



Universidade Estadual de Campinas
Instituto de Computação



Elisa de Cássia Silva Rodrigues

Mathematical and computational methods for
modeling and editing deformations

Métodos matemáticos e computacionais para
modelagem e edição de deformações

CAMPINAS
2017

Elisa de Cássia Silva Rodrigues

**Mathematical and computational methods for modeling and
editing deformations**

**Métodos matemáticos e computacionais para modelagem e
edição de deformações**

Tese apresentada ao Instituto de Computação
da Universidade Estadual de Campinas como
parte dos requisitos para a obtenção do título
de Doutora em Ciência da Computação.

Dissertation presented to the Institute of
Computing of the University of Campinas in
partial fulfillment of the requirements for the
degree of Doctor in Computer Science.

Supervisor/Orientador: Prof. Dr. Jorge Stolfi

Este exemplar corresponde à versão final da
Tese defendida por Elisa de Cássia Silva
Rodrigues e orientada pelo Prof. Dr. Jorge
Stolfi.

CAMPINAS
2017

Agência(s) de fomento e nº(s) de processo(s): CNPq, 140780/2013-0; CAPES, 01-P-04554-2013

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

R618m Rodrigues, Elisa de Cássia Silva, 1984-
Mathematical and computational methods for modeling and editing
deformations / Elisa de Cássia Silva Rodrigues. – Campinas, SP : [s.n.], 2017.

Orientador: Jorge Stolfi.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de
Computação.

1. Modelagem geométrica. 2. Computação gráfica. 3. Sistemas lineares. 4.
Mínimos quadrados. I. Stolfi, Jorge, 1950-. II. Universidade Estadual de
Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Métodos matemáticos e computacionais para modelagem e edição de deformações

Palavras-chave em inglês:

Geometric modeling

Computer graphics

Linear systems

Least squares

Área de concentração: Ciência da Computação

Titulação: Doutora em Ciência da Computação

Banca examinadora:

Jorge Stolfi [Orientador]

Helena Cristina da Gama Leitão

Rodrigo Minetto

Hélio Pedrini

Wu Shin Ting

Data de defesa: 10-02-2017

Programa de Pós-Graduação: Ciência da Computação



Universidade Estadual de Campinas
Instituto de Computação



Elisa de Cássia Silva Rodrigues

Mathematical and computational methods for modeling and
editing deformations

Métodos matemáticos e computacionais para modelagem e
edição de deformações

Banca Examinadora:

- Prof. Dr. Jorge Stolfi (*Supervisor/Orientador*)
Instituto de Computação - UNICAMP
- Profa. Dra. Helena Cristina da Gama Leitão
Instituto de Computação - UFF
- Prof. Dr. Rodrigo Minetto
Departamento Acadêmico de Informática - UTFPR
- Prof. Dr. Hélio Pedrini
Instituto de Computação - UNICAMP
- Profa. Dra. Wu Shin Ting
Faculdade de Engenharia Elétrica e de Computação - UNICAMP

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 10 de fevereiro de 2017

*“What makes the desert beautiful is that
somewhere it hides a well.”*
(Antoine Saint-Exupéry)

Acknowledgements

First, I thank my parents, Isabel and Flávio, for the psychological and financial support and especially the encouragement, understanding and unconditional love offered throughout my life, especially at this stage. My brother and my heart sister, Eduardo and Silvana, for the advice and support. All my family for being by my side at all times.

I would also like to thank old friends and colleagues from courses that collaborated, directly or indirectly, with the realization and completion of this work, sharing experiences, supporting and celebrating achievements.

Many thanks to my advisor, Jorge Stolfi, for his guidance, support, trust, patience, wisdom and professionalism offered through my research. The University of Campinas (UNICAMP), the Institute of Computing (IC), and the Visual Computer Lab (LIV) for the opportunity and available infrastructure, the teachers and staff for their support and guidance.

At last, I would also like to thank the CAPES (*Coordenação de Aperfeiçoamento de Pessoal de Nível Superior*), and CNPq (*Conselho Nacional de Desenvolvimento Científico e Tecnológico*) that granted me scholarships (01-P-04554-2013 and 140780/2013-0).

Resumo

Nesta tese, descrevemos primeiramente o algoritmo ECLES (*Editing by Constrained LEast Squares*), um método geral para edição interativa de objetos definidos por parâmetros sujeitos a restrições lineares ou afins. Neste método, as restrições e as ações de edição do usuário são combinadas usando mínimos quadrados restritos, ao invés da abordagem mais comum de elementos finitos. Usamos aritmética exata para detectar e eliminar redundâncias no conjunto de restrições e evitar falhas devido a erros de arredondamento.

O algoritmo ECLES tem diversas aplicações. Entre elas, podemos citar a edição de deformações spline com continuidade C^1 . Nesta tese, descrevemos um método interativo de edição de deformações do plano, o algoritmo 2DSD (*2D Spline Deformation*). As deformações são definidas por splines de grau 5 sobre uma malha triangular arbitrária. Estas deformações são editadas alterando-se as posições dos pontos de controle da malha. O algoritmo ECLES é usado em cada ação de edição do usuário para detectar, de forma robusta e eficiente, o conjunto de restrições de continuidade C^1 que são relevantes, garantindo que não existam redundâncias. Em seguida, como os parâmetros são modificados pelo usuário, o ECLES é chamado para calcular as novas posições dos pontos de controle satisfazendo as restrições e as posições especificadas pelo usuário.

A fim de validar nosso método 2DSD, ele foi utilizado como parte de um editor interativo para deformações do espaço 2.5D, o editor PrisMystic. Este editor foi utilizado, principalmente, para deformar modelos tridimensionais de organismos microscópicos não-rígidos de modo a coincidir com imagens reais de microscopia ótica. Também utilizamos o editor para editar modelos de terrenos.

Abstract

In this thesis, we present the ECLES algorithm (*Editing by Constrained LEast Squares*), a general method for interactive editing of objects that are defined by parameters subject to linear or affine constraints. In this method, the constraints and the user editing actions are combined using constrained least squares instead of the usual finite element approach. We use exact integer arithmetic in order to detect and eliminate redundancies in the set of constraints and to avoid failures due to rounding errors.

The ECLES algorithm has various applications. Among them, we can cite the editing of C^1 -continuous spline deformations. In this thesis, we describe an interactive editing method for deformations of the plane, the 2DSD algorithm (*2D Spline Deformation*). The deformations are defined by splines of degree 5 on an arbitrary triangular mesh. The deformations are edited by changing the positions of its control points. The ECLES algorithm is first used in each user editing action in order to detect, in a robust and efficient way, the set of relevant constraints of C^1 continuity, ensuring that there are no redundancies. Then, as the parameters are changed by the user, ECLES is called to compute the new positions of the control points satisfying the constraints and the positions specified by the user.

To validate our 2DSD algorithm, we used it as part of an interactive editor for 2.5D space deformations, the PrisMystic editor. This editor has been used, mainly, to deform 3D models of non-rigid living microscopic organisms as seen in actual optical microscope images. We also used the editor to edit terrain models.

List of Figures

1.1	(a) The morphology of the protozoan <i>Dileptus anser</i> ; (b) a control mesh surrounding a 3D model of that protozoan; (c) an actual optical microscope image; and (d) the control mesh and model deformed so as to match an actual image using our PrisMystic editor.	21
2.1	Example of a parameter editing action in a graphical editor as described in Part II. The five parameters are points of $\mathbb{R}^2$ (dots). There is one constraint represented by the gray quadrilateral, that involves its four vertices; namely, the quadrilateral must always be a parallelogram. The anchor set is $\mathcal{A} = \{0\}$ (open bold dot), and the derived set is $\mathcal{D} = \{1, 2, 3\}$ (black dots). (a) User-specified change of the anchor point p_0 to p'_0 ; (b) desired positions p'_1 , p'_2 , and p'_3 of the derived points; and (c) final positions p''_1 , p''_2 , and p''_3 of the derived points forced by the constraint.	24
2.2	Example of the classification of $n = 14$ parameters $\mathcal{P}$ and $m = 5$ constraints $\mathcal{R}$. The rectangles represent the constraints of the matrix R . The black dots are the nonzero elements of R . Note that equations 1 and 3 are not relevant since they do not involve any parameters of $\mathcal{A}$ or $\mathcal{D}$. Note also that the parameters 2, 4, 6, 8, 10, and 11 are fixed but not relevant.	25
2.3	(a) Identifying non-redundant C^1 constraints (gray quadrilaterals) in the 2DSD algorithm (see Part II) with floating-point, and (b) the exact arithmetic. Note that the floating-point version discarded one constraint which in fact is not redundant.	26
5.1	Interaction model between application and the ECLES algorithm.	32
6.1	Invariant I2 of Algorithm 6. The gray parts are nonzero elements.	39
7.1	Bit size growth in the trivial factorization without simplification for random matrices as a function of the number of rows m . The vertical axis is the maximum bit size among all entries observed by factoring 1000 matrices with $n = 15$ columns and varying number m of rows, with randomly chosen 10-bit signed integer elements. Note that the bit size stops growing when m exceeds n . The slight decrease for $m > n$ is due to the better chances of finding a small pivot as m increases.	43
7.2	Bit size growth with the GCD simplification methods for random matrices as a function of the number of rows m . The vertical axis is the maximum bit size among all entries observed by factoring 1000 matrices varying the size $m = n$ of rows and columns, with randomly chosen 10-bit signed integer elements.	44

7.3	Comparison of the bit size growth between the GCD and Turner simplification methods for random matrices as a function of the number of rows m . The vertical axis is the maximum bit size among all entries observed by factoring 1000 matrices with $n = 20$ columns and varying number m of rows, with randomly chosen 10-bit signed integer elements.	45
7.4	Comparison of the bit size growth between the GCD and Turner simplification methods for random rank deficient matrices as a function of the rank r . The vertical axis is the maximum bit size among all entries observed by factoring 1000 20×20 matrices varying the deficient rank r , with 10-bit signed integer elements.	46
7.5	Comparison of the bit size growth between the GCD and Turner simplification methods for random sparse matrices as a function of the number of rows m . The vertical axis is the maximum bit size among all entries observed by factoring 1000 sparse matrices varying the size $m = n$ of rows and columns, with densities $h \approx 0.10$ and 10-bit signed integer elements.	47
7.6	Comparison of the bit size growth between the GCD and Turner simplification methods for random sparse matrices as a function of the number of rows m . The vertical axis is the maximum bit size among all entries observed by factoring 1000 sparse matrices varying the size $m = n$ of rows and columns, with densities $h \approx 0.25$ and 10-bit signed integer elements.	47
7.7	Comparison of the bit size growth between the GCD and Turner simplification methods for random sparse matrices as a function of the density h . The vertical axis is the maximum bit size among all entries observed by factoring 1000 20×20 sparse matrices varying the density h , with 10-bit signed integer elements.	48
7.8	Comparison of the bit size growth between the GCD and Turner simplification methods for random rank deficient sparse matrices as a function of the rank r . The vertical axis is the maximum bit size among all entries observed by factoring 1000 sparse matrices varying the size $m = n$ of rows and columns and the rank deficient r , with densities $h \approx 0.10$ and 10-bit signed integer elements.	48
7.9	Comparison of the bit size growth between the GCD and Turner simplification methods for random rank deficient sparse matrices as a function of the rank r . The vertical axis is the maximum bit size among all entries observed by factoring 1000 sparse matrices varying the size $m = n$ of rows and columns and the rank deficient r , with densities $h \approx 0.25$ and 10-bit signed integer elements.	49
7.10	Comparison of the bit size growth between the GCD and Turner simplification methods for random rank deficient sparse matrices as a function of the density h . The vertical axis is the maximum bit size among all entries observed by factoring 1000 20×20 sparse matrices varying the density h and the rank deficient r , with 10-bit signed integer elements.	49
10.1	A comparison between space deformations (a) without and (b) with C^1 continuity.	59
10.2	A soft translation with 2 anchor points (black open dots) and derived points (black dots) (a) and the result of displacing the anchor points by the vector v (b).	60

12.1	Nominal positions u_{ijk} (dots) of the Bézier coefficients c_{ijk} for the Bernstein-Bézier B_{ijk}^u of degree 5 relative to a triangle, and the local Bézier control net (solid and dashed lines).	64
12.2	A deformation of $\mathbb{R}^2$ of the (a) reference mesh T in the (b) deformed reference mesh $\phi(T)$	64
12.3	(b) Bézier control points q_{ijk}^u of a degree 3 patch ϕ^u from $\Omega \rightarrow \mathbb{R}^2$ and; (a) their nominal positions u_{ijk} on the domain triangle u . The curved triangle on the right is the image of u under the deformation ϕ^u	65
12.4	(a) Nominal positions and (b) Bézier control points of a deformation ϕ of degree 3 which satisfy C^0 continuity constraints.	66
12.5	(a) Nominal positions and (b) Bézier control points of a deformation ϕ of degree 3 which satisfy C^1 continuity constraints (the gray diamonds). . . .	67
12.6	A reference mesh T for a spline deformation of degree 5, showing the Bézier control points and the quadrilateral conditions.	67
13.1	Labels of the triangles of the reference mesh T that shared the oriented edge e , and their vertices.	68
13.2	Labels of the Bézier control points that form the quadrilateral conditions on the shared edge e of the reference mesh T , for degree $d = 5$	69
13.3	A spline with consistent orientation of the quadrilaterals around the vertices.	70
13.4	Labels of the control points to obtain a consistent orientation of the quadrilaterals around the vertices.	71
14.1	Control flow for a typical editing action (soft translation).	72
14.2	Translating of two anchor points. (a) Anchor points (set $\mathcal{A}$) specified by the user; (b) initial derived points (set $\mathcal{S}$) selected by the user; and (c) derived points (set $\mathcal{D}$) specified by the <code>2DSD.ExpandDerived</code>	73
14.3	Translating of two anchor points. (a) Anchor points $\mathcal{A}$ (black open dots), derived points $\mathcal{D}$ (black dots), and non-redundant relevant constraints specified by ECLES; and (b) control points updated.	74
14.4	Examples of (a) rotation and (b) scaling of one anchor point.	74
14.5	Classification of the control points of a Bézier patch of degree 6.	76
14.6	Relevant continuity constraints (gray diamonds) and derived points added to $\mathcal{D}$ (solid dots) depending on the type of the control point p_s with $s \in \mathcal{A} \cup \mathcal{D}$ (open dot). For each type, the left figure shows a typical situation where the point p_s is sufficiently far from the triangulation's border. The right figure shows a situation near the border where some of those control points and constraints are missing. These diagrams are generalized to vertices of arbitrary degree in the obvious way.	77
14.7	Editing point q_{032}^u with the derived points q_{122}^v , q_{122}^u , q_{131}^v , q_{131}^u	79
15.1	(a) An actual microscope image of the protozoan <i>C. elegans</i> ; (b) 2D view; and (c) 3D view of a deformed model obtained with PrisMystic, matching the image (a).	81
17.1	The 3D model mesh M of a <i>Dragon</i> [82], represented by a triangular mesh.	86
17.2	The 3D model mesh M of a <i>Dragon</i> [82], represented by a point cloud.	86
17.3	A 3D model mesh M surrounded by a 3D reference mesh P , and the corresponding 2D reference mesh T	87

17.4	Barycentric coordinates $\alpha_0, \alpha_1, \beta_0, \beta_1$ and β_2 of a point p of the 3D model mesh M , related to the prism U of the 3D reference mesh P	87
17.5	A deformed reference mesh $\psi(P)$ surrounding a deformed model mesh $\psi(M)$	88
17.6	View of the deformed reference mesh $\psi(P)$ in the xy -mode, showing the control points and the global Bézier control net G (dotted line) on the deformed reference mesh $\phi(T)$	89
17.7	View of the deformed reference mesh $\psi(P)$ in the $z0$ -mode (a) and $z1$ -mode (b), showing the control points of the splines σ_0 and σ_1 , respectively.	89
18.1	The user interface of the PrisMystic editor in the initialization mode.	91
18.2	The options of the Editor Settings window of the PrisMystic editor.	92
18.3	The options of the Editing Control window of the PrisMystic editor.	93
18.4	The view mode of the PrisMystic editor.	94
18.5	The PrisMystic editor highlighting (a) the selected anchor points, and (b) the initial derived points with $\delta_{max} = 3$, in the xy -mode.	95
18.6	The PrisMystic editor highlighting (a) the initial derived points, and (b) the scaling operation applied to all control points.	95
18.7	The editing xy -mode of the PrisMystic editor.	96
18.8	(a) Before and (b) after a soft translation of two anchor points (black open dots) with $\delta_{max} = 4$, showing the derived points $\mathcal{D}$ (black dots), and the C^1 continuity constraints (quadrilaterals).	96
18.9	(a) Before and (b) after a soft rotation of one anchor point (black open dot) with $\delta_{max} = 8$ around the center c , showing the derived points $\mathcal{D}$ (black dots), and the C^1 continuity constraints (quadrilaterals).	97
18.10	(a) Before and (b) after a soft scaling of one anchor point (black open dot) with $\delta_{max} = 6$ around the center c , showing the derived points $\mathcal{D}$ (black dots), and the C^1 continuity constraints (quadrilaterals).	97
18.11	The editing modes, (a) $z1$ -mode and (b) $z0$ -mode, of the PrisMystic editor.	98
18.12	(a) Before and (b) after the editing of one anchor point with $\delta_{max} = 3$ in $z1$ -mode, showing the control points and the derived points. The relevant constraints and the global Bézier control net G of the reference mesh are also shown in (a).	98
19.1	Morphology of the organism (left). The 2D view (middle) and 3D view (right) of the reference mesh, and the organism models in a typical resting shape.	100
19.2	Actual microscope images of the nematode <i>C. elegans</i> (left); 2D view (middle); and 3D view (right) of the deformed models.	101
19.3	Actual microscope images of the protozoan <i>Dileptus anser</i> (left); 2D view (middle); and 3D view (right) of the deformed models.	102
19.4	Actual microscope images of the <i>Lacrymaria olor</i> (left); 2D view (middle); and 3D view (right) of the deformed models.	103
19.5	Actual images of the starfish <i>Asterias rubens</i> (left); 2D view (middle); and 3D view (right) of the deformed models.	104
19.6	(a) 2D view and (b) 3D view of the model of a digital terrain and its reference mesh; and (c) 2D view of the deformed model using the PrisMystic editor.	105
19.7	3D view in Blender editor [83] of the deformed model shown in Figure 19.6.	105

List of Tables

16.1 Comparison among cells of meshes consisting of tetrahedra, hexahedra, and prisms.	85
19.1 Parameters of the model mesh M and the reference mesh T for the organisms used in the tests.	99
19.2 Parameters of the model mesh M and the reference mesh T used in the test.	105

List of Algorithms

1	ECLES.Initialize	33
2	ECLES.ExtractRelevant	34
3	ECLES.CheckStrongSolvability	35
4	ECLES.Update	36
5	ECLES.CheckWeakSolvability	36
6	LinSys.LDUFactor	38
7	LinSys.Pivot	39
8	LinSys.EliminateVariable	40
9	LinSys.SimplifyURow	40
10	LinSys.SimplifyLColumn	41
11	LinSys.SimplifyURowGCD	44
12	LinSys.SimplifyLColumnGCD	44
13	LinSys.SimplifyURowTurner	46
14	LinSys.Solve	51
15	LSQ.Solve	56
16	2DSD.Select	75
17	2DSD.Translate	78

Contents

1	Introduction	19
1.1	Part I: The ECLES Algorithm	19
1.2	Part II: The 2DSD Algorithm	20
1.3	Part III: The PrisMystic Editor	20
1.4	Contributions	21
I	The ECLES Algorithm	22
2	General Parameter Editing	23
2.1	Statement of the problem	23
2.2	Relevant equations and fixed parameters	24
2.3	Solvability condition	25
2.4	The need for exact computation	25
3	Related Work	27
3.1	Constraint-based editing and modeling	27
3.2	Finite element basis	27
3.3	Optimization	28
3.3.1	Least squares	28
4	Basic Tools	29
4.1	Fraction-free LDU factoring	29
4.2	Using the hints	30
4.3	Redundant equations	31
4.4	Multidimensional parameters	31
5	The ECLES Method	32
5.1	Simplified description	32
5.2	The ECLES.Initialize procedure	33
5.2.1	The ECLES.ExtractRelevant procedure	34
5.2.2	The ECLES.CheckStrongSolvability procedure	34
5.3	The ECLES.Update procedure	35
5.3.1	The ECLES.CheckWeakSolvability procedure	36
6	Fraction-Free LDU Factoring	37
6.1	The main algorithm	37
6.2	Pivoting	39
6.3	Variable elimination	40

6.4	Row and column simplification	40
6.5	Computing cost	41
7	Simplification Techniques for LDU Factoring	42
7.1	Plain fraction-free Gaussian elimination	42
7.2	Simplifying by GCD elimination	43
7.3	Turner's GCD-free simplification	45
7.3.1	Bit size growth for rank deficient matrices	46
7.3.2	Bit size growth for random sparse matrices	47
7.3.3	Bit size growth for sparse matrices with deficient rank	48
7.3.4	Discussion about the results	49
8	Solving Exact Linear Systems	50
8.1	Solving the system	50
8.2	Checking consistency	52
8.2.1	Weak solvability	52
8.2.2	Strong solvability	52
8.3	An example	53
9	Solving the Least Squares Problem	55
9.1	Constrained least squares	55
9.2	An example	57
II	The 2DSD Algorithm	58
10	Interactive Editing of 2D Spline Deformations	59
10.1	Statement of the problem	59
10.1.1	Deformations	59
10.1.2	Meshes and splines	60
10.2	User interface	60
11	Related Work	61
11.1	Non-spline methods	61
11.2	Spline methods	62
12	Triangular Splines Deformation	63
12.1	Triangular Bézier splines	63
12.2	Using splines to model deformations	64
12.2.1	Continuity constraints	65
12.3	Local control	67
13	Spline Representation	68
13.1	Notation	68
13.1.1	Labeling and orientation of the edges	68
13.1.2	Labeling and orientation of the quadrilateral conditions	69
13.2	Data structure	69
13.2.1	Representation of the reference mesh	69
13.2.2	Representation of the spline	70
13.2.3	Representation of the C^1 constraints	70

14 The 2DSD Editing Algorithm	72
14.1 The user interaction model	72
14.1.1 Soft translation	73
14.1.2 Soft rotation and scaling	74
14.2 The 2DSD.Select procedure	75
14.2.1 The 2DSD.ExpandDerived procedure	76
14.2.2 The 2DSD.ComputeRelMagnitude procedure	78
14.3 The 2DSD.Translate procedure	78
14.4 An example	79
 III The Prismystic Editor	 80
15 Goals and Motivation	81
15.1 Goals	81
15.2 Relation to the Masters version	82
16 Related work	83
16.1 Deformation of 3D models	83
16.2 Space deformations	84
16.3 Interpolation techniques	84
16.4 Spline interpolation	85
17 Overview of the editor	86
17.1 The 3D model	86
17.2 The 3D reference mesh	87
17.2.1 Defining the barycentric coordinates	87
17.3 Deformation paradigm	88
17.4 Editing the deformation	89
17.5 Ensuring the spline continuity	90
18 Editing Paradigm	91
18.1 Prismystic user interface	91
18.1.1 <i>Editor Settings</i> window	92
18.1.2 <i>Editing Control</i> window	93
18.2 Initialization mode	93
18.3 View mode	93
18.4 Point selection sub-mode	94
18.4.1 Anchor selection	94
18.4.2 Neighborhood selection	94
18.5 Editing <i>xy</i> -deformation	95
18.5.1 Local soft translation	96
18.5.2 Local soft rotation	97
18.5.3 Local soft scaling	97
18.6 Editing <i>z</i> -deformation	98

19 Examples	99
19.1 Deformation of organism models	99
19.1.1 Results	100
19.2 Deformation of terrain models	105
20 Conclusion and Future Work	106
20.1 Part I	106
20.2 Part II	106
20.3 Part III	107
Bibliography	108
A Implementation	115
A.1 The LinSys library	115
A.1.1 Fraction-free matrix factoring	116
A.1.2 Solving linear system	116
A.2 The LSQ library	116
A.3 The ECLES library	116
A.3.1 Initializing	117
A.3.2 Updating	117
A.4 The 2DSD library	117
A.4.1 Editing the deformation	117
A.4.2 Deforming the spline	118

Chapter 1

Introduction

In this thesis, our objective is to develop mathematical and software tools for interactive editing of parameters of some model or process, subject to linear or affine constraints, in a general, robust, and efficient way.

This problem occurs in many applications of the areas of computer graphics and image processing, such as the geometric modeling and deformation, 2D and 3D spline modeling, image morphing, registration and vectorization, CAD, among others [42, 80, 91, 97]. Other possible applications include control of industrial process and power grids [19].

In Part I, we develop a general method for solving this problem. In Part II, we apply this method to the specific problem of creating and editing two-dimensional spline deformations subject to smoothness constraints. In Part III, we describe an editor of 2.5D space deformations for three-dimensional solid modeling using the method of Part II.

1.1 Part I: The ECLES Algorithm

In Part I, we consider the abstract numerical problem of interactive editing of parameters of some model or process, subject to affine constraints, in a general way.

We assume that the application has a finite list of real valued parameters $p_1, p_2, \dots, p_n$ that are subjected to a finite set of affine equality constraints; that is, polynomial equations of degree 1, like $3p_3 - 5p_8 = 7$.

We assume that, at each editing action, the user specifies new required values for some subset $\mathcal{A}$ of parameters (the *anchors*), and desirable but not required “hint” values for another subset $\mathcal{D}$ (the *derived* parameters). The application is then supposed to adjust the parameters in $\mathcal{D}$ in order to preserve the constraints, the required values for the anchors, and be as close as possible to the hints. An important feature of ECLES is that the sets $\mathcal{A}$ and $\mathcal{D}$ are not fixed, but are specified by the application at each editing action.

To solve the stated problem, we propose a general and robust method, that we call ECLES (*Editing by Constrained LEast Squares*). The ECLES algorithm uses linear system solving with exact integer arithmetic, to detect and eliminate redundancies among the constraints that are relevant to each editing action and to avoid failures due to rounding. It uses weighted and constrained least squares to minimize the discrepancy between the hints and the values for the set $\mathcal{D}$.

1.2 Part II: The 2DSD Algorithm

In Part II, we apply the ECLES algorithm to a specific application: the interactive editing of smooth simplicial spline deformations of the plane. We describe an algorithm for this application, called 2DSD (*2D Spline Deformation*).

The deformation is modeled by a spline defined on a triangular mesh. The desired deformation is specified by manipulating the Bézier control points of the spline. Each editing operation requires the automatic adjustment of several other nearby control points in order to preserve the smoothness of the spline. The set of adjusted points is determined at the time of editing through the 2DSD algorithm. We use the ECLES general parameter editing method in order to compute the new positions of the control points selected under the condition that the smoothness constraints are satisfied.

The 2DSD method can be adapted to other applications such as editing 3D spline surfaces [13, 94], editing 2.5D and 3D space deformation [31], image morphing [42, 53, 63, 91] and registration [73, 80, 97] and interactive image vectorization [92].

The main advantage of splines over other function interpolation methods is that they allow local control: if we change only one control point, the spline changes only within the corresponding cell of the mesh and perhaps a few other triangles surrounding it [25]. Simplicial (triangular or tetrahedral) Bézier patches have the advantage over quadrangular patches since they can be joined with almost arbitrary topology. On the other hand, their continuity constraints are more complicated.

