for two B-spline surfaces the maximum number of points that can be generated was changed for both the surfaces and its effect on the computational time was recorded. The maximum number of points to be generated for the two B-spline surfaces were increased independently. The number of control points for both the surfaces were kept unchanged at 8×8.

The graph representing the trend of the computation cost is shown in Figure 6.6. The computational cost for collision detection increases when the maximum number of points that can be generated at the lowest level of detail is increased. When 82×82 mesh is used, the computational time is 15 ms. A mesh of size 82×82 means that a total of 6724 points can be generated on both the B-spline surfaces. When the number of points is decreased to a mesh of 28×28, the computational time decreases to 4 ms.

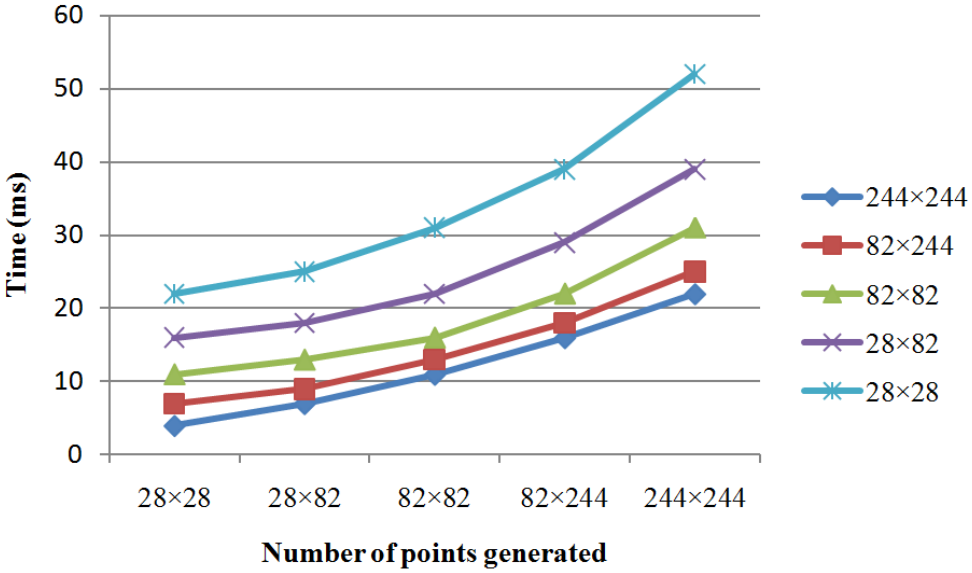


Figure 6.6 Variation of the computational time(ms) of collision detection for a B-spline surface with another deformable B-spline surface with respect to the maximum number of points generated.

6.2.3 Effect of the area of contact

When two surfaces begin to collide, the area of collision is small. Thus, denser points and subsequent spheres are generated only in a small region, at lower levels of detail. This reduces the cost of collision detection. However, in some cases, the area of contact can be very large. This would increase the region where denser points are generated and consequently, increase the number of spheres for which intersection test is to be carried out. It would also increase the number of triangles for which the triangle-triangle intersection test will have to be carried out.

When the maximum number of points to be generated on the surface is selected by the user, the size of the model will not affect the computational cost of collision detection. It will be the percentage of the surface area of the model which will affect the overall computational cost. For the closed B-spline surfaces used to represent solids, it is difficult to calculate the percentage of the surface area colliding with tool. In this case, open flat surfaces can be used to test the cost of collision detection. For simulation, a square flat B-spline surface of degree 3 and represented by an 8×8 control points mesh was used. The resolution of the surface was kept at 82×82 (82 points generated at lowest level of detail in both parametric directions). For the intersection test with a sphere, the sphere was placed at the center of the B-spline surface and its size was increased so that more and more area of the B-spline surface was inside the sphere. This increased the area of collision detection and computational time was noted for different percentages of area. Similarly, the area of contact with plane was increased by increasing the overlap of the plane with the B-spline surface.

Another square plane represented as a B-spline surface was used for B-spline - B-spline collision detection. The area of overlap was increased from 1 to 100 percent. High computational times, when a large percentage of the area is colliding, may not permit real-time collision detection and haptic interaction. The computational cost increases with an increase in the percentage of the area colliding. As both the B-spline surfaces were flat and overlapping each other, all the spheres at all levels of detail would be colliding. This increases the cost of collision detection when the area of contact is large. The trend is show as a graph in Figure 6.7.

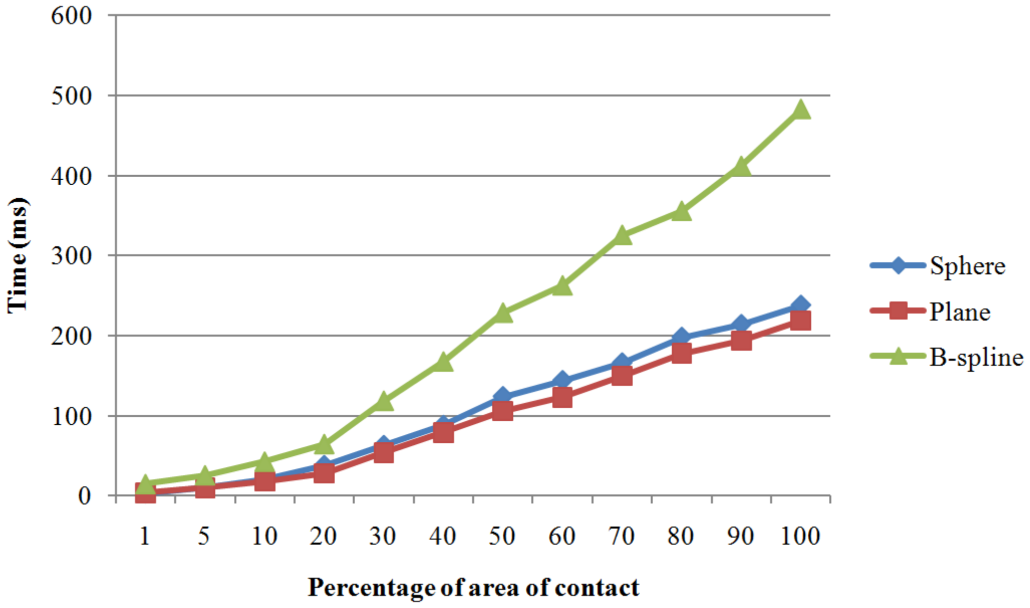


Figure 6.7 **Variation of the computational time (ms) of collision detection for a B-spline surface with a sphere, plane, and another deformable B-spline surface for different percentage area of contact.**

The algorithm presented in this thesis uses another unique feature to reduce the computational cost for collision detection, as discussed in Section 3.3.3.5. The algorithm checks the intersection of spheres at lower level of details. At any level, if all the spheres generated on a B-spline surface are intersecting with the tool, the collision detection algorithm does not go to the lower level of detail.

Interrupting the algorithm from going to lower levels of detail, the computational time will decrease significantly. This makes algorithm robust enough to efficiently tackle collision detection, when a large percentage of area is colliding, thereby permitting real-time collision detection, and haptic interaction.

The trend of computational cost is show as a graph in Figure 6.8. As the percentage of area increases, the cost of computation increases. However, this cost is much lower than that achieved without interrupting the algorithm.

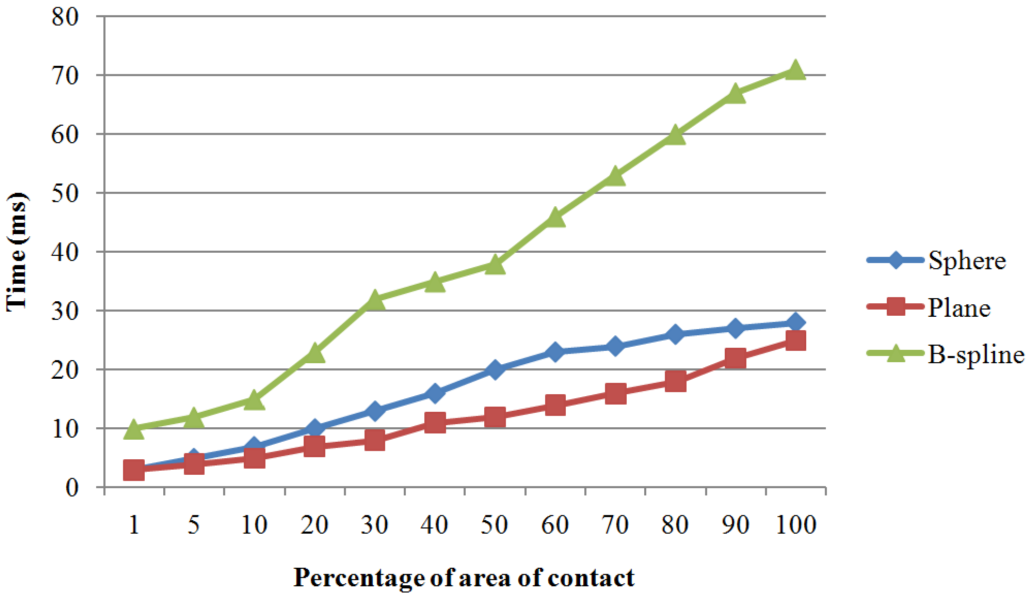


Figure 6.8 **Variation of the computational time (ms) of collision detection for a B-spline surface with a sphere, a plane, and another deformable B-spline surface for different percentage area of contact, when interrupting the algorithm from going to lower levels of detail.**

6.3 Performance of B-spline Surface Patches Merging Algorithm

One major requirement for representing 3D objects in a VR environment is that the user should interact with the virtual model as if it were a visually realistic surface or solid. In commercially available CAD software, if the underlying algorithm is unable to merge surfaces, the user has some options at modifying the surfaces to prepare them for merging. This may be achieved by tweaking control points, trimming the surface along a knot, or matching the surfaces before merging. In contrast, a virtual model in a VR application is not considered a B-spline surface by the user but a physical 3D object and hence any algorithm used to combine surfaces must be sufficiently robust to incorporate all possible cases that may arise. The algorithm described in this thesis (Chapter 4) achieves the merging process without imposing any constraints. An important factor in determining the versatility of the algorithm is that it should be able to tackle all the cases during haptic interaction. In this section, different cases are considered to compare the robustness and accuracy of the merging algorithm.

The process of merging B-spline surface patches is not carried out on regular basis during the haptic interaction. Hence, the efficiency of merging algorithm does not affect the real-time interaction. However, if the user has to wait for the merging process, it may distract an industrial designer from creative process. Thus, it is imperative that this process be made as fast as possible.

6.3.1 Robustness and accuracy of the algorithm

The virtual model is not considered a B-spline surface by the user in VR environment. Hence, any algorithm must be robust so as to incorporate all the types of situations that may arise during merging in VR environment.

The proposed algorithm is suitable for VR environments because the user does not need to directly manipulate the control points of the various surfaces. Furthermore, the algorithm does not impose any kind of restriction such as the degree of the surfaces, the knot vectors, or the type of connectivity. It utilizes the same blending matrices as those used by the collision detection algorithm and, thus, it is not necessary to perform additional or duplicate computations [Pungotra, 2010a]. For comparison purposes, a few cases were considered and the merged surfaces generated by the proposed technique were compared with the results generated by a commercially available NURBS modeling software package, Rhinoceros® (Rhino®). For this analysis, B-spline surface patches of order four are used because these are commonly combined to create complex surfaces in CAD and computer graphics. B-spline surface patches of higher order are considered in Section 6.3.2 as special cases. The algorithm was used to generate surfaces with C^0, C^1, and C^2 connectivity. As Rhino® does not permit the user to edit the knot vector, or to select an option for the type of desired connectivity during the merging process, only C^2 continuous surfaces were generated for Rhino®. Although C^0 and C^2 connectivity are presented for comparison in this study, it is possible for the proposed algorithm to create the merged surface with any desired level of connectivity (K-2 or lower).

6.3.1.1 Case 1: Similar curvatures and knot vectors

This is the simplest case that can be encountered when combining B-spline surface patches. Two arbitrary surfaces with the same degree and uniform knot vectors were used to generate a single integral surface.

Figure 6.9 shows the results for the commercial software and the proposed surface merging approach.

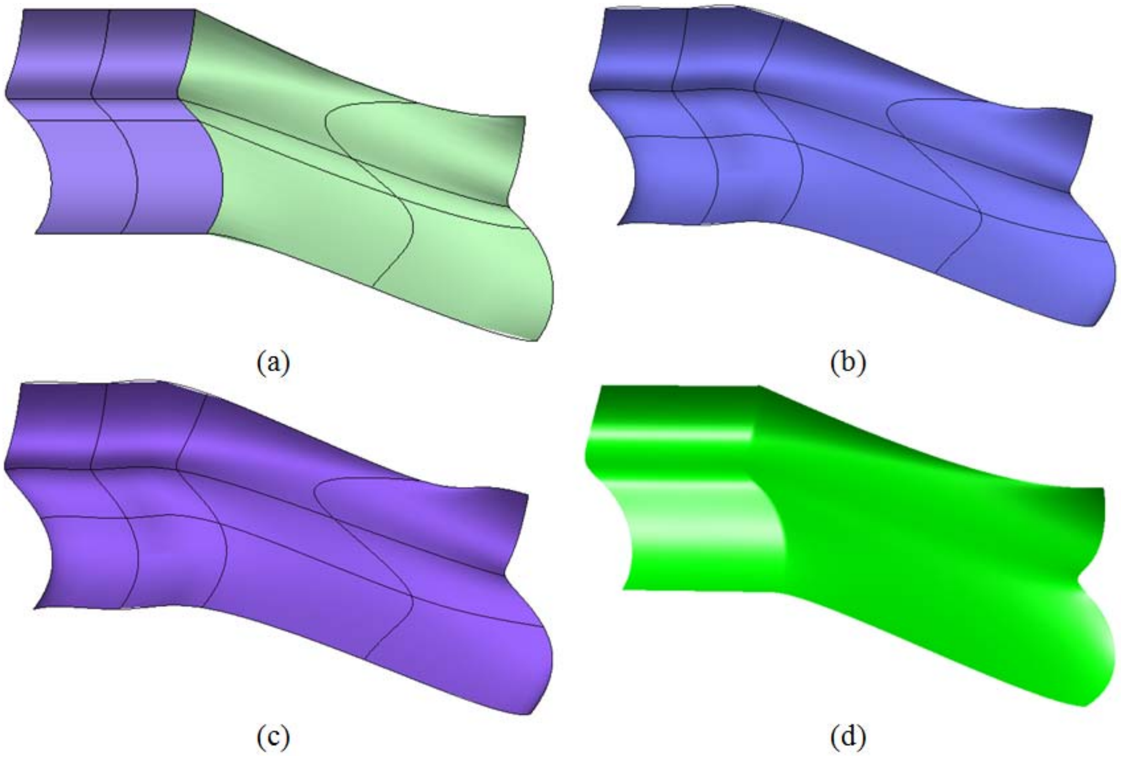
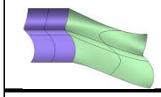
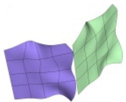
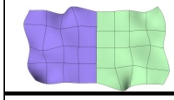
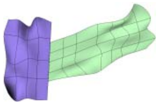


Figure 6.9 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by Rhino® (c) Merged surface with C^2 connectivity generated by proposed algorithm (d) Merged surface with C^0 connectivity generated by proposed algorithm.

Rhino® generates surface with C^2 connectivity by default (Figure 6.9 (b)) and does not allow the merging of the surfaces with C^0 or C^1 connectivity. The proposed algorithm can generate a merged surface with C^0, C^1 or C^2 connectivity. Figure 6.9(c) and Figure 6.9(d) show the merged surface with C^0, and C^2 connectivity, respectively. The results for the deviation of the merged surface from the original surfaces are shown in Table 6.3. The detailed results of the error analysis are given in Appendix A.

The results show that the standard deviation of the merged surface, generated by using the proposed technique, is lower than that generated by Rhino®. The surfaces considered in this example are, in fact, Bézier surfaces and hence the deviation is large. However, when these are joined with C^0 connectivity the standard deviation reduces to a very small value and within the tolerance required even for CAD/CAM applications. In a virtual reality environment a deviation of less than 0.5 mm can be considered adequate.

Table 6.3 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm.

Test Cases	**Error Analysis** (using point set deviation)	**Merged Surface**		
		Rhino® C^2	**Proposed Method C^2**	**Proposed Method C^0**
1: Similar curvature and knot vectors 	Average distance	0.2242	0.3568	0.0104
	Standard deviation	0.2704	0.2452	0.0164
2[1]: Different curvatures but similar knot vectors 	Average distance	0.0778	0.0837	0.0458
	Standard deviation	0.1708	0.1093	0.0621
3: Similar curvature but different knot vectors 	Average distance	0.0259	0.0273	0.0110
	Standard deviation	0.0364	0.0283	0.0096
4[2]: Similar curvatures but different knot vectors and dimensions 	Average distance	-	0.0215	0.0233
	Standard deviation	-	0.0263	0.0228

[1] The surfaces were matched (fitted without merging) before merging for Rhino®. It did not allow the merging of these surfaces for being too far apart, without matching.

[2] Rhino® did not merge the surfaces even after matching.

6.3.1.2 Case 2: Different curvatures but similar knot vectors

In many cases, particularly when the user is manipulating surfaces derived from CAD or reverse engineering, the curvature of the patch edges may not be similar, as shown in Figure 6.10(a).

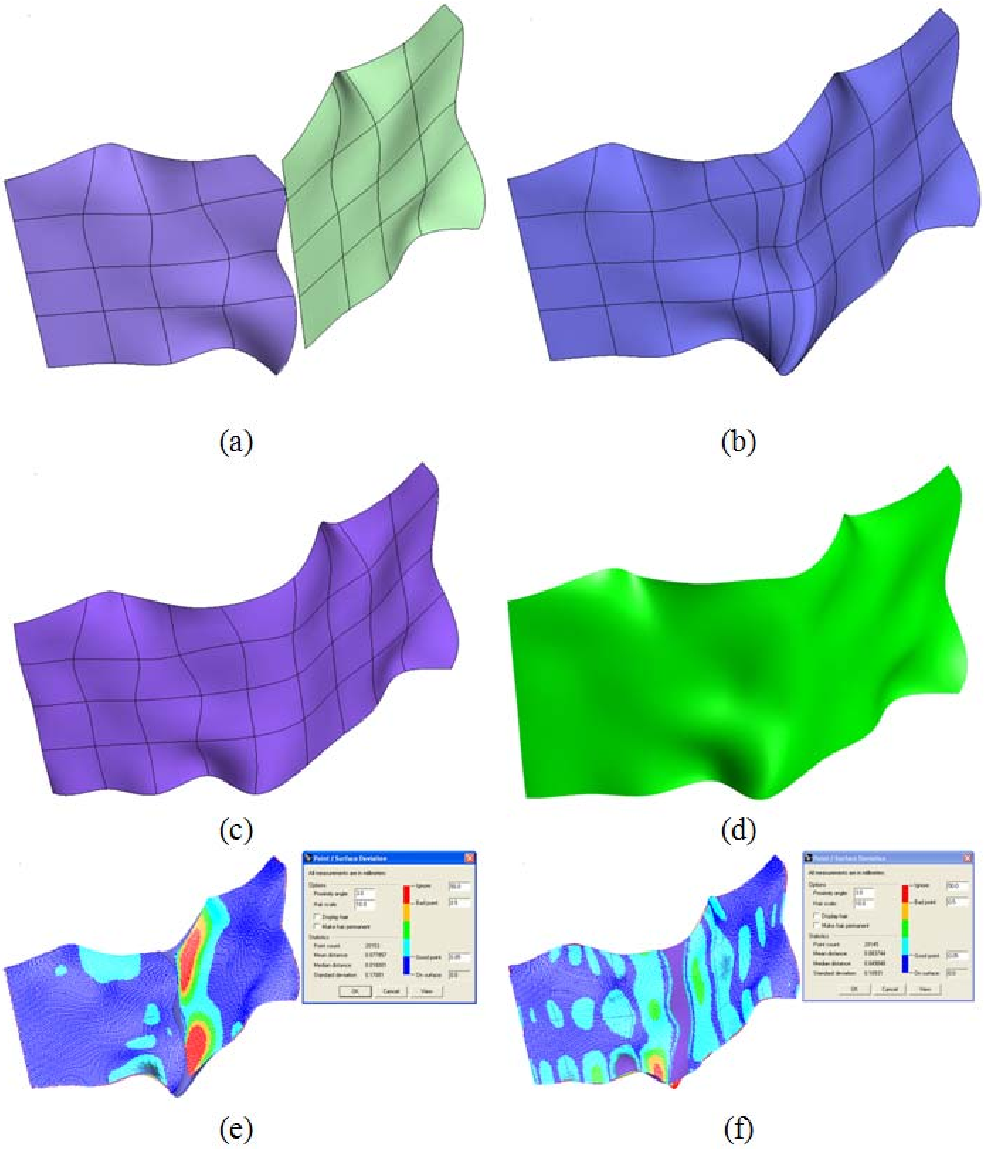


Figure 6.10 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by Rhino® after matching the surfaces (c) Merged surface with C^2 connectivity generated by proposed algorithm (d) Merged surface with C^0 connectivity generated by proposed algorithm (e) Point set deviation for the C^2 surface generated by Rhino® after matching the surfaces (f) Point set deviation for the C^2 surface generated by proposed algorithm.

The traditional methods of joining the surfaces do not work well if the curvatures do not match at the joining edge. Rhino® did not allow the surfaces to be merged because the surfaces were considered to be too far apart. In some cases, the CAD/CAM software allows the user to match surfaces (fitting the surfaces with certain connectivity without merging), thereby reducing, to a large extent, the gaps between the two surfaces. Once the gaps are closed, it might be possible to merge the surfaces. However, there is no guarantee that the surfaces will merge after matching. A merged surface was created by using this process on Rhino® as shown in Figure 6.10(b).