Since the deformation is applied to a control mesh that deforms space, rather than to the object mesh directly, the edited deformation can be applied to other models. This method is independent of the resolution and representation of the object to be deformed. Another advantage is that the deformed control mesh provides an immediate intuitive understanding of the general nature of the deformation, and of the scope of each control parameter.

Unlike radial basis methods [17, 20], the 2DSD description of the deformation remains simple and of finite size even after an arbitrarily long sequence of editing operations. At each point of the domain, except in cell boundaries, the deformation has a simple analytic formula that allows the efficient computation of derivatives.

1.3 Part III: The PrisMystic Editor

In Part III, we used the 2DSD algorithm to implement an editor of 2.5D space deformations [31, 67] to deform three-dimensional models, called PrisMystic editor.

The PrisMystic is an effective and user-friendly editor that can be used, for example, to reproduce deformations of 3D models of non-rigid cells and other organisms viewed through optical microscopes. See Figure 1.1.

The PrisMystic editor is an evolution and generalization of the editor described in my Masters dissertation [67]. The improvements include: using the ECLES algorithm, described in Part I, instead of floating-point linear algebra packages; a more flexible and general method for the selection of control points (allowing multiple anchors and more derived points); and a different goal function for the least squares method. Also, we used

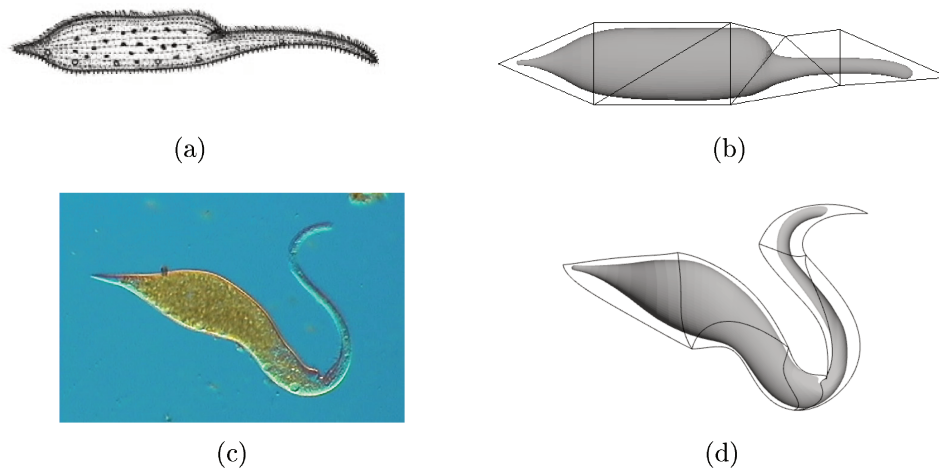


Figure 1.1: (a) The morphology of the protozoan *Dileptus anser*; (b) a control mesh surrounding a 3D model of that protozoan; (c) an actual optical microscope image; and (d) the control mesh and model deformed so as to match an actual image using our PrisMystic editor.

the general 2DSD approach, described in Part II, to connect the user interface to the ECLES solver, described in Part I. With these changes, it would be now relatively easy to extend the editor to accommodate other affine constraints, such as C^2 smoothness, vertical or horizontal alignment, fixed points, etc.

1.4 Contributions

The original contributions of this thesis are:

- a rigorous approach to parameter editing that combines exact integer arithmetic and weighted and constrained least squares for linear system solving;
- a robust and general algorithm (ECLES) for local editing of parameters with linear constraints;
- an algorithm (2DSD) for editing smooth two-dimensional spline deformations;
- an editor (PrisMystic) for 2.5D deformation of 3D solid models.
- public C/C++ implementations of ECLES, 2DSD, and PrisMystic including libraries for linear system solving using exact integer arithmetics and least squares optimization [22].

Some of these contributions were presented in conferences [71, 72], and published as technical reports [70]. The PrisMystic editor is an improved and expanded version of the editor developed in my Masters dissertation [67, 68, 69]

Part I

The ECLES Algorithm

Chapter 2

General Parameter Editing

In this chapter, we describe the problem of interactive editing of objects that are defined by parameters subject to linear or affine constraints. We present also a discussion about the solvability condition of the problem, and how the general problem can be reduced to the relevant parameters and constraints.

2.1 Statement of the problem

The problem of general parameter editing involves a set of variables (the *control parameters*) of an application subject to a fixed set of linear or affine equations (the *constraints*).

More specifically, let $p_1, p_2, \dots, p_n$ be the n control parameters, and p be the vector of the values of those parameters. We can write the constraints as a matrix equation

$$Rp = q \tag{2.1}$$

where R is a constant $m \times n$ coefficient matrix, and q is a vector of m constants (possibly zero).

For example, a chemical process may have four pumps whose flow rates p_1, p_2, p_3 , and p_4 must always satisfy $p_1 + p_2 = p_3$ and $p_4 = p_3 + 5$, that is

$$\begin{bmatrix} 1 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}. \tag{2.2}$$

Any change in one of the four rates must then be simultaneously compensated by a change in one or both of the other three.

Let $\mathcal{P} = \{1, 2, \dots, n\}$ be the set of parameter indices. For each editing action, the user (or the application) must define two disjoint subsets of $\mathcal{P}$:

- $\mathcal{A}$ (*anchor*): the indices of one or more parameters whose values will be set by the user;
- $\mathcal{D}$ (*derived*): the indices of zero or more parameters that may be adjusted if necessary to satisfy the constraints.

We denote by $\mathcal{F}$ the set $\mathcal{P} \setminus (\mathcal{A} \cup \mathcal{D})$, the *fixed* parameters whose values are not to be changed. For each parameter $s \in \mathcal{A}$, the user then specifies a new value p'_s which is mandatory. For each $s \in \mathcal{D}$, a new value p'_s is also suggested. The editing algorithm then computes a new value p''_s for each parameter. If $s \in \mathcal{A}$, p''_s will be equal to the given value p'_s . If $s \in \mathcal{D}$, p''_s is close to p'_s , but not necessarily equal to it. For every $s \in \mathcal{F}$ the value does not change, that is, the desired value p'_s and final value p''_s are equal to the current value p_s .

More generally, each parameter may be an element of some vector space $\mathbb{R}^d$. In that case, p is an $n \times d$ matrix, and q is an $m \times d$ matrix. This is the case of the application considered in Part II. See Figure 2.1.

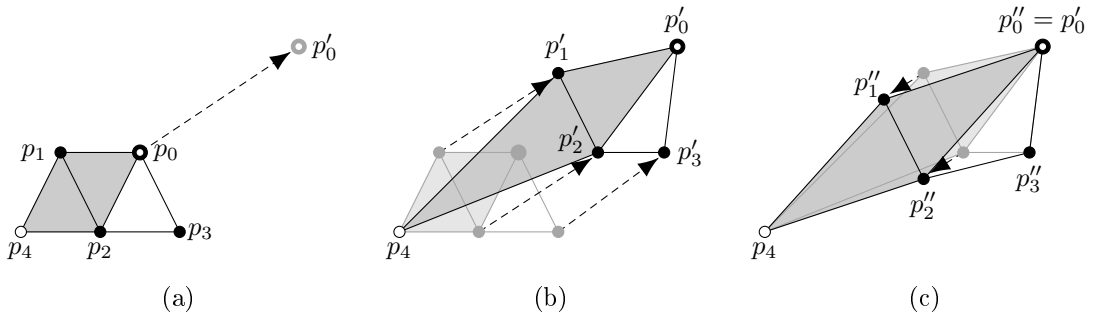


Figure 2.1: Example of a parameter editing action in a graphical editor as described in Part II. The five parameters are points of $\mathbb{R}^2$ (dots). There is one constraint represented by the gray quadrilateral, that involves its four vertices; namely, the quadrilateral must always be a parallelogram. The anchor set is $\mathcal{A} = \{0\}$ (open bold dot), and the derived set is $\mathcal{D} = \{1, 2, 3\}$ (black dots). (a) User-specified change of the anchor point p_0 to p'_0 ; (b) desired positions p'_1 , p'_2 , and p'_3 of the derived points; and (c) final positions p''_1 , p''_2 , and p''_3 of the derived points forced by the constraint.

2.2 Relevant equations and fixed parameters

The constraints that involve parameters of $\mathcal{A}$ or $\mathcal{D}$ are called *relevant constraints*. The indices of these constraints comprise the set $\mathcal{E}$, a subset of $\mathcal{R} = \{1, 2, \dots, m\}$. We define the set $\mathcal{F}'$ of *fixed relevant* parameters as comprising the indices $s \in \mathcal{F}$ such that p_s occurs in some equation of $\mathcal{E}$. See Figure 2.2.

For any subset $\mathcal{Y}$ of $\mathcal{P}$, we will denote by $p_{\mathcal{Y}}$ the subvector of $\mathcal{P}$ whose elements are all elements p_s with $s \in \mathcal{Y}$. Similarly, for any subset $\mathcal{X} \in \mathcal{R}$, we denote by $R_{\mathcal{X}\mathcal{Y}}$ the sub-matrix of R consisting of the elements R_{ij} with $i \in \mathcal{X}$ and $j \in \mathcal{Y}$.

We can replace the full constraint system (2.1) by the smaller system

$$R_{\mathcal{E}\mathcal{D}} p''_{\mathcal{D}} = q_{\mathcal{E}} - R_{\mathcal{E}\mathcal{A}} p'_{\mathcal{A}} - R_{\mathcal{E}\mathcal{F}'} p_{\mathcal{F}'} \quad (2.3)$$

which represents the relevant constraints $\mathcal{E}$. We can write this system as $Ax = b$, where $A = R_{\mathcal{E}\mathcal{D}}$, $x = p''_{\mathcal{D}}$ and $b = q_{\mathcal{E}} - R_{\mathcal{E}\mathcal{A}} p'_{\mathcal{A}} - R_{\mathcal{E}\mathcal{F}'} p_{\mathcal{F}'}$.

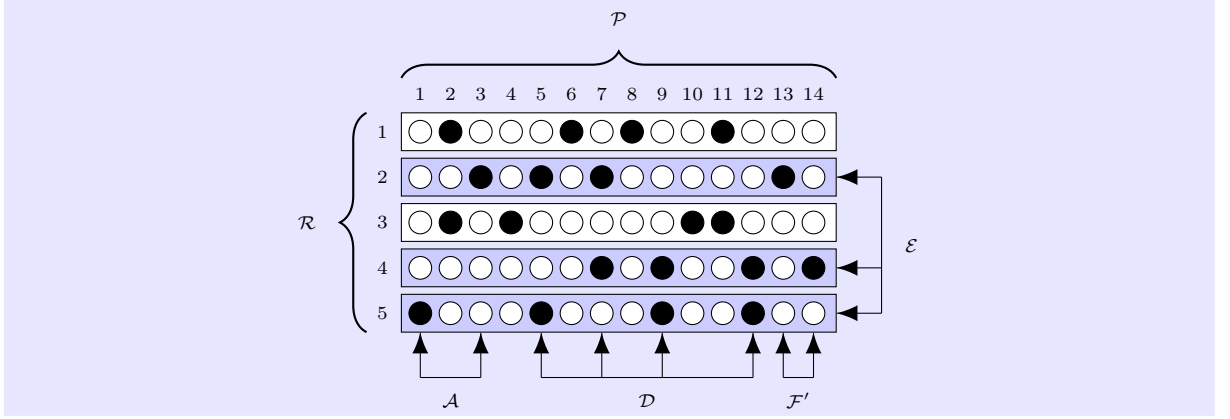


Figure 2.2: Example of the classification of $n = 14$ parameters $\mathcal{P}$ and $m = 5$ constraints $\mathcal{R}$. The rectangles represent the constraints of the matrix R . The black dots are the nonzero elements of R . Note that equations 1 and 3 are not relevant since they do not involve any parameters of $\mathcal{A}$ or $\mathcal{D}$. Note also that the parameters 2, 4, 6, 8, 10, and 11 are fixed but not relevant.

2.3 Solvability condition

Depending on the choices of $\mathcal{A}$ and $\mathcal{D}$, and of the new values p'_s , the problem may or may not have a solution.

We say that a given choice of $\mathcal{A}$ and $\mathcal{D}$ is *strongly solvable* if for any parameter vector p that satisfies all constraints, and any assignment of values for $p'_{\mathcal{A} \cup \mathcal{F}}$ there is a solution p'' , that satisfies all constraints, with $p''_{\mathcal{A}} = p'_{\mathcal{A}}$ and $p''_{\mathcal{F}} = p'_{\mathcal{F}}$. That is, the set $\mathcal{D}$ of derived parameters must be large enough to allow any combination of new values to be assigned to the control parameters in $\mathcal{A}$, while satisfying all constraints by assigning appropriate new values to all parameters in $\mathcal{D}$.

We say the problem is *weakly solvable* if such a solution exists for the particular given values $p'_{\mathcal{A}}$ and the current values of $p'_{\mathcal{F}}$.

2.4 The need for exact computation

The problem of editing parameters with linear constraints may involve redundant equations, which need to be detected and eliminated to correctly solve the system. For example, we may have $p_1 + 2p_2 = 0$, $p_2 + p_3 = 5$, $2p_3 - p_1 = 10$. Note that the last equation is a linear combination of the first two.

In general, a constraint-solving algorithm cannot assume that the application is tolerant to rounding errors. The presence of these errors, in approaches that use floating-point arithmetic, can generate numerical instabilities causing failures in the reliable detection of redundancies.

It is possible to identify and ignore the redundant constraints by using exact arithmetic to solve the linear systems. For this task, we use the *fraction-free LDU factoring*, described in Chapter 4. Figure 2.3 shows an example of application of the 2DSD algorithm, described in Part II of this thesis. In this example, the use of exact arithmetic (the

ECLES algorithm, described in Chapter 5) removed the rounding errors and numerical instabilities of the previous floating-point implementation.

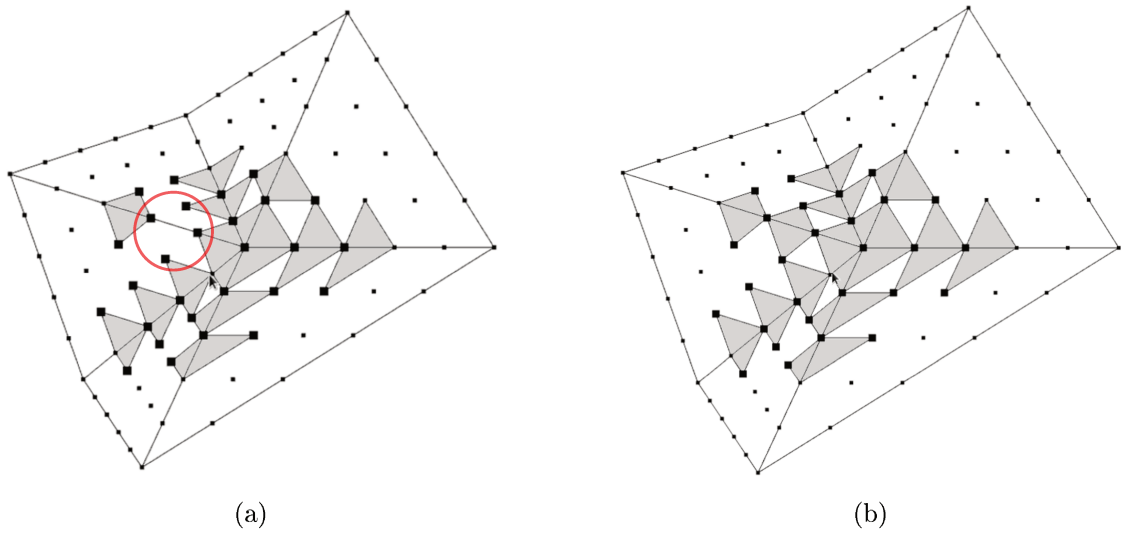


Figure 2.3: (a) Identifying non-redundant C^1 constraints (gray quadrilaterals) in the 2DSD algorithm (see Part II) with floating-point, and (b) the exact arithmetic. Note that the floating-point version discarded one constraint which in fact is not redundant.

Therefore, we assume that the constraint equations have integer coefficients (rounded by the application, as appropriate) and the parameter values are approximated by rational values. For the exact and rational computations, we used the library FLINT (Fast Library for Number Theory) [36].

Chapter 3

Related Work

There are many systems for interactive editing of objects under linear and affine constraints [10, 38, 47, 62, 79]. In this chapter, we present an overview about some techniques, described in the literature, for parameter editing with constraints.

3.1 Constraint-based editing and modeling

Typical constraint-based interactive editors recompute all parameters at each user editing request, and use penalty terms in order to minimize the changes to non-edited parameters.

The METAFONT font design system, developed by Knuth in 1979 [47], can be viewed as a precursor of constraint-based parameter editing. It is a programming language where function parameters may be subjected to linear and affine constraints. The language lets the user specify any subset of independent parameters, automatically solving for the others when enough parameters have been specified. However, it was not interactive.

In 1985, Nelson [62] described Juno, a constraint-based interactive editor for 2D drawings. Later, an extended version of the editor (Juno-2) was developed by Heydon and Nelson [38]. With Juno-2, the user can graphically define constraints, which are solved by a non-linear equation solver combined with some symbolic techniques. The user could define hints for unknown parameters.

3.2 Finite element basis

For linear and affine constraints (which we consider in this thesis), a common approach is to pre-compute a *finite element basis* of parameter change vectors that has small support and spans the space of all possible changes allowed by the constraints; and then give the user a separate “knob” for each finite element.

A typical example is the editing of splines with specified continuity. Each basis element has exactly one editable value (typically a *Bézier control point* or *value*), the other Bézier points are then computed from those. This approach was extensively studied by Schumaker [50] and others [93].

One particular case of the finite element technique is the B-spline approach [25], where there is only one control point for each patch, and the resulting spline is automatically

continuous to the maximum possible order. B-splines are well defined for tensor (quadrangular or hexahedral) type meshes with regular topology [18, 21]; extending them to irregular and simplicial meshes is possible, but rather complicated [37].

One serious limitation of finite element technique is that the finite element basis must be recomputed every time the space of allowed solutions change; that is, every time the sets $\mathcal{A}$ of anchors and $\mathcal{D}$ of derived parameters change.

3.3 Optimization

Since the constraints are usually under-determinate, the final solution must be chosen by optimizing some additional criteria.

Systems like METAFONT [47] build the set of derived parameters $\mathcal{D}$ incrementally by adding parameters to it according to certain ad hoc priority rules. For instance, variables that were most recently included in the set of anchors $\mathcal{A}$ of previous actions have lower priority. Other systems [10, 38, 62] solve for all constraints simultaneously by an iterative algorithm using the current situation p as a starting guess.

3.3.1 Least squares

To handle under-determinate constraints, we instead use the criterion of least squares with first-degree constraints [66] to find the solution $p''_{\mathcal{D}}$ that is closest to the hints $p'_{\mathcal{D}}$.

This technique was used, for instance, by Masuda et al. [57] for surface mesh editing. It allows editing any control point $\mathcal{A} = \{k\}$ and computes the remaining points in $\mathcal{D} = \mathcal{P} \setminus \mathcal{A}$ by solving a linear system that combines the criterion of least squares and the constraint equations.

Our method, described in Chapter 5, differs from Masuda's by assuming that the application provides, at each editing event, a small subset $\mathcal{D}$ of parameters that can be changed. Moreover, we use exact arithmetic to reliably detect inconsistent or redundant constraints.

Least squares were also used, by Sorkine and Cohen-Or [79] to globally approximate points given by 3D triangular meshes. However this technique has no concept of splines and smoothness. The constraints are satisfied only in the sense of least squares.

Chapter 4

Basic Tools

In this chapter, we describe the method used to solve linear systems with exact integer arithmetic. This method detects and eliminates redundancies among the constraints of the problem and avoids failures due to rounding errors. Moreover, we also described the weighted and constraint least squares to minimize the discrepancy between the desired and final solutions of the system.

4.1 Fraction-free LDU factoring

To exactly solve the linear system in Equation (2.3) whose coefficients are integers, we use a *fraction-free LDU factoring* [43] of a rectangular matrix A ($m \times n$), with rank r . It consists of five integer matrices: Π_R ($m \times m$, a permutation matrix of rows), L ($m \times r$), D ($r \times r$, diagonal), U ($r \times n$), and Π_C ($n \times n$, a permutation matrix of columns) such that

$$A = \Pi_R L D^{-1} U \Pi_C. \quad (4.1)$$

Note that, the matrices Π_R and Π_C can be represented as integer vectors to save space. The matrices L and U have specific structures:

$$L = \begin{pmatrix} \widehat{L} \\ \widetilde{L} \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} \widehat{U} & \widetilde{U} \end{pmatrix} \quad (4.2)$$

where $\widehat{L}$ is an $r \times r$ lower triangular matrix, $\widehat{U}$ is an $r \times r$ upper triangular matrix, both invertible, and $\widetilde{L}$, $\widetilde{U}$ are arbitrary integer matrices with sizes $(m - r) \times r$ and $r \times (n - r)$, respectively.

For example, the matrix A below has $m = 5$ rows, $n = 4$ columns, and rank $r = 3$:

$$A = \begin{bmatrix} 4 & 9 & 16 & 29 \\ -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 5 & 6 & -1 & -12 \\ 5 & 10 & 15 & 20 \end{bmatrix} \quad (4.3)$$

Its fraction-free LDU factoring using full pivoting is

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 15 & -80 \\ 5 & 24 & -164 \\ 5 & 20 & -120 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -80 \end{bmatrix}^{-1} \begin{bmatrix} -1 & -6 & -16 & -19 \\ 0 & 1 & -3 & 4 \\ 0 & 0 & -80 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

As detailed in Chapter 6, the factoring enables the exact solution of the system $Ax = b$ for any integer vector b , yielding a rational vector x .

4.2 Using the hints

Besides exact system solving, the other main tool that we use is the *least squares* (*quadratic optimization*) method with affine constraints. This method is used whenever there is more than one solution $p''_{\mathcal{D}}$, to find the one that is ‘closest’ to the given hints $p'_{\mathcal{D}}$.

Given a vector $x' = (x'_1, \dots, x'_n)$ of desired values, we want to find a vector $x'' = (x''_1, \dots, x''_n)$ of values that minimizes the distance between each new value x''_s and the desired value x'_s , while satisfying a set of constraints $Ax'' = b$, where A is any $m \times n$ matrix and b is a vector of m elements. More precisely, we want to find the vector x'' that minimizes the goal function

$$S(x) = \sum_{s=1}^n w_s (x_s - x'_s)^2 \quad (4.4)$$

where the coefficient w_s is a weight that indicates the importance of honoring the hint x'_s (the higher the weight value w_s is, compared to the other weights, the more the algorithm will try to make value x''_s close to x'_s).

This subproblem reduces to solving a linear system that includes the equations $Ax = b$, as detailed in Chapter 9.

4.3 Redundant equations

In many applications, there are often redundant constraints, especially when some of the parameters are considered fixed. Therefore, one of the subproblems that we need to solve is to identify and ignore such redundant constraints.

This task is a side effect of computing the fraction-free LDU factoring of the matrix A of the constraint system $Ax = b$.

If the rank r determined during the factoring is less than m , as in example (4.3), the system $Ax = b$ has redundant equations, and can be replaced by

$$\hat{A}x = \hat{b} \quad (4.5)$$

where $\hat{A}$ is an $r \times n$ matrix which is the first r rows of $\Pi_R^{-1}A = LD^{-1}U\Pi_C$, that is, $\hat{L}D^{-1}U\Pi_C$; and $\hat{b}$ is an r -element column vector, which is the first r rows of $\Pi_R^{-1}b$. For the example 4.3, we have

$$\begin{aligned} \hat{A} &= \begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 15 & -80 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -80 \end{bmatrix}^{-1} \begin{bmatrix} -1 & -6 & -16 & -19 \\ 0 & 1 & -3 & 4 \\ 0 & 0 & -80 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 4 & 9 & 16 & 29 \end{bmatrix} \end{aligned} \quad (4.6)$$

4.4 Multidimensional parameters

As observed in Chapter 2, in some applications each parameter is a vector from some Cartesian space $\mathbb{R}^d$ and each constraint is an affine equation on vectors of $\mathbb{R}^d$. For example, the ECLES algorithm generalizes trivially to this case. The application we consider in Parts II and III has $d = 2$.

Then equations (2.1) and (4.5) can be rewritten as $RP = Q$ and $AX = B$, where P , Q , X , and B are matrices with d columns. This is the case, in particular, of the application we consider in Parts II and III. So, in the rest of this thesis, we make this assumption.

Chapter 5

The ECLES Method

In this chapter, we provide a detailed description of our interactive and general parameter editing method, that we call ECLES (*Editing by Constrained LEast Squares*). This method is suitable for editing any kind of object that is defined by parameters subject to linear or affine constraints.

ECLES solves the problem defined in Chapter 2 computing a set of values $P''_{\mathcal{D}}$ for a given set $\mathcal{D}$ of derived parameters, so as to satisfy a set of linear constraints after the user changes the anchor parameters to given values $P'_{\mathcal{A}}$, and provides hints $P'_{\mathcal{D}}$ for the derived ones.

5.1 Simplified description

The ECLES method consists of two procedures, `ECLES.Initialize` and `ECLES.Update`. The `ECLES.Initialize` procedure, described with more detail in Section 5.2, is called when the user chooses the sets $\mathcal{A}$ and $\mathcal{D}$ of parameters to be adjusted through some editing software (the *user interface*). The `ECLES.Update` procedure is then called one or more times as the user specifies new values $P'_{\mathcal{A}}$ for the anchors, to modify those parameters so that all constraints remain satisfied. See Figure 5.1.

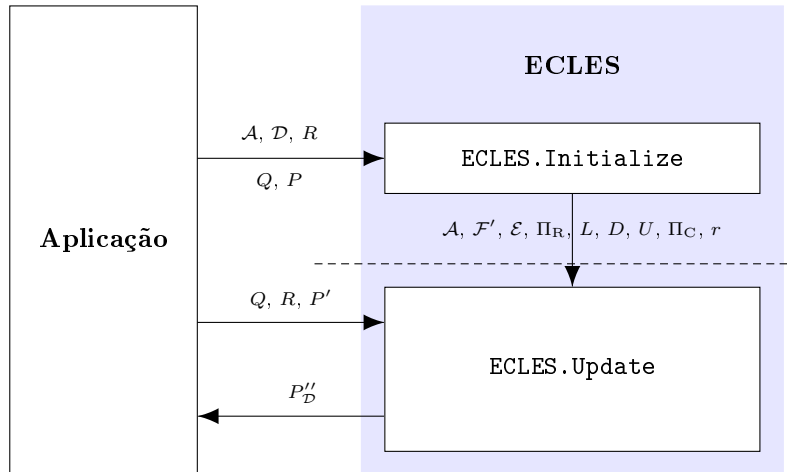


Figure 5.1: Interaction model between application and the ECLES algorithm.

The `ECLES.Initialize` procedure identifies the subset $\mathcal{E} \subseteq \mathcal{R}$ of relevant constraints and the set $\mathcal{F}' \subseteq \mathcal{F}$ of the fixed relevant parameters. This process reduces the editing problem to the relevant parameters $\mathcal{A} \cup \mathcal{D} \cup \mathcal{F}'$ and the relevant constraints $\mathcal{E}$. The `ECLES.Initialize` procedure also checks whether there are redundant constraints in the set $\mathcal{E}$.

Optionally, the procedure also verifies if the sets $\mathcal{A}$ and $\mathcal{D}$, provided by the application, satisfy the strong solvability condition of the ECLES. In any case, it returns the coefficient matrix of the non-redundant constraints, in factored form, to be used by the `ECLES.Update` procedure.