In contrast, the proposed algorithm does not consider the curvature of the edges of the merging surfaces. Figure 6.10(c) and Figure 6.10(d) show the merged surface using the proposed algorithm, without going through the process of matching the surface, with C^2 and C^0 connectivity respectively. This feature is important, particularly in a virtual reality environment, because the user considers the models as objects and does not treat them as B-spline surface patches. Although the distance between the rows of discrete points near the edges can vary significantly, causing a deviation in parameterization for these points, the overall result is still satisfactory. As shown in Table 6.4, the standard deviation, even for C^2 continuity, is about 0.1 mm, and reduces further to 0.06 mm, for C^0 continuity. In contrast, even after going through an extra process of matching the surfaces, the standard deviation for the surface generated by Rhino® is more than 0.17 mm. The results of the point set deviation analysis generated by Rhino® are also shown. Figure 6.10(e) shows the deviation between discrete points of the original surfaces and the surface generated by Rhino®, having C^2 continuity, and Figure 6.10(f) shows it for the surface with C^2 continuity generated by the proposed algorithm. It is clear from the figures that Rhino® shows large deviation in the vicinity of the common edge. This is because Rhino® manipulates the control points only in the vicinity of the common edge. The proposed algorithm, on the other hand, is able to manipulate large number of control points and hence has lower standard deviation. With the proposed algorithm, the user does not have to close the gaps by manipulating the surfaces before these can be merged. This makes the task of joining surfaces easier for the user while working in a VR environment. The tolerance is also low, which makes this technique suitable, even for other CAD/CAM applications.

6.3.1.3 Case 3: Similar curvature but different knot vectors

One constraint for NURBS-based surfaces is that an integral surface needs to have a continuous knot vector, though it can be uniform or non-uniform. If two surfaces with different knot vectors are to be merged into a single surface, then this constraint must be addressed. The traditional approach is to insert additional knots in the surfaces to achieve a continuous knot for the integral surface. Theoretically it is possible to insert a knot into a surface without changing its geometry, though many times, a slight change in the shape of the surface is observed. Even if there is no change in the geometry of the surface, additional knots increase the number of control points needed to represent the surface. These additional control points carry no significant geometric information and are added only to satisfy the constraint of having a continuous knot vector. The additional control points are a nuisance for designers because these require the designer to deal with more data and make it difficult to manipulate the geometry by moving the control points. In a virtual reality environment, the increased number of control points increases the computational cost of collision detection and manipulation.

The proposed technique does not introduce additional knot vectors. It just readjusts the knot vectors, by averaging, in such a way that the merged surface has a common knot vector. Consider a case where the two surfaces being merged have knots given by U^1 = [0, 0, 0, 0.25, 0.49, 0.67, 0.75, 0.9, 1, 1, 1] and U^2 = [0, 0, 0, 0.20, 0.45, 0.61, 0.71, 0.85, 1, 1, 1]. Then the resultant knot is calculated as average of the respective knot vectors of merging surfaces and will be given by U = [0, 0, 0, 0.22, 0.47, 0.64, 0.73, 0.87, 1, 1, 1]. Figure 6.11(a) shows the B-spline surface patches with different knot vectors and Figure 6.11(b) shows the merged surface generated by Rhino®. The merged surface created by Rhino® had four more control points in v direction as compared to the original surface patches. The surface generated by using the present algorithm with C^2 and C^0 connectivity, as shown in Figure 6.11(c) and Figure 6.11(d) respectively, had the same number of control points (6) as the surfaces that were merged. The merged surface generated by the proposed algorithm exhibits a better standard deviation even for C^2 connectivity, as shown in Table 6.4. In the case of C^0 connectivity, the standard deviation was observed to be less than 0.01 mm.

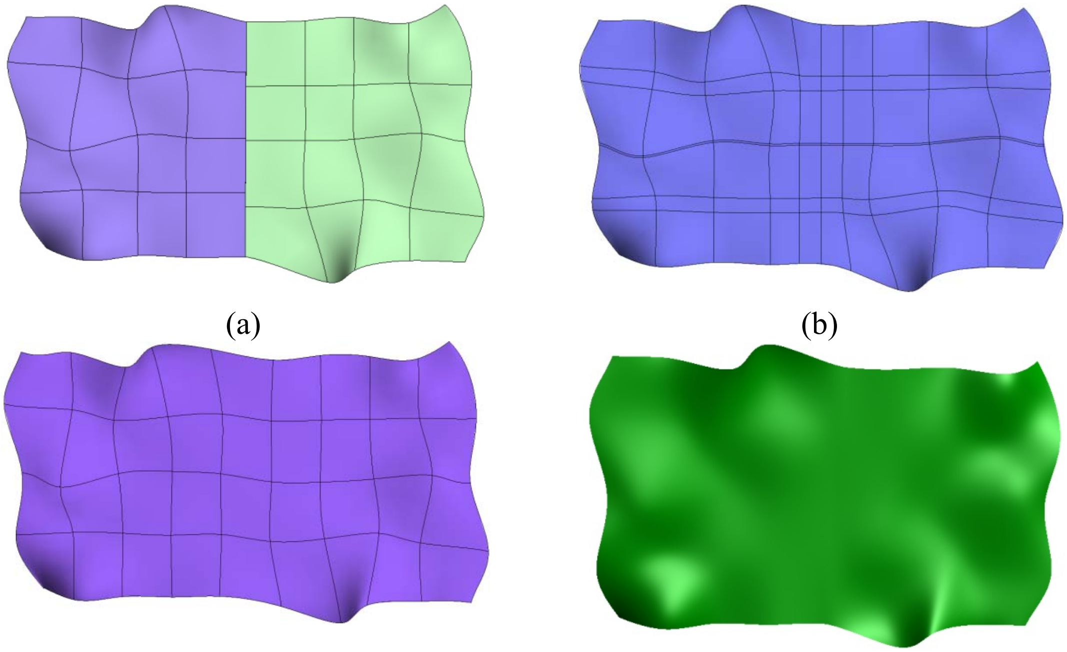

(a)
(b)
(c)
(d)

Figure 6.11 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by Rhino® with increased knots (c) Merged surface with C^2 connectivity generated by proposed algorithm (d) Merged surface with C^0 connectivity generated by proposed algorithm.

6.3.1.4 Case 4: Similar curvatures but different knot vectors and dimensions

In many cases, the edges of the patches to be merged do not have the same length. As the merged surface should have same parameter throughout the surface, the two surfaces need to deviate a lot from the original geometry. In commercially available CAD/CAM software, any two surfaces which do not have almost the same dimension cannot be merged. Rhino® could not merge the surfaces even after matching these surfaces.

The present algorithm does not put any of these constraints on the surfaces to be merged. Figure 6.12 shows initial B-spline surface patches and the merged surface generated by using the proposed algorithm. The merged surface generated by the proposed algorithm does show a deviation when compared to the original surface, but overall, the result is satisfactory. As shown in Table 6.4, the standard deviation from the surface is 0.026 mm for C^2 connectivity which is very small considering the large differences in the length of the edges at which the two surfaces were to be joined.

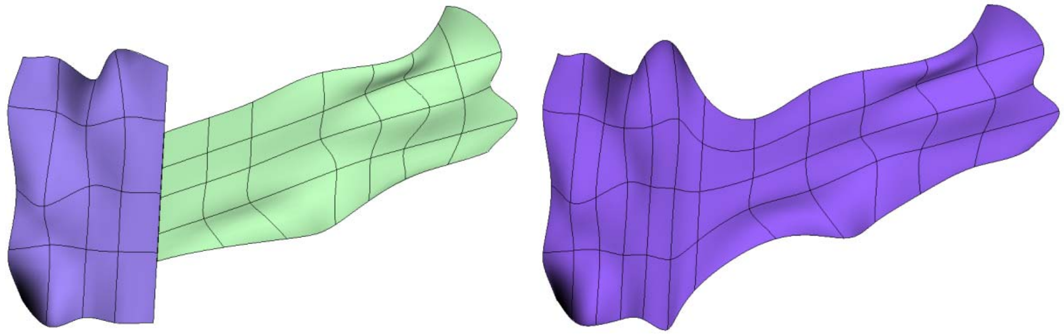

(a) (b)

Figure 6.12 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by the proposed algorithm.

6.3.1.5 Case 5: Intersecting and trimmed surfaces

If the patches to be merged are intersecting, the surface may not merge as intended by the user or may not merge at all. Rhino does not merge intersecting surfaces. The algorithm presented in this thesis uses collision detection to merge two or more intersecting B-spline surface patches, as discussed in Section 4.4.1.

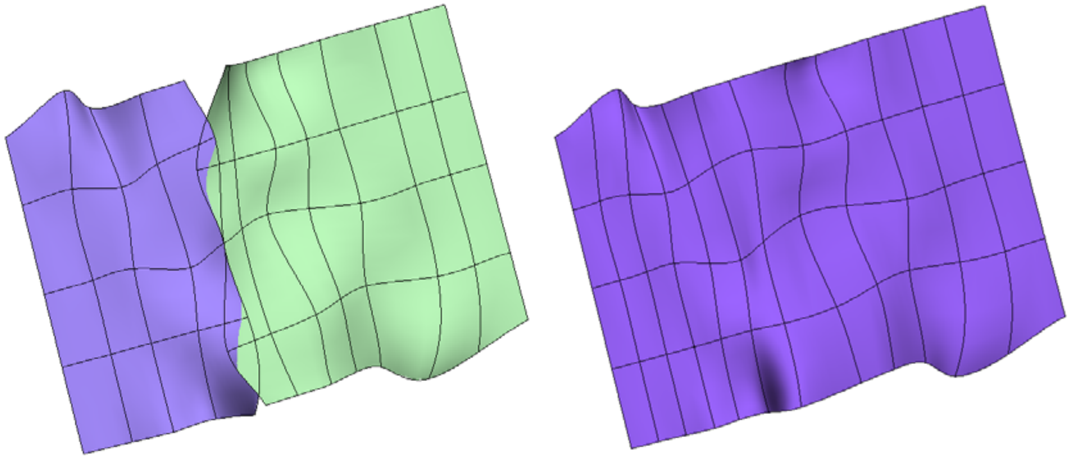
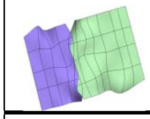
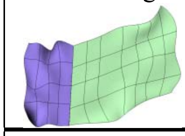
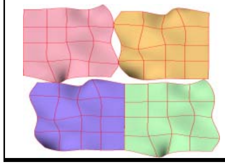

(a) (b)

Figure 6.13 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by proposed algorithm.

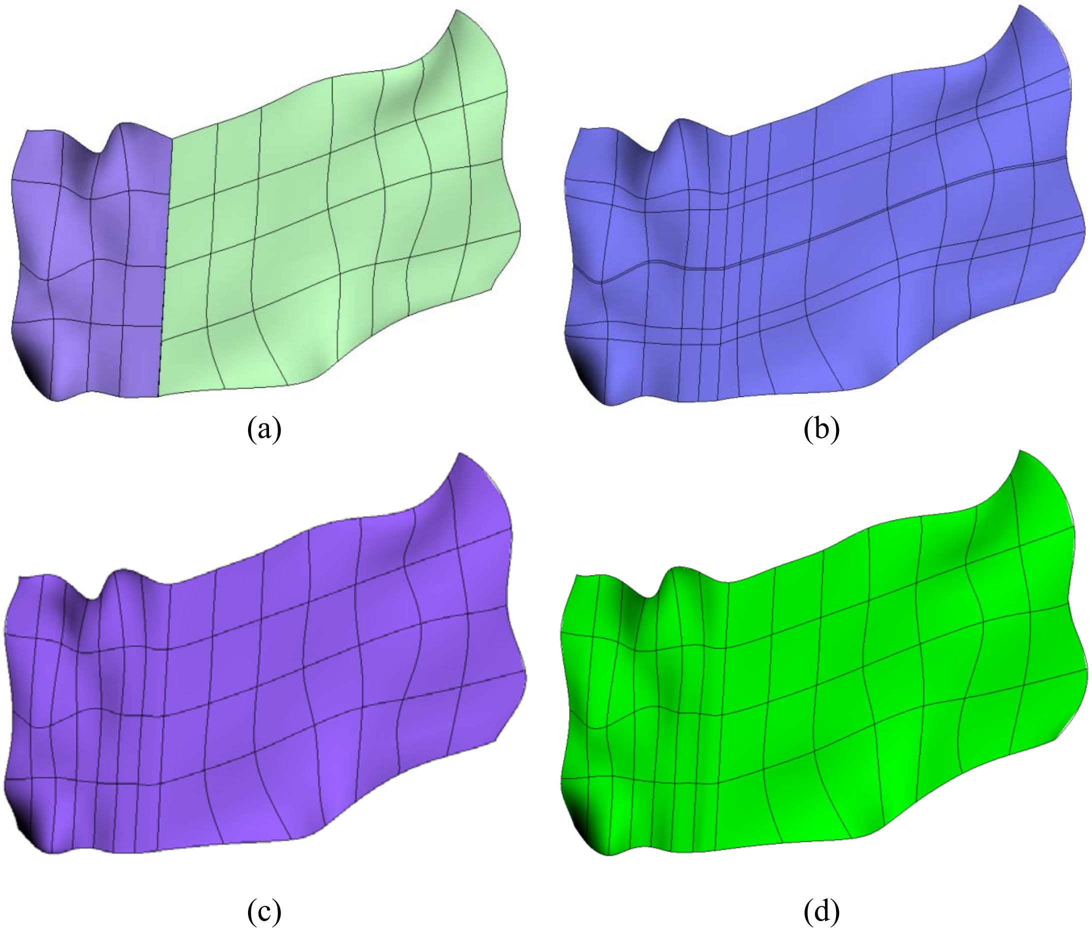
Figure 6.13 shows the merged surface generated by using the algorithm presented in this section. Except for the method of determining the matrix of combined points of discretization (**M**), the rest of the algorithm is the same as described in the previous sections. It is clear from Table 6.4 that the surface generated is within the tolerance needed during the haptic interaction with the model. Detailed error analysis of merged surface is provided in Appendix A.

Table 6.4 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm.

Test Case	Error Analysis (using point set deviation)	Merged Surface		
		Rhino® C^2	Proposed Method C^2	Proposed Method C^0
5: Intersecting and trimmed surfaces 	Average distance	-	0.0423	0.0451
	Standard deviation	-	0.0792	0.0771
6: Surfaces having different degrees 	Average distance	0.0126	0.0139	0.0157
	Standard deviation	0.0125	0.0117	0.0154
7: Multiple surfaces having different curvature and knot vectors 	Average distance	-	0.0882	0.0454
	Standard deviation	-	0.1218	0.0525

6.3.1.6 Case 6: Surfaces having different degrees

Before the surfaces having different degrees can be merged, the degree of the surfaces in *u* and *v* directions need to be made uniform. Thus, the problem reduces to increasing or decreasing the degree of a surface to make it uniform with that of the other surface. The proposed algorithm can increase or decrease the degree of the surfaces as discussed in Section 4.4.2.



(a) (b)

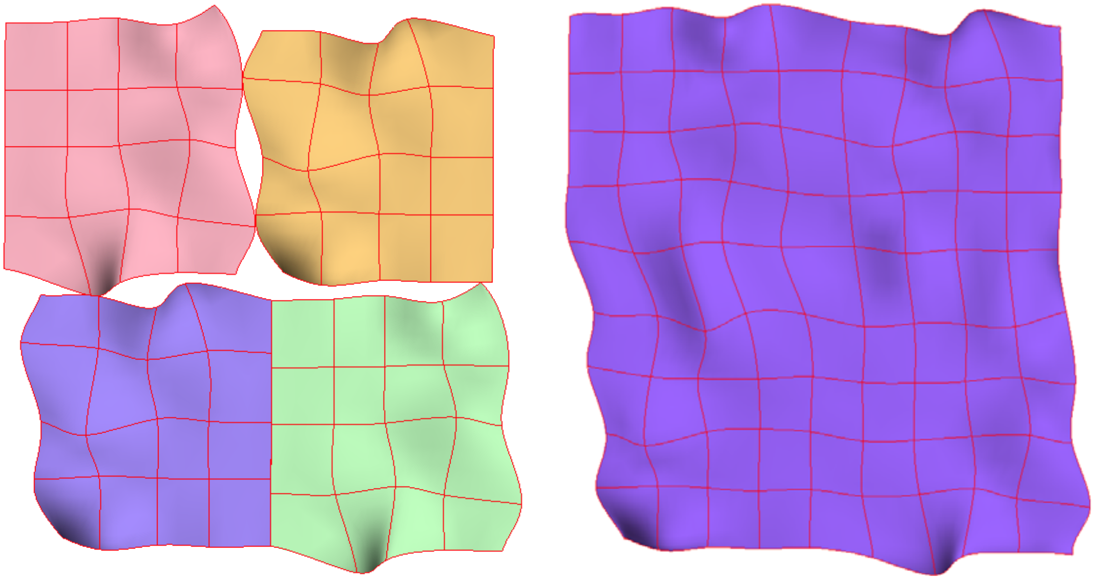
(c) (d)

Figure 6.14 (a) Initial B-spline surface patches (b) Merged surface of order 5×6, with C^4 connectivity in v direction, generated by Rhino® with increased knots (c) Merged surface of order 5×6 with C^4 connectivity in v direction, generated by proposed algorithm (d) Merged surface of order 4×4 with C^2 connectivity generated by proposed algorithm.

Figure 6.14 shows the results of merging two surfaces having different degrees. The first surface has 6×6 control points and degree three in both the directions. The second surface has 8×12 control points and degree four in u direction and degree five in v direction. The surfaces are being merged in v direction. Thus, the first surface will have to be elevated to degree four in u direction and degree 5 in v direction. Rhino®, by default increases the degree of lower order surface and merges these two surfaces with degree five in v direction with connectivity of C^4 and degree four in u direction. In order to have a common degree for the integral surface, the proposed algorithm can generate the merged surface by increasing the degree of the lower order surface or decreasing the degree of the higher order surface. The surface generated by the proposed algorithm does not require additional knots to be introduced and, hence, has a lower number of control points as compared to the one generated by Rhino®. Table 6.4 shows the deviation of the merged surface generated by Rhino® and proposed algorithm from the original surfaces.

6.3.1.7 Case 7: Multiple surfaces

The algorithm is capable of merging any number of surfaces. Figure 6.15(a) shows four B-spline surfaces to be merged and Figure 6.15(b) shows the resultant merged surface obtained by using the proposed algorithm. Rhino® could not merge the surfaces even after matching the surfaces and merging these in pairs. As shown in Table 6.4, the merged surface shows very small standard deviation from the original surfaces.



(a) (b)

Figure 6.15 (a) Initial B-spline surface patches (b) Merged surface with C^2 connectivity generated by proposed algorithm.

6.3.2 Computational efficiency of the merging algorithm

The algorithm achieves computational efficiency by utilizing all available resources. It efficiently uses the blending matrices already calculated and stored for collision detection. The revised blending matrices generated during the merging process are used later by the collision detection algorithm, thereby minimizing the number of redundant calculations. The efficiency of the algorithm stems from the fact that it works in tandem with the collision detection algorithm. The main computational cost comes from the discretization of the surfaces to be merged and the computation of the revised blending matrices and their inverses.

Discretization of the surfaces to generate common matrix **M** is done by multiplying blending matrices with their respective control point matrices. The blending matrices are band matrices, having a maximum of "*degree* + 1" number of non-zero terms in each row. Thus, the cost of discretization is minimal. Only two revised blending matrices (and their inverses) need to be calculated for all the surfaces being merged simultaneously. Again, because a revised blending matrix is a band matrix, its inverse can be safely found by Gaussian Elimination Method without using a pivot. This reduces the overall computational cost. Table 6.5 enumerates the time taken by the algorithm to merge surfaces in two different scenarios.

Table 6.5 Computational time of the algorithm for merging B-spline surface patches.

Test Cases	**Time (seconds)** when using the pre-calculated blending matrices	**Time (seconds)** when calculating all the blending matrices and loading B-spline surfaces
1: Similar curvature and knot vectors	0.0025	0.044
2: Different curvatures but similar knot vectors	0.0037	0.061
3: Similar curvature but different knot vectors	0.0028	0.073
4: Similar curvatures but different knot vectors and dimensions	0.0033	0.097
5: Intersecting and trimmed surfaces	0.0031	0.094
6: Surfaces having different degrees	0.0039	0.131
7: Multiple surfaces with different curvatures and different knot vectors	0.0048	0.145

For this study the code was efficiently run on a computer with "Intel(R) core(TM) 2 Quad CPU with 4 GB RAM @ 2.66 GHz". In the first scenario, the algorithm uses the data stored by the collision detection algorithm. The time taken to merge the surfaces is enumerated in column 2. In the second scenario, the proposed algorithm does not use the resources stored by the collision detection algorithm. The computational cost in such a scenario would include loading the surfaces being merged and calculating their blending matrices. Column 3 of Table 6.5 shows the time taken to merge the surfaces in the second scenario. This scenario does not include the computational cost to find out the regions to be discretized, during the merging of intersecting surfaces (case 5). The computational cost to find the intersecting regions will depend upon the efficiency of the collision detection algorithm. As is clear from Table 6.5, the time required to merge the test surfaces described in the chapter is less than five milliseconds. Even when this code is independently run without using the resources for collision detection algorithm (blending matrices, knot vectors, control point matrices and other related information of the B-spline surface patches to be merged), the maximum time taken to merge the surfaces is 0.145 seconds for the test case # 7. The computational time for the proposed algorithm is very small and proves its efficiency.

6.4 Computational Efficiency of Physics-based Deformation Algorithm

The B-spline collision detection and merging algorithms can work independent of the physics-based deformation system used. However, in this thesis, the mass spring mesh has been used to impart material properties to the virtual model. This mass spring mesh is created during the pre-processing stage using VSOFM [Igwe, 2008a] in order to provide more realistic virtual experience. A denser mesh of mass spring nodes increases the accuracy of the deformation behaviour for the virtual model. Unfortunately, a denser mesh will also increase the computational cost for calculating the shape change and resultant forces that must be fed back to the user.

To simulate realistic material behavior, the mass spring damper system requires small time-step to simulate physics based deformation of model characterized by a material from low to high stiffness. The running time depends upon the size of the node mesh for the mass spring system and the number of training cycles. The mass spring damper system can be used for local deformation and global deformation.

If the user desires to slightly change the shape or features of an existing object, local deformation will be a better approach. In this case the number of training cycles can be small (up to 50). During these iterations, the nodes which are in the vicinity of the colliding surfaces will move under the influence of external and internal forces. As the number of iterations (training cycles) increase, larger number of mass spring nodes will be affected by the external and internal forces. If the user desires to determine the characteristics of the model under the influence of external forces, depending upon the material properties assigned to the model to represent his/her concept, global deformation will be a better approach. In this case a large number of iterations will be needed so that all the nodes of mass spring system get sufficient time to settle under the influence of external and internal forces.

Table 6.6 shows the update time of single iteration for different size of the mass spring damper node. The update time was noted when the algorithm was run on Intel Pentium (R) D/3.2 GHz [Igwe, 2008a].

Table 6.6 Update time for single iteration of mass spring system for different node sizes.

Number of nodes	Update time (ms)
10×10×10	0.1
15×15×15	0.4
30×30×30	3.0

6.5 Computational Efficiency of the Interactive Design Framework

The interactive design framework consists of collision detection algorithm, B-spline surface patches merging algorithm, and mass spring system. In previous sections, these algorithms were independently implemented to determine the efficiency and accuracy of these algorithms. In this section, these algorithms were combined and the overall computational time was determined. As discussed in Sections 3.3 and 5.2, the collision detection and mass spring system algorithm has two phases. In the pre-processing phase, blending matrices and their inverses are determined. At the same time the nodes for the mass spring system and the properties of the springs and dampers are determined, based on the material properties assigned to the virtual model. The computational cost of pre-processing phase does not affect the run-time computational time. During the run-time phase, the total computational cost includes the cost of collision detection, and cost of determining the deformation of the model using mass spring system.

6.5.1 Pre-processing phase

During the pre-processing phase the algorithm receives input of the control points, knot vector, type of surface (open or closed), maximum number of points to be generated on the B-spline surface at the lowest level of detail, the number of nodes for the mass spring system, and the material properties to be assigned to the model. If the knot vector is not given, it is calculated by the algorithm.

As discussed in Section 3.3, two blending matrices are generated for each B-spline surface. At the same time, the inverse of these blending matrices is calculated. The computation time for calculating these blending matrices is small. For a B-spline surface with a control point mesh of size 50×50 and having a maximum of 244×244 number of points that can be generated on the B-spline surface at the lowest level of detail, the computational time to calculate blending matrices and their inverses is 1.434 seconds. The size of the blending matrix in u direction will be 50×244 and the size of the blending matrix in v direction will be 244×50. This time reduces to 8 milliseconds for a B-spline surface having a control point mesh of size 8×8 and 82×82 number of points that can be generated on the B-spline surface at the lowest level of detail.

The computation time to generate nodes for mass spring system depends upon the size of the mass spring mesh and the number of learning cycles used to generate VSOFM mesh. More cycles (up to 1000) are needed when the VSOFM models is required to generate the mass spring mesh from a very large cloud point set. In the algorithm presented in this thesis, very small number of nodes is given for generation of mass spring nodes. Thus, fewer learning cycles (100) give very good results. In the present algorithm, 300 learning cycles were used. For a 12×12×12 mass spring mesh, it takes 3.471 seconds to generate the mass spring damper model. Table 6.7 shows the computational time for generation of mass spring mesh of different sizes. Even for 20×20×20 mass spring mesh the pre-processing time is less than half minute, which is very reasonable.

Table 6.7 Pre-processing time for generation of mass spring system for different node sizes.

Number of nodes	Pre-Processing Time (seconds)
10×10×10	2.44
15×15×15	7.88
20×20×20	28.48
25×25×25	76.90
30×30×30	213.02
35×35×35	353.19
40×40×40	958.10

The computation time increase with increased size nodes for the mass spring system. Figure 6.16 shows the correlation between the size of the mass spring mesh being generated and the pre-processing time for calculating the mesh by using volumetric self organizing feature map.

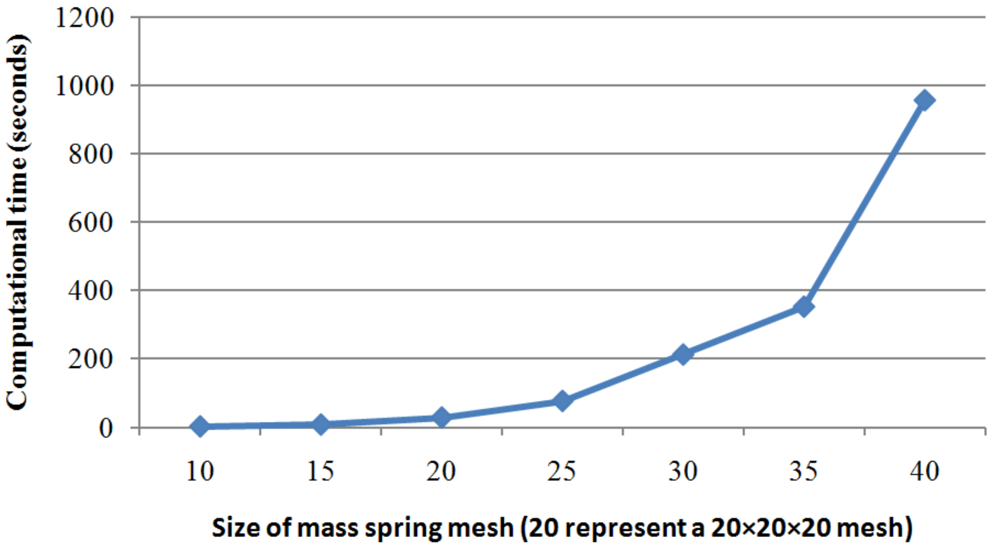
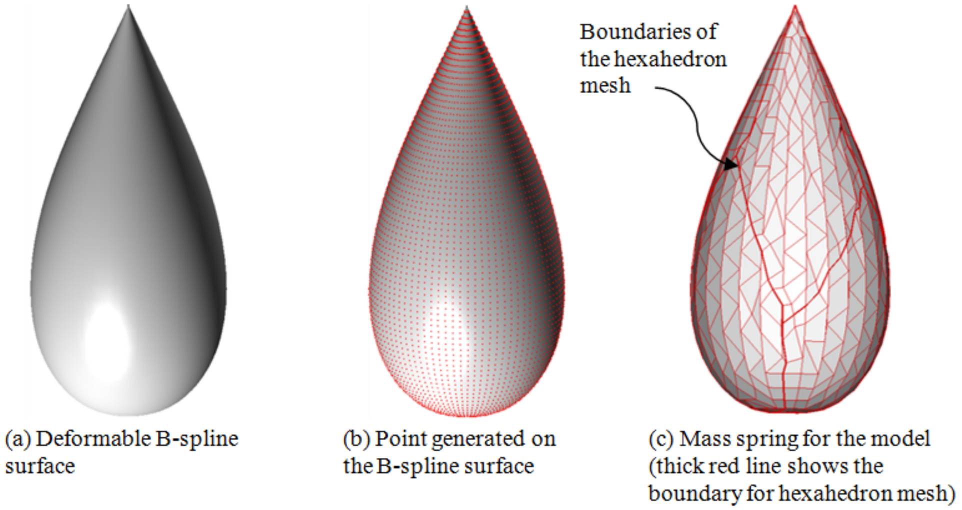


Figure 6.16 Pre-processing time for calculating the mass spring mesh of different sizes.

6.5.2 Run-time phase

During the run-time phase the total computational time is sum total of the computation times for collision detection and force response system. Number of control points used to represent the B-spline surface and the maximum number of points to be generated on the B-spline surface determine the computational time for collision detection. At the same time, the number of nodes needed to represent mass-spring system mesh and the number of iterations performed to calculate deformation of the model determines the computational cost for the force response model.

The computational times for different algorithms were enumerated in Sections 6.2, 6.3, and 6.4 under different conditions. For implementing the interactive design framework B-spline surface was represented by a control point net of size 8×8. Maximum of 82×82 points can be generated on the B-spline surface at the lowest level of detail. Thus the B-spline surface can be represented at very good resolution with sufficient number of control points. The size of the nodes for the mass spring system was 12×12×12. The variables were assigned for a general case. However, these variables can be varied by the user as per his/her requirements.



(a) Deformable B-spline surface

(b) Point generated on the B-spline surface

(c) Mass spring for the model (thick red line shows the boundary for hexahedron mesh)

Figure 6.17 (a) Deformable B-spline model (b) Points for collision detection (c) Mass spring mesh.

Figure 6.17(a) show a deformable tear drop B-spline model, the maximum number of points that can be generated at the lowest level of detail (Figure 6.17(b)), and the mass spring system for the model (Figure 6.17(c)). The thick red lines in Figure 6.17(c) show the boundaries of the hexahedron mass spring mesh. These boundaries were the edges of a cube and adapt to the shape of the B-spline model. The model was made to interact with different types of tools. These included point-based, implicit surface-based (sphere, plane), and deformable B-spline surface-based tools. The algorithm can generate frame rate in excess of 30 Hz.

6.6 Concluding Remarks

In this chapter, the algorithms developed for collision detection, merging B-spline surface patches, and mass spring damper based deformation system were implemented. Computational efficiency and robustness of these algorithms were checked during the implementation. The proposed algorithms are robust as these are capable of handling all the cases that are acceptable for NURBS-based surfaces. These algorithms do not impose any restriction on the degree of the surface patches, the number of control points, convexity or concavity of the surface, and extent of deformation.

Collision detection algorithm can efficiently carry out the intersection test for a variety of surfaces, such as, point, implicit surfaces, tessellated surfaces, and deformable B-spline surfaces. Even when the area of contact is large, the collision detection algorithm can maintain reasonable frame rate. B-spline surface patch merging algorithm additionally does not impose any restriction regarding the type of continuity required at the common edge, the knot vector, or the number of surfaces being merged simultaneously. The proposed algorithm can efficiently merge B-spline surface patches having dissimilar curvatures at the common edge, intersecting B-spline surface patches, and the B-spline surface patches having trimmed edges. The user will not have to face a situation where the two surfaces cannot be merged due to intersection, dissimilar curvature at common edge or similar situations. This adds to the robustness of the algorithm. All the algorithms work in tandem and use the pre-computed blending matrices efficiently. This increases the robustness and efficiency of the framework developed in this thesis.

The virtual interactive design module which consists of the collision detection and merging of B-spline surfaces and mass spring mesh, can achieve frame rates in excess of 30 Hz. This will allow the user to interaction with the deformable model in real-time. However, if a very dense mass spring mesh is used or if the numbers of control points of the B-spline surfaces are very large, the computation time will increase.

CHAPTER 7 DEFORMABLE MODELS FOR INTERACTIVE DESIGN AND USER TRAINING

7.1 Introduction

Modeling deformable objects for freeform interactive design is an active research topic in engineering. Physically based virtual models provide a sense of realism to the user for many applications in engineering. Over the past decade, B-spline modeling has become the standard mathematical model for representing freeform or organic objects in CAD/CAM systems. Using a B-spline surface to represent the virtual model in a haptic interactive design module helps to streamline the exchange of the information with existing CAD/CAM systems. B-spline surfaces also help increase visual realism because these represent continuous surface. At the same time, the algorithms used for interaction with the virtual model should be computationally efficient to maintain acceptable level of virtual realism of 30 Hz required for human eye to perceive the dynamic simulation as continuous with no time lag.

Figure 7.1 shows the various modeling techniques developed in this thesis and their relationship to the interactive design problem. These techniques are integrated to develop a B-spline surface based interactive design module for various applications in virtual reality environment. The same virtual reality issues exist for user training and the modeling techniques developed in thesis can be used to provide a virtual environment to the user similar to the real world applications. This will help users to benefit from the unique features of the design concept. At the same time, the users can provide valuable insight to an industrial designer during their interaction with various design concepts.

Simple illustrations of an application familiar to everyone are presented in this chapter to demonstrate the capability of the developed algorithm for freeform interactive shape design and user training. B-spline surface based deformable models have been used for the illustrations. The tools have been represented as rigid implicit surfaces (plane and sphere) as well as deformable B-spline surfaces.

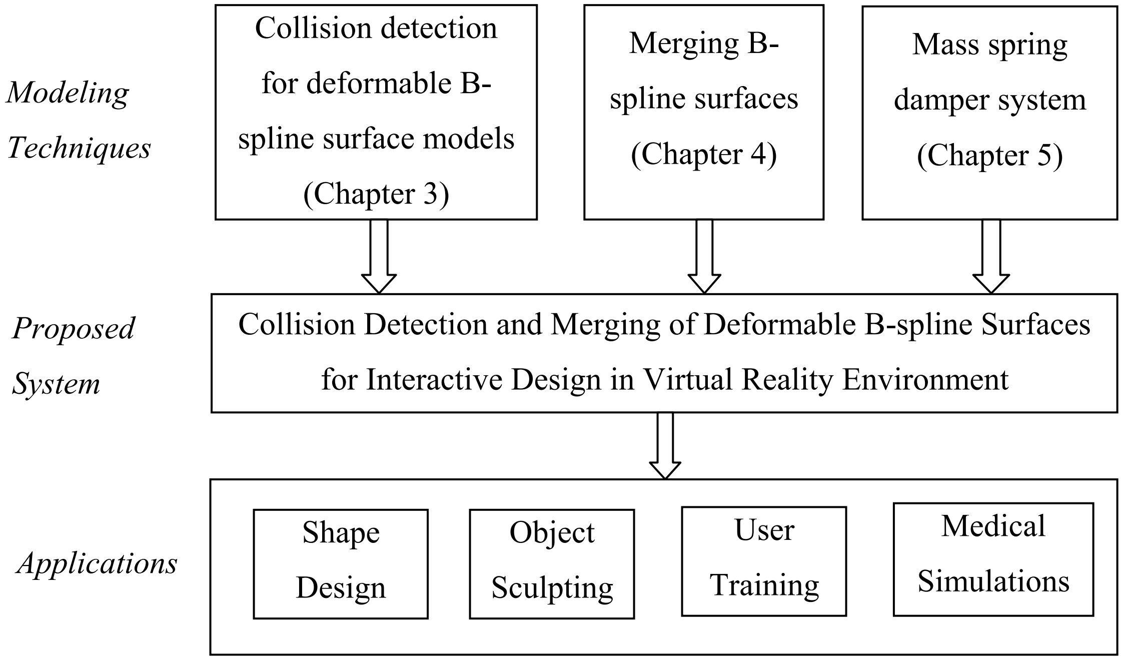

Modeling Techniques
Collision detection for deformable B-spline surface models (Chapter 3)
Merging B-spline surfaces (Chapter 4)
Mass spring damper system (Chapter 5)
Proposed System
Collision Detection and Merging of Deformable B-spline Surfaces for Interactive Design in Virtual Reality Environment
Applications
Shape Design
Object Sculpting
User Training
Medical Simulations

Figure 7.1 Integration of various modeling techniques proposed in this thesis to develop B-spline surface based interactive design module for various applications in virtual reality environment.

7.2 Conceptual Product Design

The concept design process needs creativity and freedom to innovate and explore alternative solutions [Morris, 2009]. During the concept generation phase a rough idea, which can come from the background research or from a previous design, is expanded into several solution alternatives. Physical product design and production may require major investments and can lead to significant financial implications in the event of a solution not meeting design requirements or specifications. However, these risks can be managed by developing and testing new solutions at the concept stage. The product concepts can be evaluated depending on the design considerations and identified customer needs.

Based on the results of a user study, an industrial designer first outlines requirements of the users. Concept design process starts by defining the user groups and describing the usage of the final concept product. The designer then defines a hypothetical user activity and starts generating more detailed solutions for the products to support this activity. This process includes a description of the basic functions, design specification requirements, functional requirements, ergonomic shape features, constraints, and other important technical attributes.

The evaluation of product concept solutions is one of the critical steps in the concept development process. The target of the evaluation is to make a decision on whether to discontinue the concept, further iterate the concept, or start utilising the concept. Valuable insight into refining the concepts can be gained using evaluation methods involving end-users. An important goal is to identify whether the new product concepts find acceptance amongst the intended target user group. Another goal may be to evaluate the design from a human factors perspective. User evaluation is useful as a tool for iteratively refining the designs based on user feedback in accordance with the concept of user-centred design.

The modeling techniques developed in this thesis can enable an industrial designer to sculpt and validate the concepts in the virtual reality environment. A group of users, fairly representative of the intended user's segment, can evaluate the concept in VR environment. Further design modifications may be guided by different sets of users. These different sets of user may be based on different cultures, geographies, age groups, or medical conditions. These modifications will often deal with a variety of needs that can be fulfilled using innovative but easily implemented changes to the existing generic model.

Concept design focuses on the fundamental characteristics of the product that distinguishes it from the existing products or concepts. These characteristics may include appearance, product size, shape, ergonomics, interaction, and intended user segment of population. These characteristics may result in tangible benefits for different user segments. In this context, the term *easy-to-use* may be a vague generalized idea. Different user segments will find varying degree of ease while using a product.

In order for product design to fulfill the requirements of diverse set of users, several methods for supporting design activities have been developed. Computer Aided Design (CAD) systems support the precise and detailed specifications of the geometry of the product. However, before the precise and detailed specification of product design is carried out, several concepts must be developed. These concepts must be tested by various user segments. Virtual reality can be used to design and test these concepts. The modeling tools developed in this thesis can be used for the purpose. B-spline surface representation of the model ensures streamlined exchange of information with CAD systems for precise and detailed specifications. The following sections illustrate the use of the modeling methodologies developed in this thesis for shape design, validation of concept, and user training.

7.3 Interactive Design of an Ergonomic Spoon

Real-time interaction with a product model during interactive development provides a quick insight into the overall performance of the proposed solution. The example of designing a functional, stylistic spoon for different user segments is used to establish the application and desirability of the modeling tools developed in this thesis. Figure 7.2 shows many of the commercially available spoons.

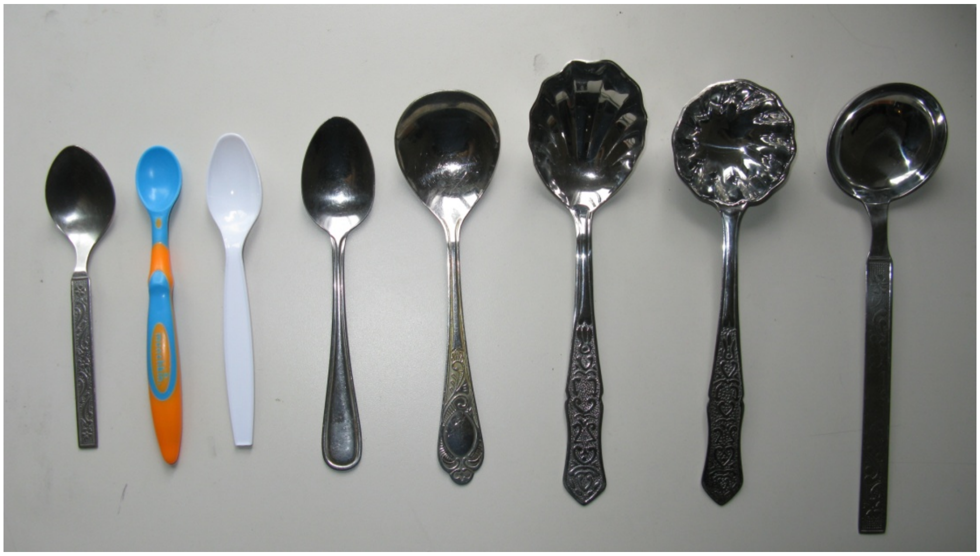
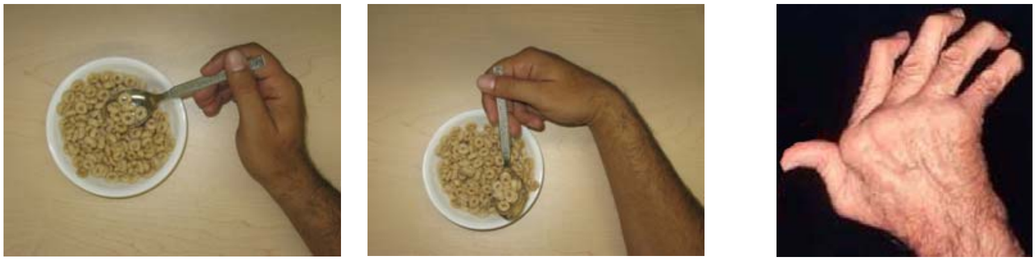


Figure 7.2 Photograph of typical commercially available spoons. Product designs mainly focus on contemporary style, ease of use, and comfort of the user.

These designs presume that the intended users can efficiently work with their hands and fingers. However, for different sets of users, these designs may not be suitable from an ergonomic point of view in terms of *ease-of-use* and providing comfortable grip. One of the user segments is of the patients suffering from rheumatoid arthritis. Rheumatoid arthritis is a chronic inflammatory disorder that most typically affects the small joints in hands and feet [The, Arthritis Society, 2010]. The patients often experience lack of movement in their upper body joints. Eating is one of basic tasks that can be impaired by rheumatoid arthritis. The spoons that are commercially available may not be helpful for this user segment. Virtual reality can make it possible for an industrial designer to imitate a rheumatoid arthritis patient and develop ergonomic spoons for the users.

Figure 7.3(a-b) shows natural way of fetching and eating food. Figure 7.3(c) shows the hand of a patient suffering from rheumatoid arthritis. This limits the movement of fingers and wrist. It is generally accepted that a user would hold a spoon in particular way and turn wrist to eat as shown in Figure 7.3(a-b). However, this may not be possible for different users such as patients suffering from rheumatoid arthritis, children, and aged people. Due to a weak grip (as in case of children and aged people) or restricted motion (as in case of patients suffering from rheumatoid arthritis), it possible that a user cannot put the food properly in the spoon or can get the food in the spoon, but cannot turn his/her wrist enough to bring it to mouth without spilling it. Even for other users, it is possible to come up with different concepts to develop ergonomic spoons.