Basically, the `ECLES.Update` procedure, described with more detail in Section 5.3, solves the linear system that combines the set of non-redundant relevant constraint, the specified anchor values P'_A , and the hints P'_D , and computes the derived values P''_D .

5.2 The `ECLES.Initialize` procedure

After defining the set $\mathcal{E}$, `ECLES.Initialize` extracts the relevant equations and rewrites them in the form of the system (2.3).

In general, there may be redundancies in system (2.3). The `ECLES.Initialize` procedure computes the fraction-free LDU factoring (described in Section 4.1) of the matrix $R_{\mathcal{ED}}$ into integer matrices L, D, U , and permutation matrices Π_R and Π_C , such that $R_{\mathcal{ED}} = \Pi_R L D^{-1} U \Pi_C$. In the process, it obtains the rank r of $R_{\mathcal{ED}}$. The first r rows of $\Pi_R^{-1} R_{\mathcal{ED}}$ are a set of non-redundant equations. These steps are formalized in Algorithm 1.

Algorithm 1: `ECLES.Initialize`

Input: $\mathcal{A}$: set of indices of the a anchor parameters;
 $\mathcal{D}$: set of indices of the n derived parameters;
 R : $l \times c$ coefficient matrix of all constraint equations;
 Q : $m \times j$ matrix of independent terms of all constraints;
 P : $n \times j$ matrix of the current values of all parameters;
 $strong$: boolean flag requesting strong satisfiability;
 $method$: the simplification method of the factoring.

Output: $\mathcal{F}'$: set of indices of the f relevant fixed parameters;
 $\mathcal{E}$: set of indices of the m relevant constraints;
 Π_R, L, D, U, Π_C, r : fraction-free LDU factoring of $R_{\mathcal{ED}}$.

```

1 begin
2    $(\mathcal{E}, \mathcal{F}') \leftarrow \text{ECLES.ExtractRelevant}(\mathcal{A}, \mathcal{D}, R)$ 
3    $(\Pi_R, L, D, U, \Pi_C, r) \leftarrow \text{LinSys.LDUFactor}(R_{\mathcal{ED}}, method)$ 
4   if  $strong$  then
5     if not  $\text{ECLES.CheckStrongSolvability}(R, \mathcal{E}, \mathcal{A}, \mathcal{F}', Q, P, \Pi_R, L, r)$  then
6       error "Strong solvability is not satisfied!"

```

If the strong solvability (see Section 5.2.2) is required, the `ECLES.Initialize` procedure checks it in step 5. If the solvability condition of the ECLES method is not satisfied, then the `ECLES.Update` is not called. Otherwise, the information returned by the `ECLES.Initialize` procedure will be used by the `ECLES.Update` procedure in order to constructs a non-redundant linear system (see Section 5.3).

5.2.1 The `ECLES.ExtractRelevant` procedure

The `ECLES.ExtractRelevant` procedure identifies the equations of R that depend on any parameter of the sets $\mathcal{A}$ or $\mathcal{D}$. The set $\mathcal{E}$ of relevant constraints consists of these equations. Based on the set $\mathcal{E}$, the `ECLES.ExtractRelevant` procedure defines the set $\mathcal{F}'$ of fixed relevant parameters. These steps are formalized in Algorithm 2.

Algorithm 2: `ECLES.ExtractRelevant`

Input: $\mathcal{A}$: set of indices of the a anchor parameters;
 $\mathcal{D}$: set of indices of the n derived parameters;
 R : $l \times c$ coefficient matrix of all constraint equations.
Output: $\mathcal{E}$: set of indices of the m relevant constraints;
 $\mathcal{F}'$: set of indices of the f relevant fixed parameters.

```

1 begin
2   for  $i \leftarrow 1$  to  $l$  do
3     if  $(\exists j \in \{1, \dots, c\}), j \in \mathcal{A} \cup \mathcal{D}$  then
4        $\mathcal{E} \leftarrow \mathcal{E} \cup \{i\}$ 
5   for each  $i \in \mathcal{E}$  do
6     for  $j \leftarrow 1$  to  $c$  do
7       if  $((R_{\mathcal{E}})_{ij} \neq 0)$  and  $(j \notin \mathcal{A} \cup \mathcal{D})$  then
8          $\mathcal{F}' \leftarrow \mathcal{F}' \cup \{j\}$ 

```

5.2.2 The `ECLES.CheckStrongSolvability` procedure

The `ECLES.CheckStrongSolvability` procedure verifies whether the relevant constraints can be satisfied for any assignment of values of the anchor parameters. The mathematical justification is described in Section 8.2.2. These steps are formalized in Algorithm 3.

Algorithm 3: ECLES.CheckStrongSolvability

Input: R : $l \times c$ coefficient matrix of all constraint equations;
 $\mathcal{E}$: set of indices of the m relevant constraints;
 $\mathcal{A}$: set of indices of the a anchor parameters;
 $\mathcal{F}'$: set of indices of the f relevant fixed parameters;
 Q : $m \times j$ matrix of independent terms of all constraints;
 P : $n \times j$ matrix of the current values of all parameters;
 Π_R : $m \times m$ permutation matrix of rows;
 L : $m \times r$ lower triangular matrix of integer coefficients;
 r : rank of the matrix $R_{\mathcal{ED}}$.

Output: Boolean flag indicating if the strong solvability condition is satisfied.

```

1 begin
2    $C \leftarrow \Pi_R^{-1}(Q_{\mathcal{E}} - R_{\mathcal{EF}'}P_{\mathcal{F}'})$ 
3    $K \leftarrow \Pi_R^{-1}R_{\mathcal{EA}}$ 
4    $(\widehat{C}, \widetilde{C}) \leftarrow \text{LinSys.SplitRows}(C, r)$ 
5    $(\widehat{K}, \widetilde{K}) \leftarrow \text{LinSys.SplitRows}(K, r)$ 
6    $X_1 \leftarrow \widehat{L}^{-1}\widehat{C}$ 
7    $X_2 \leftarrow \widehat{L}^{-1}\widehat{K}$ 
8    $Y_1 \leftarrow \widetilde{L}X_1$ 
9    $Y_2 \leftarrow \widetilde{L}X_2$ 
10  return  $Y_1 = \widetilde{C}$  and  $Y_2 = \widetilde{K}$ 

```

5.3 The ECLES.Update procedure

If strong solvability was not checked in ECLES.Initialize, the ECLES.Update procedure first uses the method, described in Section 5.3.1, to determine if the weak solvability condition of the linear system (2.3) is satisfied for the given values $P'_{\mathcal{A}}$ and current values of $P_{\mathcal{F}'}$. That is, if there is a displacement $P''_{\mathcal{D}}$ of the derived points such that all constraints in $R_{\mathcal{E}}$ are satisfied. When this condition is not satisfied, the ECLES.Update procedure returns a message to the application notifying that the specified anchor parameter values are not valid. Then, for example, the procedure cancels the editing action and the user must select new values for the anchors.

Otherwise, if the solvability condition is satisfied, the values $P''_{\mathcal{D}}$ of the derived parameters are computed by the ECLES.Update procedure, each time the suggested values $P'_{\mathcal{D}}$ are given by the user interface, by solving the system

$$\widehat{A}P''_{\mathcal{D}} = \widehat{B} \quad (5.1)$$

where $\widehat{A}$ is the first r rows of $LD^{-1}U\Pi_C$ and $\widehat{B}$ is the first r rows of $\Pi_R^{-1}B$, obtained from the ECLES.Initialize.

The ECLES.Update procedure solves the least squares system, equations (9.6) and (9.8), obtaining the new computed values $P''_{\mathcal{D}}$ for the derived parameters. This method is described with more detail in Section 9.1.

The steps described in this section are formalized in Algorithm 4.

Algorithm 4: ECLES.Update

Input: $\mathcal{A}$: set of indices of the a anchor parameters;
 $\mathcal{F}'$: set of indices of the f relevant fixed parameters;
 $\mathcal{E}$: set of indices of the m relevant constraints;
 R : $l \times c$ coefficient matrix of all constraint equations;
 Q : $m \times j$ matrix of independent terms of all constraints;
 Π_R, L, D, U, Π_C, r : fraction-free LDU factoring of $R_{\mathcal{ED}}$;
 P' : $n \times j$ matrix of the suggested values of all parameters.
Output: $P''_{\mathcal{D}}$: $n \times j$ matrix of the new values of the n derived parameters.

```

1 begin
2    $B \leftarrow Q_{\mathcal{E}} - R_{\mathcal{EA}}P'_{\mathcal{A}} - R_{\mathcal{EF}'}P'_{\mathcal{F}'}$ 
3    $(\widehat{B}, \widetilde{B}) \leftarrow \text{LinSys.SplitRows}(\Pi_R^{-1}B, r)$ 
4    $(\widehat{L}, \widetilde{L}) \leftarrow \text{LinSys.SplitRows}(L, r)$ 
5   if ECLES.CheckWeakSolvability $(\widehat{L}, \widetilde{L}, \widehat{B}, \widetilde{B})$  then
6      $P''_{\mathcal{D}} \leftarrow \text{LSQ.Solve}(M, \widehat{L}, D, U, \Pi_C, \widehat{B}, P')$ 
7   else
8     error "Weak solvability is not satisfied!"

```

5.3.1 The ECLES.CheckWeakSolvability procedure

The *ECLES.CheckWeakSolvability* procedure verifies whether the relevant constraints can be satisfied for given values of the anchor parameters. The mathematical justification is described in Section 8.2.1. These steps are formalized in Algorithm 5.

Algorithm 5: ECLES.CheckWeakSolvability

Input: $\widehat{L}$: $r \times r$ lower triangular matrix of integer coefficients;
 $\widetilde{L}$: $(m - r) \times r$ lower triangular matrix of integer coefficients;
 $\widehat{B}$: $r \times r$ matrix of the right-hand side of integer coefficients;
 $\widetilde{B}$: $(m - r) \times r$ matrix of the right-hand side of integer coefficients.
Output: Boolean flag indicating if the weak solvability condition is satisfied.

```

1 begin
2    $X \leftarrow \widehat{L}^{-1}\widehat{B}$ 
3    $Y \leftarrow \widetilde{L}X$ 
4   return  $Y = \widetilde{B}$ 

```

Chapter 6

Fraction-Free LDU Factoring

In this chapter, we describe the technique used to solve linear systems with integer coefficients. We need to exactly solve the system in order to check for linearly dependent equations. The basic idea has been described by Jeffrey [43]. We reimplemented his algorithm with minor differences in the pivoting strategy (full pivoting instead of partial pivoting) column permutation.

6.1 The main algorithm

We now describe the full fraction-free LDU factoring procedure used by ECLES. The factorization (4.1) can be obtained by adapting the Gauss-Jordan elimination algorithm to use fraction-free integer arithmetic that uses cross multiplication to avoid fractions. Namely, we use the pivot value to multiply the target row, instead of dividing the pivot row. As we shall see, it is important to simplify the matrices during this algorithm by removing common factors of the coefficients in each equation.

The factoring is executed by the procedure `LinSys.LDUFactor` (Algorithm 6) which factors the matrix A into matrices Π_R , L , D^{-1} , U and Π_C . Recall that the matrices Π_R and Π_C can be represented as integer vectors to save space. This procedure is more complicated than necessary because it allows the user to choose between three simplification methods, as discussed further on.

The rank r of the matrix is the number of nonzero rows of the triangularized matrix U . In step 15 of `LinSys.LDUFactor`, the procedure `LinSys.RemoveNullParts` discards all zero rows and columns of the resulting matrices L , D , and U .

The main loop invariant of this algorithm (I1) states that formula (4.1) is valid; that is, the matrices Π_R , L , D^{-1} , U and Π_C , in that order, are a factorization of the input array A . This invariant is true before and after every step.

Algorithm 6: `LinSys.LDUFactor`

Input: A : $m \times n$ matrix of integers;
 $method$: the simplification method: “None”, “GCD” or “Turner”.

Output: Π_R : $m \times m$ row permutation matrix;
 L : $m \times r$ lower triangular matrix of integer coefficients;
 D : $r \times r$ diagonal matrix of integer coefficients;
 U : $r \times n$ upper triangular matrix of integer coefficients;
 Π_C : $n \times n$ column permutation matrix;
 r : rank of the matrix A .

```

1 begin
2    $\Pi_R \leftarrow I_{m \times m}$ ;  $L \leftarrow I_{m \times m}$ ;  $D \leftarrow I_{m \times m}$ ;  $U \leftarrow A_{m \times n}$ ;  $\Pi_C \leftarrow I_{n \times n}$ 
3    $i \leftarrow 1$ ;  $j \leftarrow n$ 
4   while  $i \leq m$  and  $i \leq j$  do
5      $(j, \Pi_R, L, U, \Pi_C) \leftarrow \text{LinSys.Pivot}(i, j, m, n, \Pi_R, L, U, \Pi_C)$ 
6     if  $U_{ii} \neq 0$  then
7       if  $i = 1$  then
8          $(L, D, U) \leftarrow \text{LinSys.SimplifyURow}(i - 1, i, n, m, L, D, U, method)$ 
9          $L_{ii} \leftarrow L_{ii}U_{ii}$ ;  $D_{ii} \leftarrow D_{ii}U_{ii}$ 
10        for  $t$  from  $i + 1$  to  $m$  do
11           $(L, D, U) \leftarrow \text{LinSys.EliminateVariable}(i, t, n, L, D, U)$ 
12           $(L, D, U) \leftarrow \text{LinSys.SimplifyURow}(i, t, n, L, D, U, method)$ 
13           $(L, D) \leftarrow \text{LinSys.SimplifyLColumn}(i, m, L, D, method)$ 
14           $i \leftarrow i + 1$ 
15    $r \leftarrow i - 1$ ;  $(L, D, U) \leftarrow \text{LinSys.RemoveNullParts}(L, D, U, r)$ 

```

A secondary loop invariant (I2) says that the matrices L and U are partially triangulated and the rank of A is at least i . Specifically:

- the first $i - 1$ rows and columns of U are an upper triangular matrix with nonzero elements in the diagonal;
- the elements in rows i to m and columns 1 to $i - 1$ and $j + 1$ to n of U are zero;
- the first $i - 1$ rows and columns of L are a lower triangular matrix with nonzero elements in the diagonal;
- all elements of the diagonal of D are nonzero;
- the elements in rows 1 to $i - 1$ and columns i to m of L are zero;
- rows i to m and columns i to m of L are an identity matrix.

Loop invariant I2 is true before step 2 and after step 14, inclusive. See Figure 6.1. In addition, after step 3 an additional condition holds: the element U_{ii} is nonzero and elements $U_{i+1,j}$ to U_{mj} are all zero (invariant I2').

The factorization is complete when $j < i$, that is, rows i to m of U are zero.

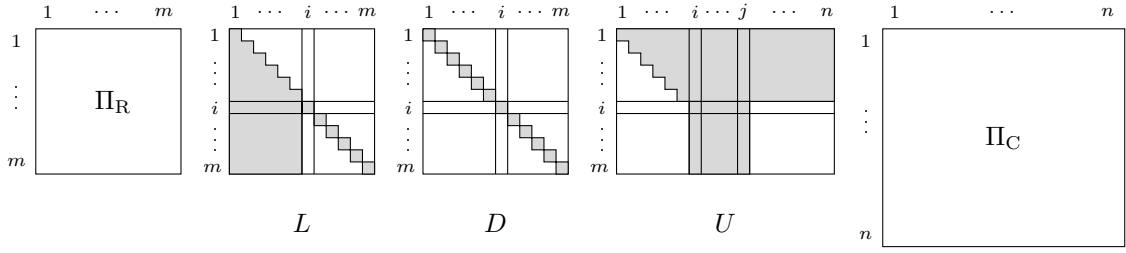


Figure 6.1: Invariant I2 of Algorithm 6. The gray parts are nonzero elements.

6.2 Pivoting

The auxiliary procedure `LinSys.Pivot` swaps the rows and columns of L , D , U (and Π_R , Π_C) to bring the chosen pivot element U_{ks} to position U_{ii} . It may also reduce j .

Algorithm 7: `LinSys.Pivot`

Input: i : current row in the factoring process;

j : last column nonzero in matrix U ;

m : number of rows in matrix A ;

n : number of columns in matrix A ;

Π_R, L, U, Π_C : factoring satisfying invariants I1 and I2.

Output: j : last column nonzero in matrix U ;

Π_R, L, U, Π_C : updated factoring.

```

1 begin
2    $x \leftarrow i + 1; y \leftarrow i; s \leftarrow i; t \leftarrow i;$ 
3    $(U, \Pi_C, j) \leftarrow \text{LinSys.GatherNonzeroColumns}(i, j, U, \Pi_C)$ 
4   if  $j < i$  then
5      $\text{return } (j, \Pi_R, L, U, \Pi_C)$ 
6    $pivot \leftarrow U_{ii}$ 
7   for  $p$  from  $i$  to  $m$  do
8     for  $q$  from  $i$  to  $j$  do
9       if  $U_{pq} \neq 0$  and  $(pivot = 0 \text{ or } |U_{pq}| < |pivot|)$  then
10          $pivot \leftarrow U_{pq};$ 
11          $s \leftarrow p; t \leftarrow q$ 
12   if  $pivot = 0$  then
13      $j \leftarrow i - 1$ 
14      $\text{return } (j, \Pi_R, L, U, \Pi_C)$ 
15   if  $i \neq t$  then
16      $(U, \Pi_C) \leftarrow \text{LinSys.SwapColumns}(i, t, U, \Pi_C)$ 
17   if  $i \neq s$  then
18      $(\Pi_R, L, D, U) \leftarrow \text{LinSys.SwapRows}(i, s, \Pi_R, L, D, U)$ 

```

6.3 Variable elimination

The auxiliary procedure `LinSys.EliminateVariable` modifies the matrix U so that row t ($> i$) does not depend on permuted variable i .

Algorithm 8: `LinSys.EliminateVariable`

Input: i : current row in the factoring process;
 t : next row in the factoring process;
 n : number of columns in matrix A ;
 L, D, U : matrices of the factoring.

Output: L, D, U : updated factoring.

```

1 begin
2    $L_{ti} \leftarrow L_{tt}U_{ti}$ 
3    $D_{tt} \leftarrow D_{tt}U_{ii}$ 
4   for  $s$  from  $i + 1$  to  $n$  do
5      $U_{ts} \leftarrow (U_{ii}U_{ts}) - (U_{ti}U_{is})$ 
6    $U_{ti} \leftarrow 0$ 

```

On input, invariants I1 and I2' are valid. On output, row t of U , L_{ti} , and D_{tt} are modified so that invariants I1 and I2' are still valid, and $U_{ti} = 0$.

6.4 Row and column simplification

The procedure `LinSys.SimplifyURow` eliminates common factors of the row t of the matrix U , adjusting L and D so that invariant I1 is preserved. It does not affect invariant I2'. Depending on the *method* argument, it can use GCD (Greatest Common Divisor) elimination, or Turner's GCD-free method, or (for comparison purposes) no simplification at all. See Chapter 7.

Algorithm 9: `LinSys.SimplifyURow`

Input: i : current row in the factoring process;
 t : next row in the factoring process;
 n : number of columns in matrix A ;
 L, D, U : matrices of the factoring;
 $method$: the simplification method: "None", "GCD" or "Turner".

Output: L, D, U : updated factoring.

```

1 begin
2   if  $method = \text{"GCD"}$  then
3      $(L, U) \leftarrow \text{LinSys.SimplifyURowGCD}(i, t, n, L, U)$ 
4   else if  $method = \text{"Turner"}$  then
5      $(D, U) \leftarrow \text{LinSys.SimplifyURowTurner}(i, t, n, D, U)$ 

```

If the simplification *method* is "GCD", the procedure `LinSys.SimplifyLColumn` eliminates common factors of the column i of the matrix L , adjusting D so that invariant I1 is

preserved. See Chapter 7. It does not affect invariant I2'. This procedure does nothing when *method* is “Turner” or “None”.

Algorithm 10: `LinSys.SimplifyLColumn`

Input: *i*: current row in the factoring process;
 m: number of rows in matrix *A*;
 L, D: matrices of the factoring;
 method: the simplification method: “None”, “GCD” or “Turner”.

Output: *L, D*: updated factoring.

```

1 begin
2   if method = “GCD” then
3      $(L, D) \leftarrow \text{LinSys.SimplifyLColumnGCD}(i, m, L, D)$ 

```

6.5 Computing cost

The procedure `LinSys.LDUFactor` (Algorithm 6) executes $\Theta(m^2n)$ arithmetic operations for a general $m \times n$ matrix of rank m . Multiplication of two t -bit numbers takes time $\Theta(t^2)$. Since the bit size of the matrix entries grows like $\Theta(m)$ (whether with GCD or Turner’s simplification), the running time of the `LinSys.LDUFactor` algorithm is $\Theta(m^4n)$.

Chapter 7

Simplification Techniques for LDU Factoring

In this chapter, we justify the need for elimination of common factors presenting the plain fraction-free Gaussian elimination, and describe two simplification methods (*GCD* and *Turner*). We compare both methods according to the bit size growth of the matrix elements resulting of the fraction-free LDU factoring, described in Chapter 4

7.1 Plain fraction-free Gaussian elimination

The following example [87] shows the execution of the straightforward fraction-free Gauss-Jordan elimination on a matrix A without elimination of common factors, obtaining the matrix U . We omit the processing of the L and D matrices and the pivoting step, for clarity.

$$\begin{aligned}
 A = U = \begin{bmatrix} 8 & 7 & 4 & 1 \\ 4 & 6 & 7 & 3 \\ 6 & 3 & 4 & 6 \\ 4 & 5 & 8 & 2 \end{bmatrix} &\rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & -18 & 8 & 42 \\ 0 & 12 & 48 & 12 \end{bmatrix} \\
 &\rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & 880 & 1200 \\ 0 & 0 & 480 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & 880 & 1200 \\ 0 & 0 & 0 & -576000 \end{bmatrix}
 \end{aligned}$$

A practical problem when using this algorithm is that the bit size of the integers generated during the elimination grows exponentially with the number of equations [33]. In this example, the elements of the original matrix U are in signed 3-bit integers (excluding sign) but the resulting U matrix has 21-bit integers. In general, if the input numbers have t bits (excluding sign), the final matrices will have $t \cdot 2^{r-1}$ bits, where r is the rank (the number of non-redundant equations). See Figure 7.1.

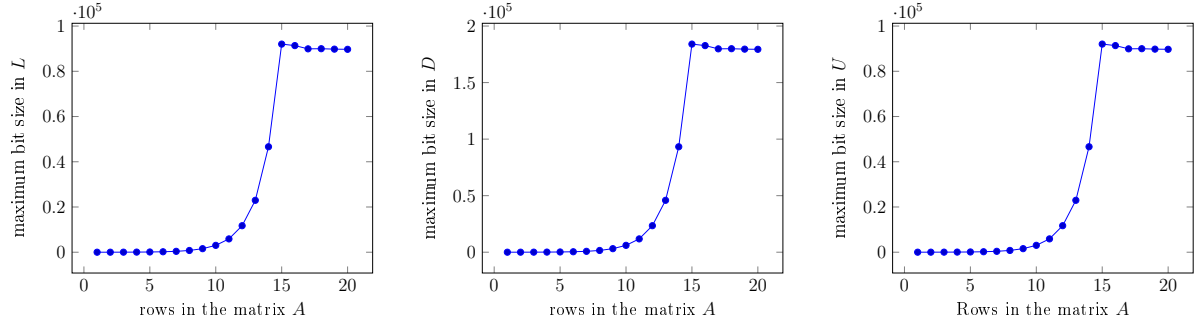


Figure 7.1: Bit size growth in the trivial factorization without simplification for random matrices as a function of the number of rows m . The vertical axis is the maximum bit size among all entries observed by factoring 1000 matrices with $n = 15$ columns and varying number m of rows, with randomly chosen 10-bit signed integer elements. Note that the bit size stops growing when m exceeds n . The slight decrease for $m > n$ is due to the better chances of finding a small pivot as m increases.

7.2 Simplifying by GCD elimination

The bit size growth of the numbers can be greatly reduced by dividing each computed row by the GCD of its coefficients. Therefore, the range growth in subsequent stages of the elimination is restricted to a level inherent in the problem. The following example [87] shows the result for on the same matrix U .

$$U = \begin{bmatrix} 8 & 7 & 4 & 1 \\ 4 & 6 & 7 & 3 \\ 6 & 3 & 4 & 6 \\ 4 & 5 & 8 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & -9 & 4 & 21 \\ 0 & 1 & 4 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 11 & 15 \\ 0 & 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 11 & 15 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In this example the elements of the resulting matrix U can be stored in 5-bit signed integers. Note that the L matrix must be adjusted too to maintain the invariants. With this optimization, we observe experimentally that the bit size of the elements in the matrix U grows linearly with the number of equations (specifically, close to $t \cdot r$), instead of exponentially.

To obtain a linear growth of the matrices L and D , the columns of the matrix L and the related elements of the matrix D can be divided by the GCD of its coefficients. See Figure 7.2.

These simplifications are described by the procedure `LinSys.SimplifyURowGCD`. This procedure finds the largest common factor GCD in row t and divides that factor into that row, and multiplies it into column t of L .

The procedure `LinSys.SimplifyLColumnGCD` finds the GCD common factor between the column i of the matrix L and the element D_{ii} of the matrix D . Then, it divides the column i of L and the element D_{ii} by that GCD factor.

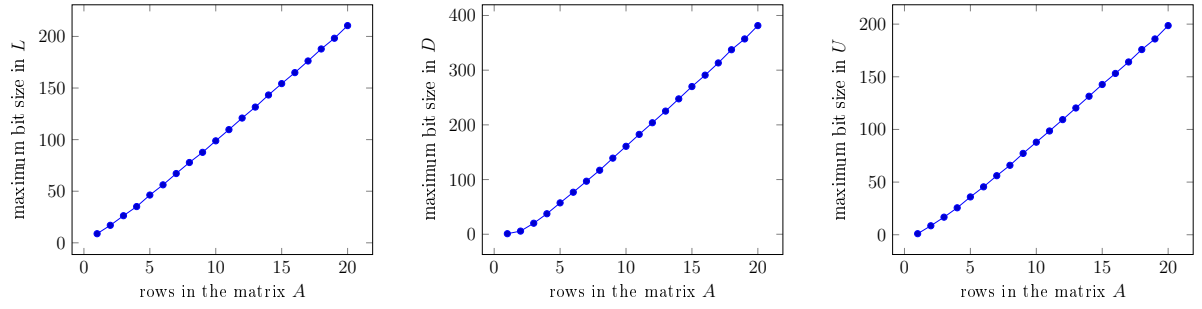


Figure 7.2: Bit size growth with the GCD simplification methods for random matrices as a function of the number of rows m . The vertical axis is the maximum bit size among all entries observed by factoring 1000 matrices varying the size $m = n$ of rows and columns, with randomly chosen 10-bit signed integer elements.

Algorithm 11: `LinSys.SimplifyURowGCD`

Input: i : current row in the factoring process;
 t : row to simplify;
 n : number of columns in matrix A ;
 L, U : factoring matrices.

Output: L, U : updated factoring.

```

1 begin
2    $gcd \leftarrow U_{t,i+1}$ 
3   for  $s$  from  $i + 1$  to  $n$  and  $(gcd \neq 1)$  do
4      $gcd \leftarrow \text{LinSys.CalculateGCD}(gcd, U_{t,s})$ 
5   if  $gcd > 1$  then
6     for  $s$  from  $i + 1$  to  $n$  do
7        $U_{ts} \leftarrow U_{ts} / gcd$ 
8      $L_{tt} \leftarrow L_{tt} * gcd$ 

```

Algorithm 12: `LinSys.SimplifyLColumnGCD`

Input: i : current row in the factoring process;
 m : number of rows in matrix A ;
 L, D : factoring matrices.

Output: L, D : updated factoring.

```

1 begin
2    $gcd \leftarrow D_{ii}$ 
3   for  $t$  from  $i$  to  $m$  while  $gcd \neq 1$  do
4      $gcd \leftarrow \text{LinSys.CalculateGCD}(gcd, L_{ti})$ 
5   if  $gcd > 1$  then
6     for  $t$  from  $i$  to  $m$  do
7        $D_{ii} \leftarrow D_{ii} / gcd$ 
8      $L_{ti} \leftarrow L_{ti} / gcd$ 

```

7.3 Turner's GCD-free simplification

In 1968, G. H. Bareiss [4] observed that some common factors in the computed rows are predictable and can be avoided by using more complex formulas in the elimination step. Later, in 1995, Peter R. Turner [87] observed that some of these factors can be found easily. Specifically, after elimination of variable i , the diagonal element of row $i - 2$ of the partially triangulated matrix divides all the elements of the new rows $i, i + 1, \dots, m$.

The following example shows the execution of the fraction-free Gaussian elimination with Turner's simplification on the same matrix U .

$$\begin{aligned}
 U = \begin{bmatrix} 8 & 7 & 4 & 1 \\ 4 & 6 & 7 & 3 \\ 6 & 3 & 4 & 6 \\ 4 & 5 & 8 & 2 \end{bmatrix} &\rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & -18 & 8 & 42 \\ 0 & 12 & 48 & 12 \end{bmatrix} && \text{(first stage)} \\
 \rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & \mathbf{880} & \mathbf{1200} \\ 0 & 0 & \mathbf{480} & \mathbf{0} \end{bmatrix} &\rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & 110 & 150 \\ 0 & 0 & 60 & 0 \end{bmatrix} && \text{(second stage)} \\
 \rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & \mathbf{20} & 40 & 20 \\ 0 & 0 & 110 & 150 \\ 0 & 0 & 0 & \mathbf{-9000} \end{bmatrix} &\rightarrow \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & 110 & 150 \\ 0 & 0 & 0 & -450 \end{bmatrix} && \text{(third stage)}
 \end{aligned}$$

Note that, in the second stage, the pivot $U_{11} = 8$ divides all elements of sub-matrix U_{ij} with $i, j \in \{3, 4\}$. In this example, the original matrix has 3-bit integer elements, and the computation can be performed using 16-bit integer arithmetic.

Turner's algorithm has the advantage that the determinant of the original matrix U is automatically detected as the final value of U_{nn} (apart from signed changes, if pivoting is used). Still, Turner's algorithm reduces the growth in the bit size of elements to linear instead of exponential [4, 33, 87].

A comparison between GCD and Turner's simplification shows that the bit size growth in the factored matrices is similar in both methods. See Figure 7.3.

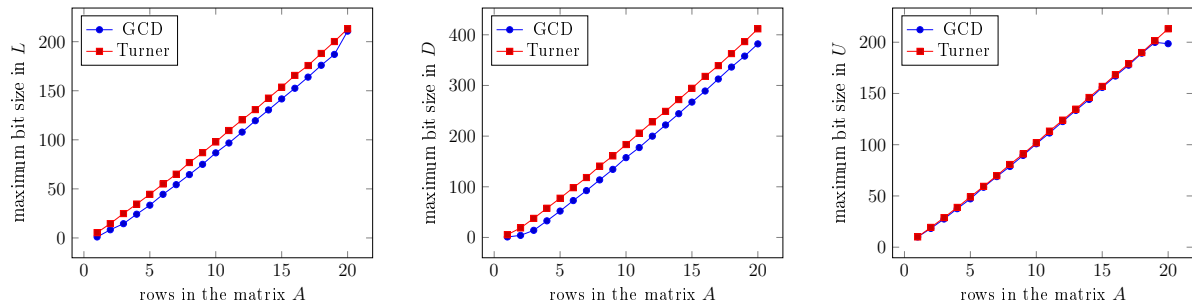


Figure 7.3: Comparison of the bit size growth between the GCD and Turner simplification methods for random matrices as a function of the number of rows m . The vertical axis is the maximum bit size among all entries observed by factoring 1000 matrices with $n = 20$ columns and varying number m of rows, with randomly chosen 10-bit signed integer elements.

The procedure `LinSys.SimplifyURowTurner` implements the Turner’s algorithm to eliminate the common factors in row t . Note that Turner’s simplification does not modify the matrix L .

Algorithm 13: `LinSys.SimplifyURowTurner`

Input: i : current row in the factoring process;
 t : next row in the factoring process;
 n : number of columns in matrix A ;
 D, U : factoring matrices.

Output: D, U : updated factoring.

```

1 begin
2   if  $i \geq 2$  then
3     for  $s \leftarrow i + 1$  to  $n$  do
4        $U_{ts} \leftarrow U_{ts} / U_{i-1,i-1}$ 
5      $D_{tt} \leftarrow U_{tt} / U_{i-1,i-1}$ 

```

However, while Turner’s method saves some time by eliminating the computation of the GCD, it may result slightly in a bit larger numbers, because it fails to eliminate some common factors that may arise by coincidence – like the factor 20 in row 2 of the example. It also saves time by not modifying the matrix L . Other authors have identified additional common factors in the matrices L , D , and U that can be eliminated computing the GCD [58].

Originally, the fraction-free factoring method was defined only for square or non-singular matrices [4, 24, 60, 65, 96]. In this thesis, we follow the presentation of D. J. Jeffrey [43] which can be applied to arbitrary rectangular matrices of any rank r .

7.3.1 Bit size growth for rank deficient matrices

The previous results hold also for rank-deficient square matrices: the bit size grows linearly in the rank r . The factoring of matrices with rank deficient using the GCD simplification generates smaller matrix elements. See Figure 7.4. In these tests, each matrix was obtained by multiplying two matrices of size $20 \times r$ and $r \times 20$ with random elements in the appropriate range $[-\lfloor \sqrt{2^{t-1}/r} \rfloor \dots \lfloor \sqrt{2^{t-2}/r} \rfloor]$, where t is the bit size of the final matrix elements.

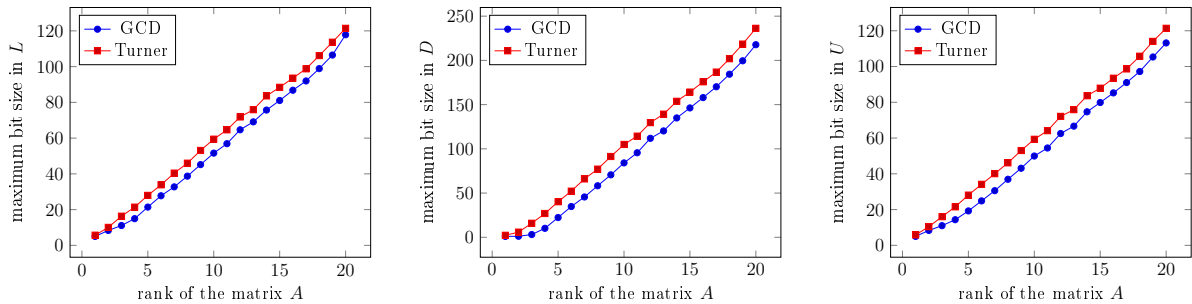


Figure 7.4: Comparison of the bit size growth between the GCD and Turner simplification methods for random rank deficient matrices as a function of the rank r . The vertical axis is the maximum bit size among all entries observed by factoring 1000 20×20 matrices varying the deficient rank r , with 10-bit signed integer elements.

7.3.2 Bit size growth for random sparse matrices

In this section, we determine the bit size growth for the case when the matrix is sparse, as a function of its density h of non-zero elements. Figures 7.5 and 7.6 show how the bit size grows as a function of the number $m = n$ of rows and columns for fixed densities $h \approx 0.10$ and $h \approx 0.25$. Figure 7.7 show how the bit size varies with h for a fixed matrix size.

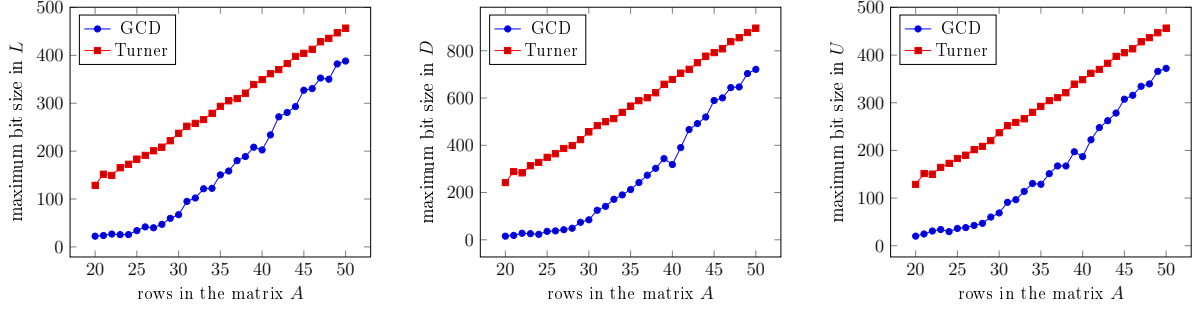


Figure 7.5: Comparison of the bit size growth between the GCD and Turner simplification methods for random sparse matrices as a function of the number of rows m . The vertical axis is the maximum bit size among all entries observed by factoring 1000 sparse matrices varying the size $m = n$ of rows and columns, with densities $h \approx 0.10$ and 10-bit signed integer elements.

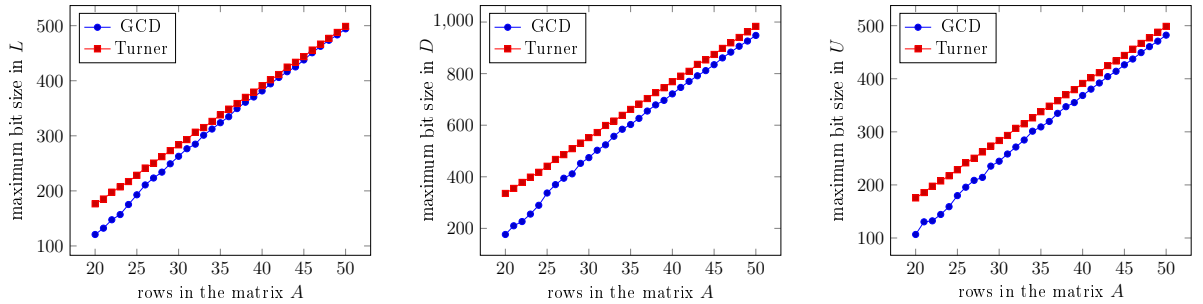


Figure 7.6: Comparison of the bit size growth between the GCD and Turner simplification methods for random sparse matrices as a function of the number of rows m . The vertical axis is the maximum bit size among all entries observed by factoring 1000 sparse matrices varying the size $m = n$ of rows and columns, with densities $h \approx 0.25$ and 10-bit signed integer elements.

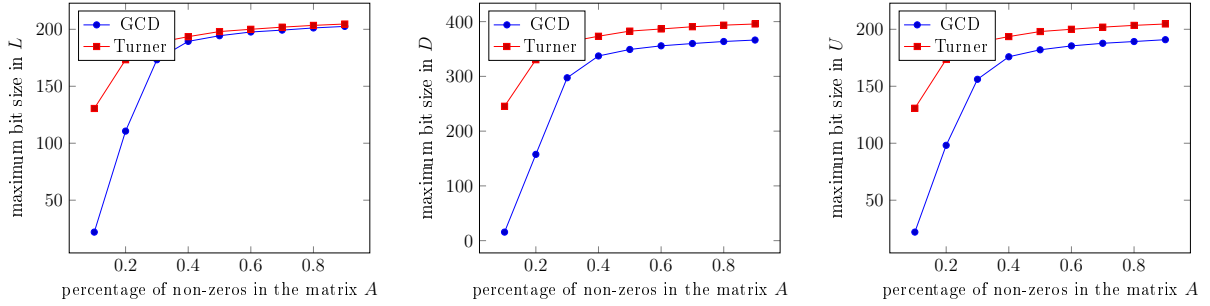


Figure 7.7: Comparison of the bit size growth between the GCD and Turner simplification methods for random sparse matrices as a function of the density h . The vertical axis is the maximum bit size among all entries observed by factoring 1000 20×20 sparse matrices varying the density h , with 10-bit signed integer elements.

7.3.3 Bit size growth for sparse matrices with deficient rank

Figures 7.8 and 7.9 show how the bit size grows as a function of rank r for densities $h \approx 0.10$ and $h \approx 0.25$. Figure 7.10 show how the bit size varies with h for a fixed matrix size.

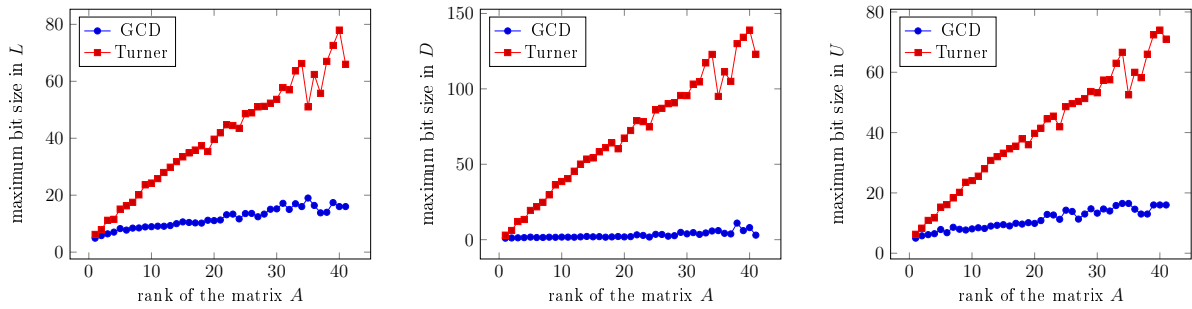


Figure 7.8: Comparison of the bit size growth between the GCD and Turner simplification methods for random rank deficient sparse matrices as a function of the rank r . The vertical axis is the maximum bit size among all entries observed by factoring 1000 sparse matrices varying the size $m = n$ of rows and columns and the rank deficient r , with densities $h \approx 0.10$ and 10-bit signed integer elements.

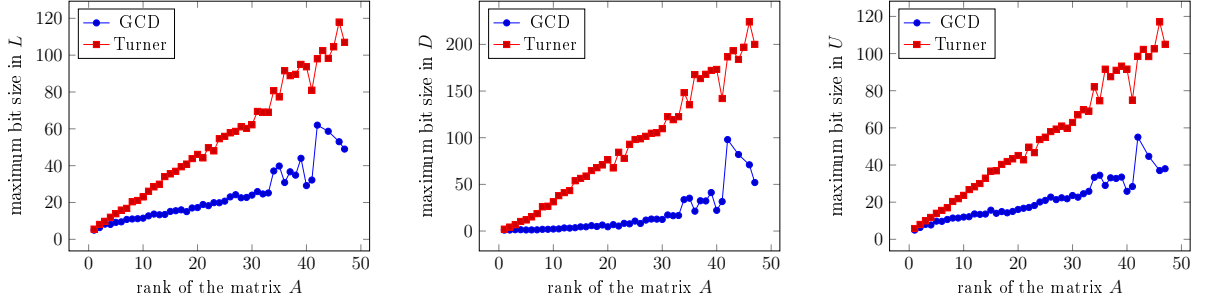


Figure 7.9: Comparison of the bit size growth between the GCD and Turner simplification methods for random rank deficient sparse matrices as a function of the rank r . The vertical axis is the maximum bit size among all entries observed by factoring 1000 sparse matrices varying the size $m = n$ of rows and columns and the rank deficient r , with densities $h \approx 0.25$ and 10-bit signed integer elements.

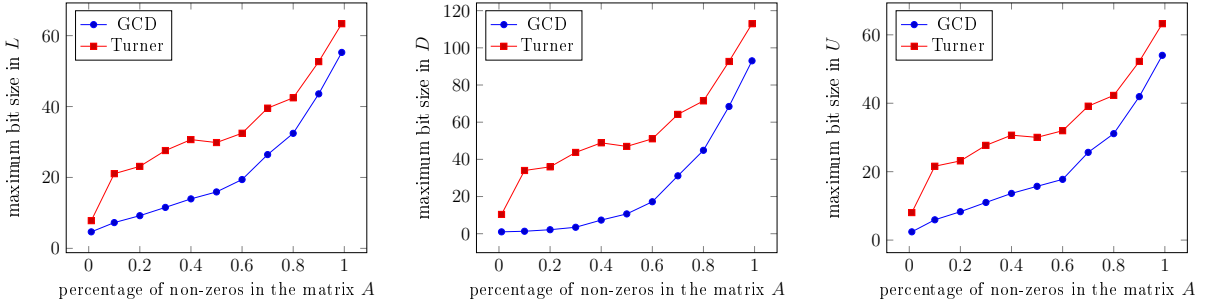


Figure 7.10: Comparison of the bit size growth between the GCD and Turner simplification methods for random rank deficient sparse matrices as a function of the density h . The vertical axis is the maximum bit size among all entries observed by factoring 1000 20×20 sparse matrices varying the density h and the rank deficient r , with 10-bit signed integer elements.

7.3.4 Discussion about the results

We can conclude from these tests that sparsity and low rank considerably reduce the running time of the `ECLES.Update` algorithm both by reducing the size of the matrices $\hat{L}$ and $\hat{U}$ (to $m \times r$ and $r \times n$ instead of $m \times m$ and $n \times n$, respectively) and by reducing the bit size of their elements. The last factor is quite significant if full GCD simplification is used instead of the Turner method, even though the two are nearly equivalent for dense full-rank matrices.

Chapter 8

Solving Exact Linear Systems

In this chapter, we describe the technique used to solve linear systems with integer coefficients in factored form [43], obtained from the fraction-free LDU factoring algorithm, described in Chapter 6.

8.1 Solving the system

We assume in general that the linear system to be solved is

$$AP = B \quad (8.1)$$

where P is an unknown array of n rational numbers, and B is a known vector of m integers. Substituting formulas (4.1) and (4.2) into system (8.1), we have

$$\Pi_R \begin{pmatrix} \hat{L} \\ \tilde{L} \end{pmatrix} D^{-1} \begin{pmatrix} \hat{U} & \tilde{U} \end{pmatrix} \Pi_C P = B. \quad (8.2)$$

Letting $X = \Pi_C P$, $Y = UX$ and, $\hat{B}$ and $\tilde{B}$ be the first r and last $m - r$ elements of $\Pi_R^{-1}B$, respectively, Equation (8.2) becomes

$$\begin{pmatrix} \hat{L} \\ \tilde{L} \end{pmatrix} D^{-1} Y = \begin{pmatrix} \hat{B} \\ \tilde{B} \end{pmatrix}. \quad (8.3)$$

We can split the system (8.3) into two systems

$$\hat{L}D^{-1}Y = \hat{B} \quad \text{and} \quad \tilde{L}D^{-1}Y = \tilde{B}. \quad (8.4)$$

Since $\hat{L}D^{-1}$ is an $r \times r$ invertible matrix, we can solve the first system for Y

$$Y = D\hat{L}^{-1}\hat{B}. \quad (8.5)$$

The matrix $D\hat{L}^{-1}$ turns out to be integer [43], therefore Y is a array of integers.

To solve the system (8.2), we can find X by solving

$$\begin{pmatrix} \widehat{U} & \widetilde{U} \end{pmatrix} \begin{pmatrix} \widehat{X} \\ \widetilde{X} \end{pmatrix} = Y \quad (8.6)$$

that is, by solving

$$\widehat{U}\widehat{X} = Y - \widetilde{U}\widetilde{X} \quad (8.7)$$

where $\widetilde{X}$ can be chosen arbitrarily. Setting $\widetilde{X} = 0$, we get

$$\widehat{X} = \widehat{U}^{-1}Y. \quad (8.8)$$

Then we can get the solution P by

$$P = \Pi_C^{-1}X. \quad (8.9)$$

The matrix $\widehat{U}^{-1}$ is not integer, therefore the computation (8.8) must be done with rational arithmetic, and $\widehat{X}$ is a vector of rational numbers.

The steps for solving the system are formalized in Algorithm 14. The rank r and the matrices Π_R , L , D , U and Π_C are received of the `LinSys.LDUFactor` procedure.

Algorithm 14: `LinSys.Solve`

Input: $\widehat{L}$: $r \times r$ lower triangular matrix of integer coefficients;
 D : $r \times r$ diagonal matrix of integer coefficients;
 $\widehat{U}$: $r \times r$ upper triangular matrix of integer coefficients;
 Π_C : $n \times n$ permutation matrix of columns;
 $\widehat{B}$: $n \times j$ matrix of the right-hand side of integer coefficients.

Output: P : $n \times j$ matrix of coordinates of the new values of the n parameters.

```

1 begin
2    $Y \leftarrow D\widehat{L}^{-1}\widehat{B}$ 
3    $\widehat{X} \leftarrow \widehat{U}^{-1}Y$ 
4    $\widetilde{X} \leftarrow 0$ 
5    $X \leftarrow \text{LinSys.JoinRows}(\widehat{X}, \widetilde{X})$ 
6    $P \leftarrow \Pi_C^{-1}X$ 

```

8.2 Checking consistency

The second system $\tilde{L}D^{-1}Y = \tilde{B}$ in Equation (8.4) is non-empty only if the rank r of A is less than the number of rows m , in which case either some constraints are redundant, or there are incompatible constraints.

8.2.1 Weak solvability

To verify whether the original system (8.1) is consistent, for the given right-hand-side matrix B , it is necessary and sufficient to check if the bottom half of the Equation (8.3) is satisfied with this value Y , that is

$$\tilde{L}\hat{L}^{-1}\hat{B} = \tilde{B}. \quad (8.10)$$

8.2.2 Strong solvability

Strong solvability means that the system (8.1) can be solved assignment P'_j to the anchor parameters. Recall that $B = Q_{\mathcal{E}} - R_{\mathcal{E}\mathcal{A}} P'_{\mathcal{A}} - R_{\mathcal{E}\mathcal{F}'} P_{\mathcal{F}'}$ then

$$\Pi_{\mathbf{R}}^{-1}B = \Pi_{\mathbf{R}}^{-1}(Q_{\mathcal{E}} - R_{\mathcal{E}\mathcal{F}'} P_{\mathcal{F}'}) - \Pi_{\mathbf{R}}^{-1}R_{\mathcal{E}\mathcal{A}} P'_{\mathcal{A}}. \quad (8.11)$$

We can rewrite the matrix $\Pi_{\mathbf{R}}^{-1}B$ as

$$\Pi_{\mathbf{R}}^{-1}B = C - KP'_{\mathcal{A}}$$

where $C = \Pi_{\mathbf{R}}^{-1}(Q_{\mathcal{E}} - R_{\mathcal{E}\mathcal{F}'} P_{\mathcal{F}'})$ and $K = \Pi_{\mathbf{R}}^{-1}R_{\mathcal{E}\mathcal{A}}$.

The matrices $\Pi_{\mathbf{R}}^{-1}B$, C and K can be split as

$$\begin{pmatrix} \hat{B} \\ \tilde{B} \end{pmatrix} = \begin{pmatrix} \hat{C} \\ \tilde{C} \end{pmatrix} - \begin{pmatrix} \hat{K} \\ \tilde{K} \end{pmatrix} P'_{\mathcal{A}} \quad (8.12)$$

where $\hat{B}$, $\hat{C}$, and $\hat{K}$ are the r first rows of the matrices $\Pi_{\mathbf{R}}^{-1}B$, C , and K . Then, we have two equations

$$\hat{B} = \hat{C} - \hat{K}P'_{\mathcal{A}} \quad \text{and} \quad \tilde{B} = \tilde{C} - \tilde{K}P'_{\mathcal{A}}. \quad (8.13)$$

Substituting $\hat{B}$ and $\tilde{B}$ in the Equation (8.10), we have

$$\begin{aligned} \tilde{L}\hat{L}^{-1}(\hat{C} - \hat{K}P'_{\mathcal{A}}) &= \tilde{C} - \tilde{K}P'_{\mathcal{A}} \\ \tilde{L}\hat{L}^{-1}\hat{C} - \tilde{C} &= (\tilde{L}\hat{L}^{-1}\hat{K} - \tilde{K})P'_{\mathcal{A}}. \end{aligned} \quad (8.14)$$

So, to verify whether the original system (8.1) is consistent for any B , it is necessary and sufficient to check if

$$\tilde{L}\hat{L}^{-1}\hat{C} = \tilde{C} \quad (8.15)$$

$$\tilde{L}\hat{L}^{-1}\hat{K} = \tilde{K}. \quad (8.16)$$

8.3 An example

Considering the system $AP = B$, where A is the matrix of the example (4.3), and B the following integer vector.

$$A = \begin{bmatrix} 4 & 9 & 16 & 29 \\ -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 5 & 6 & -1 & -12 \\ 5 & 10 & 15 & 20 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 142 \\ 42 \\ 40 \\ -128 \\ 50 \end{bmatrix}. \quad (8.17)$$

We have the following fraction-free LDU factoring matrices, described in Chapter 4.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 15 & -80 \\ 5 & 24 & -164 \\ 5 & 20 & -120 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -80 \end{bmatrix}^{-1} \begin{bmatrix} -1 & -6 & -16 & -19 \\ 0 & 1 & -3 & 4 \\ 0 & 0 & -80 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

The system (8.3) can be obtained using the factoring matrices. Then, we have the system

$$\begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 15 & -80 \\ 5 & 24 & -164 \\ 5 & 20 & -120 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -80 \end{bmatrix}^{-1} Y = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 142 \\ 42 \\ 40 \\ -128 \\ 50 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 15 & -80 \\ 5 & 24 & -164 \\ 5 & 20 & -120 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -80 \end{bmatrix}^{-1} Y = \begin{bmatrix} 42 \\ 40 \\ -128 \\ 50 \end{bmatrix}. \quad (8.18)$$

Checking the solvability condition (8.10), we have that the original system A is consistent

$$\begin{bmatrix} 5 & 24 & -164 \\ 5 & 20 & -120 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 15 & -80 \end{bmatrix}^{-1} \begin{bmatrix} 42 \\ 40 \\ 142 \end{bmatrix} = \begin{bmatrix} -128 \\ 50 \end{bmatrix}$$

$$\begin{bmatrix} -128 \\ 50 \end{bmatrix} = \begin{bmatrix} -128 \\ 50 \end{bmatrix}. \quad (8.19)$$

Solving the first system (8.5) for Y , we have

$$Y = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -80 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 15 & -80 \end{bmatrix}^{-1} \begin{bmatrix} 42 \\ 40 \\ 142 \end{bmatrix} = \begin{bmatrix} 42 \\ -82 \\ -920 \end{bmatrix}. \quad (8.20)$$

Solving the second system (8.8) for $\hat{X}$, we have

$$\hat{X} = \begin{bmatrix} -1 & -6 & -16 \\ 0 & 1 & -3 \\ 0 & 0 & -80 \end{bmatrix}^{-1} \begin{bmatrix} 42 \\ -82 \\ -920 \end{bmatrix} = \begin{bmatrix} 59 \\ -95/2 \\ 23/2 \end{bmatrix}. \quad (8.21)$$

Setting $\tilde{X} = 0$, and solving of Equation (8.9) for P , we have

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 59 \\ -95/2 \\ 23/2 \\ 0 \end{bmatrix} = \begin{bmatrix} 59 \\ -95/2 \\ 23/2 \\ 0 \end{bmatrix}. \quad (8.22)$$

The parameter vector P may be converted from rational to floating-point. For example, converting the rational solution P in Equation (8.22), we have

$$P = \begin{bmatrix} 59 \\ -95/2 \\ 23/2 \\ 0 \end{bmatrix} = \begin{bmatrix} 59 \\ -47.5 \\ 11.5 \\ 0 \end{bmatrix}. \quad (8.23)$$

Chapter 9

Solving the Least Squares Problem

In this chapter, we describe the quadratic optimization (*least squares*) method [66] with affine constraints. This method is used whenever there is more than one solution, to find values of $P''_{\mathcal{D}}$ that are “close” the given hints $P'_{\mathcal{D}}$.