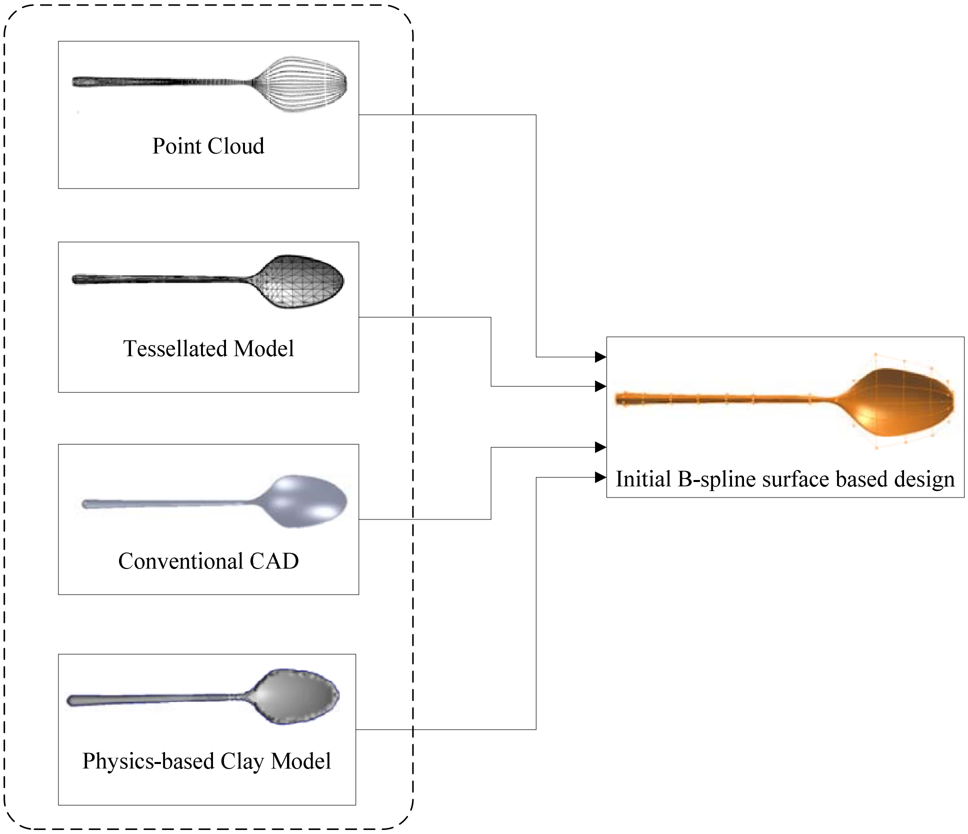


(a) (b) (c)

Figure 7.3 Generally accepted motions (a) Holding a spoon and (b) Rotating wrist while eating (c) Rheumatoid arthritis restricts movements of fingers and wrist, modified picture from [Joint, Pain Solutions, 2010].

In a similar fashion, different materials can be incorporated in product design to reduce overall weight and enhance features such as high friction gripping surface to prevent slippage of spoon during use. In this context, a variety of concepts that cannot be addressed effectively using conventional CAD design packages may be examined using interactive design simulation tools in virtual reality environment. The haptic technologies associated with virtual reality based design can also enable the designer to alternate concepts depending upon the range of motion and examine the effectiveness of the concept while holding and manipulating the proposed solution.

An initial design can be obtained by digitizing an existing design using reverse engineering approach. Point cloud data or tessellated surface can be used to obtain an initial design represented as a B-spline surface. Conventional CAD system can also be used to create the initial form. Alternatively, the initial design can be created from a virtual *lump of clay*. Figure 7.4 shows the generation of initial design by different methods.



(a) User defined model

(b) B-spline-based virtual model

Figure 7.4 Different methods to generate initial design represented as a B-spline surface.

Initial analysis can be carried out in VR space to determine if the product meets the requirements for all/intended user segments or if it needs modifications for particular segments of users. The industrial designer can investigate different shapes, sizes, and material of the spoon to determine a valid concept before finally deciding on the best design for intended user segments.

In terms of function, a spoon is primarily used for serving or eating food, although, these spoons can be further classified based on the basis of drink or food with which they are most often used. Based on the function, two different designs are considered in this thesis, a spoon for eating food and a spoon for serving food.

A spoon has two distinct parts, a handle for holding and a small shallow bowl (or shell), oval or round, at the end of the handle for fetching food. Figure 7.5 shows two parts of a spoon.

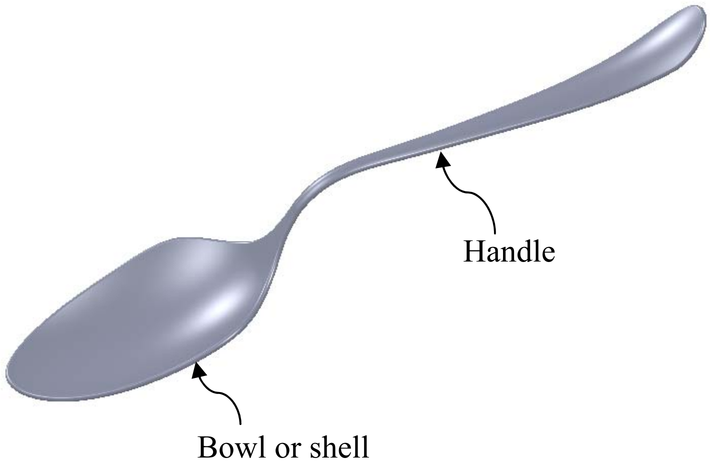


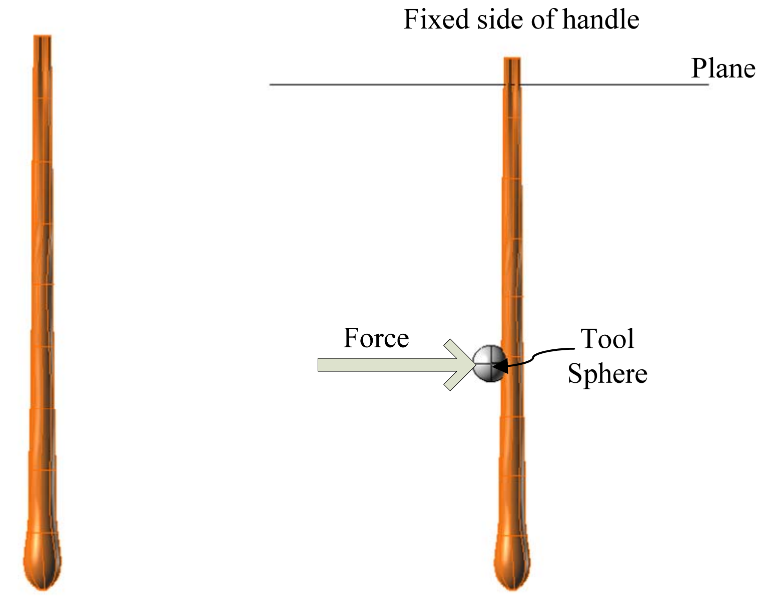
Figure 7.5 Parts of a spoon.

A spoon can be modified as a whole or the two parts (handle and bowl) can be designed separately and later merged to get the desired shape. Separate parts are easier to manipulate and can give rise to more concept models by different combinations.

7.3.1.1 Spoon for eating food

The handle of a spoon can be modified in different shapes and sizes to accommodate impaired wrist movements or weak grip of fingers. To reduce the computational cost, the bowl (shell) part and handle are used as separate B-spline models. One of the solutions is to bend the handle so that the bowl part of the spoon faces the person and he/she does not need to bend the wrist to eat from the spoon.

Using the interactive design frame work, bending can be achieved by fixing one end of the spoon handle and applying force on the other part. Figure 7.6(a) shows the handle of initial spoon design. In the design framework, a plane is used to fix a portion of handle and the force is applied by a sphere based tool. Figure 7.6(b) shows the vertical plane and the sphere used to push the handle.

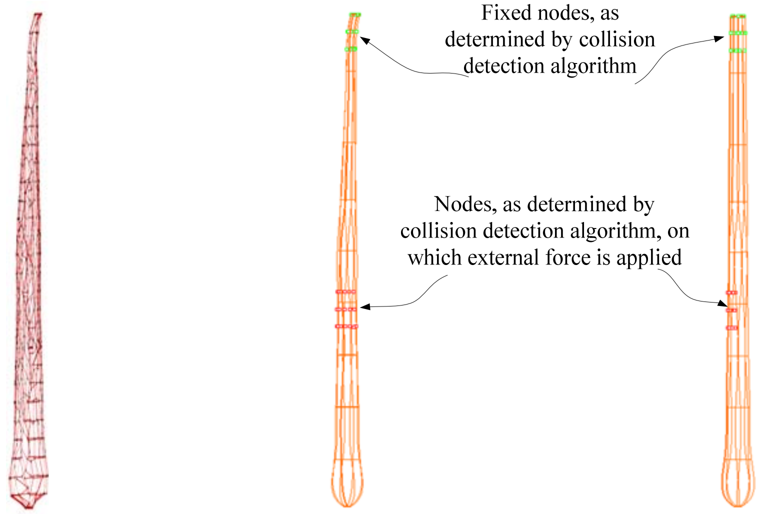


(a) B-spline model of the spoon handle

(b) Using plane to fix nodes at one side and a tool sphere to apply force

Figure 7.6 (a) Handle of spoon from the initial design (b) A plane is used to fix nodes on one side of the plane (opposite to the direction of normal vector) and a sphere is used to apply force.

During the pre-processing phase, a mass spring mesh is created for the handle shown in Figure 7.6(a) (Please refer to Chapter 5 for detail). A 12×12×12 node mesh was created, from the points generated by using blending matrices. Figure 7.7(a) shows the mass spring mesh of the spoon handle.


Fixed nodes, as determined by collision detection algorithm
Nodes, as determined by collision detection algorithm, on which external force is applied

(a) Mass spring mesh of the handle

(b) Fixed and colliding nodes, determined by the collision detection algorithm

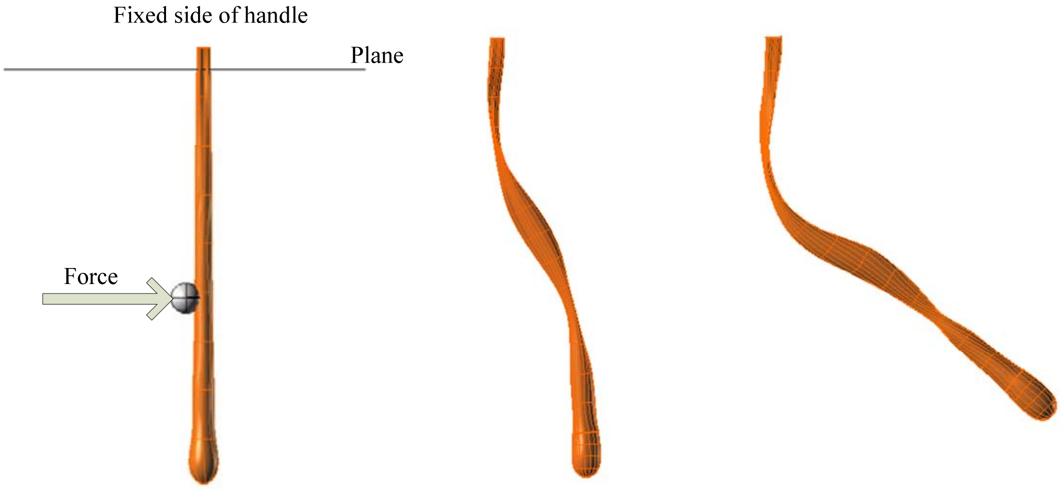
(c) Side view of the fixed and colliding nodes

Figure 7.7 (a) Mass spring mesh for the spoon handle (b) Fixed nodes (green) and the nodes colliding with the sphere tool (red) (c) Side view the B-spline model showing fixed and colliding nodes.

Collision detection algorithm, developed in this thesis, checks the intersection of the plane with the spoon handle (Please refer to Chapter 3 for detail). All the nodes on one side of the plane (opposite side of the normal vector of plane) are fixed. The fixed nodes do not experience any movement or deformation. Even if these nodes experience internal or external force through spring damper system, these nodes do not move. The nodes (green color) shown in Figure 7.7(b-c), are the fixed nodes detected by the collision detection algorithm.

At the same time, the collision detection algorithm checks the intersection of the tool sphere and the B-spline model. It first determines the intersection of sphere and the convex hull of the spoon handle. Points are generated within the minimum and maximum range of *u* and *v* parameters, determined from the intersection of the tool sphere and the convex hull. Spheres are generated using these points and intersection test is carried out between the tool spheres and the spheres created on the B-spline surface. More points are generated within the intersecting spheres at the lower level of detail. This process is carried out until the collision detection algorithm determines the region of the B-spline model colliding with the tool sphere and determines the nodes of the mass spring system which will be experiencing external force. The process of collision detection was explained in Section 3.3. The information about the magnitude of the external force applied through the sphere and the nodes experiencing this force is transmitted the mass spring deformation system. Figure 7.7(b-c) shows the nodes (red color), determined by the collision detection algorithm, which will experience external force.

As soon as a force is applied through the sphere, the B-spline model starts deforming. The upper part of the handle shown in Figure 7.7(b-c) has fixed nodes and therefore, cannot move under the influence of applied force. However, the bottom side of handle is free to move. Figure 7.8 (a) shows the initial design of the spoon handle, experiencing the external force through the tool sphere. Figure 7.8(b) shows an intermediate shape of the handle after a force was applied. Initially the middle portion of the handle was deformed and the free end of the spoon lagged behind the middle portion. However, as the number of iterations increased, the middle and bottom portions got straightened up. Figure 7.8(c) shows the bending of the handle under application of force.



(a) Initial shape of the model

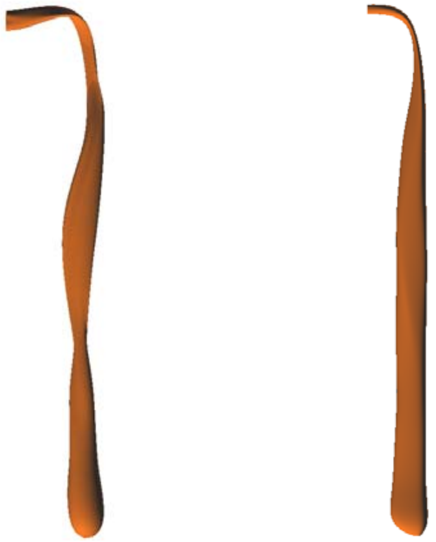
(b) Intermediate shape of the model. The middle part deforms first while the bottom part lags behind

(c) After a number of iterations, the bottom part moves enough to straighten up

Figure 7.8 Deformation of handle under the application of external force with one end fixed.

It should be noted that the lower end of the handle will remain straight only if global deformation is allowed. With higher number of iterations the free side of the handle gets enough time to straighten up. In case of local deformation, the portion where force is applied will deform but this deformation will not get enough time to propagate throughout the mass spring mesh. Due to global deformation, the free end of the handle moves enough to keep the bottom side of the handle straight.

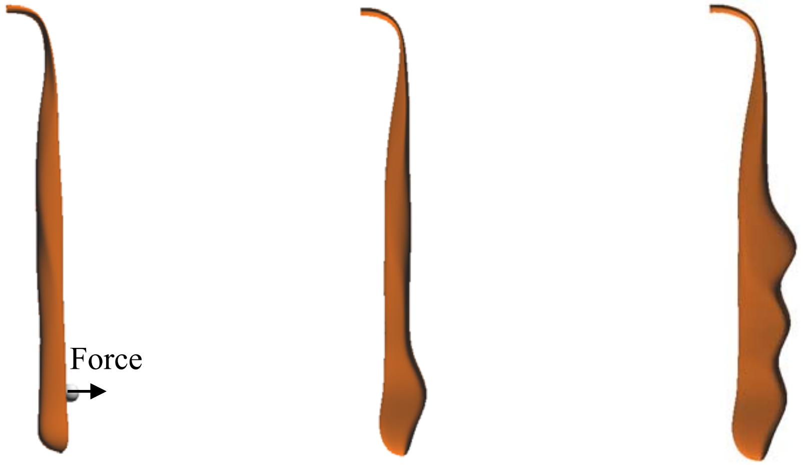
By applying forces at different points, the shape is further changed as shown in Figure 7.9(a). Once the initial spoon handle shape is modified using the tools developed in this thesis, the B-spline model is imported in commercially available CAD software (Rhino®) to refine and add finer details to the model. The refined model for the handle of the spoon is shown in Figure 7.9(b).



(a) (b)

Figure 7.9 Modified shape of spoon to accommodate lack of wrist movement.

It is possible that the user cannot use fingers to hold the spoon due to lack of movement of fingers, or weak grip. If the user cannot hold the spoon with fingers, then the handle can be modified to for a firm grip using all the fingers. This will also help to effectively counter the torque resulting from the modified design. Again, the tools developed in this thesis can be used to change the shape of the handle.

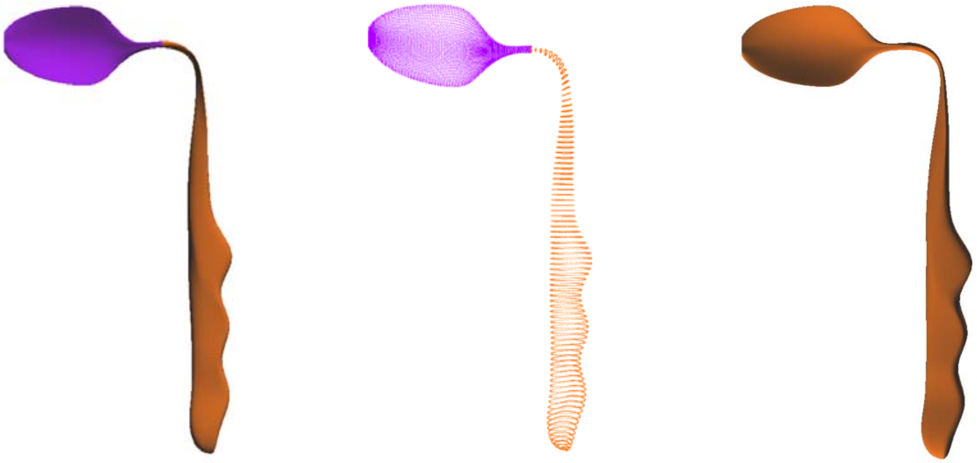


(a) Handle of the spoon interacting with too sphere (b) Deformation due to applied force (c) Modified shape of handle after several interactions

Figure 7.10 Modifying the shape of spoon handle to grip it with all the fingers of hand.

Figure 7.10(a) shows the spoon handle a tool sphere. Again, the collision detection algorithm determines the colliding nodes of the mass spring mesh, which would experience the external force applied through this sphere. As soon as the tool sphere is pulled out, the nodes of the mass spring mesh colliding with the tool sphere (determined by the collision detection algorithm) experience force in the outward direction. The external force acting on the colliding nodes starts pulling these mass spring nodes in the direction of the external force. In this case, local modification is more appropriate because we only intend to change the shape in a small region. Local modification uses small number of iterations and hence, the nodes which are away from the colliding surface do not get enough time to move under the application of the external force. Due to this reason, there is no need to have a plane to fix some of the nodes. Figure 7.10(b) shows the deformation of the handle under the influence of external force applied through the sphere tool. By repeating this process at different points, the handle can be modified to accommodate all the four fingers to grab the handle while eating. Figure 7.10(c) shows the spoon the with modified handle shape.

Once various shapes of the spoon handle are generated, these handles need to be merged with the bowl section of the spoon. The B-spline surface patches merging algorithm can be used to merge the bowl and handle part to generate various models (Please refer to Chapter 4 for detail).



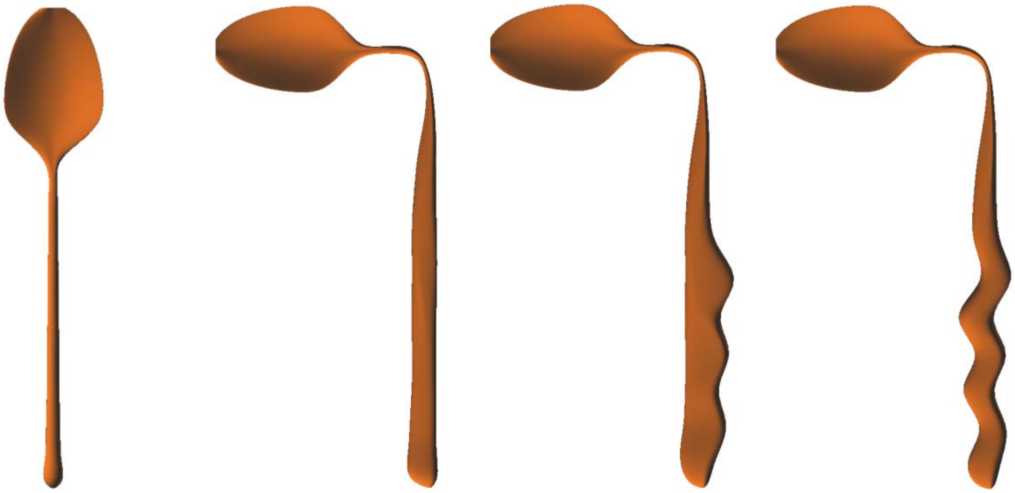
(a) Bowl and handle of spoon (b) Point cloud of the bowl and handle of the spoon (c) Merged B-spline model of spoon

Figure 7.11 Merging bowl and modified spoon handle to generate the B-spline model of a of spoon.

Figure 7.11(a) shows the bowl and the modified spoon model. These parts must be merged to generate an integrated B-spline model of the design concept for a spoon. Due to the modification of the shape of the handle, the knot vector and the common edge of the bowl and handle did not match. Rhino could not merge these B-spline models to generate a single B-spline surface patch. Even when the edges were matched (establishing tangency at the common edge without merging) Rhino did not allow the merging of the surface for being too far apart. The main reason was that when the knot vectors are different, even after matching of the surfaces, certain gaps remain at the common edges. However, the B-spline merging algorithm can merge these surfaces even without matching or manipulating the control points of the B-spline surfaces. Figure 7.11(b) shows the point cloud of the bowl and handle, generated by using blending matrices stored during the pre-processing phase. These point clouds are merged to generate single matrix of the point clouds, **M**. Revised number of control points, knot vector are calculated as discussed in Chapter 4. Using the matrix of points **M**, revised knot vector, and revised number of control points new blending matrices are generated.

These revised blending matrices are further sued to generate a merged, single patch B-spline surface is of the spoon as shown in Figure 7.11(c).

This design can accommodate the lack of wrist movement and the user does not have to rotate his/her wrist. However, due to the bend, the user will experience a torque. Figure 7.12(a) shows the initial design and Figure 7.12 (b-d) show different variations of the design. The design shown in Figure 7.12 (b) can accommodate lack of wrist movement of the user. The designs shown in Figure 7.12 (c-d) provide better grip for different sets of users. These grips can be further modified to suit grip of the user.