We combine the least squares optimization with the fraction-free LDU factoring, in order to obtain a best solution for the problem of interactive editing of parameters with constraints.

9.1 Constrained least squares

We now describe in detail the procedure `LSQ.Solve` used by `ECLES.Update`, described in Section 5.3. It receives the fraction-free LDU factoring of the $m \times n$ constraint matrix A , the right-hand side vector B and the hints P' for the n unknowns. If $r = n$, there is a single solution which is computed by solving the constraint system (2.3) $AX = B$ as described in Section 8.1.

Otherwise, if $r < n$, the system has many solutions and the procedure has to minimize the goal function S defined by Equation (4.4) while satisfying the constraints (4.5), as described in Chapter 4.

At the maximum point, the gradient ∇S of the goal function S must be perpendicular to the solution set of the non-redundant constraints. The gradient consists of the derivatives

$$\frac{\partial S}{\partial P_s}(P) = 2w_s(P_s - P'_s) \quad (9.1)$$

for $s = 1, 2, \dots, n$. To be perpendicular to the constraint solution space, the gradient $\nabla S(P'')$ must satisfy

$$\frac{\partial S}{\partial P_s}(P'') = - \sum_{k=1}^n \Lambda_k \hat{A}_{ks} \quad (9.2)$$

where each variable Λ_k is an indeterminate real coefficient, the *Lagrange multiplier* [64] of the constraint expressed by row k of the system (4.5). From (9.1) and (9.2) we get

$$2w_s P''_s + \sum_{k=1}^n \Lambda_k \hat{A}_{ks} = 2w_s P'_s. \quad (9.3)$$

Equation (9.3) can also be written in matrix form

$$MP'' + \hat{A}^\top \Lambda = MP' \quad (9.4)$$

where M is the $n \times n$ invertible diagonal matrix with $M_{ss} = 2w_s$; $\hat{A}^\top$ is the transpose of $\hat{A}$; and Λ is an array with the Lagrange multipliers $\Lambda_1, \dots, \Lambda_m$.

We can combine Equation (9.4) with the constraints (4.5) obtaining the *least squares linear system*

$$\begin{aligned} MP'' &= MP' - \hat{A}^\top \Lambda \\ \hat{A}P'' &= \hat{B}. \end{aligned} \quad (9.5)$$

System (9.5) can be solved in two steps; namely, since $\hat{A}M^{-1}\hat{A}^\top$ is $r \times r$ invertible matrix, we can solve the first system for Λ

$$\hat{A}M^{-1}\hat{A}^\top \Lambda = \hat{A}P' - \hat{B}. \quad (9.6)$$

Then we can compute P'' by solving

$$MP'' = MP' - \hat{A}^\top \Lambda. \quad (9.7)$$

That is,

$$P'' = P' - M^{-1}\hat{A}^\top \Lambda. \quad (9.8)$$

These steps are formalized in Algorithm 15.

Algorithm 15: LSQ.Solve

Input: M : $n \times n$ matrix of weight;

$\hat{L}$: $r \times r$ lower triangular matrix of integer coefficients;

D : $r \times r$ diagonal matrix of integer coefficients;

U : $r \times n$ upper triangular matrix of integer coefficients;

Π_C : $n \times n$ permutation matrix of columns;

$\hat{B}$: $r \times j$ matrix of the right-hand side of integer coefficients;

P' : $n \times j$ matrix of coordinates of the suggested values of the parameters.

Output: P'' : $n \times j$ matrix of coordinates of the new values of the parameters.

```

1 begin
2   if  $r = n$  then
3      $(\hat{U}, \tilde{U}) \leftarrow \text{LinSys.SplitColumns}(U, r)$ 
4      $P''_D \leftarrow \text{LinSys.Solve}(\hat{L}, D, \hat{U}, \Pi_C, \hat{B})$ 
5   else
6      $\hat{A} \leftarrow \hat{L}D^{-1}U\Pi_C$ 
7      $\Lambda \leftarrow (\hat{A}M^{-1}\hat{A}^\top)^{-1}(\hat{A}P' - \hat{B})$ 
8      $P'' \leftarrow P' - M^{-1}\hat{A}^\top \Lambda$ 

```

9.2 An example

Considering the system $AP = B$, where A is the matrix of the example (4.3), and B the following integer vector.

$$A = \begin{bmatrix} 4 & 9 & 16 & 29 \\ -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 5 & 6 & -1 & -12 \\ 5 & 10 & 15 & 20 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 142 \\ 42 \\ 40 \\ -128 \\ 50 \end{bmatrix}. \quad (9.9)$$

The matrices $\hat{A}$ and $\hat{B}$ are

$$\hat{A} = \begin{bmatrix} -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 4 & 9 & 16 & 29 \end{bmatrix} \quad \text{and} \quad \hat{B} = \begin{bmatrix} 42 \\ 40 \\ 142 \end{bmatrix}. \quad (9.10)$$

Considering the matrices M and P' following

$$M = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad \text{and} \quad P' = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}. \quad (9.11)$$

We solve the system (9.5) in two steps. First, solving the Equation (9.6) for Λ

$$\Lambda = \begin{bmatrix} 327 & -310 & -413 \\ -310 & 306 & 420 \\ -413 & 420 & 597 \end{bmatrix}^{-1} \begin{bmatrix} -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 4 & 9 & 16 & 29 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 42 \\ 40 \\ 142 \end{bmatrix} = \begin{bmatrix} -123/20 \\ -23/4 \\ -7/20 \end{bmatrix}. \quad (9.12)$$

To find P'' , we solve the Equation (9.8)

$$P'' = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} -1 & 1 & 4 \\ -6 & 5 & 9 \\ -19 & 15 & 16 \\ -16 & 19 & 29 \end{bmatrix} \begin{bmatrix} -123/20 \\ -23/4 \\ -7/20 \end{bmatrix} = \begin{bmatrix} 3/2 \\ -3/2 \\ -23/2 \\ 23/2 \end{bmatrix}. \quad (9.13)$$

Converting the rational solution P'' (9.13) obtained by least squares, we have

$$P'' = \begin{bmatrix} 3/2 \\ -3/2 \\ -23/2 \\ 23/2 \end{bmatrix} = \begin{bmatrix} 1.5 \\ -1.5 \\ -11.5 \\ 11.5 \end{bmatrix}. \quad (9.14)$$

Note that, unlike the solution obtained in Section 8.3, the least squares solution is close to the vector P' .

Part II

The 2DSD Algorithm

Chapter 10

Interactive Editing of 2D Spline Deformations

In this chapter, we describe the problem of interactive editing of 2D spline deformations that are defined by Bézier control points subject to smoothness constraints.

10.1 Statement of the problem

10.1.1 Deformations

Let Ω be some region of $\mathbb{R}^2$, the *domain*. A *deformation* is a function ϕ from Ω to Ω . We consider here deformations that are polynomial splines defined on a triangular mesh that covers Ω . See Chapter 12. As defined in Section 12.2, the deformation is determined by a set of Bézier control points that are subject to various continuity constraints.

A deformation is *applied* to an object (image, 3D model, etc.) by mapping each point p of the latter to the point $\phi(p)$, and assigning to this point the same properties (color, texture, etc) that p had.

For many applications, such as solid modeling and image morphing, the deformation must be *smooth*, that is, its derivatives must be continuous. Otherwise, it will introduce corners or creases in embedded smooth objects. See Figure 10.1.



Figure 10.1: A comparison between space deformations (a) without and (b) with C^1 continuity.

10.1.2 Meshes and splines

A t -dimensional mesh T is a partition of some domain $\Omega \subseteq \mathbb{R}^d$ into simple *parts* (or *cells*) with pairwise disjoint interiors. We define a (*polynomial*) *spline function* on a mesh T with domain Ω as a function $f : \Omega \rightarrow \mathbb{R}$ such that the restriction f^u of f to each part u of the mesh is a polynomial on the coordinates of the argument.

In this thesis, only polynomial spline deformations are considered. Both coordinates of the spline $\phi(p)$ are polynomial spline functions on the same mesh T , the *reference mesh*.

10.2 User interface

As described in Chapter 14, the spline to be edited is defined by a set of Bézier control points which belong to the Bézier patches associated with the faces of the reference mesh. The user is expected to edit the deformation by moving those control points (e.g. with the mouse).

A typical implementation of the 2DSD algorithm is described in Part III. Generally, we assume that, in each operation, the user first chooses the operation type (translation, rotation, etc), the anchor set $\mathcal{A}$, and an initial set $\mathcal{D}$ of derived points (that may be enlarged by the application in order to ensure solvability). Then, the user drags the anchor points with the mouse. For each new placement of the anchors the editor adjusts the deformation and applies it to the object being deformed.

For example, in a *local soft translation* all the anchor points are translated by the same displacement vector v . The derived points $\mathcal{D}$ are automatically updated in order to maintain the smoothness of the deformation. If there are enough degrees of freedom, the displacement of the derived points decreases gradually in their distance from the set $\mathcal{A}$. See Figures 10.2.

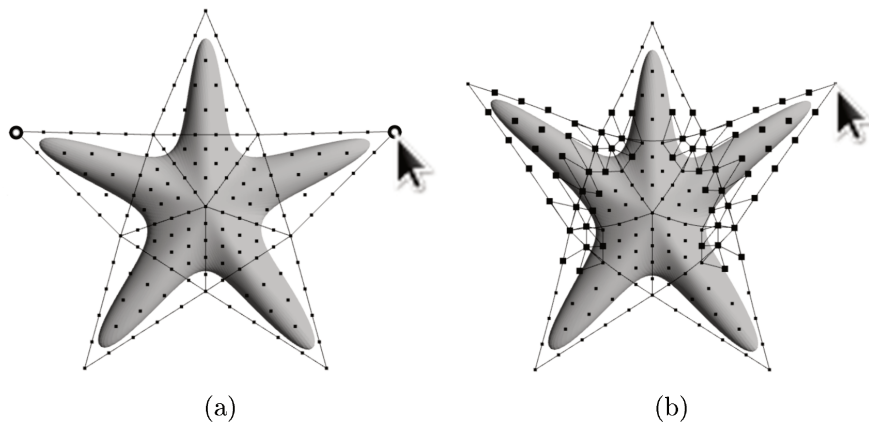


Figure 10.2: A soft translation with 2 anchor points (black open dots) and derived points (black dots) (a) and the result of displacing the anchor points by the vector v (b).

Other operations (*local soft rotation*, *local soft scaling*, etc.) are described in Part III.

Chapter 11

Related Work

In this chapter, we present a literature review about methods for deformations of two-dimensional models and interpolation techniques. We focus, specifically, on techniques that use 2D control meshes to define the deformation.

2D deformations have many specific applications, and therefore there is no single approach that is optimal for all of them.

Two-dimensional deformation techniques such as radial basis functions and free-form deformations have been extensively studied in the context of image morphing and registration. These techniques have been surveyed by Wolberg [91] and, more recently, by Islam et al. [42]. Zitova and Flusser [97] and Sotiras et al. [80] provide reviews of image registration methods, including 2D deformation techniques.

Smooth space deformations also have been extensively studied in the context of three-dimensional shape editing. See Chapter 16.

Many existing space deformation methods provide continuity (C^0) but not smoothness (C^1). The few existing techniques for C^1 deformation either provide little control (like B-splines which are practical only with a regular grid mesh); or are hard to edit because they have a very large number of free parameters with non-intuitive effects and constraints; or yield deformations whose representations become increasingly complex as they are edited.

11.1 Non-spline methods

Some mesh-based space deformation methods attempt to obtain C^1 smoothness by the use of non-polynomial interpolating functions, which are determined only by the control mesh vertices and/or faces. Some techniques are:

- **Mean value coordinates:** this interpolation technique was one early approach, it is infinitely smooth almost everywhere, but is not C^1 at the vertices of the control mesh [29, 30, 40, 45, 52].
- **Harmonic coordinates:** this interpolation is smooth everywhere, but does not have closed formulas, and is expensive to compute numerically [44].
- **Green coordinates:** this technique is one of the most recent approaches to interpolation. It has closed formulas, but is still expensive to compute. It also yields quasi-conformal deformations, which partially preserves the shape of the deformed objects. [54, 78].

- **Radial basis functions:** another popular approach to non-spline modeling uses a linear combination of *radial basis* functions [17, 20]. Each time the deformation is edited, one radial element is added to its description and its coefficient is manipulated directly by the user. This approach is very flexible, but has the drawback that the complexity of the deformation increases without bound as editing goes on. It is also difficult to ensure that the deformation remains one-to-one, without “fold-over”.

11.2 Spline methods

Many deformation modeling methods use polynomial splines, that is, piecewise-defined functions where each piece is a polynomial on the domain coordinates, developed by Paul de Casteljau and Pierre Bézier [25].

Barr [5], and Sederberg and Parry [74] were the pioneers in the development of a space deformation method using splines as interpolation technique, namely free-form deformation (FFD). Due to its advantages, the FFD approach has been widely investigated allowing several extensions and variations [15, 31, 56].

Spline-based deformation editors for modeling of 3D objects generally use a control mesh consisting of either hexahedra [51, 74] or tetrahedra [8, 41, 95] defined by Bézier control points. As we shall see in Part III, one can also use prismatic elements [67].

In the two-dimensional context, most spline-based deformation techniques use quadrangular or triangular patches. Some applications include morphing [53, 63], registration [73], and vectorization [92]. Simplicial (triangular or tetrahedral) Bézier patches have the advantage over quadrangular patches since they can be joined with almost arbitrary topology. On the other hand, their continuity constraints are more complicated.

Compared to non-spline methods, splines generally use more control points, but can use a control mesh with fewer cells. An important advantage of the spline approach is that the complexity of the deformation is independent of its editing history. Namely, the number of patches and control points is fixed by the choice of the control mesh. Moreover, the existence of a simple analytic expression for the deformation around a point is an important advantage in many applications that require derivatives of the deformation.

Chapter 12

Triangular Splines Deformation

In this chapter, we review the theory of Bézier splines defined on *simplicial meshes*, whose cells are geometric t -dimensional simplices (intervals, triangles, tetrahedra, etc.). We consider here specifically the case $t = 2$, that is $\Omega \subseteq \mathbb{R}^2$, so the cells are triangles.

12.1 Triangular Bézier splines

As is well known [50], any polynomial f from $\mathbb{R}^2$ to $\mathbb{R}$ can be conveniently expressed as a linear combination of the *Bernstein-Bézier simplicial polynomials* relative to any simplex u . Let p be a point of $\mathbb{R}^2$, and β_0, β_1 and β_2 be the barycentric coordinates of p relative to the vertices of a triangle u . Let $d \in \mathbb{N}$ be a degree, and i, j and k be non-negative integers such that $i + j + k = d$. Then the two-dimensional Bernstein-Bézier polynomial of degree d with indices i, j , and k is defined as

$$B_{ijk}^u(p) = B_{ijk}(\beta_0, \beta_1, \beta_2) = \frac{d!}{i!j!k!} \beta_0^i \beta_1^j \beta_2^k. \quad (12.1)$$

There are $(d+1)(d+2)/2$ Bernstein-Bézier polynomials (and hence Bézier coefficients) for each triangle. The set of all polynomials B_{ijk} with $i + j + k = d$ is a basis for the bivariate polynomials of total degree at most d defined on $\mathbb{R}^2$. That is, every such polynomial can be written uniquely as

$$f(p) = \sum_{i+j+k=d} c_{ijk} B_{ijk}^u(p) \quad (12.2)$$

for all $p \in \mathbb{R}^2$. The coefficients c_{ijk} are called the *Bézier coefficients of f relative to the triangle u* [50]. The Equation (12.2) says that $f(p)$ is a linear combination of the control coefficients c_{ijk} . In fact, for any point p in $\mathbb{R}^2$, and any simplex u , it can be shown that

$$\sum_{i+j+k=d} B_{ijk}^u(p) = 1. \quad (12.3)$$

Therefore $f(p)$ is actually an affine combination of the c_{ijk} . Moreover, for any point p in the simplex u , the values of the polynomials $B_{ijk}^u(p)$ lie between 0 and 1. Therefore these values form a *partition of unity* and $f(p)$ is a *convex combination* of the c_{ijk} .

Each coefficient c_{ijk} can be associated to a *nominal position* u_{ijk} in the triangle u , whose barycentric coordinates are, by definition, $(i/d, j/d, k/d)$ relative to u . See Figure 12.1.

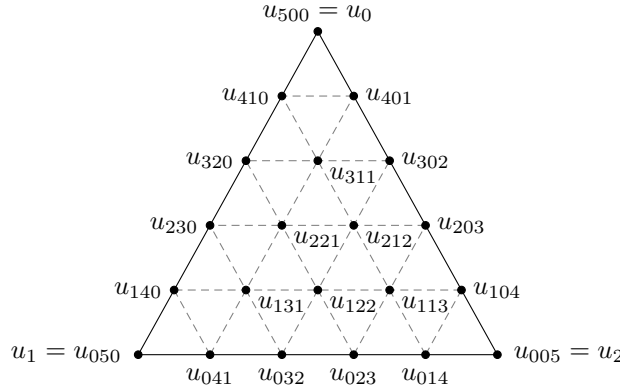


Figure 12.1: Nominal positions u_{ijk} (dots) of the Bézier coefficients c_{ijk} for the Bernstein-Bézier B_{ijk}^u of degree 5 relative to a triangle, and the local Bézier control net (solid and dashed lines).

The triangular grid defined by those points and the edges, shown in Figure 12.1, will be called the *local Bézier control net* of the triangle; the union of all those local nets is the *global Bézier control net*.

12.2 Using splines to model deformations

A deformation of a region $\Omega \subseteq \mathbb{R}^n$ can be defined as a function $\phi : \Omega \rightarrow \mathbb{R}^n$. A convenient way of modeling such functions is to let each coordinate of $\phi(x)$ be a spline function $\phi_r(x)$, with $0 \leq r \leq n$; all these splines being of the same degree and defined on the same mesh T . We call such function a *spline deformation*. The function ϕ deforms T , the *reference mesh*, into a new mesh $\phi(T)$ with curved boundaries, the *deformed reference mesh*. See Figure 12.2.

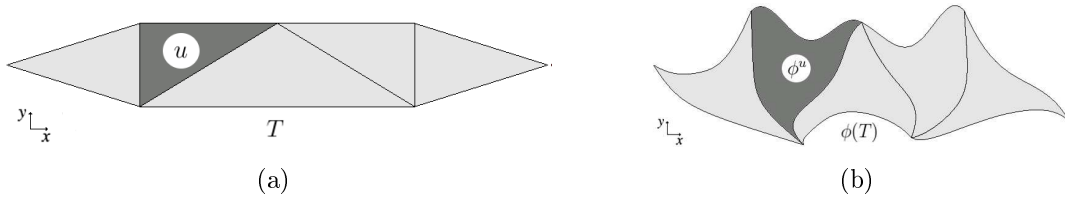


Figure 12.2: A deformation of $\mathbb{R}^2$ of the (a) reference mesh T in the (b) deformed reference mesh $\phi(T)$.

The Bernstein-Bézier polynomial representation can be used to describe the deformation ϕ . For $n = 2$, let u be a triangle of T and ϕ^u be the part of ϕ with domain u . For each coordinate r (0 for x or 1 for y), the Bézier coefficient $c_{ijk;r}^u$ of ϕ_r^u can be viewed as coordinate r of a point q_{ijk}^u , the *Bézier control point* of ϕ^u with indices i, j and k . The function ϕ^u can be modified by moving the points q_{ijk}^u . See Figure 12.3.

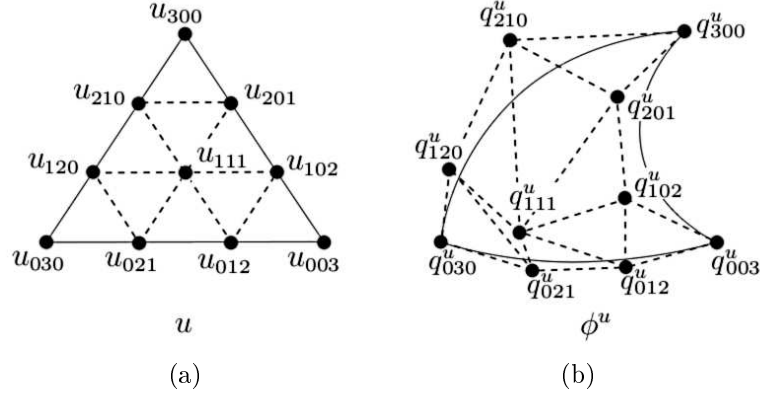


Figure 12.3: (b) Bézier control points q^u_{ijk} of a degree 3 patch ϕ^u from $\Omega \rightarrow \mathbb{R}^2$ and; (a) their nominal positions u_{ijk} on the domain triangle u . The curved triangle on the right is the image of u under the deformation ϕ^u .

Note that the control points q^u_{ijk} are distinct from their nominal positions u_{ijk} . They are also distinct from the images $\phi^u(u_{ijk})$ of those nominal positions, except at the corners. That is, $\phi^u(u_{d00}) = q^u_{d00}$, $\phi^u(u_{0d0}) = q^u_{0d0}$ and $\phi^u(u_{00d}) = q^u_{00d}$, but otherwise $\phi^u(u_{ijk}) \neq q^u_{ijk}$ in general.

12.2.1 Continuity constraints

Generally, we say that a deformation $\phi : \Omega \rightarrow \mathbb{R}^2$ is *continuous to order r* (C^r) if each coordinate of ϕ is continuous to order r . This is called *parametric continuity* which is distinct from the *geometric continuity* (G^r) sometimes used in computer graphics [25]. The latter is not appropriate here since the parametrization of the deformed mesh $\phi(T)$ is relevant, not just its shape.

Ensuring C^0 continuity

A spline function has C^0 continuity when there are no discontinuities across cell boundaries. For a spline function f defined on a triangulation T , the C^0 condition can be easily expressed in terms of the Bézier coefficients.

Let u and v be two adjacent triangles of T with Bézier coefficients c^u_{ijk} and $c^v_{i'j'k'}$. It is well known that the condition for f to be continuous across the common edge of u and v is that $c^u_{ijk} = c^v_{i'j'k'}$ for all i, j, k, i', j', k' such that the nominal positions coincide, that is, such that $u_{ijk} = v_{i'j'k'}$. Similarly, a deformation $\phi : \Omega \rightarrow \mathbb{R}^2$ defined by spline is C^0 -continuous across an edge if have $q^u_{ijk} = q^v_{i'j'k'}$ whenever $u_{ijk} = v_{i'j'k'}$. See Figure 12.4.

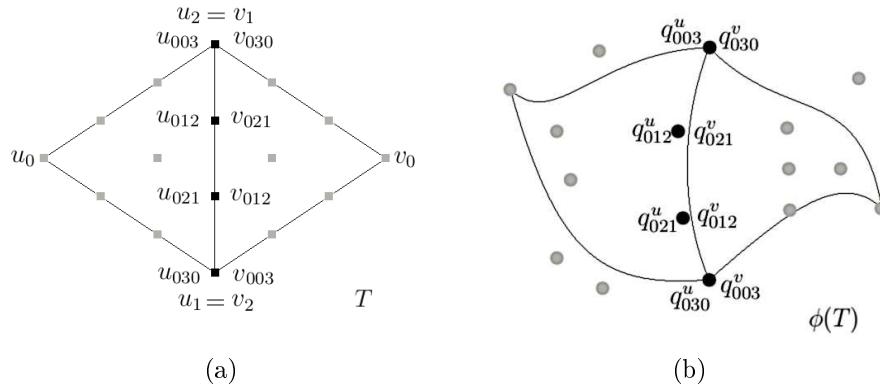


Figure 12.4: (a) Nominal positions and (b) Bézier control points of a deformation ϕ of degree 3 which satisfy C^0 continuity constraints.

Ensuring C^1 continuity

We say that a function is *smooth* when it has at least C^1 continuity. A polynomial spline is always smooth in the interior of any cell. Thus the spline is smooth along the edge between two adjacent cells u and v if the first derivatives of the corresponding polynomials f^u and f^v in any direction coincide at any point on that boundary. For simplicial polynomial splines, this requirement translates into a set of linear constraints on the Bézier coefficients of f^u and f^v . Specifically, f has C^1 continuity along that common edge if and only if

$$c_{0jk}^v = c_{0jk}^u \quad (12.4)$$

$$c_{1jk}^v = \beta_0 c_{1,j,k}^u + \beta_1 c_{0,j+1,k}^u + \beta_2 c_{0,j,k+1}^u \quad (12.5)$$

for all j, k such that $j + k = d - 1$, where β_0, β_1 and β_2 are barycentric coordinates of v_0 relative to u_0, u_1 and u_2 [50].

For a spline deformation $\phi : \Omega \rightarrow \mathbb{R}^2$, the C^1 continuity is given by analogous conditions to equations in (12.4) and (12.5) over the Bézier control points q_{ijk}^t instead of the coefficients c_{ijk}^t . Namely, the deformation ϕ is continuous across the shared edge if and only if

$$q_{0jk}^v = q_{0jk}^u \quad (12.6)$$

$$q_{1jk}^v = \beta_0 q_{1,j,k}^u + \beta_1 q_{0,j+1,k}^u + \beta_2 q_{0,j,k+1}^u. \quad (12.7)$$

We call Equation (12.7) the *quadrilateral condition*. It says that the quadrilateral formed by the control points $q_{1jk}^v, q_{1jk}^u, q_{0,j+1,k}^u, q_{0,j,k+1}^u$ must be an affine image of the quadrilateral formed by their nominal positions. See Figure 12.5.

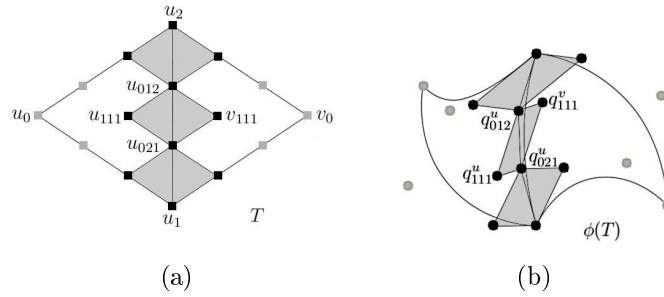


Figure 12.5: (a) Nominal positions and (b) Bézier control points of a deformation ϕ of degree 3 which satisfy C^1 continuity constraints (the gray diamonds).

12.3 Local control

The theory of C^1 -continuous 2D splines with triangular cells has been extensively studied, for example, by Schumaker [50]. It is known that there is a minimum degree d of the interpolating spline that allows local editing of the spline while maintaining its smoothness [34, 93]. If the degree d is too low, the constraints are interconnected in such a way that the required changes propagate from triangle to triangle over all the reference mesh T , so that local control is not possible.

In particular, for triangle meshes in $\mathbb{R}^2$, the smallest degree that allows local control with C^1 continuity is $d = 5$, which we use in the examples that follow. In this case, each triangle has 21 Bézier control points as shown in Figure 12.6.

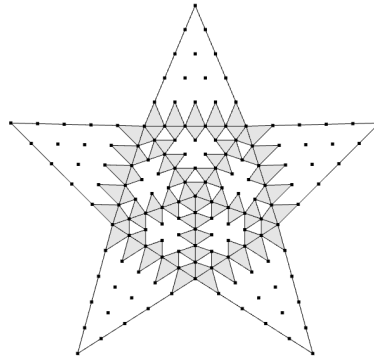


Figure 12.6: A reference mesh T for a spline deformation of degree 5, showing the Bézier control points and the quadrilateral conditions.

Let n be the number of parts and l be the number of edges in the free border of the mesh, it can be proved (by induction on n) that a simplicial spline of that size has $\frac{25}{2}n + O(l)$ distinct control points and $\frac{13}{2}n + O(l)$ degrees of freedom; that is, about $\frac{25}{13} = 1.9$ control points for each degree of freedom.

In comparison, a quadrangular bicubic tensor spline of degree 3 (the smallest degree that provides local control and C^1 continuity) with n patches has $9n + O(l)$ control points and $4n + O(l)$ degrees of freedom; that is, about $9/4 = 2.25$ control points for each degree of freedom. Therefore, despite requiring a higher degree than tensor splines to obtain locality, the triangular spline deformation needs fewer control points to achieve the same flexibility.

Chapter 13

Spline Representation

In this chapter, we define the representation and the data structure to store the reference mesh T , the spline deformation on that mesh, and its continuity constraints. This definition is important to ensure the consistency of data stored in the structure, which are basis for the 2DSD algorithm. This data structure was used in the previous version of our editor [67].

13.1 Notation

13.1.1 Labeling and orientation of the edges

For each oriented edge e of the reference mesh T , we denote by l^e and r^e its source and destination vertices, respectively. Moreover, we denote by u^e the adjacent triangle to the left of edge e , and v^e the adjacent triangle to the right of edge e . The notation p^e and n^e are the other vertices of those triangles. See Figure 13.1. Note that, if we consider the same edge in the opposite direction, the labels are swapped in pairs u^e with v^e , l^e with r^e , and p^e with n^e .

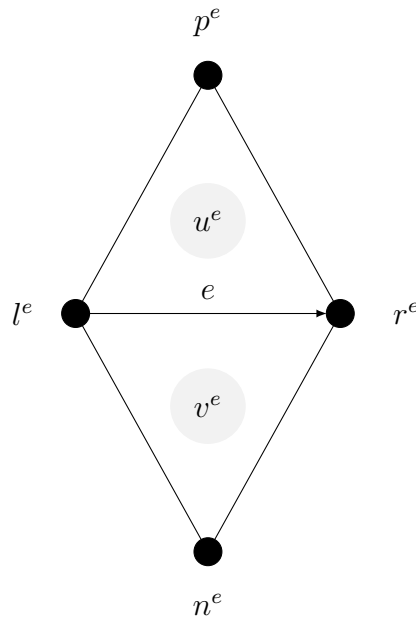


Figure 13.1: Labels of the triangles of the reference mesh T that shared the oriented edge e , and their vertices.

13.1.2 Labeling and orientation of the quadrilateral conditions

There are d quadrilateral conditions $Q_0^e, Q_1^e, \dots, Q_{d-1}^e$ for each oriented edge e of the reference mesh T , numbered according to the direction of the edge e . For each quadrilateral condition, we denote l_i^e, r_i^e, p_i^e , and n_i^e the four Bézier control points of that condition, as shown in Figure 13.2. We say that l_i^e and r_i^e are *left* and *right medial members*; and p_i^e and n_i^e are *previous* and *next extreme members* of the quadrilateral condition i , respectively.

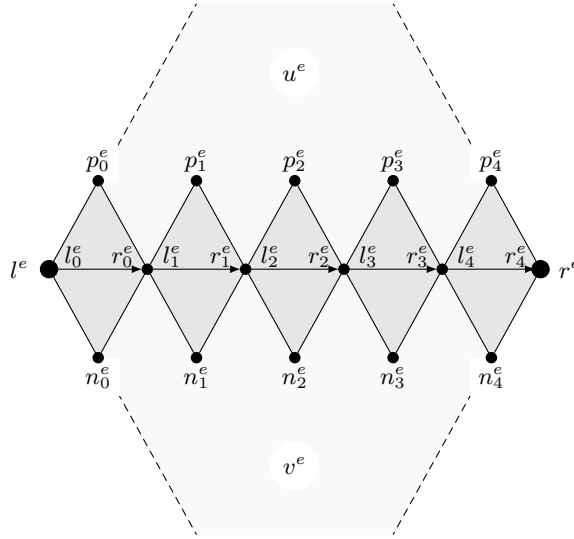


Figure 13.2: Labels of the Bézier control points that form the quadrilateral conditions on the shared edge e of the reference mesh T , for degree $d = 5$.

If e' is the edge e taking in the opposite direction, then condition Q_i^e coincides with $Q_{d-i-1}^{e'}$, p_i^e with $n_{d-i-1}^{e'}$, and l_i^e with $r_{d-i-1}^{e'}$.

13.2 Data structure

13.2.1 Representation of the reference mesh

The reference triangulation T is represented in our editor by the set of its vertices, the set of its edges, and the set of its faces. So, there are three structures to represent a triangulation:

- **Point**: has the coordinates $\mathbf{x}$ and $\mathbf{y}$ on the triangulation T .
- **Edge**: has the pointers $\mathbf{l}$ and $\mathbf{r}$ to the source and destination vertices of the edge, respectively. The orientation of the edge is chosen arbitrarily.
- **Face**: has the pointers $\mathbf{p0}$, $\mathbf{p1}$, and $\mathbf{p2}$ to its vertices and $\mathbf{e0}$, $\mathbf{e1}$, and $\mathbf{e2}$ to its edges. The vertices and edges are numbered in counter clockwise orientation from a arbitrary vertex. So that $\mathbf{p0}$, $\mathbf{p1}$, and $\mathbf{p2}$ are the source vertices of $\mathbf{e0}$, $\mathbf{e1}$, and $\mathbf{e2}$, respectively.

13.2.2 Representation of the spline

The spline deformation ϕ is represented by a data structure with three record types:

- **ControlPoint**: represents a Bézier control point of the spline ϕ . It has the current coordinates $\mathbf{x}, \mathbf{y}, \mathbf{z}$ in $\mathbb{R}^3$, and an integer **type** that indicates its position on a triangle of the reference mesh T . Note that two or more control points, that must be identified to obtain C^1 continuity, are represented by a single record of type **ControlPoint**.
- **BezierTriangle**: represents a triangle of the reference mesh T in one particular orientation. It has a pointer **f** to the corresponding face of the T , and a vector **Cp** of pointers to its $(d+1)(d+2)/2$ control points q_{ijk}^f . These points are stored in lexicographic order of the indices ijk .
- **BezierEdge**: represents a edge of the reference mesh T shared by two triangles. It has a pointer **e** to the corresponding record of type **Edge**; and two pointers **p** and **n**, to the **Point** record of the other two vertices of the triangles that share the edge, according to Figure 13.1. It also has pointers to the two records of the type **BezierTriangle** **u** and **v**, and a vector **Qc**[0...d-1] of pointers to the d records of the type **Quadrilateral** that represent the C^1 continuity constraints related to this edge. See Section 13.2.3.

13.2.3 Representation of the C^1 constraints

Each quadrilateral condition is represented by a record of the type:

- **Quadrilateral**: has a four pointers **p**, **l**, **n** and **r**, to the four **ControlPoint** record affected by the quadrilateral condition. It also has an integer **orientation** that indicates the orientation of the quadrilateral with respect to the direction of the shared edge (0 - same direction; 1 - opposite direction).

The quadrilateral condition is stored only once in the data structure, that is, it is stored in the attribute **Qc**[**i**] of each shared edge e , either the quadrilateral Q_i^e or the quadrilateral $Q_{d-i-1}^{e'}$, where e' is the edge e in the opposite direction.

The orientation of the quadrilaterals $Q_0^{e'}$ and $Q_1^{e'}$ or $Q_{d-1}^{e'}$ and $Q_{d-2}^{e'}$ must be opposite to the orientation of the **BezierEdge** record because it is necessary to ensure that the quadrilaterals around a vertex have consistent orientations. See Figure 13.3.

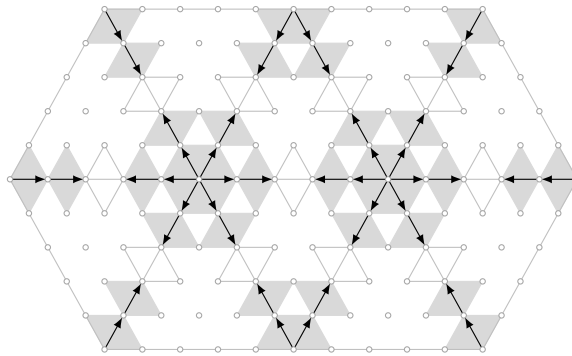


Figure 13.3: A spline with consistent orientation of the quadrilaterals around the vertices.

The other quadrilaterals Q_i^e with $(2 \leq i \leq d-3)$ can be oriented arbitrarily. In Figure 13.4 the quadrilateral $Q_c[2] = Q_2^e$, that is, it has the same orientation of the edge e .

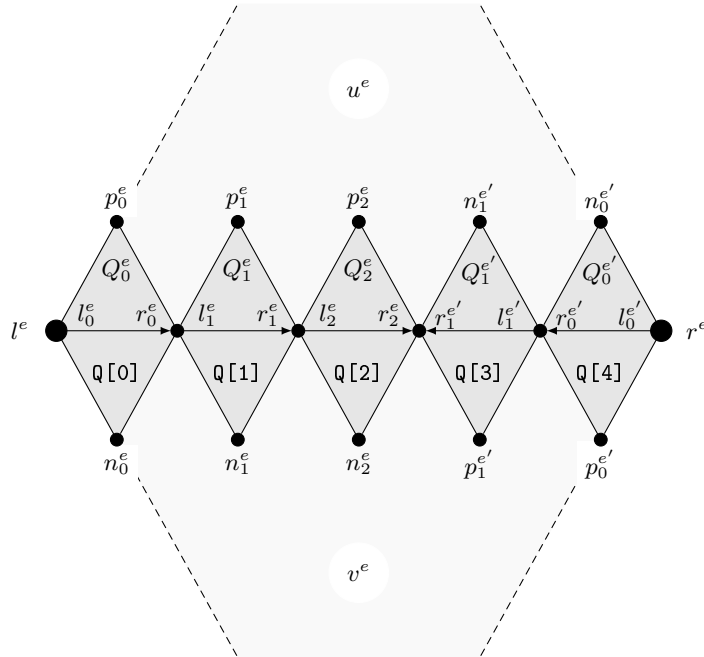
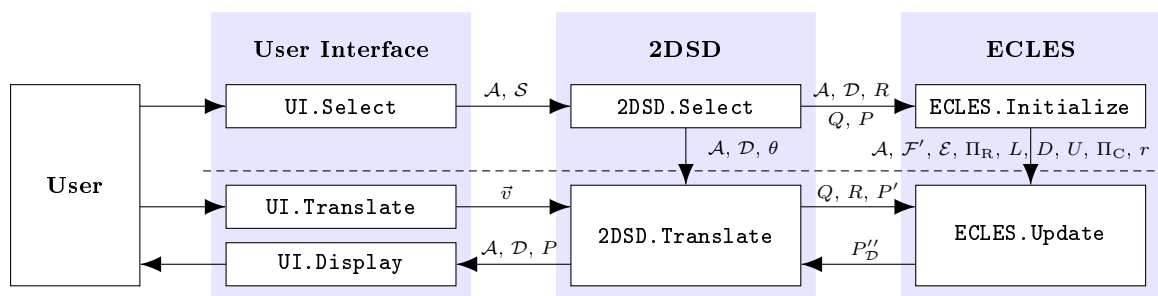


Figure 13.4: Labels of the control points to obtain a consistent orientation of the quadrilaterals around the vertices.

The 2DSD Editing Algorithm

Each constraint can be expressed by a linear equation with integer coefficients, if the coordinates of the vertices of the reference mesh T are rational. Therefore, we can use the ECLES general parameter editing method, as part of our interactive algorithm for editing of 2D spline deformations, in order to adjust the control points preserving the $\mathbf{C}^1$ continuity of the spline while trying to obey the changes indicated by the user.

In Part III, we describe in detail a typical local editing action, as seen by the user of the editor. Here the focus is on the back stage of the editor, namely the interaction of the user interface with the 2DSD algorithm. Figure 14.1 shows the interaction model between the user, the application's user interface, our interactive algorithm 2DSD, and the ECLES general method.



The interaction process has two steps: the first occurs once in each editing action when the user chooses the control points to be adjusted; and the second step occurs one or more times when the user modifies the position of those control points, e.g. by dragging them with the mouse. To simplify, only one editing operation is shown (translation of the anchor points described in Section 18.5.1). Other operations, such as rotation and scaling of the anchor points can be implemented in similar ways.

Initially, through appropriate gestures of the user interface, the user selects a set of anchors ($\mathcal{A}$) and a set of initial derived control points ($\mathcal{S}$). These sets are transmitted by the interface method (`UI.Select`) to the first part of the 2DSD algorithm, the `2DSD.Select` procedure (see Section 14.2). This procedure chooses the final derived control points (set $\mathcal{D}$) based on the user-selected set $\mathcal{S}$, and computes the relative magnitude θ_s of the desired displacement for each control point p_s .

Then, the sets $\mathcal{A}$ and $\mathcal{D}$ of control points and the coefficient matrix of constraints R are given to `ECLES.Initialize` (see Section 5.2). This procedure constructs the coefficient matrix of the linear system, in factored form (Π_R, L, D, U, Π_C) , and computes the true rank r of that matrix.

14.1.1 Soft translation

In soft translation, the user displaces all the anchors by the same vector v . See Figure 14.2.

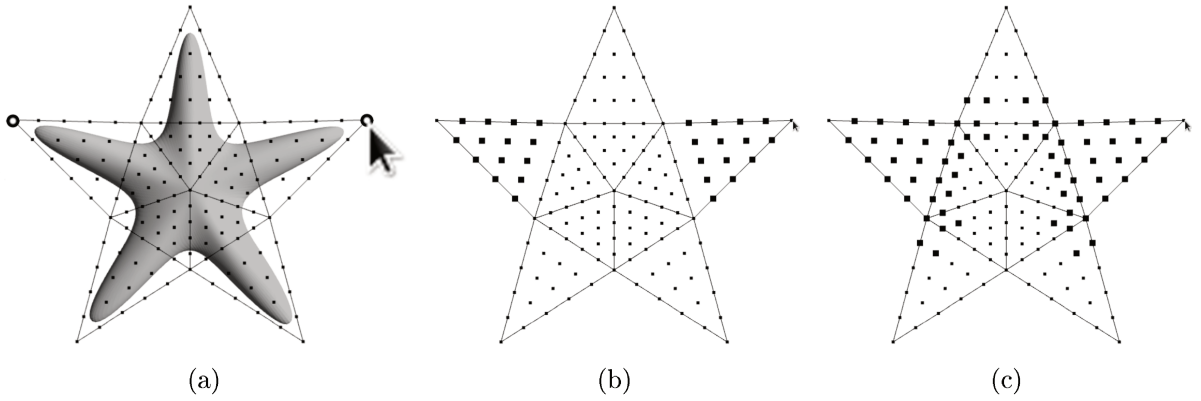


Figure 14.2: Translating of two anchor points. (a) Anchor points (set $\mathcal{A}$) specified by the user; (b) initial derived points (set $\mathcal{S}$) selected by the user; and (c) derived points (set $\mathcal{D}$) specified by the `2DSD.ExpandDerived`.

The suggested translation $\theta_s v$ for a derived point p_s decreases in magnitude as one goes away from the anchor points.

The second part of our algorithm is executed when the user specifies (e.g. by dragging with the mouse) a change in the anchor points, represented by some list v of change arguments, summarized by a displacement vector v , in Figure 14.1. This data is passed by the corresponding interface method (`UI.Translate`) to the second part of the 2DSD algorithm, the `2DSD.Translate` procedure (see Section 14.3). This procedure computes the new positions P'_A of the anchors and the suggested positions P'_D for the derived control points. The `2DSD.Translate` then passes those informations to `ECLES.Update` (see Section 5.3), that computes the new positions P''_D of the derived control points satisfying the constraints. Then `2DSD.Translate` updates the current position P of the control points in $\mathcal{A}$ and $\mathcal{D}$, and gives that information to the user interface for visual feedback - namely, to display the effects of the change on the deformed mesh and/or the deformed object. See Figure 14.3.

The 2DSD ensures (by rounding, if necessary) that the vector v has rational coordinates. The suggested displacements $\theta_s v$ to the derived points can be computed with floating point and then rounded. The choice of $\mathcal{D}$ (see Section 14.2.1) ensures that the constraints can always be solved, for any choice of $\mathcal{A}$ and vector v . Note that if $P'_A = P_A + \theta_s v$ any constraint that involve only points of $\mathcal{A}$ are satisfied by P'_A if and only if they are satisfied by P_A . More generally, this is true if P'_A is an exact image of P_A by an affine map.

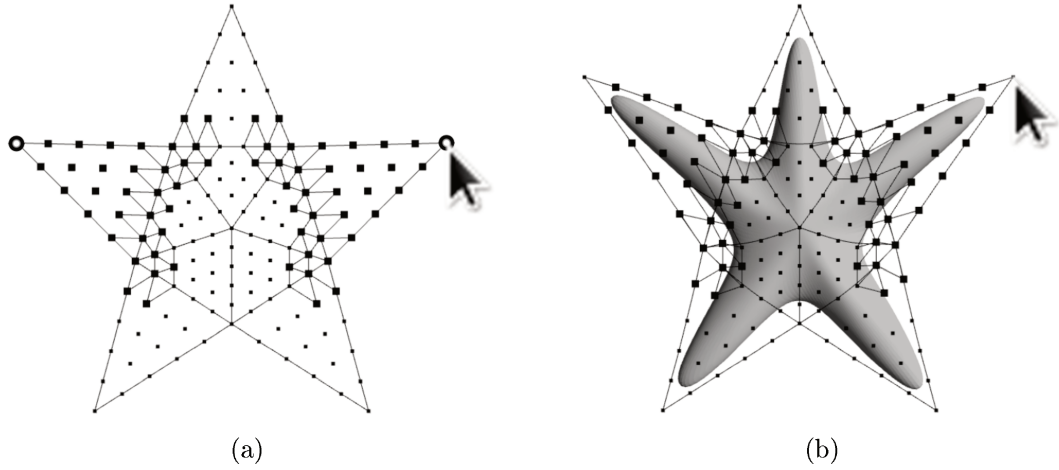


Figure 14.3: Translating of two anchor points. (a) Anchor points $\mathcal{A}$ (black open dots), derived points $\mathcal{D}$ (black dots), and non-redundant relevant constraints specified by ECLES; and (b) control points updated.

The **ECLES.Update** procedure then computes the new positions $P''_{\mathcal{D}}$ of the derived points satisfying the constraints. Then, **2DSD.Translate** updates the current position P_s of each control point $s \in (\mathcal{A} \cup \mathcal{D})$, and gives that information to the user interface to display the effects of the change on the deformed mesh and/or the deformed object.

14.1.2 Soft rotation and scaling

Other editing operations can be easily added to 2DSD, by adding a new procedure for each operation. Our prototype editor (see Chapter 18) also supports the operations of local soft rotation and local soft scaling of one or more anchor points. See Figure 14.4.

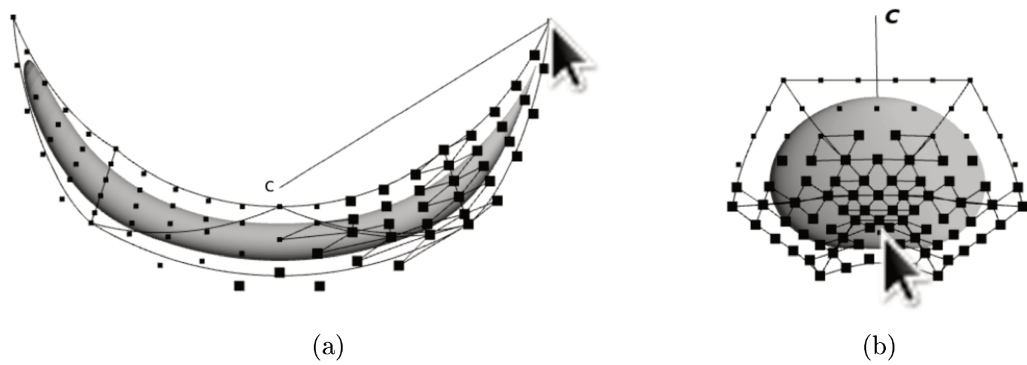


Figure 14.4: Examples of (a) rotation and (b) scaling of one anchor point.

For the soft rotation (see Section 18.5.2), the user defines an arbitrary angle α and center $c \in \mathbb{R}^2$. The suggested angle α'_s of each derived point s will be $\theta_s \alpha$. In soft scaling (see Section 18.5.3), the user defines a center c and a scale factor γ . The suggest scaling for each derived point s will be γ^{θ_s} .

The new positions p'_s of the anchors are computed in floating point (they are usually irrational) and then rounded to rationals with suitable precision. As a result, they are not a

linear map of the original positions p_s . If there are four or more anchors, the new anchor positions may violate some constraints. Therefore, for this operation, `ECLES.Initialize` calls the `ECLES.CheckStrongSolvability` procedure to check the strong solvability condition in equation 2.3, and, if it is not satisfied, fails and notifies the application. Then, `2DSD.Select` returns a message to the user, asking her to select a different (usually smaller) set of anchor points.

14.2 The `2DSD.Select` procedure

The `2DSD.Select` procedure is described in Algorithm 16.

Algorithm 16: `2DSD.Select`

Data: $\mathcal{A}, \mathcal{S}$: set of anchor and initial derived points;
 G : the control graph;
 R : $l \times c$ coefficient matrix of all constraint equations;
 Q : $n \times j$ matrix of independent terms of the constraints;
 P : $n \times j$ matrix of coordinates of the current positions of the control points.
Result: $\mathcal{D}, \mathcal{F}'$: sets of derived and relevant fixed points;
 $\mathcal{E}$: set of relevant constraints;
 θ : relative magnitude to the displacement of point;
 Π_R, L, D, U, Π_C, r : fraction-free LDU factoring of $R_{\mathcal{ED}}$.

```

1 begin
2    $\mathcal{D} \leftarrow \text{2DSD.ExpandDerived}(\mathcal{A}, \mathcal{S}, G)$ 
3    $\theta \leftarrow \text{2DSD.ComputeRelMagnitude}(\mathcal{A}, \mathcal{D}, G)$ 
4    $(\mathcal{F}', \mathcal{E}, \Pi_R, L, D, U, \Pi_C, r) \leftarrow \text{ECLES.Initialize}(\mathcal{A}, \mathcal{D}, R, Q, P, \text{true}, \text{Turner})$ 

```

The `2DSD.Select` procedure expands the initial set $\mathcal{S}$ of derived points to the set $\mathcal{D}$ using the `2DSD.ExpandDerived` procedure, in step 2 (see Section 14.2.1).

In step 3, the `2DSD.Select` procedure uses `2DSD.ComputeRelMagnitude` (see Section 14.2.2) to compute a real coefficient θ_s for each control point p_s relative to the displacement of the anchor points, which will be used by `2DSD.Translate` (see Section 14.3) to compute the positions P'_s .

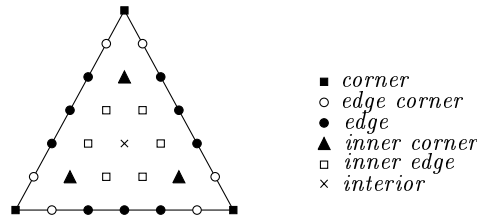
Finally, in step 4, `2DSD.Select` calls the `ECLES.Initialize` procedure, described in Section 5.2, to identify the relevant and non-redundant C^1 continuity constraints for the editing action, and to obtain a collection of matrices Π_R, L, D, U, Π_C and the rank r needed for the `ECLES.Update` procedure, described in Section 5.3.

14.2.1 The 2DSD.ExpandDerived procedure

The 2DSD.ExpandDerived procedure is used in step 2 of 2DSD.Select to determine a superset $\mathcal{D}$ of the specified derived points $\mathcal{S}$ in order to ensure the ECLES solvability condition.

Initially, 2DSD.ExpandDerived sets $\mathcal{D} \leftarrow \mathcal{S}$. Then, for each $s \in (\mathcal{A} \cup \mathcal{D})$, the algorithm finds all quadrilateral conditions that involve the control point p_s ; and then adds the indices of zero or more control points that enter into these conditions to the set $\mathcal{D}$. The process is iterated until all points in $\mathcal{A} \cup \mathcal{D}$ have been examined. When $d \geq 5$, the final set of derived control points $\mathcal{D}$ can be confined to the triangles that own the control points in $(\mathcal{A} \cup \mathcal{S})$ and only a few adjacent triangles.

For the purpose of this step, each Bézier control point $p_s = q_{ijk}^u$ is classified into six types according to its nominal position u_{ijk} in the triangle u . See Figure 14.5.



Type	Description
<i>corner</i>	$i = d$ or $j = d$ or $k = d$.
<i>edge corner</i>	$i = 0$ and $(j = 1$ or $k = 1)$ or $j = 0$ and $(i = 1$ or $k = 1)$ or $k = 0$ and $(i = 1$ or $j = 1)$.
<i>edge</i>	none other above and $(i = 0$ or $j = 0$ or $k = 0)$.
<i>inner corner</i>	$i = j = 1$ or $i = k = 1$ or $j = k = 1$.
<i>inner edge</i>	none other above and $(i = 1$ or $j = 1$ or $k = 1)$.
<i>interior</i>	$i \geq 2$ and $j \geq 2$ and $k \geq 2$.

Figure 14.5: Classification of the control points of a Bézier patch of degree 6.

The type of the point p determines the set of quadrilateral constraints that apply to that point and the additional points inserted in the set $\mathcal{D}$. A point p of type *interior* does not take part in any quadrilateral condition, so it does not contribute to the set $\mathcal{D}$. A point p of any other type contributes the additional derived points according to rules are shown in Figure 14.6.

When applying these rules, the algorithm skips any control points that would lie on non-existing triangles, and any quadrilateral conditions that would depend on them. These rules ensure that there is at least one derived point for each quadrilateral involved in the editing action. The set $\mathcal{D}$ obtained ensures the strong solvability condition of the ECLES.

The relevant quadrilateral constraints determined by the $\mathcal{A}$ and $\mathcal{D}$ points may include redundant equations. This happens, for example, when $\mathcal{A} \cup \mathcal{D}$ includes a *corner* or *edge corner* point of an interior vertex.