(a) (b) (c) (d)

Figure 7.12 Investigation of different shapes of a spoon for eating food to accommodate impaired wrist movements or weak grip of fingers. The original design is shown in (a) and design modifications are presented from (b) to (d).

These designs can be evaluated by an industrial designer or a user in the virtual reality environment. The evaluation of the model and user training with a finalized model is presented in Section 7.4.

7.3.1.2 Spoon for serving food

A serving spoon, in general, would have deeper shell and longer handle. A general design, as shown in Figure 7.13(a), consists of a straight handle and a bowl. While serving from a deep container, the user must raise his/her arm to fetch food from the container. Restricted movement of the arm (or short arm in case of children) may make it harder to serve food. Hence, the design must be modified to accommodate this user group. Figure 7.13(b) shows a design which can be used to fetch food without raising arm. The handle of the spoon was bent in manner as discussed in previous section. The handle allows the user to fetch food from a deep container without raising the arm. However, it would take more torque to pour food in plate due to the bent handle. To accommodate it, one side of the bowl part of the spoon can be dipped to allow easy serving of the food. Figure 7.13(c) shows this variation which makes it easy to pour food into the plate. These modifications can be carried out using the methodologies developed in this thesis.

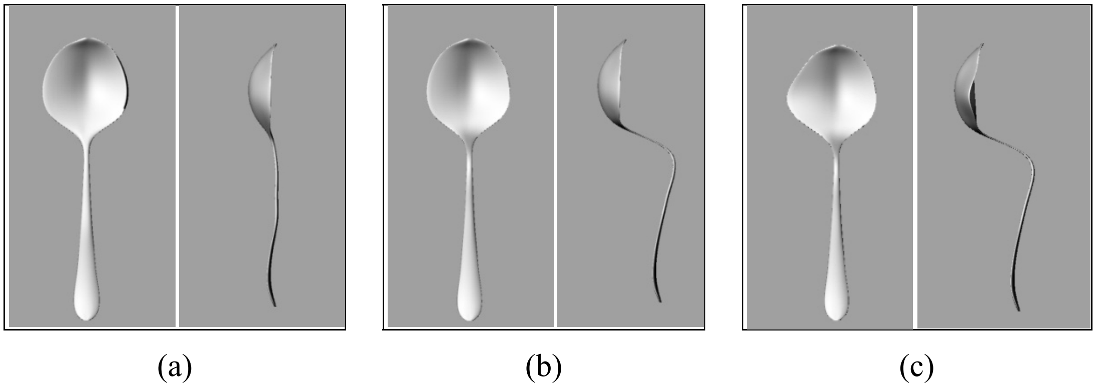

(a) (b) (c)

Figure 7.13 Investigation of different shapes of a serving spoon to accommodate impaired shoulder movements. The original design is shown in (a) and design modifications are presented in (b) and (c).

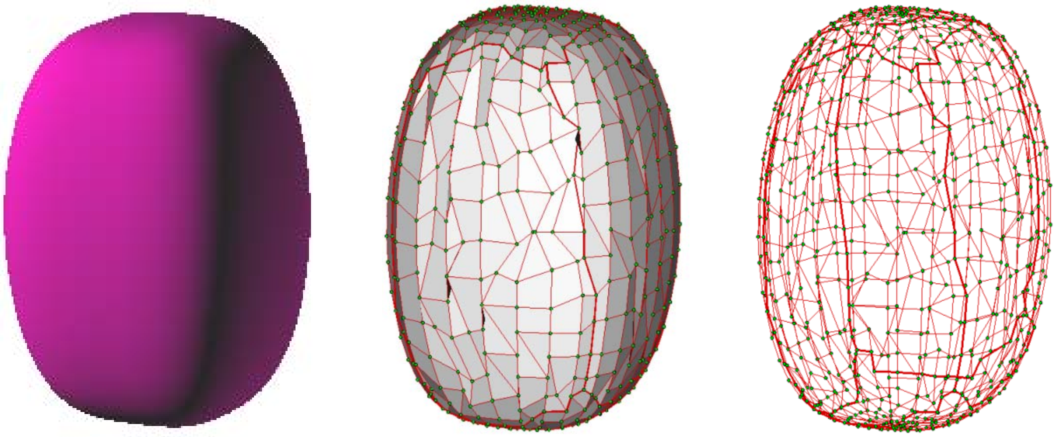
7.4 Evaluation of Model and User Training

Once an industrial designer comes up with different concepts, these can be evaluated before moving on to the next step of detailed design. The models can be evaluated in relation to its ergonomic shape and size, suitability for intended user group, strength and weight. Once the design is finalized for form and function, it can be used to impart training to the user group.

7.4.1 Evaluation of model

Considering the group of users affected by rheumatoid arthritis, aged people, and children, it will be pertinent to know if these users can eat with the spoon without spilling the food. The collision detection algorithm, developed in this thesis is capable of handling collision of two or more deformable B-spline surfaces.

A B-spline surface patch having 8×8 control points net was used to represent food (jelly) for interaction with the spoon to mimic eating with a spoon. A 12×12×12 mass spring damper mesh was used to incorporate material properties of jelly to this model. Figure 7.14(a) shows the B-spline model and Figure 7.14 (b-c) show the mass spring mesh of this model.



(a) B-spline model

(b) Mass spring model shown with facets. The nodes are shown as green dots

(c) Mass spring model. The thick red lines represent the boundries of hexahedron mesh

Figure 7.14 (a) B-spline model representing food (jelly) (b-c) Mass spring mesh to incorporate material properties to the model.

The methodologies developed in this thesis, make it possible to model food (jelly) as a B-spline deformable model and simulate an environment in which a user can interact with food and spoon represented as B-spline surfaces. Figure 7.15(a) shows a spoon with a jelly on it, while the spoon was kept straight. Again, the collision detection algorithm detects the regions of the B-spline models colliding in the virtual reality environment.

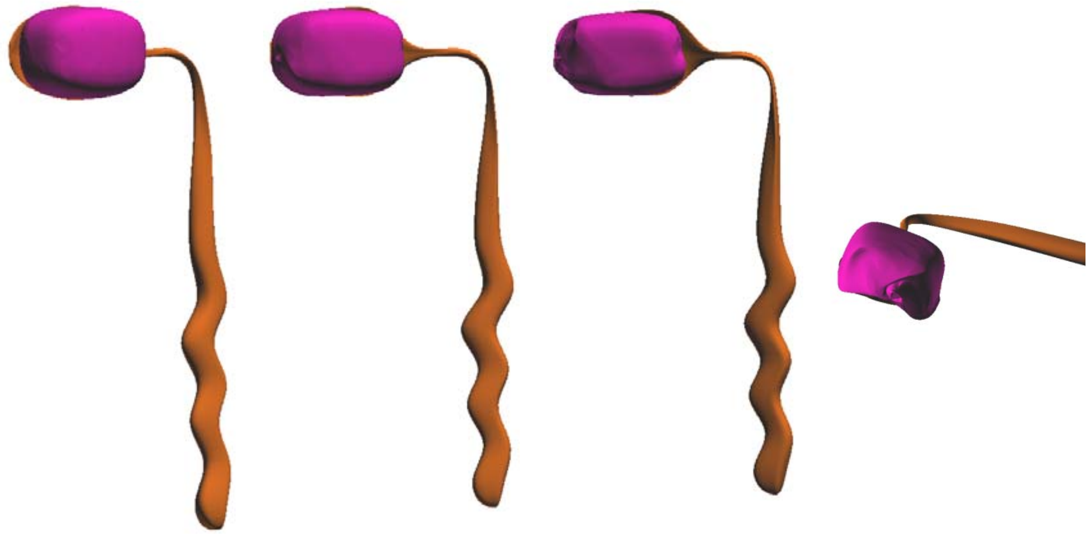
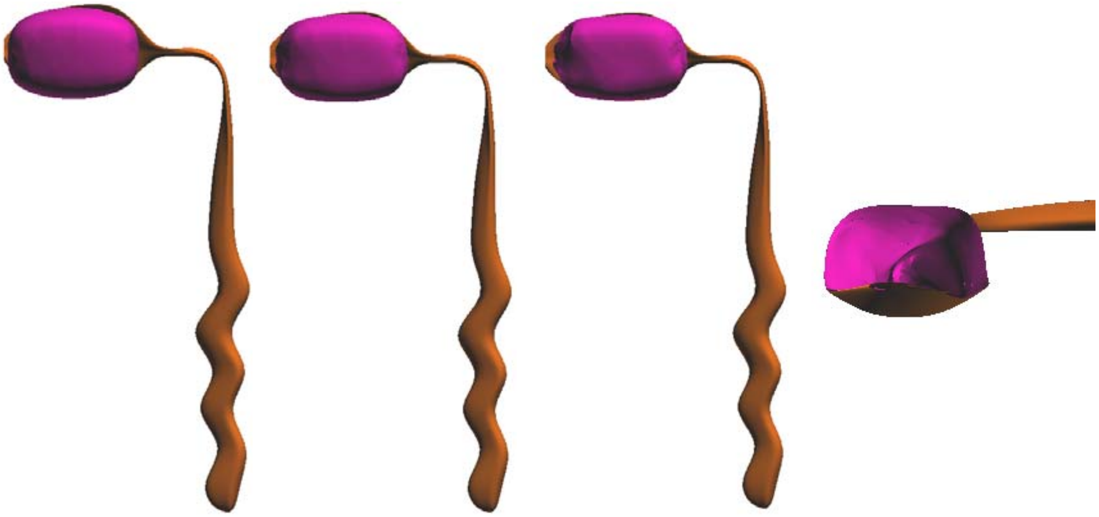

(a) Jelly on spoon (b) Jelly starts flowing under its own weight (c) Jell stops flowing further due to its own weight (d) Other view of the jelly flowing due to its own weight

Figure 7.15 Simulation of food (jelly) in a spoon without tilting it.

Due to its own weight, jelly started flowing downwards as shown in Figure 7.15(b). As soon as jelly deforms, the control points net of this B-spline model gets updated to represent deformed jelly. By using the blending matrices and the revised matrix of control points, points are once again generated on jelly. Spheres are generated on the jelly surface using these points. Same process is carried out for the spoon. The spheres generated on jelly and spoon are checked for intersection and more points and subsequently spheres are generated at the lower levels of detail. At the lowest level, the collision detection algorithm determines the nodes which will experience external force due to the collision. Thus the nodes of the mass spring mesh of jelly, which collide with spoon experience reactive force and do not move downwards. However, other nodes continue to move and their movements are determined by the mass spring mesh and the physical properties assigned to it. The control points of the B-spline models are updated and this process continues. As the mass spring model of spoon has more stiffness (given properties of steel), there is no noticeable change in the shape of spoon. However, if different material is chosen, the shape of spoon may also change.

As shown in Figure 7.15(b-d), jelly starts spilling out of the spoon. This means that if this design is used, the user cannot eat jelly by using this size without spilling it. Figure 7.15(d) shows the close up of the bowl of spoon. It is clear that the jelly is spilling from the front portion of the spoon bowl. At this point, an industrial designer can start modifying the design in such a way that the jelly does not get spilled. It is clear from Figure 7.15 (b-d) that the jelly was spilling from the front portion of the spoon bowl while the back portion was empty. The design of the spoon can be modified to see how the jelly will behave, if the bowl is tilted about 10 degree in the backward (clockwise for the spoon shown in Figure 7.15, when seen from the handle side along the handle) direction. Figure 7.16(a) shows the jelly put in a spoon with tilted bowl. The jelly starts flowing due to its own weight as shown in Figure 7.16(a-d). However, this time it does not spill out of the spoon as shown in Figure 7.16(d). It spreads in the spoon evenly. This shows that there was improvement in the design.



(a) Jelly on spoon

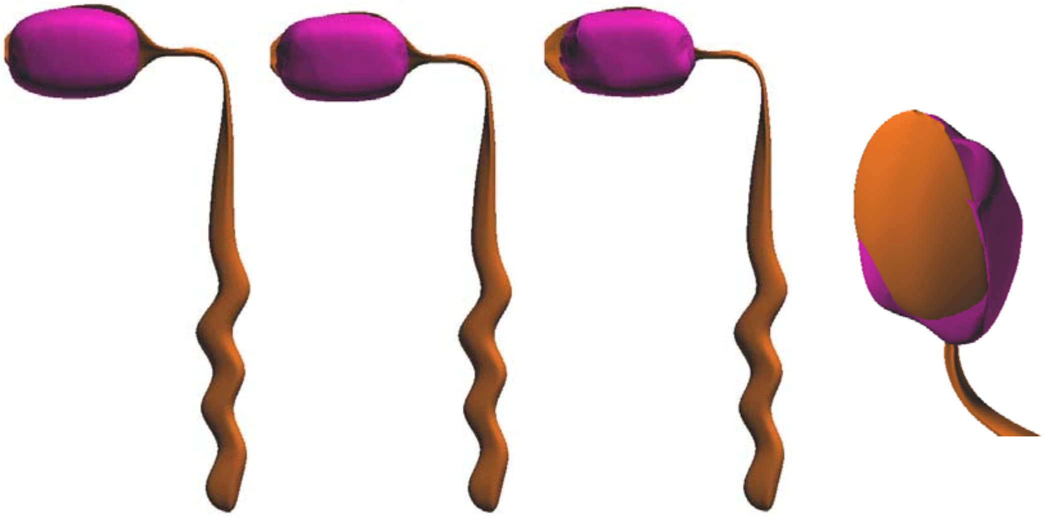
(b) Jelly starts flowing under its own weight

(c) Jell stops flowing further due to its own weight

(d) Other view of the spoon. Jelly does not spill out of spoon

Figure 7.16 Simulation of food (jelly) in a spoon when the bowl is tilted by ten degree.

The bowl was tilted by 20 degree to see if it further improves the design. However, as shown in Figure 7.17, the jelly started flowing from the back part of the spoon. This means that by tilting the bowl by twenty degrees, the design deteriorated rather than improving its functionality.



(a) Jelly on spoon

(b) Jelly starts flowing under its own weight

(c) Jell stops flowing further due to its own weight

(d) Bottom view of the spoon. Jelly spills out of spoon

Figure 7.17 Simulation of food (jelly) in a spoon when the bowl is tilted by twenty degree.

The simulation suggests that the spoon bowl should be tilted by 10 degree to improve the design. In a similar fashion, other parameters of the spoon such as depth of the bowl, curvature of the bowl, or the width of the bowl, can be varied to determine best suitable design.

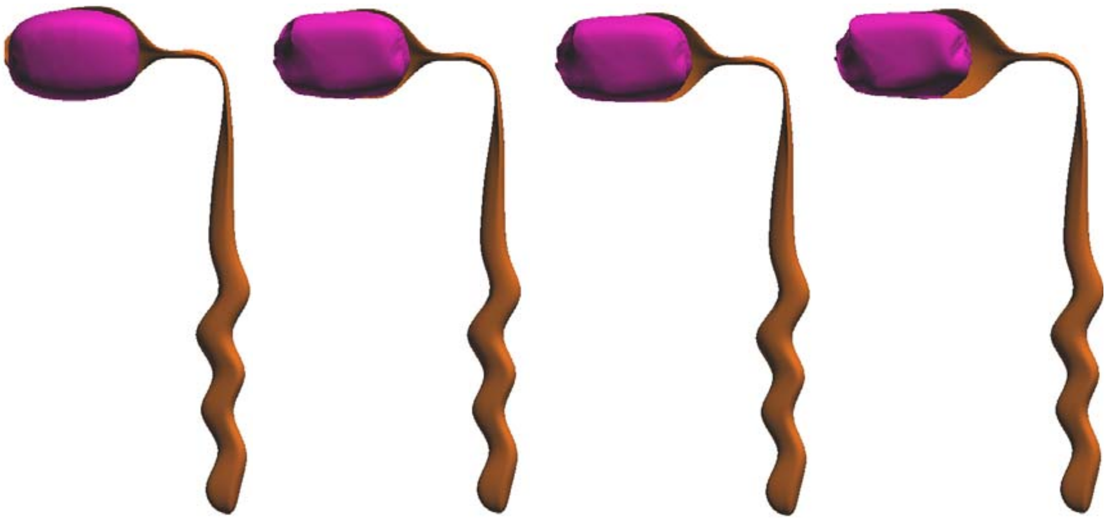
The industrial designer can also determine weight of the spoon, the deflection of the spoon due to its own weight and that of the food, while using different materials for the spoon. The preliminary evaluation of the concepts in the virtual reality environment would help the industrial designer to come up with the designs which can be successfully implemented. The tools developed in this thesis, can provide this environment to the industrial designer.

7.4.2 User training

Even when a product design for a given user segment is ready, the users may require training to benefit from the unique features of the design. Even when a variety of shapes is available for a spoon, adaptations needed for eating may still be overwhelming for patients suffering from rheumatoid arthritis or aged people having weak grips. An occupational therapist can help train these people. However, this would require that products are readily available in the market. In the absence of a suitable product, it might be difficult for an occupational therapist to train this user segment. Virtual reality environment can be used to efficiently train these users. Various shapes discussed in Section 7.3 can be used to assess how a user eats his/her food and determine which design will work for a particular user or a set of users. At the same time, an industrial designer can have better understanding of the difficulties of the user group. This can help the industrial designer to come up with better design after receiving valuable inputs from the users.

A major challenge for patients of rheumatoid arthritis or aged people is to eat food with a spoon without spilling it. There are primarily two reasons for spilling food from the spoon; tilting of spoon and shaking of hands while eating with a spoon. Virtual reality environment can provide various scenarios in which a user can interact with spoon while eating food. A variety of spoons developed during the interactive design phase can be used to determine the *best fit* for the user.

In the simulation study, different scenarios were considered, which included different angle of tilt for the spoon and shaking of hands. Acceleration was imparted to spoon to simulate shaking of hands. When the spoon gets tilted, jelly may start spilling out of the spoon. A larger tilt will increase the rate of spilling of jelly. However, by practicing with the virtual spoon, a user can be trained to eat without spilling food. Figure 7.18(a) shows the interaction of spoon and food to simulate a user tilting his/her spoon while eating food. Different angle of tilt were considered which resulted in spilling of jelly by different magnitudes. Figure 7.18(b) shows spilling of jelly when the spoon was tilted by ten degrees. The spilling of jelly increased with increased angle of tilt as shown by Figure 7.18(b-d). Figure 7.18(e-h) show the side view of the spoon and jelly.

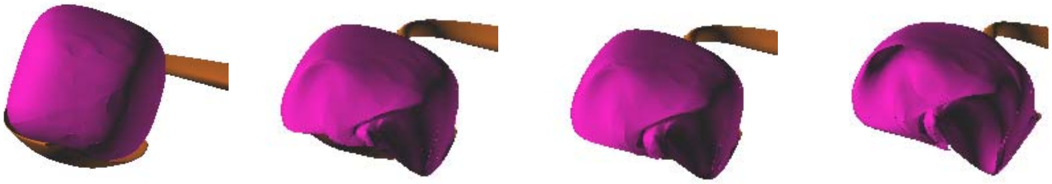


(a) Jelly on spoon

(b) Jelly spills out of spoon when spoon is tilted by 10 degrees

(c) Jelly spills out of spoon when spoon is tilted by 20 degrees

(d) Jelly spills out of spoon when spoon is tilted by 30 degrees



(e) Side veiw of the jelly on spoon

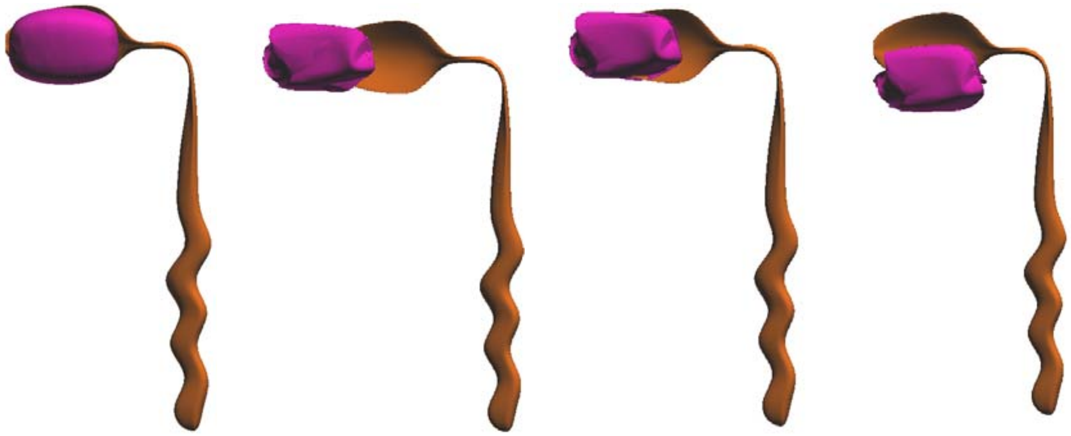
(f) Side view of the spilling jelly, when the spoon tilted by 10 degree

(g) Side view of the spilling jelly, when the spoon tilted by 20 degree

(h) Side view of the spilling jelly, when the spoon tilted by 30 degree

Figure 7.18 Simulation of food (jelly) in a spoon when the spoon is tilted by different degrees.

The spilling of spoon gets aggravated when the tilting of spoon is accompanied by the shaking of hands while eating with a spoon. By incorporating the acceleration to the spoon and eventually to the food, both the tilting of spoon as well as shaking of hands can be simulated. Figure 7.19(a) shows the jelly on the spoon bowl. Its side view is shown in Figure 7.19(e). If the user tilts the spoon (anti-clockwise rotation of the spoon when seen from the side of the handle of the spoon) and his/her hands are shaking in the same direction, the jelly will experience force due to its own weight as well as due to the inertial forces. Figure 7.19(b and f) show the result of simulation. It is clear from the figure that shaking of hand exacerbates the slipping of jelly from the spoon. However, when the shaking of hands happened in the transverse direction, the jelly slipped to lesser extent as shown in Figure 7.19(c and g).