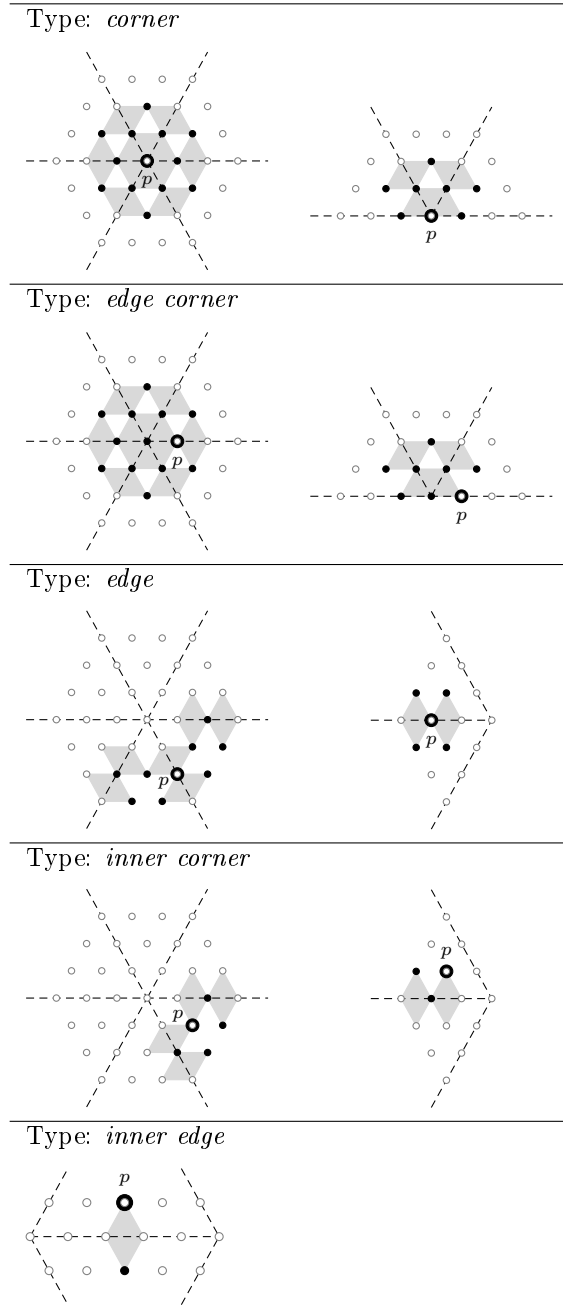


Figure 14.6: Relevant continuity constraints (gray diamonds) and derived points added to $\mathcal{D}$ (solid dots) depending on the type of the control point p_s with $s \in \mathcal{A} \cup \mathcal{D}$ (open dot). For each type, the left figure shows a typical situation where the point p_s is sufficiently far from the triangulation's border. The right figure shows a situation near the border where some of those control points and constraints are missing. These diagrams are generalized to vertices of arbitrary degree in the obvious way.

14.2.2 The 2DSD.ComputeRelMagnitude procedure

The `2DSD.ComputeRelMagnitude` procedure computes the value θ_s , which defines the relative magnitude of the desired displacement of each point p_s , that is, how much the suggested position P'_s is affected by the specified displacement v of the anchor points. The value θ_s is a number between 0 and 1 given by the formula

$$\theta_s = \frac{\delta''_s}{\delta'_s + \delta''_s} \quad (14.1)$$

where δ'_s is the distance between the point p_s and the nearest anchor point; and δ''_s is the distance between the point p_s and the nearest fixed point. Both δ'_s and δ''_s are graph-theoretical distances measured on the global Bézier control net G . The distances are computed by Dijkstra's algorithm [16]. The value θ_s computed for anchor points is 1, and for fixed points is 0.

14.3 The 2DSD.Translate procedure

The `2DSD.Translate` procedure is called by the user interface to implement soft translation, every time the anchor points are moved by the user to a new position. It is described in Algorithm 17.

Algorithm 17: 2DSD.Translate

Data: v : displacement applied to the anchor points;
 Q : independent terms of the constraints;
 $\mathcal{A}, \mathcal{D}, \mathcal{F}'$: set of anchor, derived and relevant fixed points;
 θ : relative magnitude to the displacement of points;
 Π_R, L, D, U, Π_C, r : matrices and rank r returned by `ECLES.Initialize`;
 P : matrix of coordinates of the current positions of the control points.
Result: P : updated matrix of coordinates of the control points.

```

1 begin
2   for each  $s \in \mathcal{P}$  do  $P'_s \leftarrow P_s + \theta_s v$ 
3    $P''_{\mathcal{D}} \leftarrow \text{ECLES.Update}(\mathcal{A}, \mathcal{F}', \mathcal{E}, R, Q, \Pi_R, L, D, U, \Pi_C, r, P')$ 
4   for each  $s \in \mathcal{D}$  do  $P_s \leftarrow P''_s$ 
5   for each  $s \in \mathcal{A}$  do  $P_s \leftarrow P'_s$ 

```

In step 2, the `2DSD.Translate` procedure computes the suggested positions P'_s of each control point s using the value θ_s . Then, in step 3, the new positions $P''_{\mathcal{D}}$ of the derived points are computed by the `ECLES.Update` procedure (see Section 5.3). Finally, in steps 4 and 5, `2DSD.Translate` sets the final position P_s of each control point.

14.4 An example

Suppose that the set $\mathcal{A}$ is the point $p = q_{032}^u$ between the triangles u (right) and v (left), shown in Figure 14.7, and $\mathcal{S} = \emptyset$.

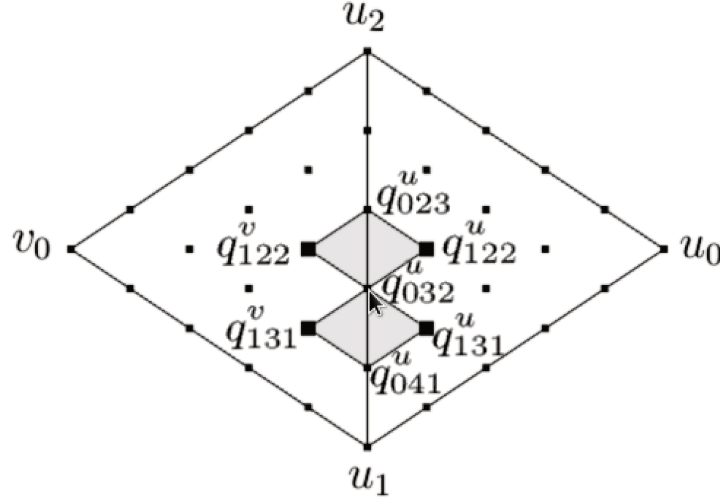


Figure 14.7: Editing point q_{032}^u with the derived points q_{122}^v , q_{122}^u , q_{131}^v , q_{131}^u .

According to Figure 14.6, the algorithm will select the four derived control points $\mathcal{D} = \{q_{122}^v, q_{122}^u, q_{131}^v, q_{131}^u\}$ (large black dots in Figure 14.7). Based on these points and on the anchor point q_{032}^u , **ECLES.Initialize** identifies two relevant constraints, which are included in the set $\mathcal{E}$:

$$-(\beta_0 + \beta_1 + \beta_2)q_{122}^v + (\beta_0)q_{122}^u + (\beta_1)q_{032}^u + (\beta_2)q_{023}^u = 0 \quad (14.2)$$

$$-(\beta_0 + \beta_1 + \beta_2)q_{131}^v + (\beta_0)q_{131}^u + (\beta_1)q_{041}^u + (\beta_2)q_{032}^u = 0 \quad (14.3)$$

where β_0 , β_1 , and β_2 are the barycentric coordinates of v_0 relative to u_0 , u_1 , and u_2 . The set $\mathcal{F}'$ has the two points: $q_{023}^u = q_{023}^v$ and $q_{041}^u = q_{041}^v$.

The matrix form of equations (14.2) and (14.3) of the example in Figure 14.7 is

$$\begin{bmatrix} -\beta & 0 & \beta_0 & 0 \\ 0 & -\beta & 0 & \beta_0 \end{bmatrix} P_{\mathcal{D}}'' = - \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} P_{\mathcal{A}}' - \begin{bmatrix} \beta_2 & 0 \\ 0 & \beta_1 \end{bmatrix} P_{\mathcal{F}'} \quad (14.4)$$

where the constant matrix $Q_{\mathcal{E}}$ is zero.

In the example in Figure 14.7, the equations are linearly independent and the system is indeterminate. Therefore, the new positions $P_{\mathcal{D}}''$ are computed by **ECLES.Update** which solves the system (14.4) using the least squares criterion.

Part III

The PrisMystic Editor

Chapter 15

Goals and Motivation

In this chapter, we describe the goals and motivation for the interactive editor of deformations for 3D models that we developed, called PrisMystic.

15.1 Goals

The original motivation for developing the PrisMystic editor was to help the analysis of images of organisms and organic structures. Specifically, the editor was used to deform 3D models of those organisms to match the images of real specimens obtained by optical microscopes. See Figure 15.1.

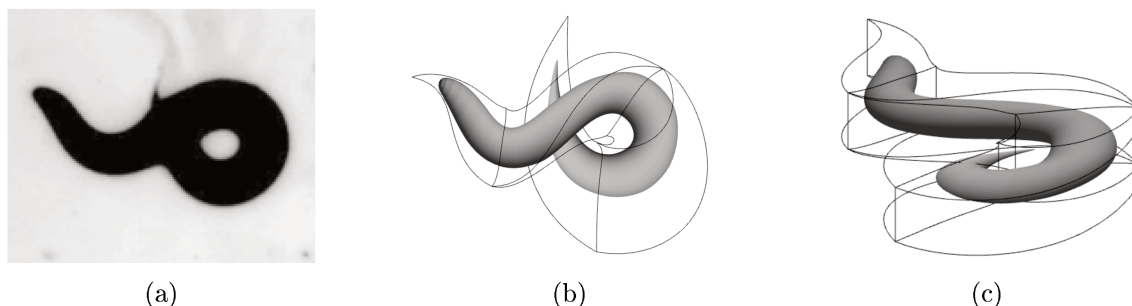


Figure 15.1: (a) An actual microscope image of the protozoan *C. elegans*; (b) 2D view; and (c) 3D view of a deformed model obtained with PrisMystic, matching the image (a).

Although the models for this application are three-dimensional, the deformations are essentially two-dimensional with little change in depth, because the third dimension cannot be easily perceived through a microscope.

This editor can also be used for models of other mostly planar biological structures, such as blood cells or neurons grown on a flat surface. As it turns out, the editor is also to edit 2D deformations of other non-biological objects, such as editing terrains (see Section 19.2).

15.2 Relation to the Masters version

The PrisMystic editor is an improved version of the editor previously described in my Masters dissertation [67]. This editor uses a 2.5D space deformation technique (see Section 17.3) also previously proposed.

The improvements include: using the ECLES algorithm, described in Part I, instead of floating-point linear algebra packages; a more flexible and general method for the selection of control points (allowing multiple anchors); and a different goal function for the least squares method. Also, we used the general 2DSD approach, described in Part II, to connect the user interface to the ECLES solver, described in Part I.

With these changes, it would be now relatively easy to extend the editor to accommodate other affine constraints, such as C^2 smoothness, vertical or horizontal alignment, fixed points, among others.

Chapter 16

Related work

The literature presents some approaches to deformations of three-dimensional models. We focus, specifically, on space deformation methods that use volumetric meshes as a control tool to deform the space surrounding the 3D model. In this approach, the deformation of the control mesh is transferred to the embedded model using interpolation techniques.

16.1 Deformation of 3D models

There are many approaches for modeling of 3D objects, which can be classified in *Physics-based* or *geometry-based* methods. The goal of the Physics-based deformation methods is to simulate the physical behavior of objects under internal and external forces. Realistic simulations require the use of these methods. However, they are not adequate for interactive applications or real-time simulations due to the high computational cost [61]. In this thesis, we did not consider using these approaches since they would require accurate physical models of the interior of the models. For example, the model of a microorganism should include information of elasticity and viscosity of tissues, which is unlikely to be available, even for the best-studied organisms.

The geometric methods are simpler, faster, and relatively easier to implement. Moreover, they allow arbitrary deformations without regard to the laws of physics [6]. The challenge of these methods is to produce deformations that are close to reality, and simultaneously to maintain the initial characteristics of the objects, such as smoothness of their surface. The geometric approaches can be classified in *surface* or *space* deformation methods.

The surface deformation methods assume that the object is represented by its external surface. The parameters or vertices of this surface are manipulated directly by the user. Usually, these methods are used when the deformation must preserve details of the surface of the object. Linear techniques produce good results for small deformations [13]. However for large deformations, they can generate unwanted artifacts due to linearization errors, which leads to the popularization of the nonlinear techniques [94].

The space deformation methods allow the user modifies the object mesh indirectly by deforming the space surrounding it [31]. These methods can have fewer parameters than the object mesh itself, reducing the editing work of the deformation. Usually, they are used in interactive applications where the model has thousands of vertices. In these applications, the surface deformation techniques are impractical due to the large number of parameters to be manipulated by the user, and because it is difficult to maintain the smoothness of the surface. Another advantage of space deformation methods is that the modeling of the deformation is independent of the model's representation.

16.2 Space deformations

Smooth space deformations have been extensively studied in the context of three-dimensional shape editing. A survey focusing on interactivity is presented by Gain and Bechmann [31]. They classify the space deformation methods according to the dimension of the *control objects* used to define the deformation:

- **Point-based deformations (0D):** the user handles control points which are freely positioned on the space surrounding the model. Each control point has a region of influence within which the movement of the point causes displacements of the model's points. Some methods of this class are: Constraint-based Deformation [3], Directly Manipulated FFD [31], Dirichlet FFD [31], and Simple Radial Deformation [1, 2, 11, 49].
- **Curve-based deformations (1D):** the user handles one or more curves or control axes for local or global deformations of the model. Examples of such methods are: Regular Global Deformations [31], Generalized de Casteljau Deformation [31], Axial Deformation [31], Wires [31], Bender [31], and Skinning Frameworks [5, 55];
- **Surface-based deformations (2D):** the user handles one or more control surfaces for local or global deformations of the model. Some of these methods are: Parametric Patch Tools [26, 27], Star-convex Tools [23], Triangular Meshes [45, 48], and Field-based Tools [88, 89];
- **Volume-based deformations (3D):** the user handles a coarse mesh surrounding the model. Sederberg and Parry [74] introduced the space deformation approaches using volumetric meshes formed by hexahedra: the Bézier Free-Form Deformation (FFD). Then, other methods were developed: FFD with Local Control [39, 51] and FFD with Arbitrary Topology Grid [8, 15, 41, 56, 95].

Space deformation editors that use three-dimensional control meshes seem better suited to arbitrary smooth deformations. The deformed control mesh provides an immediate intuitive understanding of the general nature of the deformation, and of the scope of each control parameter. In particular, it becomes easier to notice and avoid singularities in the deformation (places where the deformation is not injective, meaning that space is being folded over itself). For these reasons, we have opted for a 3D (or “2.5D”) control method in this work.

Most 3D space deformation methods described in the literature use either hexahedral [51, 74] or tetrahedral [8, 41, 95] control meshes defined by Bézier control points. Other works use prismatic elements [12, 67], including the method described in this thesis. We have opted using a control mesh of triangular prisms with vertical walls. So, our deformation method is 2.5D according to the classification of Gain and Bechmann [31].

16.3 Interpolation techniques

Interpolation techniques are used to transfer the deformation of the control mesh to the embedded model in the deformed space. Some space deformation methods use non-spline interpolating functions, which are determined by the control mesh vertices and/or faces only, in order to obtain smooth deformations (see Section 11.1).

Another interpolation technique widely adopted in graphical applications, that provide local control and smooth deformations are the spline functions [25], defined in Chapter 12. Barr [5],

and Sederberg and Parry [74] were the pioneers in the development of a space deformation method using splines as interpolation technique, namely Free-Form Deformation (FFD). Due to its advantages, the FFD approach has been widely investigated allowing several extensions and variations [15, 31, 56].

16.4 Spline interpolation

The splines functions can also be used to maintain the continuity and smoothness of the control mesh and of the inner 3D model[8, 67, 74]. Simplicial Bézier patches have the advantage over quadrangular ones since they can be joined with almost arbitrary topology. On the other hand, their continuity constraints are more complicated.

However, the requirement of C^1 continuity makes this approach very hard to edit because of the large number of control parameters. For example, with a tetrahedral mesh, the degree of the interpolating spline must be at least 5 to allow localized editing of the mesh. Therefore, to specify each tetrahedron in the mesh the user must specify the position of 56 control points ($56 \times 3 (x, y, z) = 168$ real parameters) [8]. With an hexahedral mesh one can have C^1 splines of degree 3, however each cell requires 64 control points (192 parameters) [31].

With so many control points, it becomes difficult to identify and select the ones that must be edited to achieve a desired effect. Furthermore, the editing software must automatically move many additional control points in order to satisfy the C^1 continuity constraints, increasing the user's confusion. Complex user interfaces, with high-level abstractions, have been developed to address this problem [8, 31], but they do not solve it completely.

Compared to these methods, meshes with prismatic cells require only 42 control points per cell. All these points are outside the control mesh, and its coordinates are more restricted so that there are only 84 free real parameters ($21 \times 2 (x, y) + 42 \times 1 (z)$). Moreover, the necessary editing interface is much simpler than other 3D control meshes because these coordinates can be separately edited. So, the user needs to edit at most 21 control points per cell, at a time. See Table 16.1.

	Degree	Number of points	Inner points	Real parameters
Tetrahedron	5	56	yes	$56 \times 3(x,y,z) = 168$
Hexahedron	3	64	yes	$64 \times 3(x,y,z) = 192$
Prism	5	42	no	$21 \times 2(x,y) + 42 \times 1(z) = 84$

Table 16.1: Comparison among cells of meshes consisting of tetrahedra, hexahedra, and prisms.

For our application, we believe that our 2.5D method provides a reasonable balance between control mesh size, naturalness of editing, and computation speed.

Chapter 17

Overview of the editor

In this chapter, we describe an overview of PrisMystic editor and its deformation paradigm, that is, the 2.5D space deformation approach [67, 68].

17.1 The 3D model

We assume that the organism to be deformed is given as a dense triangular *model mesh* M with tens of thousands of triangles, which is read from a file in the Wavefront format (.obj). See Figure 17.1.

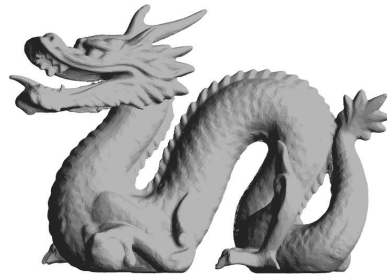


Figure 17.1: The 3D model mesh M of a *Dragon* [82], represented by a triangular mesh.

Note that the space deformation approach allows other representations of the model, and not just triangular meshes. The PrisMystic editor also supports point cloud models. See Figure 17.2.

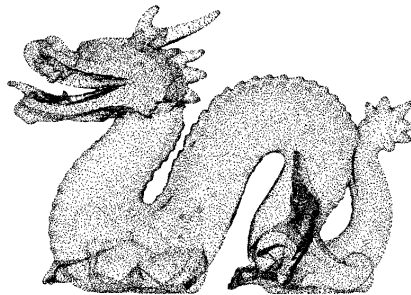


Figure 17.2: The 3D model mesh M of a *Dragon* [82], represented by a point cloud.

17.2 The 3D reference mesh

To use the PrisMystic editor, the user must provide a *3D reference mesh* P around the object consisting of a single layer of triangular prisms with flat top and bottom faces, parallel to the xy plane, and vertical walls. The projection of the 3D reference mesh on the xy plane is the *2D reference mesh* T of Part II. The projection of P and its projection on the z axis is an interval $[a, b]$. See Figure 17.3.

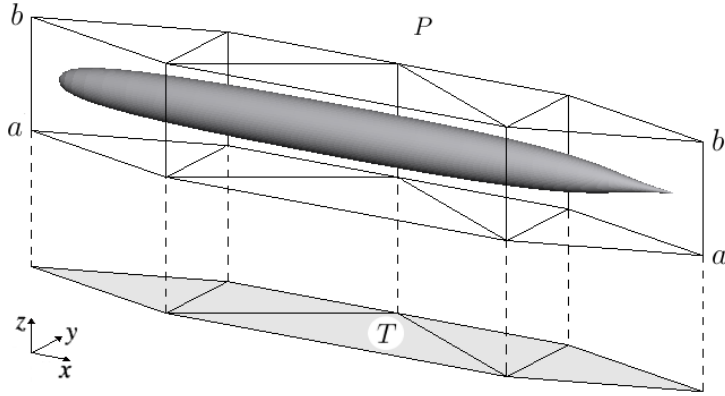


Figure 17.3: A 3D model mesh M surrounded by a 3D reference mesh P , and the corresponding 2D reference mesh T .

17.2.1 Defining the barycentric coordinates

In order to compute the deformed image $\psi(p)$ of a point p of the object, the program finds the prism U of P that contains it and computes the barycentric coordinates β_0, β_1 and β_2 of p with respect to the triangle u of the 2D reference mesh T corresponding to the prism U . The program also computes the vertical position of p relative to both triangular faces u^0 and u^1 of U , that is, the two numbers $\alpha_0 = (b - z)/(b - a)$ and $\alpha_1 = (z - a)/(b - a)$. See Figure 17.4.

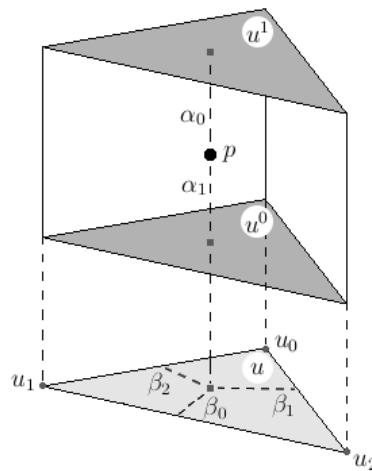


Figure 17.4: Barycentric coordinates $\alpha_0, \alpha_1, \beta_0, \beta_1$ and β_2 of a point p of the 3D model mesh M , related to the prism U of the 3D reference mesh P .

If the object is defined by mesh M of triangles, the deformation is applied only to the vertices of the mesh assigning that the triangles remain flat. This the resulting mesh M is not exactly the deformation $\psi(M)$ applied to the original object. However, if M is sufficiently fine, the discrepancy between M and $\psi(M)$ can be negligible.

17.3 Deformation paradigm

The deformations allowed by our editor consist of 2D spline deformations of the x and y coordinates combined with (x, y) -dependent 1D stretching maps of the z coordinates. All three functions are defined on a same reference mesh T in $\mathbb{R}^2$.

In many applications, the viewing conditions are such that the organism is almost viewed from the same angle (always sideways, or always from the top), so that these “2.5D” deformations are sufficient. If the organism can be viewed from different angles, then the model should be appropriately rotated before being loaded into Prismystic.

More precisely the deformation is a function ψ from P to $\mathbb{R}^3$ that consists of a two-dimensional deformation $\phi : T \rightarrow \mathbb{R}^2$, and two spline functions $\sigma_0 : T \rightarrow \mathbb{R}$ and $\sigma_1 : T \rightarrow \mathbb{R}$, all with the same degree d ,

$$\psi(p) = (\psi(p).x; \psi(p).y; \psi(p).z), \quad (17.1)$$

where $p = (x, y, z) \in \mathbb{R}^3$ and

$$\begin{aligned} \psi(x, y, z).x &= \phi(x, y).x \\ \psi(x, y, z).y &= \phi(x, y).y \\ \psi(x, y, z).z &= \alpha_0(x, y)\sigma_0(x, y) + \alpha_1(x, y)\sigma_1(x, y). \end{aligned}$$

Note that, for each position p inside of P , the coordinate z of the point $\psi(p)$ varies between $\sigma_0(x, y)$ and $\sigma_1(x, y)$.

The deformation function ψ then takes the reference mesh P to the *3D deformed reference mesh* $\psi(P)$, which consists of a set of prisms with vertical walls, whose top and bottom faces are curved triangles forming two spline surfaces. The *deformed model mesh* $\psi(M)$ is another triangular mesh with the same topology of M , obtained by mapping all vertices of M through the function ψ . See Figure 17.5.

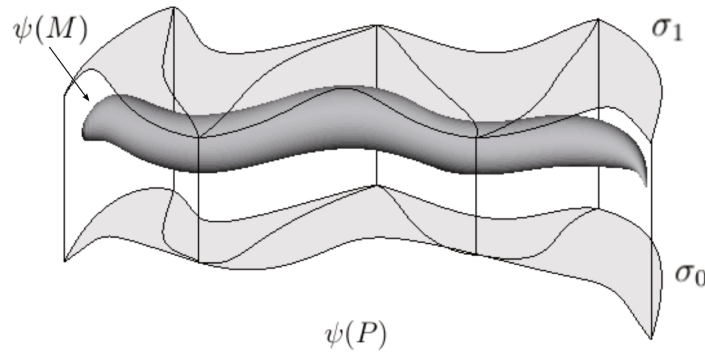


Figure 17.5: A deformed reference mesh $\psi(P)$ surrounding a deformed model mesh $\psi(M)$.

17.4 Editing the deformation

The splines σ_0 , σ_1 and ϕ , described in Section 17.3, are defined by their Bézier control points. The user can modify the deformation by moving the control points with the mouse. Considering splines with degree $d = 5$, each function has pieces defined by $(d+1)(d+2)/2 = 21$ Bézier control points, according to Chapter 12.

The editor has separate modes for adjusting the (x, y) deformation (a spline mapping $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^2$) and the top and bottom surfaces (two spline functions $\sigma_0, \sigma_1 : \mathbb{R}^2 \rightarrow \mathbb{R}$). The algorithm 2DSD, described in Part II, is used by the editor to edit these three functions.

The user can select one of three editing modes available in the editor: *xy-mode*, *z0-mode* and *z1-mode*. In the *xy-mode*, the user has a view of the top of the deformed reference mesh $\psi(P)$, that is, the 2D deformed reference mesh $\phi(T)$. Thus, the user can only modify the coordinates x and y of each control point q_{ijk}^u of ϕ . See Figure 17.6.

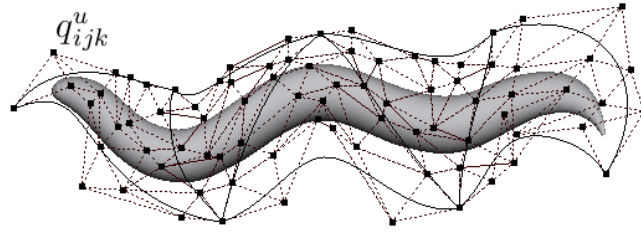


Figure 17.6: View of the deformed reference mesh $\psi(P)$ in the *xy-mode*, showing the control points and the global Bézier control net G (dotted line) on the deformed reference mesh $\phi(T)$.

In the *z0-mode* and *z1-mode*, the user can edit the coefficients $c_{ijk;r}^u$ of the splines σ_0 or σ_1 , respectively. In these modes, the user has an oblique view of the deformed reference mesh $\psi(P)$. For each triangle $u \in T$ and each set of indices i, j and k with $i + j + k = d$, there are two coefficients: $c_{ijk;0}^u$, for the bottom surface; and $c_{ijk;1}^u$, for the top surface (as described in Section 12.2). Therefore, there are $2(d+1)(d+2)/2 = (d+1)(d+2)$ control points, for each prism of P but only the coordinate $z0$ or $z1$ can be modified by the user. Each one of the two splines σ_0 and σ_1 is edited independently, by moving the point with the mouse along the vertical line that contains it. See Figure 17.7.

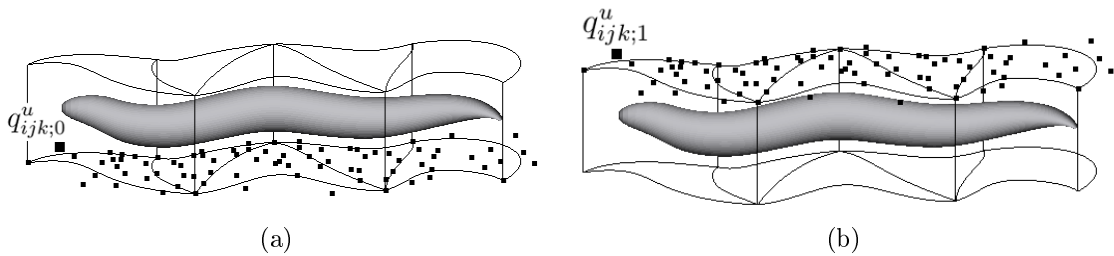


Figure 17.7: View of the deformed reference mesh $\psi(P)$ in the *z0-mode* (a) and *z1-mode* (b), showing the control points of the splines σ_0 and σ_1 , respectively.