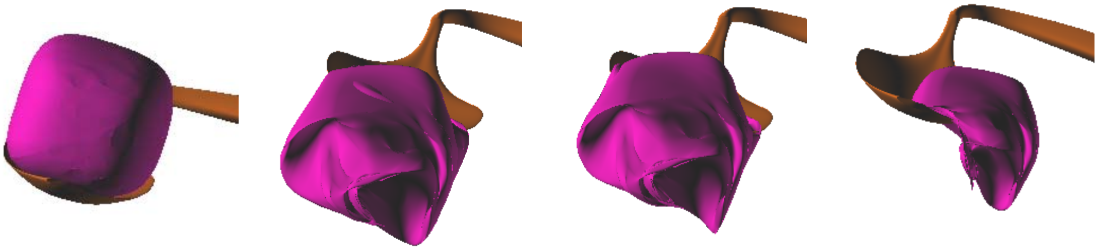


(a) Jelly on spoon

(b) Jelly spills out of spoon when spoon is tilted by 30 degrees and the user's hand is shaking in the same direction

(c) Jelly spills out of spoon when spoon is tilted by 30 degrees and the user's hand is shaking in the transverse direction

(d) Jelly spills out of spoon when spoon is tilted by 30 degrees sideways and the user's hand is shaking in the same direction



(e) Side veiw of the jelly on the spoon

(f) Side view of the spilling jelly

(g) Side view of the spilling jelly

(h) Side view of the spilling jelly

Figure 7.19 Simulation of food (jelly) in a spoon when the spoon is tilted by different degrees and the hand of the user is shaking.

In the same way, when the tilting and the shaking of hand happens in transverse direction (along the major axis of the handle of the spoon), the jelly slips and starts falling down from the side. This is shown in Figure 7.19(d and h). At this point if an industrial designer concludes that the user cannot eat without the tiling and shaking of hands, the spoon design can be reviewed. Some modification such as raising one side of the spoon bowl or tilting the bowl part of the spoon in the other direction can be carried out at this stage.

7.5 Concluding Remarks

Various techniques developed in this thesis can be used to simulate various scenarios in virtual reality environment. The collision detection algorithm allows both rigid implicit surfaces and deformable B-spline surface-based tools. The B-spline surface merging algorithm allows the users to merge different deformable models to generate a large number of shapes.

The simulations can provide an industrial designer, an enhanced insight into the form and function of the concept required for a given group of users. A variety of models can be generated and tested by the industrial engineer and real-time information can be used to improve their form and function. At the same time, the interactive design framework can be used to provide training to the users in the virtual reality environment. All these processes can be carried out without physically fabricating various prototypes, thereby, reducing the wastage caused by scraped models and prototypes which were fabricated at the preliminary stage.

The simulations proved the effectiveness of the methodologies developed in this thesis to provide an intuitive environment to an industrial designer or engineer. The manipulation of deformable handle to change its shape and merging of spoon handle with bowl was accomplished in real-time. The simulation of spoon and jelly was performed at a higher rate than 30 Hz.

CHAPTER 8 CONCLUSIONS AND RECOMMENDATIONS

8.1 Review of Methodologies Developed for Deformable Modelling

In this thesis, the development of methodologies for enabling seamless interaction of deformable virtual models was presented. The main focus was to develop a framework for efficient collision detection, merging of B-spline surfaces representing the deformable virtual objects, and modeling the shape of the virtual objects based on physical properties. The application for illustrating the proposed methodology was virtual interactive design. However, the core algorithms are versatile for use in medical simulations, games and other haptic interactive applications. It was assumed that all the deformable models were represented as B-spline surfaces. Although, the mass damper spring mesh had been used in thesis, other geometric or physically based deformation models can also be used in conjunction with the collision detection and B-spline surface merging algorithms.

All the components of the framework for interaction with deformable models used pre-computed blending matrices. These blending matrices are independent of the control points of the B-spline surface and hence, can be pre-computed. Once computed, these blending matrices could be used to find the new position of control points to represent deformed B-spline surface without calculating blending functions. As there was no need to calculate computationally intensive blending functions, various algorithms could work efficiently. In fact these blending matrices enabled the algorithm to efficiently merge B-spline surface patches, accurately check the collision, and generate nodes for the mass spring system to determine deformation using the physics-based model.

The collision detection algorithm was capable of handling intersection test of virtual models with haptic tools. These haptic tools were represented as a point, an implicit surface, a tessellated surface, and B-spline surface. No restriction was imposed on the number of control points representing a B-spline surface, degree of the B-spline surface, knot vectors, or extent of deformation. The B-spline surfaces can be merged without imposing restrictions which are generally imposed by commercially available software. The algorithm can merge surfaces which are intersecting, trimmed or do not have common edge at which these are joined. Most of the commercially available software cannot tackle these cases. Material properties like Young's modulus and Poisson ratio were incorporated into the model while generating mass spring system by volumetric self organizing feature maps.

8.2 Novel Features of the Proposed Method

In order to demonstrate the novel features and computation efficiency of the collision detection algorithm, B-spline surface patches algorithm, mass spring system, and integrated interactive design module, different simulations were performed. Deformable model was represented as B-spline surface for easy exchange with commercially available CAD software. Merging of B-spline surface patches algorithm provided an efficient and robust framework for integrating B-spline surface models to generate more complex and interesting designs. The mass spring system allowed simulating real material behaviour for the model depending upon the physical properties assigned to it by the user. All the algorithms used common resources (blending matrices) for efficient use of the pre-computed information and worked in tandem.

8.2.1 Blending matrices

B-spline representation is one of the main methods for free-form surface modeling and has become the standard for CAD systems. However, the high computational cost of continuously computing the blending functions for merging, collision detection and physics-based deformation system, while the model is deforming, restricts the use of B-spline representation in a Virtual Reality (VR) environment. In this thesis an alternative method to represent B-spline surface patches had been presented for an interactive VR environment.

A uniformly discretized B-spline surface patch can be represented by a set of control points and two pre-calculated B-spline blending matrices. The proposed technique exploited the fact that these B-spline blending matrices were independent of the position of control points and therefore could be pre-calculated. The blending matrices enabled the algorithm to merge B-spline surface patches, accurately check the collision, and generate nodes for the mass spring system to determine deformation using the physics-based model. This technique does away with the need to calculate computationally intensive blending functions for the B-spline surfaces, and inverse of large matrices during the run-time. The computational efficiency achieved by using blending matrices helped to achieve real time interactions between the virtual model and the tool, in a virtual reality environment.

An essential aspect in B-spline surface modeling is the conversion between different representations of B-spline surfaces. These arise in various aspects of B-spline surface modeling/manipulation such as shape control, degree control, and merging of B-spline surface patches. Blending matrices can be used to establish a uniform mathematical model for all aspects of B-spline surface modeling/manipulation needed in a virtual interactive design process and provide a general tool for the conversions between different representations of B-spline surfaces.

Once these blending matrices are calculated and stored, they can be used for a variety of applications in a VR environment. Two or more B-spline patches can be merged in a VR environment. The same blending matrices can be used for efficient collision detection as well as generating nodes for the mass spring system. In this manner, all the aspects of B-spline manipulation work in tandem and reduce the computational cost without affecting the accuracy of various interactions. For all these applications, the blending matrices played a significant role in making the whole process computationally efficient.

8.2.2 Collision detection algorithm

The collision detection algorithm utilized the best qualities of parametric representation for free form surfaces and the ease and efficiency of triangle-triangle intersection test. The density of points generated on the surface was increased at lower levels of detail within the area of probable collision. The *on-the-fly* generation of points and triangles also helped maintaining the *quality* of the triangles. Although the algorithm used triangle-triangle intersection test for collision detection at the lowest level of detail, it was more efficient than a tessellated surface deformable model. The novel method of generating spheres to find out the regions of the surface likely to collide, allowed multiple contact collision detection. The algorithm is also capable of detecting collision with a tessellated surface, implicit surface, or point-based tool. Hence, a variety of rigid and deformable tools could be used during sculpting or validation of the model. No limitations were imposed on the shape, complexity, degree or the number of control points of the B-spline surface representing the tool or model. Both the model and the tool could have complex shapes, elastic or plastic properties, and multiple contacts. This would allow the user to use rigid or deformable tools with complex shapes with greater ease and productivity during the sculpting exercise or model validation within a virtual reality environment.

The novel technique of comparing the normals of the points generated on a flat surface reduced the computational cost of collision detection between two flat B-spline surfaces. A fewer number of points were generated for tessellation on a flat surface which further reduced the number of triangles generated for triangle-triangle intersection test. This made the algorithm robust as it could handle a potentially computationally expensive situation at a much lower computational cost. The calculation and storage of transformation matrices and their inverse during the preprocessing stage also ensured that no inverse was needed to be computed during the run-time phase of the algorithm. The matrix inverse calculations would have increased the computational cost. This reduced the computational cost of rendering the deformation of the B-spline surface and made the algorithm robust and efficient.

The collision detection algorithm can detect collision with multiple B-spline patches. However, it creates problems for the mass spring system. Hence, the B-spline surface used to represent the model or the tool was limited to single patch. If there are multiple patches, these can be combined into single B-spline patch by using the B-spline surface patches merging algorithm.

However, the collision detection algorithm, presented in this chapter, cannot detect the extent of tool penetration. The calculation of tool penetration helps to calculate the magnitude of the forces to be fed back to the user during haptic interaction. In future, this algorithm will be extended further to include the calculation of tool penetration depth.

Another limitation of the collision detection algorithm is that it cannot detect self collision.

8.2.3 B-spline surface patches merging algorithm

B-spline surfaces have been used in a VR environment but one problem, which had not been addressed so far, is efficiently creating complex shapes by combining multiple dissimilar B-spline surface patches. The algorithm presented in this thesis allows the user to combine multiple B-spline surface patches in to a single B-spline surface. This algorithm also exploited the blending matrices of the surface patches used by the collision detection algorithm, thereby, making it computationally efficient. It created new blending matrices for the merged surface, which replaced those for the original surface patches. The revised blending matrices generated during the merging process were used later by the collision detection algorithm, thereby minimizing the number of redundant calculations. In this manner, the B-spline surface patches merging algorithm worked in tandem with the collision detection algorithm and only a small number of additional computations were performed during the merging process.

A major constraint in a VR environment is that the user treats the virtual model as a surface or solid rather than a B-spline surface. For this reason, the user is not supposed to tweak the control points before merging the surfaces. This algorithm is capable of handling all the cases, which are acceptable for NURBS-based surfaces. It does not impose any restriction on the degree of the surface patches, the number of control points, the type of continuity required at the common edge, knot vector or the number of surfaces being merged simultaneously. It could efficiently merge B-spline surface patches having dissimilar curvatures at the common edge, intersecting B-spline surface patches, and the B-spline surface patches having trimmed edges. These types of patches cannot be merged by using the traditional approach used by commercially available CAD software. Overall the proposed algorithm was efficient, accurate, and robust. The surface generated by merging of two or more patches had better tolerance than that is acceptable for many VR applications. The user would not face a situation where the two surfaces were not being merged due to intersection, dissimilar curvature at common edge or similar situations. This added to the robustness of the algorithm.

8.2.4 Integration of mass spring system

In this thesis a mass spring damper system was integrated with the collision detection algorithm for real-time interactive simulation of deformable models. Both the model and the tool could have complex shapes, elastic or plastic properties, and multiple contacts. This allows the user to use rigid or deformable tools with complex shapes with greater ease and productivity during the sculpting or model validation in a virtual reality environment. However, the collision detection algorithm could work independent of a physics-based model used to calculate deformation.

The number of nodes of the mass spring system was independent of the number of points generated for collision detection. This allowed the collision detection and the haptic force response through the mass spring system to work independent of each other.

The process of mechanical interactive design is essentially moving the exterior nodes in response to the applied external forces. The interior nodes will move according to the dynamics of the mass spring structure and applied forces. Global and local deformations were possible and were achieved by changing the number of iterations used by deformation algorithm. Lower number of iteration allowed only local deformation and global deformation was achieved by increasing number of iterations. The mass spring system allowed the interactive design framework to efficiently simulate deformation of deformable models.

8.3 Other applications

Aside from interactive product design, the modeling tools developed in this thesis can be used for many other applications. Collision detection can be independently used for medical applications and games. The B-spline surface patches algorithm can be used for merging B-spline patches in commercially available software as it can tackle all the cases that are mathematically possible to merge. The overall interactive design module can be used for training in the area of surgery simulation and many cases of rehabilitation. In biomedical applications, soft tissues can be modeled as B-spline surface and assigned appropriate properties using mass spring system.

8.4 Recommendations to Resolve Limitations

The collision detection algorithm, presented in this thesis, cannot detect self collision. However, this algorithm can be integrated with a mass spring damper mesh to impart physical properties to a deformable model. The springs used in the mass spring damper mesh have minimum solid length and hence, the model would resist self intersection. Unfortunately, during high deformation, the mass spring system may not be valid and self intersection may happen.

The collision detection algorithm cannot calculate the tool penetration depth while interacting with the virtual model. This information is required to calculate the resultant force feedback to be provided to the user. This information will be necessary when this algorithm is interfaced with haptic tools. The goal of this thesis was to develop techniques which can be used for interactive design. Hence, no haptic tool was interfaced with these algorithms. However, the collision detection algorithm will need to be extended to include tool penetration depth before it is integrated with a haptic tool.

The continuity conditions for the joined multiple B-spline surface patches allow the user to represent a model having multiple surfaces. The technique used in this thesis to detect collision works efficiently with single or multiple B-spline surface patches. However, if one or more of these patches are deformed, the continuity will no longer exist. Hence, these techniques cannot be used in a VR environment when the user intends to deform the surface to obtain the desired shape. A mass spring system can be developed to integrate multiple patches which are joined rather than merged.

In the absence of a mass spring system which can integrate multiple B-spline surface patches, a B-spline surface patches merging algorithm was developed. However, NURBS surface representation imposes certain limitations on merging process and in many situations, virtual models represented as B-spline surfaces cannot be merged. As an example, the branching of a surface is not possible to achieve by partly merging two or more surfaces. In such a scenario, an enhanced mass spring system that can integrate different patches joined together by some continuity conditions can be developed. This will make it possible for the designer to come up with more shapes than that are possible while merging these patches.

8.5 Future Work

The work presented in this thesis was primarily to demonstrate the efficiency and robustness achieved by using the collision detection algorithm, B-spline surface patches merging algorithm and integrating these algorithms with mass spring damper system to develop interactive product design framework. The deformable virtual model was represented as a B-spline surface patch and the tools could be represented as a point, an implicit surface, a tessellated surface, or a B-spline surface patch. Research is underway to make these algorithms computationally more efficient to make interactive modeling more realistic and effective.

The implementation of collision detection algorithm was done in Microsoft® Visual Studio® using C++. During the implementation, more emphasis was put on making the coding readable than on making it efficient. The intersection test between spheres, to generate more points at the lower levels of details, was carried out using hierarchical "*for loop*". However, if these intersection tests are performed simultaneously, using multithreading techniques, computational cost of collision detection can be further reduced. As the intersection test of each sphere is independent of the result of intersection test of other spheres, multithreading should be performed to reduce the computational cost.

A uniform mass spring system is used in this thesis. Although more points are generated in the vicinity of collision detection to accurately check the collision of virtual objects, the number of springs in the area remains constant. By subdividing the hexahedron mass spring mesh and appropriately distributing the properties assigned to springs and dampers, denser mass spring mesh can be generated *on-the-fly* in the vicinity of collision detection. This will increase the accuracy and resolution of the deformation in the vicinity of collision detection. In this manner, a sparse uniform mass spring system can be used in conjunction with a denser mass spring system in the vicinity of collision detection. The efficiency and the effectiveness of *on-the-fly* generation of a denser mass spring mesh, to realistically deform the virtual object as per the assigned properties, should be explored.

A hexahedral mesh of mass spring was used so as to mimic B-spline surface as a solid. However, if the user would like to work with surfaces, a two dimensional mass spring system can be used. This way, a designer can manipulate surfaces and later on merge or join these surfaces to come up with various shapes. A two dimensional mass spring system should also be developed for this purpose.

Though mass spring system has been used in this thesis, other physics based deformation model can also be used in conjunction with the collision detection and B-spline surface patches merging algorithms. Different types of physics based techniques may be appropriate for a variety of applications. Hence it is recommended that the integration of other physics based techniques with the collision detection algorithm be explored and compared with the mass spring system implemented in this thesis.

8.6 Final Remarks

A preliminary design and analysis tool that supports quick conceptualization and modification of 3D geometry has been proposed in this thesis. The proposed technique provides rapid verification of early design ideas and adds more information to the interactive design paradigm. This is particularly suitable for modifying and identifying an optimal concept for a particular user segment so as to increase the chances of satisfying customers. By providing industrial designers a variety of rigid and deformable tools to quickly create, modify and analyze alternative concepts, a large number of models can be created before choosing the most appropriate model for a given user segment. The users can be trained in virtual reality environment and their inputs can also be used by an industrial designer to generate better concepts.

In conclusion, this research work helped in understanding several aspects of collision detection, manipulation and merging of B-spline surfaces, and the mass spring damper system. It has clearly established that there is merit in perusing further research in deformable model represented as B-spline surface using the tools developed in this thesis.

The research work has also established a definite need to further explore the potential of interactive design framework; developed in this thesis, by integrating it with commercially available NURBS based software.

BIBLIOGRAPHY

Anthony, L., Regli, W. C., John, J. E. and Lombeyda, S. V., 2001, " CUP: A computer-aided conceptual design environment for assembly modeling", *The ASME Journal of Computer and Information Science in Engineering*, 1(2), 186-192.

Astheimer, P., Dai, F., Felger, W., Göbel, M., Haase, H., Müller, S. and Ziegler, R., 1995, "Virtual design II – an advanced VR system for industrial applications", In: *Proceedings of Virtual Reality World '95*, Stuttgart, Germany, 337-363.

Attali, D. and Montanvert, A., 1997, "Computing and simplifying 2D and 3D semi-continuous skeletons of 2D and 3D shapes", *Computer Vision and Image Understanding*, 67(3), 261-273.

Avila, R. S. and Sobierajski, L. M., 1996, "A haptic interaction method for volume visualization", In: *Proceedings of the 7th Conference on Visualization '96*, San Francisco, California, United States, 197 - 204.

Avis, D. and Bremner, D., 1995, "How good are convex hull algorithms?", In: *Proceedings of the 11th Annual Symposium on Computational Geometry*, Vancouver, Canada, 20-28.

Avril, Q., Gouranton, V. and Arnaldi, B., 2010, "A broad phase collision detection algorithm adapted to multi-cores architectures", In: *Proceedings of Virtual Reality International Conference (VRIC 2010)*, Laval, France, 1-6.

Baerentzen, J. A., 1998, "Octree-based volume sculpting?", In: *Proceedings of the IEEE Visualization Conference (Vis98)*, NC, USA, 9-12.

Bainville, E., Chaffanjon, P. and Cinouin, P., 1995, "Computer generated visual assistance during retroperitoneoscopy ", *Computers in Biology and Medicine*, 25(2), 165-171.

Baraff, D., 1990, "Curved surfaces and coherence for non-penetrating rigid body simulation", *Computer Graphics*, 24(4), 19-28.

Barber, C. B., Dobkin, D. P. and Huhdanpaa, H., 1996, "The quickhull algorithm for convex hulls", *ACM Trans. on Mathematical Software*, 22(4), 469-483.

Barequet, G., Chazelle, B., Guibas, L., Mitchell, J. and Tal, A., 1996, "Boxtree: A hierarchical representation of surfaces in 3D", In: *Proceedings of Eurographics'96*, 387-396.

Barr, A. H., 1984, "Global and local deformations of solid primitives", *SIGGRAPH Computer Graphics*, 18(3), 21-30.

Basdogan, C., Ho, C., Srinivasan, M., Smal, S. and Dawson, S., 1998, "Force interactions in laparoscopic simulations: haptic rendering of soft tissues", In: *Proceedings of Medicine Meets Virtual Reality Conference*, San Diego, CA, USA, 385-391.

Basdogan, C., Suvranue, D., Jung, K., Muniyandi, M., Kim, H. and Srinivasan, M., 2004, "Haptics in minimally invasive surgical simulation and training", *IEEE Trans. on Computer Graphics and Applications*, 24(2), 56-64.

Bathe, K. J., 1996, *Finite element procedures*, Prentice Hall, Englewood Cliffs, NJ.

Baumann, R. and Glauser, D., 1996, "Force feedback for virtual reality based minimally invasive surgery simulator", In: *Proceedings of Medicine Meets Virtual Reality*, San Diego, CA, USA, 564-579.

Benko, P., Martin, R. R. and Varady, T., 2001, "Algorithm for reverse engineering boundary representation models", *Computer Aided Design*, 33(11), 839-851.

Berkley, J., Turkiyyah, G., Berg, D., Ganter, M. and Weghorst, S., 2004, "Real-time finite element modeling for surgery simulation: an application to virtual suturing", *IEEE Trans. on Visualization and Computer Graphics*, 10(3), 314-325.

Bloomenthal, J., 1997a, "Bulge elimination in convolution surfaces", *Computer Graphics Forum*, 16(1), 31-41.

Bloomenthal, J. and B, W., 1990, " Interactive techniques for implicit modeling", *Computer Graphics*, 24(2), 109-116.

Bloomenthal, J., Bajaj, C., Blin, J., Gascuel, M., Rockwood, A., Wyvill, B. and Wyvill, G., 1997b, *Introduction to implicit surfaces*, Morgan Kaufmann Publishers Inc, San Fransisco, CA, USA.

Bloomenthal, J. and Chek, L., 1999, "Skeletal methods of shape manipulation", In: *Proceedings of Proceeding of Shape Modeling and Applications 1999*, Aizu-Wakamatsu, Japan, 44-47.

Bloomenthal, J. and Shoemaker, K., 1991, "Convolution surfaces", In: *Proceedings of SIGGRAPH '91 on Computer Graphics*, 251-256.

Bordegoni, M., Colombo, G. and Formentini, L., 2006, "Haptic technologies for the conceptual and validation phases of product design", *Computer & Graphics*, 30(3), 377-390.

Bourdot, P., Convard, T., Picon, F., Ammi, M., Touraine, D. and Vézien, J.-M., 2010, "VR–CAD integration: Multimodal immersive interaction and advanced haptic paradigms for implicit edition of CAD models", *Computer-Aided Design*, 42(5), 445-461.

Bowman, D., 1996, *Conceptual design space – beyond walk-through to immersive design*, (a chapter in Designing digital space), John Wiley & Sons, New York, USA.

Bradshaw, G. and O'Sullian, C., 2002, "Sphere-tree construction using dynamic medial axis approximation", In: *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, San Antonio, Texas, USA, 33-40.

Butterworth, J., Davidson, A., Hench, S. and Olano, T. M., 1992, "3DM: a three-dimensional modeler using a head-mounted display", In: *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, Cambridge, Massachusetts, USA, 135-138.

Cani-Gascuel, M.-P. and Desbrun, M., 1997, "Animation of deformable models using implicit surfaces", *IEEE Trans. on Visualization and Computer Graphics*, 3(1), 39-50.

Cazals, F. and Giesen, J., "Delaunay triangulation based surface reconstruction: Ideas and algorithms", Technical report 5393, INRIA, 2004, 42 pages.

Chapin, W. L., Lacey, T. A. and Leifer, L., 1994, "DesignSpace: a manual interaction environment for computer aided design", In: *Proceedings of the Conference Companion on Human Factors in Computing Systems*, Boston, Massachusetts, USA, 33-34.

Charrot, P. and Gregory, J., 1984, "A pentagonal surface patch for computer-aided geometric design", *Computer-Aided Geometric Design*, 1(1), 87-94.

Che, X., Liang, X. and Li, Q., 2005, "G^1 continuity conditions of adjacent NURBS surfaces", *Computer Aided Geometric Design*, 22(4), 285-298.

Cheng, M. and Wang, G., 2008, "Approximate merging of multiple Bezier segments", *Progress in Natural Science*, 18(6), 757-762.

Cheshire, D., Evans, M. and Dean, C., 2001, "Haptic modeling an alternative industrial design methodology?", In: *Proceedings of EuroHaptics 2001*, Birmingham, UK, 124-129.

Chew, P., "Guaranteed quality triangular meshes", Technical Report TR89-983, BU-CS-96-006, Cornell University, 1989, 20 pages.

Christensen, J., Marks, J. and Ngo, J. T., 1997, "Automatic motion synthesis for 3D mass-spring models", *The Visual Computer*, 13(1), 20-28.

Chu, C. C., Dani, T. H. and Gadh, R., 1997, "Multisensory interface for a virtual reality based computer aided design system", *Computer-Aided Design*, 29(10), 709-725.

Coquillart, S., 1990, "Extended free-form deformation: a sculpturing tool for 3D geometric modeling", *Computer Graphics*, 24(4), 187-193.

Cotin, S., Delingette, H. and Ayache, N., 2000, "A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation", *The Visual Computer*, 16(8), 437-452.

Cotin, S., Delingette, H., Clement, J. M., Bro-Nielsen, M., Ayache, N. and Marescaux, J., 1996, "Geometrical and physical representations for a simulator of hepatic surgery", *Studies in Health Technology and Informatics*, 29(1), 139-151.

Dachille, F., Kaufman, A. and Qin, H., 2001, "A novel haptics based interface and sculpting system for physics-based geometric design", *Computer Aided Design*, 33(5), 403-420.

Dachille, F., Qin, H., Kaufman, A. and El-Sanat, J., 1999, "Haptic sculpting of dynamic surfaces", In: *Proceedings of Symposium on Interactive 3D Graphics*, Atlanta, GA, USA, 103-110.

Dani, T. H. and Gadh, R., 1997, "Creation of concept shade designs via a virtual reality interface", *Computer Aided Design*, 29(8), 555-563.

Davis, O. R. and Burton, R. P., 1991, "Free-form deformation as an interactive modeling tool", *Imaging Technology*, 17(4), 181-187.

Dey, T. K., "Delaunay mesh generation of three dimensional domains", Technical Report OSU-CISRC - TR64, 2007, 32 pages.

Dobashi, Y., Kaneda, K., Yamashita, H., Okita, T. and Nishita, T., 2000, "A simple, efficient method for realistic animation of clouds", In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, New Orleans, Louisiana, USA, 19-28.

Du, W.-H. and Schmitt, F. J. M., 1990, "On the G^1 continuity of piecewise Bézier surfaces: a review with new results", *Computer- Aided Design*, 22(9), 556-573.

Duriez, C., Dubois, F., Kheddar, F. and Andriot, C., 2006, "Realistic haptic rendering of interacting deformable objects in virtual reality environments", *IEEE Trans. on Visualization and Computer Graphics*, 12(1), 36-47.

Ehmann, S. and Lin, C., 2001, "Accurate and fast proximity queries between polyhedra using convex surface decomposition", *Computer Graphics Forum*, 20(3), 500-510.

Ericson, C., 2005, *Real-time collision detection*, The Morgan Kaufmann Series in Interactive 3D Technology, Morgan Kaufmann Publishers, San Francisco, USA.

Faloutsos, P., Panne, M. and Terzopoulos, D., 1997, "Dynamic free-form deformations for animation synthesis", *IEEE Trans. on Visualization and Computer Graphics*, 3(3), 201-214.

Farin, G., 1997, *Curves and surfaces for computer-aided geometric design: a practical guide*, Academic Press, New York, USA.

Ferley, E., Cani, M.-P. and Gascuel, J.-D., 2000, "Practical volumetric sculpting", *Visual Computer*, 16(8), 469-480.

Ferley, E., Cani, M.-P. and Gascuel, J.-D., 2001, "Resolution adaptive volume sculpting", *Graphical Models*, 63(6), 459-478.

Galoppo, N., Tekin, S., Otaduy, M. A., Gross, M. and Lin, M. C., 2007, "Interactive haptic rendering of high-resolution deformable objects", In: *Proceedings of 2nd Int. Conference on Virtual Reality*, Beijing, China, 215-223.

Galyean, T. A. and Hughes, J. F., 1991, "Sculpting: an interactive volumetric modeling technique", *Computer Graphics*, 25(4), 267-274.

Gao, Z. and Gibson, I., 2005, "Haptic B-spline surface sculpting with a shaped tool of implicit surface", *Computer Aided Design & Applications* 2(1-4), 263-272.

Gao, Z. and Gibson, I., 2006, "Haptic sculpting of multi-resolution B-spline surfaces with shaped tools", *Computer Aided Design*, 38(6), 661-676.

Gilbert, E., Johnson, D. and Keerthi, S., 1988, "A fast procedure for computing the distance between complex objects in three-dimensional space", *IEEE Robotics and Automation*, 4(2), 193-203.

Gironimo, G. D., Lanzotti, A. and Vanacore, A., 2006, "Concept design for quality in virtual environment", *Computers & Graphics*, 30(6), 1011-1019.

Gottschalk, S., Lin, M. C. and Manocha, D., 1996, "OBBTree: A hierarchical structure for rapid interference detection", In: *Proceedings of 23rd Annual Int. Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 96*, New Orleans, Louisiana, USA, 171-180.

Gregory, A., Lin, M., Gottschalk, S. and Taylor, R., 2000a, "Fast and accurate collision detection for haptic interaction using a three degree of freedom force feedback device", *Computational Geometry*, 15(1), 69-89.

Gregory, A. D., Ehmann, S. A. and Lin, M. C., 2000b, "inTouch: interactive multiresolution modeling and 3D painting with a haptic interface", In: *Proceedings of the IEEE Virtual Reality 2000 Conference*, New Brunswick, New Jersey, USA, 45-52.

Griessmair, J. and Purgathofer, W., 1989, "Deformation of solids with trivariate B-splines", In: *Proceedings of Eurographics '89*, Elsevier Science Publishers, North-Holland, 137 - 148.

Guigue, P. and Devilers, O., 2003, "Fast and robust triangle-triangle overlap test using orientation predicates", *Graphics Tools*, 8(1), 25-42.

Herzen, B., Barr, A. and Zatz, H., 1990, "Geometric collisions for time-dependent parametric surfaces", In: *Proceedings of ACM SIGGRAPH'90*, Dallas, Texas, USA, 39-48.

Hirota, G., Maheshwari, R. and Lin, M. C., 1999, "Fast volume-preserving free-form deformation using multi-level optimization", In: *Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications, SMA '99*, Ann Arbor, Michigan, United States, 234-245.

Ho, C., Basdogan, C. and Srinivasan, M., 1999, "Efficient point-based rendering techniques for haptic display of virtual objects", *Presence*, 8(5), 447-491.

Hoffmann, C. M. and Hopcroft, J. E., 1987, "Simulation of physical systems from geometric models", *IEEE Robotics and Automation*, 3(3), 194-206.

Hsu, W. M., Hughes, J. F. and Kaufman, H., 1992, "Direct manipulation of free-form deformations", *ACM Computer Graphics*, 26(2), 177 - 184.

Hu, S.-M., Tong, R.-F., Ju, T. and Sun, J.-G., 2001, "Approximate merging of a pair of Bézier curves", *Computer-Aided Design*, 33(2), 125-136.

Hua, J. and Qin, H., 2003, "Free-form deformations via sketching and manipulating scalar fields", In: *Proceedings of the 8th ACM Symposium on Solid Modeling and Applications*, Seattle, Washington, USA, 328 - 333.

Hubbard, P. M., 1995, "Collision detection for interactive graphics applications", *IEEE Transactions on Visualization and Computer Graphics*, 1(3), 218-230.

Hughes, M., DiMattia, C., Lin, M. and Manocha, D., 1996, "Efficient and accurate interference detection for polynomial deformation", In: *Proceedings of the IEEE Computer Animation Conference*, Washington DC, USA, 155-166.

Igwe, P. C., *Deformable volumetric self-organising feature maps and physics-based mdoeling for concept design, Ph.D Thesis*, Department of Mechanical and Material Engineering, The University of Western Ontario, London, 2008a

Igwe, P. C. and Knopf, G. K., 2006, "Modeling deformable objects for computer-aided sculpting (CAS)", In: *Proceedings of the IEEE Conference on Geometric Modeling and Imaging: New Trends*, IEEE Computer Society, Washington, DC, USA, 9 - 14.

Igwe, P. C., Knopf, G. K. and Canas, R., 2008b, "Developing alternative design concepts in VR environments using volumetric self organizing feature maps", *Intelligent Manufacturing*, 19(6), 661-675.

Jimenez, P., Thomas, F. and Torras, C., 2001, "3D collision detection: a survey", *Computer and Graphics*, 25(2), 269-285.

Jin, X., Li, Y. and Peng, Q., 2000, "General constrained deformation based on generalized metaballs", *Computer Graphics*, 24(2), 219-231.

Joint Pain Solutions, 2010 http://www.joint-pain-solutions.com/rheumatoid-arthritis-pictures.html.

Klosowski, J. T., Held, M., Mitchell, J., Sowizral, H. and Zikan, K., 1998, "Efficient collision detection using bounding volume hierarchies of k- dops", *IEEE Trans. on Visualization and Computer Graphics*, 4(1), 21-36.

Knopf, G. K. and Igwe, P. C., 2005, "Deformable mesh for virtual shape sculpting", *Robotics and Computer Integrated Manufacturing*, 21(4), 302-311.

Knopf, G. K. and Pungotra, H., 2007, "Visual exploration of numeric data using 3D self organising feature maps", In: *Proceedings of Int. Conference on Artificial Neural Network in Engineering (ANNIE 2007)*, St. Louis, Missouri, USA, 17, 297-302.

Knopf, G. K. and Sangole, A., 2002, "Intelligent systems for interactive design and visualization", In: *Proceedings of IEEE Conference of Industrial Electronics Society*, Sevilla, Spain, 2995-3001.

Knopf, G. K. and Sangole, A., 2004, "Interpolating scattered data using 2D self-organizing feature maps", *Graphical Models*, 66(1), 50-69.

Knopf, G. K., Sangole, A. and Igwe, P., 2003, "Parameterization of scattered surface points using a SOFM", In: *Proceedings of Intelligent Engineering Systems Through Artificial Neural Networks: ASME Press*, Missouri, St. Louis, USA, 33-38.

Koch, R. M., Gross, M. H., Carls, F. R., von-Büren, D. F., Fankhauser, G. and Parish, Y. I., 1996, "Simulating facial surgery using finite element models", In: *Proceedings of the 23rd Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '96*, 421-428.

Kohonen, T., 2001, *Self organizing map*, Springer Series in Information Sciences, Heidelberg, New York.

Krishnan, S., Pattekar, A., Lin, M. C. and Manocha, D., 1998, "A higher order bounding volume for fast proximity queries", In: *Proceedings of 3rd International Workshop on Algorithmic Foundations of Robotics*, 122–136.

Kuehnapfel, U. and Neisius, B., 1993, "CAD-based graphical computer simulation in endoscopic surgery", *Endoscopic Surgery and Allied Technologies*, 1(3), 181-184.

Lamousin, H. J. and Waggenspack-Jr., W. N., 1994, "NURBS-based free-form deformations", *IEEE Computer Graphics and Applications*, 14(6), 59 - 65.

Larsen, E., Gottschalk, S., Lin, M. and Manocha, D., 2000, "Fast distance queries with rectangular swept sphere volumes", In: *Proceedings of IEEE Int. Conference on Robotics and Automation*, San Francisco, USA, 3719 - 3726.

Larsson, T. and Akenine-Möller, T., 2001, "Collision detection for continuously deforming bodies", In: *Proceedings of Eurographics 2001*, 325-333.

Lauterbach, C., Garland, M., Sengupta, S., Luebke, D. and Manocha, D., 2009, "Fast BVH construction on GPUs", *Computer Graphics Forum*, 28(2), 375-384.

Lee, Y., Terzopoulos, D. and Waters, K., 1995, "Realistic modeling for facial animation", In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '95*, 55-62.

Lin, M. C., Baxter, W., Foskey, M., Otaduy, M. A. and Scheib, V., 2002, "Haptic interaction for creative processes with simulated media", In: *Proceedings of the IEEE International Conference on Robotics and Automation*, Washington DC, USA, 598–604.

Lin, M. C. and Canny, J. F., 1991, "A fast algorithm for incremental distance calculation", In: *Proceedings of IEEE International Conference on Robotics and Automation*, Sacramento, CA, 1008 - 1014.

Lin, M. C. and Gotttschalk, S., 1998, "Collision detection between geometric models: A survey", In: *Proceedings of IMA Conference of Mathematics of Surfaces*, UK, 602-608.

Lin, M. C. and Manocha, D., 2004, *Collision and proximity Queries*, in J. E. Goodman and J. O'Rourke, eds., Handbook of Discrete and Computational Geometry, Chapman & Hall, Boca Raton, Florida, USA.

Liu, Y., Pottmann, H. and Wang, W., 2006, "Constrained 3D shape reconstruction using a combination of surface fitting and registration", *Computer Aided Design*, 38(6), 572-583.

Luo, Q. and Xiao, J., 2007, "Contact and deformation modeling for interactive environments", *IEEE Trans. on Robotics*, 23(3), 416-430.

Martin, W. and Cohen, E., 2001, "Representation and extraction of volumetric attributes using trivariate splines: a mathematical framework", In: *Proceedings of Sixth*

ACM Symposium on Solid Modeling and Applications, SMA '01, Ann Arbor, Michigan, United States, 234-240.

McCormack, J. and Sherstyuk, A., 1998, "Creating and rendering convolution surfaces", *Computer Graphics Forum*, 17(2), 113-120.

McDonnell, K. T. and Qin, H., 2007, "PB-FFD: A point-based technique for free-form deformation ", *Graphics, GPU, & Game Tools*, 12(3), 25 - 41.

McNeely, W. A., Puterbaugh, K. D. and Troy, J. J., 1999, "Six degree-of-freedom haptic rendering using voxel sampling", In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, Los Angeles, California, USA, 401 - 408.

Mine, M. R., 1997, "ISAAC: a Meta-CAD system for virtual environments", *Computer-Aided Design*, 29(8), 547-553.

Mirtich, B., "Efficient algorithms for two-phase collision detection", Technical Report, TR-97-23, MERL, 1997, 26 pages.

Möller, T., 1997, "A fast triangle-triangle intersection test", *Graphics Tools* 2(2), 25-30.

Morris, R., 2009, "*The fundamentals of product design*", AVA Publishing SA, Switzerland.

Nedel, L. P. and Thalmann, D., 1998, "Real time muscle deformations using mass spring systems", In: *Proceedings of the Computer Graphics International Conference, CGI '98*, Hannover, Germany, 156–165.

Nishita, T., Iwasaki, H., Dobashi, Y. and Nakamae, E., 1997, "A modeling and rendering method for snow by using metaballs", *Computer Graphics Forum*, 16(3), 357-364.

Ohtake, Y. and Belyaev, A. G., 2002, "Dual-primal mesh optimization for polygonized implicit surfaces with sharp features", *Transactions of ASME, Journal of Computing and Information Science in Engineering*, 2(2), 277-284.

Okabe, A., Boots, B., Sugihara, K. and Chiu, S., 2000, *Spatial tessellations: Concepts and applications of Voronoi diagrams*, John Wiley, Chichester, UK.

Øyvind, H. and Morten, D., 2006, "*Triangulations and Applications*", Springer-Verlag Berlin Heidlberg, The Netherlands.

Park, S. and Kunwoo, L., 1997, "High-dimensional trivariate NURBS representation for analyzing and visualizing fluid flow data", *Computers & Graphics*, 21(4), 473-482.

Pasko, A., Adzhiev, V., Sourin, A. and Savchenko, V., 1995, "Function representation in geometric modeling: concepts, implementation and applications", *Visual Computer*, 11(8), 429-446.

Piegl, L. and Tiller, W., 1994, "Software engineering approach to degree elevation of B-spline curves", *Computer Aided Design*, 26(1), 17-28.

Piegl, L. and Tiller, W., 1995, "Algorithm for degree reduction of B-spline curves", *Computer Aided Design*, 27(2), 101-110.

Piegl, L. and Tiller, W., 1997, *The NURBS book*, Monographs in visual communications, Springer, New York.

Platt, J. C. and Barr, A. H., 1988, "Constraint methods for flexible models", *Computer Graphics*, 22(4), 279-288.

Pungotra, H., Knopf, G. K. and Canas, R., 2008, "Efficient algorithm to detect collision between deformable B-spline surface for virtual sculpting", *Computer-Aided Design*, 40(10-11), 1055-1066.

Pungotra, H., Knopf, G. K. and Canas, R., 2009a, "Framework for modeling and validating conept designs in virtual reality environments", In: *Proceedings of the IEEE Virtual Reality 2009 Conference (Symposium on Human Factors and Ergonomics)*, Toronto, Canada, 393-398.

Pungotra, H., Knopf, G. K. and Canas, R., 2009b, "Novel collision detection algorithm for physics-based simulation of deformable B-spline shapes", *Computer Aided Design & Applications*, 6(1), 43-54.

Pungotra, H., Knopf, G. K. and Canas, R., 2010a, "Merging multiple B-spline surface patches in a virtual reality environment", *Computer Aided Design*, 42(10), 847 - 859.

Pungotra, H., Knopf, G. K. and Canas, R., 2010b, "Representation of objects in virtual reality environment using B-spline blending matrices", In: *Proceedings of the 21st IASTED Int. Conference on Modeling and Simulation (MS 2010)*, Banff, Alberta, Canada, 281-288.

Quinlan, S., 1994, "Efficient distance computation between non-convex objects", In: *Proceedings of IEEE Int. Conference on Robotics and Automation*, New Orleans, LA, USA, 3324-3329.

Raviv, A. and Elber, G., 2000, "Three-dimensional freeform sculpting via zero sets of scalar trivariate functions", *Computer-Aided Design*, 32(8-9), 513-526.

Reuter, P., Joyot, P., Trunzler, J., Boubekeur, T. and Schlick, C., "Reconstructing implicit surfaces with sharp edges via enriched reproducing kernel approximation", Research report RR-1334-04, 2004, 20 pages.

Robinson, G., Ritchie, J. M., Day, P. N. and Dewar, R. G., 2007, "System design and user evaluation of Co-Star: an immersive stereoscopic system for cable harness design", *Computer-Aided Design*, 39(4), 245-257.

Rockwood, A., 1989, "The Displacement method for implicit blending surfaces in solid models", *ACM Trans. on Graphics*, 8(4), 279-297.

Sachs, E., Roberts, A. and Stoops, D., 1991, "3-Draw: a tool for designing 3D shapes", *IEEE Computer Graphics and Applications*, 11(6), 18-24.

Sagar, M. A., Bullivant, D., Mallinson, G. D., Hunter, P. J. and Hunter, I. W., 1994, "A virtual environment and model of the eye for surgical simulation", In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH '94*, ACM, New York, NY, 205-212.

Savchenko, V., Pasko, A., Okunev, O. and Kunii, T., 1995, "Function representation of solids reconstructed from scattered surface points and contours", *Computer Graphics Forum*, 14(4), 181- 188.

Schein, S. and Elber, G., 2004, "Discontinuous free-form deformations", In: *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, 227 - 236.

Schmitt, B., Pasko, A. and Schlick, C., 2001, "Constructive modeling of FRep solids using spline volumes", In: *Proceedings of Sixth ACM Symposium on Solid Modeling and Applications*, Ann Arbor, Michigan, USA, 321 - 322.

Sederberg, T. and Perry, S., 1986, "Free-form deformation of solid geometric models", In: *Proceedings of the 13th annual conference on computer graphics and interactive techniques*, 20 (4), 151-160.

Sener, B., Wormald, P. and Campbell, R., 2002, "Evaluating a haptic modeling system with industrial designers", In: *Proceedings of EuroHaptics International Conference*, Edinburgh, Scotland, 165 -169.

Shen, C., O'Brien, J. F. and Shewchuk, J. R., 2004, "Interpolating and approximating implicit surfaces from polygon soup", In: *Proceedings of ACM SIGGRAPH04*, Los Angeles, California, 1-9.

Shi, X., Wang, T., Wu, P. and Liu, F., 2004, "Reconstruction of convergent G^1 smooth B-spline surfaces", *Computer Aided Geometric Design*, 21(9), 893-913.

Speeter, T. H., 1992, "Three-dimensional finite element analysis of elastic continua for tactile sensing", *Robotics Research*, 11(1), 1-19.

Székely, G., Brechbühlera, C., Huttera, R., Rhomberga, A., Ironmongera, N. and Schmida, 2000, "Modelling of soft tissue deformation for laparoscopic surgery simulation", *Medical Image Analysis*, 4(1), 57-66.

Taia, C. L., Hub, S. M. and Huangb, Q. X., 2003, "Approximate merging of B-spline curves via knot adjustment and constrained optimization", *Computer-Aided Design*, 35(10), 893-899.

Terzopoulos, D. and Fleischer, K., 1988a, "Modeling inelastic deformation: viscoelasticity, plasticity, fracture", *Computer Graphics*, 22(4), 269-278.

Terzopoulos, D., Platt, J., Barr, A. and Fleischer, K., 1987, "Elastically deformable models", *Computer Graphics*, 21(4), 205 - 214.

Terzopoulos, D., Platt, J. and Fleischer, K., 1991, "Heating and melting deformable models", *Visualization and Computer Animation*, 2(2), 68-73.

Terzopoulos, D. and Waters, K., 1990, "Physically-based facial modeling, analysis, and animation", *Visualization and Computer Animation*, 1(2), 73 - 80.

Terzopoulos, D. and Witkin, A., 1988b, "Physically based model with rigid and deformable components", *IEEE Trans. on Visualization and Computer Graphics*, 8(6), 41-51.

Teschner, M., Kimmerle, S., Zachmann, G., Heidelberger, B., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnetat-Thalmann, N. and Strasser, W., 2005, "Collision detection for deformable objects", *Computer Graphics Forum*, 24(1), 61-81.

The Arthritis Society, 2010 http://www.arthritis.ca/types%20of%20arthritis/ra/default.

asp?s=1&province=on.

Thompson, T., Johnson, D. and Cohen, E., 1997, "Direct haptic rendering of sculptured models", In: *Proceedings of 1997 Symposium on Interactive 3D Graphics*, Providence, Rhode Island, United States, 167-176.

Tu, X. and Terzopoulos, D., 1994, "Artificial fishes: physics, locomotion, perception, behavior", In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, 43 - 50.

Tuohy, S. T. and Bardis, L., 1993, "Low-Degree Approximation of High-Degree B-Spline Surfaces", *Engineering with Computers*, 9(4), 198-209.

Turk, G. and O'Brien, J. F., "Variational implicit surfaces", Technical Report GIT-GVU-99-15, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, 1999a, 9 pages.

Turk, G. and O'Brien, J. F., 2002, "Modelling with implicit surfaces that interpolate", *ACM Transaction on Graphics*, 21(4), 855-873.

Turk, G. and O'Brien, J. F., 1999b, "Shape transformation using variational implicit functions", In: *Proceedings of ACM SIGGRAPH 99*, Los Angeles, California, 335-342.

van-den-Bergen, G., 1997, "Efficient collision detection of complex deformable models using AABB trees", *Graphics Tools*, 2(4), 1-13.

Wang, S. W. and Kaufman, A. E., 1995, "Volume sculpting", In: *Proceedings of the 1995 Symposium on Interactive 3D Graphics, I3D '95*, Monterey, California, USA, 151-156.

Waters, K., 1992, "A physical model of facial tissue and muscle articulation derived from computer tomography data", In: *Proceedings of Visualization in Biomedical Computing (VBC '92)*, Chapel Hill, N.C., USA, 574–583.

Weidlich, D., Cser, L., Polzin, T., D.Cristiano and Zickner, H., 2007, "Virtual reality approaches for immersive design", *CIRP Annals - Manufacturing Technology*, 56(1), 139-142.

Witkin, A. P. and Heckbert, P. S., 1998, "Using particles to sample and control impict surfaces", In: *Proceedings of 21st Annual Conference on Computer Graphics and Interactive Techniques*, Orlando, Florida, USA, 269-277.

Wyvill, B., Galin, E. and Guy, A., 1999, "Extending the CSG tree: warping, blending and boolean operations in an implicit surface modeling system", *Computer Graphics Forum*, 18(2), 149 -158.

Wyvill, B. and Wyvill, G., 1989, "Field functions for implicit surfaces", *Visual Computer*, 5(1-2), 75-82.

Ye, J., *Integration of virtual reality techniques into computer aided product design, PhD Thesis*, Department of Design and Technology, Loughborough University, Leicestershire, UK, 2005

Ye, J. and Campbell, R., 2006a, "Supporting conceptual design with multiple VR based interfaces", *Virtual and Physical Prototyping*, 1(3), 171-181.

Ye, J., Campbell, R., Page, T. and Badni, K., 2006b, "An investigation into the implementation of virtual reality technologies in support of conceptual design", *Design Studies*, 27(1), 77-97.

Ye, J. and Campbell, R. I., 2002, "New CAD interfaces for the conceptual design process", In: *Proceedings of the 3rd Annual International Conference on Rapid Product Development*, Bloemfontein, South Africa, 150 - 162.

Yong, J.-H., Hu, S.-M., Sun, J.-G. and Tan, X.-Y., 2001, "Degree reduction of B-spline curves", *Computer Aided Geometric Design*, 18(2), 117-127.

Zeid, I., 1991, *CAD/CAM – theory and practice*, McGraw-Hill, New York, USA.

Zheng, J. M., Chan, K. W. and Gibson, I., 2003, "Constrained deformation of freeform surfaces using surface features for interactive design", *Advanced Manufacturing Technology*, 22(2), 54-67.

Zhong, Y., Ma, W. and Shirinzadeh, B., 2005, "A methodology for solid modelling in a virtual reality environment", *Robotics and Computer-Integrated Manufacturing*, 21(6), 528-549.

Zienkiewicz, O. C., Taylor, R. L., Taylor, R. L. and Zhu, J. Z., 2005, "*The finite element method: its basis and fundamentals*", 6th ed., Elsevier Butterworth–Heinemann, Burlington, MA, USA.

APPENDIX A: ERROR ANALYSIS OF THE MERGED B-SPLINE SURFACE

A.1 Introduction

One major requirement for representing 3D objects in a VR environment is that the user should interact with the virtual model as if it were a visually realistic surface or solid. The algorithm described in this thesis (Chapter 4) achieves the merging process without imposing any constraints. An important factor in determining the versatility of the algorithm is that it should be able to tackle all the cases during haptic interaction. At the same time, the deviation of the merged surface should be small.

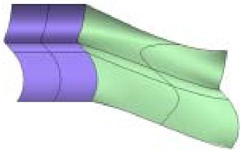
In this appendix, different cases are compared for accuracy of the merging algorithm and detailed error analysis is presented. For comparison purposes, a few cases were considered and the merged surfaces generated by the proposed technique were compared with the results generated by a commercially available NURBS modeling software package, Rhinoceros® (Rhino®). The algorithm was used to generate surfaces with C^0, C^1, and C^2 connectivity. As Rhino® does not permit the user to edit the knot vector, or to select an option for the type of desired connectivity during the merging process, only C^2 continuous surfaces were generated for Rhino®.

The data generated for error analysis depends upon the characteristics of the surfaces used for merging. Hence, the results may vary if the characteristics of the surfaces being merged are changed or the number of points generated for merging process is varied. However, this data can be used for comparing the results achieved by the proposed algorithm and that by generated by Rhino®. Same cases were discussed in Chapter 6 and concise data was presented for error analysis.

A.2 Case 1: Similar Curvatures and Knot Vectors

This is the simplest case that can be encountered when combining B-spline surface patches. Two arbitrary surfaces with the same degree and uniform knot vectors were used to generate a single integral surface. The results of the deviation analysis of the merged surface from the original surfaces are shown in Table A.1. The surface generated by proposed algorithm results in lower standard deviation. Similarly, the maximum distance of the points is lower for the proposed algorithm. The results are much better when only C^0 connectivity is required.

Table A.1 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for surface having similar curvatures and knot vectors.

Test Cases	**Error Analysis** (using point set deviation)	**Merged Surface**		
		Rhino® C^2	**Proposed Method C^2**	**Proposed Method C^0**
1: Similar curvature and knot vectors 	Total points	20402	20402	20402
	Close points	18136	18262	20134
	Average distance	0.2242	0.3568	0.0104
	Median distance	0.3154	0.3119	0.0056
	Standard deviation	0.2704	0.2452	0.0164
	Maximum distance	0.9726	0.9561	0.0855

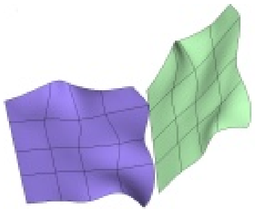
A.3 Case 2: Different Curvatures but Similar Knot Vectors

In many cases, the curvature of the patch edges may not be similar. The traditional methods for merging surfaces require a common edge and do not work well if the curvatures do not match at the joining edge. Rhino® did not allow the surfaces to be merged because the surfaces were considered to be too far apart. In some cases, the CAD/CAM software allows the user to match surfaces (fitting the surfaces with certain connectivity without merging), thereby reducing, to a large extent, the gaps between the two surfaces. Once the gaps are closed, it might be possible to merge the surfaces.

However, there is no guarantee that the surfaces will merge after matching. A merged surface was created by using this process on Rhino®.

In contrast, the proposed algorithm does not to match the surfaces at common edge. This feature is important, particularly in a virtual reality environment, because the user considers the models as objects and does not treat them as B-spline surface patches. As shown in Table A.2, the standard deviation, even for C^2 continuity, is about 0.1 mm, and reduces further to 0.06 mm, for C^0 continuity. The maximum deviation of the merged surface by using proposed algorithm is much less compared to that merged by Rhino, even after matching the surfaces.

Table A.2 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for surface having different curvatures but similar knot vectors.

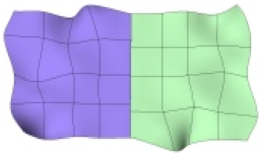
Test Cases	**Error Analysis** (using point set deviation)	**Merged Surface**		
		Rhino® C^2	**Proposed Method C^2**	**Proposed Method C^0**
2[1]: Different curvatures but similar knot vectors 	Total points	20402	20402	20402
	Close points	20153	20145	19974
	Average distance	0.0778	0.0837	0.0458
	Median distance	0.0180	0.0498	0.0273
	Standard deviation	0.1708	0.1093	0.0621
	Maximum distance	0.8071	0.5046	0.2713

[1] The surfaces were matched (fitted without merging) before merging for Rhino®. It did not allow the merging of these surfaces for being too far apart, without matching.

A.4 Case 3: Similar Curvature but Different Knot Vectors

One constraint for NURBS-based surfaces is that an integral surface needs to have a continuous knot vector. The traditional approach is to insert additional knots in the surfaces to achieve a continuous knot for the integral surface. Theoretically it is possible to insert a knot into a surface without changing its geometry. Even if there is no change in the geometry of the surface, additional knots increase the number of control points needed to represent the surface. In a virtual reality environment, the increased number of control points increases the computational cost of collision detection and manipulation. Table A.3 shows the results of merging for proposed algorithm and Rhino®.

Table A.3 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for surface having similar curvature but different knot vectors.

Test Cases	**Error Analysis** (using point set deviation)	**Merged Surface**		
		Rhino® C^2	**Proposed Method C^2**	**Proposed Method C^0**
3[1]: Similar curvatures but different knot vectors 	Total points	20402	20402	20402
	Close points	20137	20133	20098
	Average distance	0.0259	0.0273	0.0110
	Median distance	0.0133	0.0190	0.0084
	Standard deviation	0.0364	0.0283	0.0096
	Maximum distance	0.1888	0.1394	0.0411

[1] The merged surface created by Rhino increased the number of knots and control points. The proposed algorithm merged surfaces without increasing the number of control points.

The merged surface created by Rhino® had four more control points in *v* direction as compared to the original surface patches. These additional control points carry no significant geometric information and are added only to satisfy the constraint of having a continuous knot vector. The surface generated by using the present algorithm with C^2 and C^0 connectivity had the same number of control points (6) as the surfaces that were merged. In addition, the merged surface generated by the proposed algorithm exhibits a better standard deviation even for C^2 connectivity, as shown in Table A.3. In the case of C^0 connectivity, the standard deviation was observed to be less than 0.01 mm. At the same time, the maximum deviation of the surface from the original surface is less when surfaces are merged using the proposed algorithm. The maximum deviation is the least for merged surface with C^0 connectivity using the proposed algorithm.

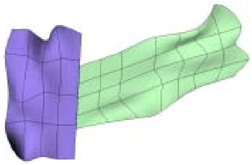
A.5 Case 4: Similar Curvatures but Different Knot Vectors and Dimensions

In many cases, the edges of the patches to be merged do not have the same length. As the merged surface should have same parameter throughout the surface, the two surfaces need to deviate a lot from the original geometry. In commercially available CAD/CAM software, any two surfaces which do not have almost the same dimension cannot be merged. Rhino® could not merge the surfaces even after matching these surfaces.

The algorithm proposed in this thesis does not put any of these constraints on the surfaces to be merged. The merged surface generated by the proposed algorithm does show a deviation when compared to the original surface, but overall, the result is satisfactory. As shown in Table A.4, the standard deviation from the surface is 0.026 mm for C^2 connectivity which is very small considering the large differences in the length of the edges at which the two surfaces were to be joined.

The maximum deviation of the merged surface is about 0.11 mm while maintaining C^2 connectivity. These results show that in addition to the robustness of the algorithm, the deviation of the merged surface is also very small.

Table A.4 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for surface having similar curvatures but different knot vectors and dimensions.

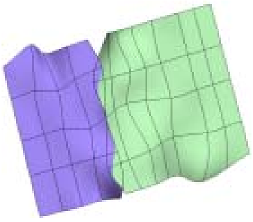
Test Cases	**Error Analysis** (using point set deviation)	**Merged Surface**		
		Rhino® C^2	**Proposed Method C^2**	**Proposed Method C^0**
4[1]: Similar curvatures but different knot vectors and dimensions 	Total points	-	20402	20402
	Close points	-	19891	19935
	Average distance	-	0.0215	0.0233
	Median distance	-	0.0137	0.0159
	Standard deviation	-	0.0263	0.0228
	Maximum distance	-	0.1131	0.1254

[1] Rhino® did not merge the surfaces even after matching.

A.6 Case 5: Intersecting and Trimmed Surfaces

If the patches to be merged are intersecting, the surface may not merge as intended by the user or may not merge at all. Rhino® does not merge intersecting surfaces. The algorithm presented in this thesis uses collision detection to merge two or more intersecting B-spline surface patches. The algorithm for merging intersecting or trimmed surface is discussed in detail in Section 4.4.1. The point set deviation analysis of the merged surfaces is shown in Table A.5. It is clear from Table A.5 that the surface generated is within the tolerance needed during the haptic interaction with the model. However, as some of the regions of intersecting surfaces were not used for merging, the maximum deviation of the merged surfaces from the original surfaces is large in this case.

Table A.5 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for intersecting and trimmed surfaces.

Test Cases	**Error Analysis** (using point set deviation)	**Merged Surface**		
		Rhino® C^2	**Proposed Method C^2**	**Proposed Method C^0**
5[1]: Intersecting and trimmed surfaces 	Total points	-	20402	20402
	Close points	-	19339	19112
	Average distance	-	0.0423	0.0451
	Median distance	-	0.0144	0.0221
	Standard deviation	-	0.0792	0.0771
	Maximum distance	-	0.4759	0.3251

[1] Rhino® did not merge the surfaces even after matching.

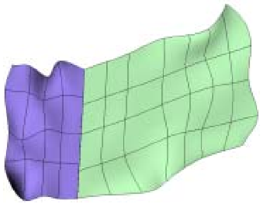
A.7 Case 6: Surfaces having Different Degrees

Before the surfaces having different degrees can be merged, the degree of the surfaces in *u* and *v* directions need to be made uniform. Thus, the problem reduces to increasing or decreasing the degree of a surface to make it uniform with that of the other surface. The proposed algorithm can increase or decrease the degree of the surfaces as discussed in Section 4.4.2.

For the analysis, two surfaces of different degrees were considered. The first surface had 6×6 control points and degree three in both the directions. The second surface had 8×12 control points and degree four in *u* direction and degree five in *v* direction. The surfaces were merged in *v* direction. Thus, the first surface was elevated to degree four in *u* direction and degree 5 in *v* direction. Rhino®, by default increases the degree of lower order surface and merges these two surfaces with degree five in *v* direction with connectivity of C^4 and degree four in *u* direction. The surface generated by the proposed algorithm did not require additional knots to be introduced and, hence, had a lower number of control points as compared to the one generated by Rhino®.

Table A.6 shows the deviation of the merged surface generated by Rhino® and proposed algorithm from the original surfaces. It is clear from the point set deviation analysis that the proposed algorithm results in a merged surface with lower standard deviation compared to that merged by Rhino®. The maximum deviation of the merged surface is also lower for the surface merged by using the proposed algorithm.

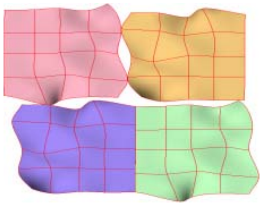
Table A.6 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for surface having different degrees.

Test Cases	**Error Analysis** (using point set deviation)	**Merged Surface**		
		Rhino® C^2	**Proposed Method C^2**	**Proposed Method C^0**
6: Surfaces having different degrees 	Total points	20402	20402	20402
	Close points	20109	20178	20121
	Average distance	0.0126	0.0139	0.0157
	Median distance	0.0087	0.0085	0.0081
	Standard deviation	0.0125	0.0117	0.0154
	Maximum distance	0.0563	0.0455	0.0396

A.7 Case 7: Multiple Surfaces

The algorithm is capable of merging any number of surfaces. Rhino® could not merge the surfaces even after matching the surfaces and merging these in pairs. As shown in Table A.7, the merged surface shows very small standard deviation from the original surfaces.

Table A.7 Comparison of the point set deviation of the merged surfaces generated by Rhino® and the proposed algorithm for multiple surfaces.

Test Cases	Error Analysis (using point set deviation)	Merged Surface		
		Rhino® C^2	Proposed Method C^2	Proposed Method C^0
7[1]: Multiple surfaces 	Total points	-	40804	40804
	Close points	-	39151	39469
	Average distance	-	0.0811	0.0448
	Median distance	-	0.0523	0.0241
	Standard deviation	-	0.0961	0.0705
	Maximum distance	-	0.3880	0.3050

[1] Rhino® did not merge the surfaces even after matching.

A.8 Concluding Remarks

B-spline surface patch merging algorithm does not impose any restriction regarding the type of continuity required at the common edge, the knot vector, or the number of surfaces being merged simultaneously. The proposed algorithm can efficiently merge B-spline surface patches having dissimilar curvatures at the common edge, intersecting B-spline surface patches, and the B-spline surface patches having trimmed edges. The user will not have to face a situation where the two surfaces cannot be merged due to intersection, dissimilar curvature at common edge or similar situations. This adds to the robustness of the algorithm. At the same time, the algorithm is capable of merging multiple B-spline surfaces with small deviation. This deviation is lower than that exhibited by the surfaces merged by Rhino®. In many cases, Rhino® was not able to merge surfaces, whereas the proposed algorithm could achieve it.

However, it should be noted that the mathematical representation of NURBS surfaces does not allow certain type of merging. Two surfaces cannot be merged if these produce 'T junction'. As an example, if a B-spline cylindrical surface is lying on a flat open B-spline surface, these cannot be merged. The algorithm proposed in this thesis cannot merge surfaces if this merging is not allowed due to the constraints imposed by the mathematical representation of NURBS surfaces.

CURRICULUM VITAE

Name:	**Harish Pungotra**
Post-secondary Education and Degrees:	Panjab University Chandigarh, Punjab, India 1989-1993 B.E.(Mechanical Engineering)
	Panjab University Chandigarh, Punjab, India 1993-1997 M.E.(Mechanical Engineering)
	The University of Western Ontario London, Ontario, Canada 2006-2010 Ph.D.
Honors and Awards:	Ministry of Human Resources Graduate Scholarship (GATE) India 1993-1995
	Distinction in M.E. Panjab University Chandigarh, Punjab, India 1997
Related Work Experience	Teaching/Research Assistant, Mechanical & Material Engineering The University of Western Ontario London, Ontario, Canada 2006-2010
	Guest Worker, National Research Council of Canada (NRC) Institute for Research in Construction London, Ontario, Canada 2006-2010

Publications:

Theses

Pungotra, H. (2010) *Collision Detection and Merging of Deformable B-spline Surfaces in Virtual Reality Environment*. Doctor of Philosophy (Ph.D.), Department of Mechanical and Materials Engineering, Faculty of Engineering. The University of Western Ontario, London, Ontario, Canada.

Pungotra, H. (1997) *Computer Aided Design of Bearings and Design Charts for Industrial Applications*. Master of Engineering (M.E.), Department of Mechanical Engineering, Faculty of Engineering. Panjab University, Chandigarh, Punjab, India.

Refereed Journal Papers

Pungotra H., Knopf G.K., Canas R. 2010, “Merging multiple B-spline surface patches in a virtual reality environment ”, *Computer Aided Design*, 42(10), 847-859.

Pungotra, H., Knopf, G. K. and Canas, R., 2009, "Novel collision detection algorithm for physics-based simulation of deformable B-spline shapes", *Computer Aided Design & Applications*, 6(1), 43-54.

Pungotra, H., Knopf, G. K. and Canas, R., 2008, "Efficient algorithm to detect collision between deformable B-spline surface for virtual sculpting", *Computer-Aided Design*, 40(10-11), 1055-1066.

Refereed Conference Papers

Pungotra H., Knopf G. K., Canas R., 2010, "Representation of objects in virtual reality environment using B-spline blending matrices", In: *Proceedings of the* 21st *IASTED Int. Conference on Modeling and Simulation (MS 2010)*, Banff, Alberta, Canada, 281-288.

Pungotra, H., Knopf, G. K. and Canas, R., 2009, "Framework for modeling and validating concept designs in virtual reality environments", In: *Proceedings of the IEEE Virtual Reality 2009 Conference* (*Symposium on Human Factors and Ergonomics*), Toronto, Canada, 393-398.

Knopf, G. K. and **Pungotra, H.**, 2007, "Visual exploration of numeric data using 3D self organizing feature maps", In: *Proceedings of Int. Conference on Artificial Neural Network in Engineering* (*ANNIE 2007*), St. Louis, Missouri, USA, 297-302.

Pungotra H., 2000,"Computer aided design of ball bearings and design charts for industrial applications", In: *Proceedings of Int. Conference on Intelligent and Flexible Manufacturing Systems* (*IAFMS 2000*), 172-176.

Conference Presentations:

Pungotra H., Knopf G.K., Canas R., 2009, "Novel collision detection algorithm for physics-based simulation of deformable B-spline shapes", *CAD Conference '09*, June 08-12, Reno, Nevada, USA.

Igwe P. C., **Pungotra H.**, Canas R. and Knopf G. K., 2007, "Conceptual design by haptic modeling", *The Seventh Int. Conference on Advanced Manufacturing Technologies*, June 4-6, London, ON, Canada.