17.5 Ensuring the spline continuity

As observed in Section 12.2.1, the C^0 continuity condition is ensured because only one control point is available for two or more nominal positions that coincide, that is, nominal positions on the shared edges or vertices by two or more triangles in T .

The C^1 continuity constraints are represented by quadrilateral conditions, as shown in Figure 12.6. When the user edits one or more control points (anchors), the program uses the 2DSD algorithm to determine one or more additional control points that must be modified in order to ensure the C^1 continuity, and to apply the necessary adjustments as the anchors are moved by the user, as described in Part II.

Chapter 18

Editing Paradigm

In this chapter, we present the user interface of the PrisMystic editor and its main controls. We also describe the editing paradigm used by the editor that allows the view and editing in distinct modes: *xy*-mode, *z0*-mode, and *z1*-mode.

18.1 PrisMystic user interface

The user interface of the PrisMystic editor has a main editing window, a menu bar, and two control windows. See Figure 18.1.

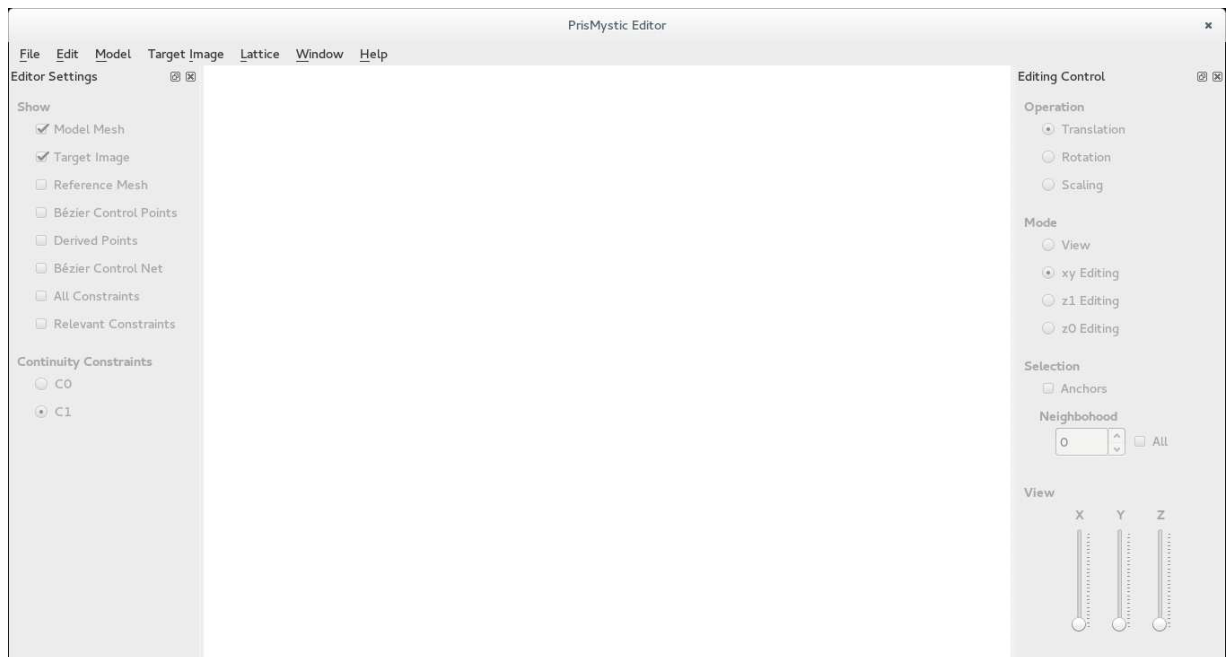


Figure 18.1: The user interface of the PrisMystic editor in the initialization mode.

The main window is used for editing and visualization of the deformations. Various options and parameters of the editor are selected in the windows *Editor Settings* (see Section 18.1.1) and *Editing Control* (see Section 18.1.2).

The PrisMystic has five distinct *editor mode*:

1. Initialization;
2. 3D viewing;
3. Editing the deformation spline ϕ in xy plane;
4. Editing the deformation spline σ_0 in z axis;
5. Editing the deformation spline σ_1 in z axis.

When started the editor is in the initialization mode and the other modes are disabled, as shown in Figure 18.1.

18.1.1 *Editor Settings* window

The possible settings for the PrisMystic editor include items which are to be shown in the main editing window, and the available constraints. See Figure 18.2.

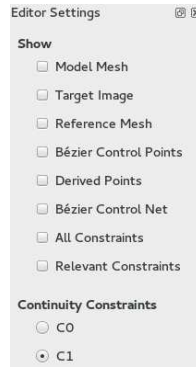


Figure 18.2: The options of the Editor Settings window of the PrisMystic editor.

Among the available options, the user can choose to display in the main editing window the model mesh, and/or the reference mesh, and/or the target image to be matched (displayed in the background).

When the reference mesh is displayed, the user can choose to display any combination of:

- Bézier control points of the deformed reference mesh;
- Derived control points $\mathcal{D}$;
- Global Bézier control net;
- Constraints on the control points (for example, the C^1 continuity constraints);
- Non-redundant relevant constraints defined during the editing of a deformation.

The current set of anchors $\mathcal{A}$, is always displayed. The user can also choose between enforcing only (that is, no constraints on the control point) C^0 , or the C^1 continuity constraints as described in Section 12.2.1.

18.1.2 Editing Control window

The Editing Control window lets the user choose between view and editing modes (xy , $z0$, and $z1$), and controls the view mode. Moreover, the user can select the type of editing operation to be applied to the anchor points: translation, rotation, or scaling. See Figure 18.3.

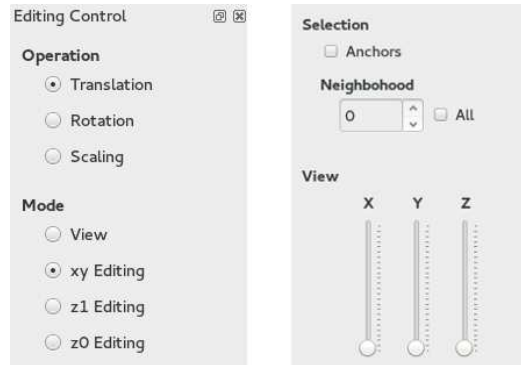


Figure 18.3: The options of the Editing Control window of the PrisMystic editor.

In both view and editing mode, it is possible display any object available in the *Show* group of the *Editor Settings*. See Figure 18.2.

18.2 Initialization mode

When the editor is initialized all its options are disabled, except the menu options. See Figure 18.1. In the menu bar, there are options to load the model mesh (it can be a 2D or 3D mesh), the reference mesh of prisms, and a target image that is to be matched, if necessary. There are also options to clear these three objects, and to save the model and the reference mesh displayed in the main editing window.

The model and the reference mesh are loaded from files in Wavefront format `.obj`, which contain lists of vertices and faces of a triangular mesh. The reference mesh P file should contain a 2D mesh of triangles, that is the top face of P , whose vertices should have the z coordinate $z = b$, for some positive number b . The bottom face of P is assumed to be the same triangulation, with z coordinate $a = -b$ (see Section 17.2).

From this information, the editor builds the global Bézier control net, defines the Bézier control points, and locates the vertices of the model mesh M relative to the reference mesh P (see Section 17.2). Then, the editor enables the editing xy -mode by default.

18.3 View mode

The view mode is enabled by selecting *View* among the available modes in the *Editing Control* window. See Figure 18.3.

The default setting of the editor view mode displays the model and the reference mesh of prisms. The view mode also allows the user to rotate the objects, displayed in main editing window, in any direction through the sliders X , Y and Z . See Figure 18.3.

Figure 18.4 shows the PrisMystic editor in the view mode, displaying the deformed reference mesh $\psi(P)$ and the deformed 3D model mesh $\psi(M)$.

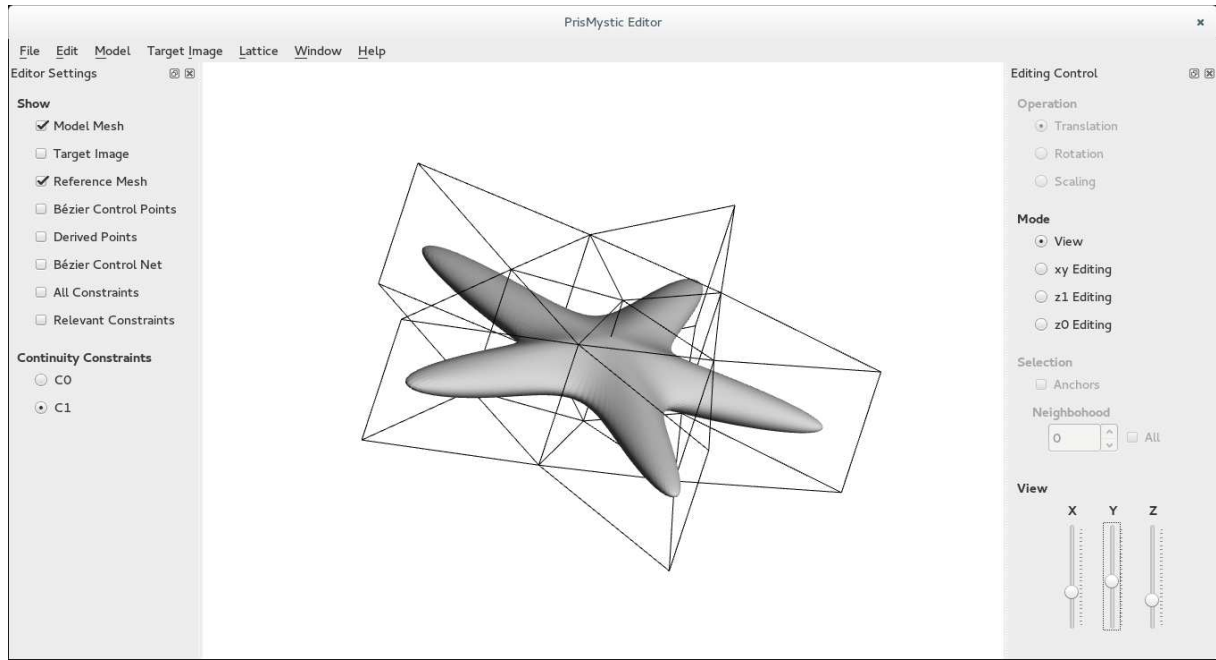


Figure 18.4: The view mode of the PrisMystic editor.

18.4 Point selection sub-mode

The xy , $z0$, and $z1$ editing modes have a *selection* sub-mode, in which the user selects the set $\mathcal{A}$ of anchor points of the editing action, and the set $\mathcal{S}$ of initial derived points.

By default of this sub-mode, the user sees the model mesh, the reference mesh, and its control points. The selection sub-mode is enabled in all editing modes, and the editor maintains the view according to the selected editing mode: xy -mode, $z0$ -mode, and $z1$ -mode. As a shortcut, if the editor is in any of the editing modes, and there are no anchors selected, clicking one control point makes it the only anchor and sets the set $\mathcal{S}$ to empty.

18.4.1 Anchor selection

When the checkbox *Anchors* is checked, the user can choose a set $\mathcal{A}$ of anchor points by clicking on them, with the mouse. To move the anchor, the user needs to uncheck the *Anchors* option. See Figure 18.5(a)

18.4.2 Neighborhood selection

The user can choose the radius of the *Neighborhood* δ_{max} in the editing region, through a numeric widget. See Figure 18.3. The initial set $\mathcal{S}$ of derived points is then set to all points at maximum distance δ_{max} from the anchors, in the global Bézier control net G .

For example, Figure 18.5(b) shows the initial derived points $\mathcal{S}$ selected in the maximum neighborhood defined by the user.

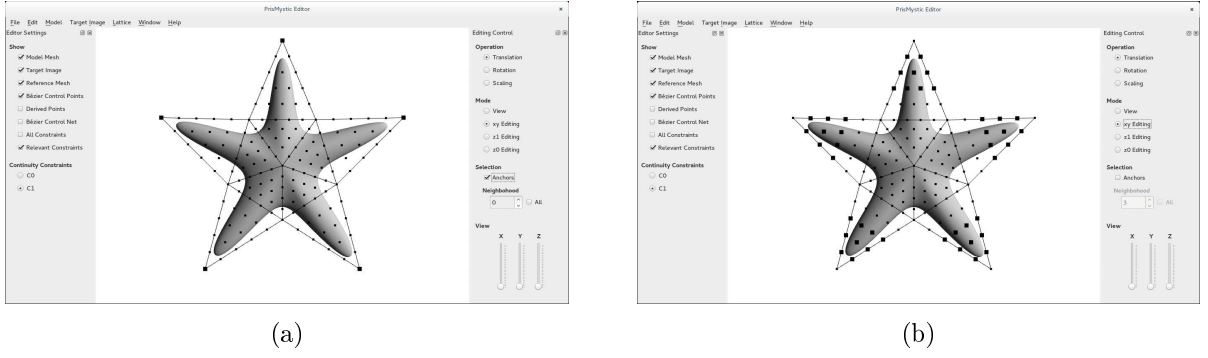


Figure 18.5: The PrisMystic editor highlighting (a) the selected anchor points, and (b) the initial derived points with $\delta_{max} = 3$, in the xy -mode.

When the checkbox *All* is checked, the numeric widget is disabled, and the maximum distance is defined automatically as the greatest possible ($\delta_{max} = +\infty$). So, all control points are automatically included in the set $\mathcal{S}$, except the anchor point clicked by the user. This function allows that the operation applied to the anchor point to be reproduced at all control points, that is, it is useful to apply global translation, rotation and scaling. See Figure 18.6.

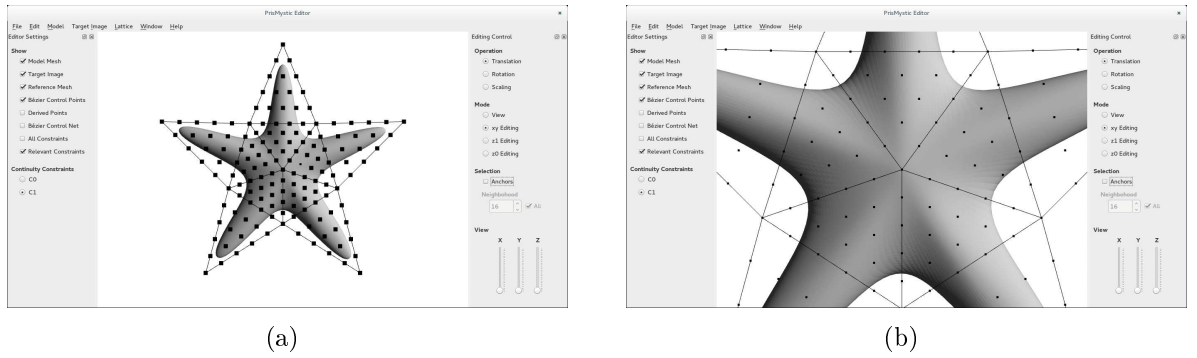
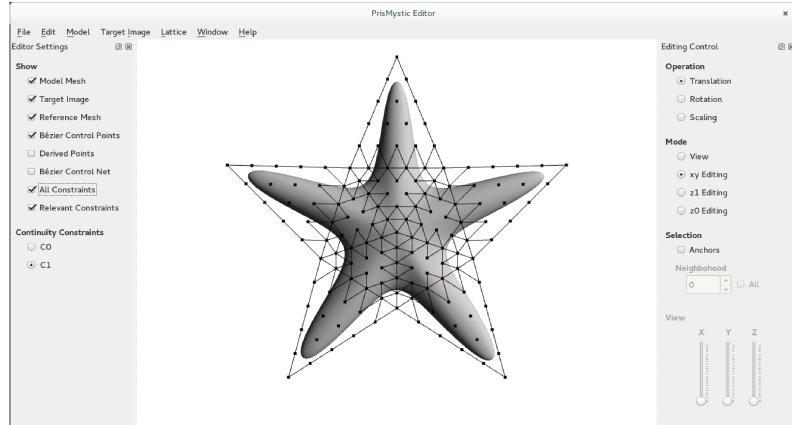


Figure 18.6: The PrisMystic editor highlighting (a) the initial derived points, and (b) the scaling operation applied to all control points.

When the user exits the selection sub-mode and clicks on any one anchor point to drag it, the initial derived set $\mathcal{S}$ is automatically selected by the interface. Then, it is augmented with the extra control points needed for solvability, obtaining the set $\mathcal{D}$ of derived points (see Section 14.1).

18.5 Editing xy -deformation

In the editing xy -mode, the user is presented with a top view of the reference mesh of prisms, that is, the 2D deformed reference mesh T and the Bézier control points. See Figure 18.7. The user can modify the coordinates x and y of the control points by selecting one operation (translation, rotation, or scaling), and dragging any one anchor point with the mouse.

Figure 18.7: The editing xy -mode of the PrisMystic editor.

18.5.1 Local soft translation

In the xy -mode of our editor, the local soft translation is the default editing operation. The user defines a displacement $\vec{v}$ for any one anchor points by clicking on one of the anchor points and dragging it to a new position, with the mouse. The same displacement will be applied to all anchors. Then, the new positions of the derived points is computed by 2DSD algorithm. See Figure 18.8.

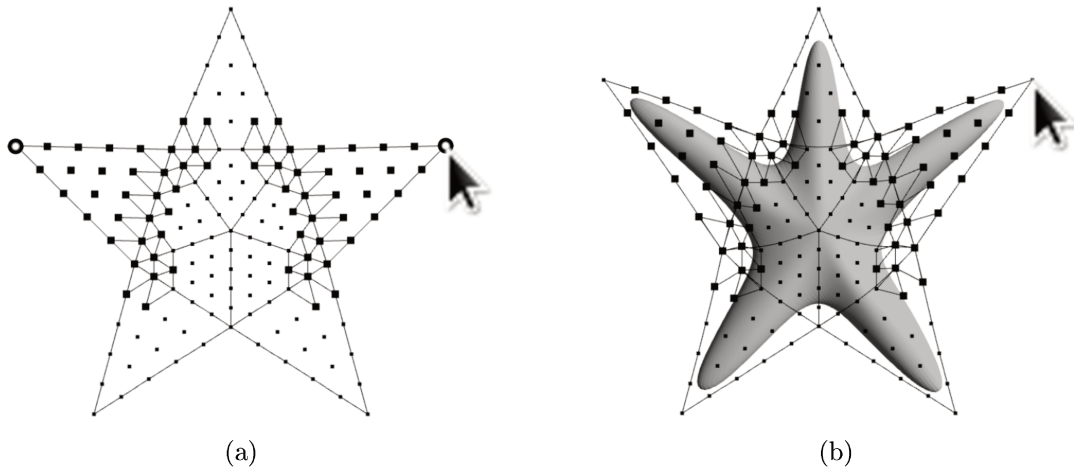


Figure 18.8: (a) Before and (b) after a soft translation of two anchor points (black open dots) with $\delta_{max} = 4$, showing the derived points $\mathcal{D}$ (black dots), and the C^1 continuity constraints (quadrilaterals).

18.5.2 Local soft rotation

Another editing operation available in xy -mode is the local soft rotation of the anchor points around a user-chosen center point. The user defines a center c in the xy plane by clicking on the desired position in the main editing window, with the right button of the mouse. Then, the user chooses an angle α by clicking on any one of the anchor points and dragging it to a new position, with the mouse. A guide line between the center c and the new position is displayed during the dragging. The same rotation angle α around c is applied to all other anchors. Then, the new positions of the derived points is computed by 2DSD algorithm. See Figure 18.9.

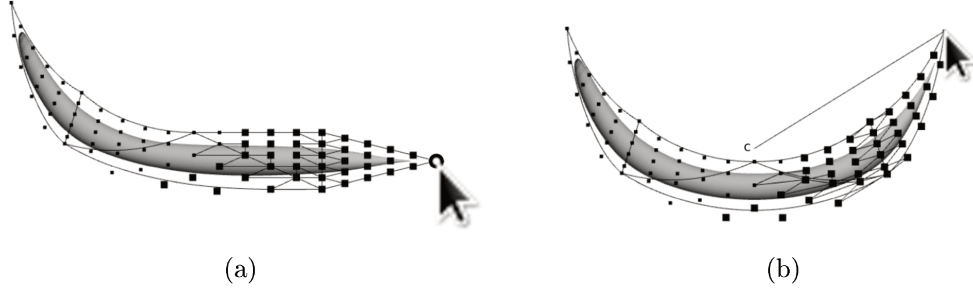


Figure 18.9: (a) Before and (b) after a soft rotation of one anchor point (black open dot) with $\delta_{max} = 8$ around the center c , showing the derived points $\mathcal{D}$ (black dots), and the C^1 continuity constraints (quadrilaterals).

18.5.3 Local soft scaling

The local soft scaling operation also available in the xy -mode, expands or contracts the anchor points relative to a user-chosen center point. As in the rotation operation, the user defines a center c in the xy plane by clicking on the desired position in the main editing window, with the right button of the mouse. Then, the user chooses a scale factor γ by clicking on one of the anchor points and dragging it to a new position with the mouse. A guide line between the center c and the new position is displayed during the dragging. The same scale factor γ around c is applied to all other anchors. Then, the new positions of the derived points are computed by the 2DSD algorithm. See Figure 18.10.

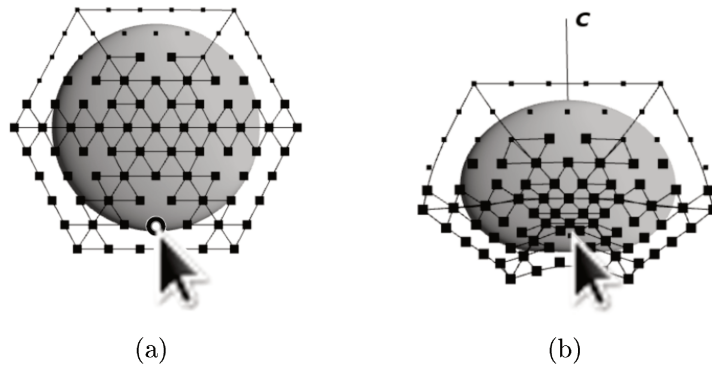


Figure 18.10: (a) Before and (b) after a soft scaling of one anchor point (black open dot) with $\delta_{max} = 6$ around the center c , showing the derived points $\mathcal{D}$ (black dots), and the C^1 continuity constraints (quadrilaterals).

18.6 Editing z -deformation

In the editing modes $z0$ -mode and $z1$ -mode, the user is presented with a oblique view of the deformed reference mesh $\psi(P)$.

In the $z1$ -mode, the control points of the top spline σ_1 of $\psi(P)$ is displayed. For the user's convenience, the view is automatically rotated 180° so that the spline σ_0 appears on top. See Figure 18.11.

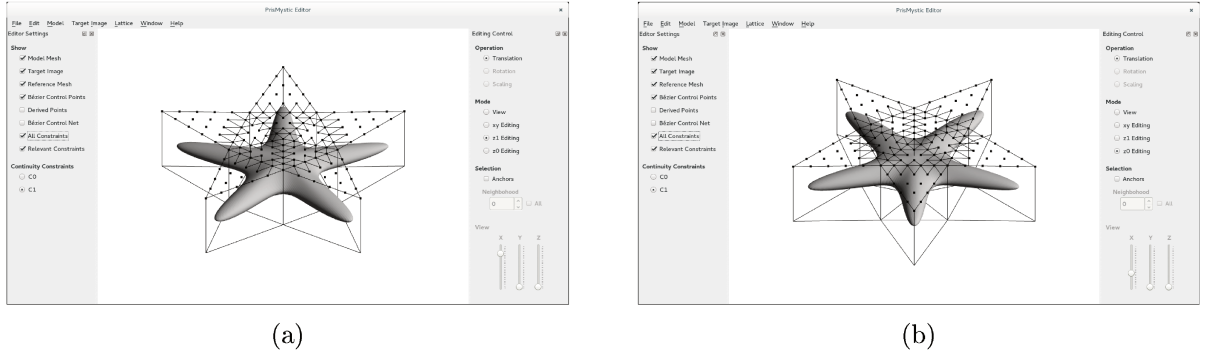


Figure 18.11: The editing modes, (a) $z1$ -mode and (b) $z0$ -mode, of the PrisMystic editor.

In $z0$ -mode or $z1$ -mode the user can modify the z coordinate of the selected anchor points by dragging, with the mouse, any one anchor point up or down along the vertical line that passes through it. The same z displacement is applied to all anchors. The 2DSD algorithm then computes the vertical displacements for the derived points. See Figure 18.12.

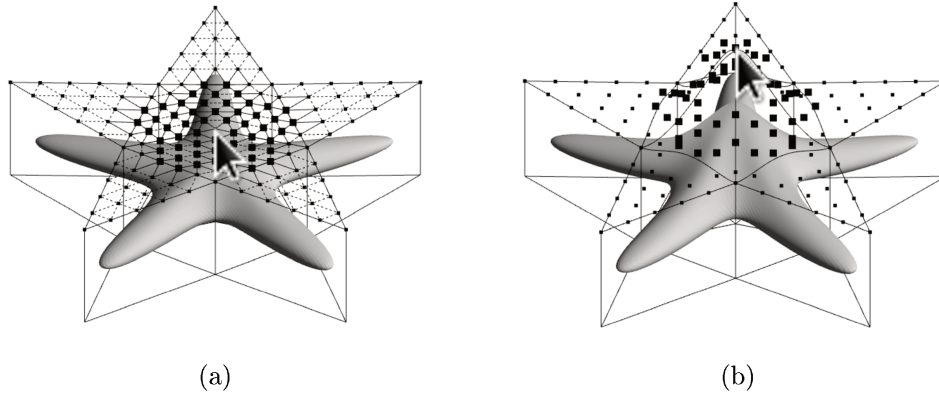


Figure 18.12: (a) Before and (b) after the editing of one anchor point with $\delta_{max} = 3$ in $z1$ -mode, showing the control points and the derived points. The relevant constraints and the global Bézier control net G of the reference mesh are also shown in (a).

By default, the set of anchors $\mathcal{A}$ contain only the point clicked with the mouse, and the set of initial derived points $\mathcal{S}$ is empty. Optionally, the user can select larger sets $\mathcal{A}$ and $\mathcal{S}$ (see Section 18.4).

Chapter 19

Examples

In this chapter, we present images of the obtained results using the PrisMystic editor to deform models. We test the 2.5D space deformation method, described in this part of the thesis, which uses the 2DSD algorithm, described in Part II.

19.1 Deformation of organism models

Our experiments tested the suitability of the developed algorithms, using the PrisMystic editor to reproduce some deformations of organisms which were observed in actual images.

We used examples of three microorganisms in the experiments: the nematode worm *Caenorhabditis elegans*, and the protozoa *Dileptus anser* and *Lacrymaria olor*. We also used the organism starfish *Asterias rubens*. The organism models were created based on its morphology. See left column in Figure 19.1.

The models of the organisms were generated using the Blender editor [83]. The model is a dense triangular mesh consisting of tens of thousands of triangles, showing a typical resting shape of the organism. For each model was appropriately defined a file in the Wavefront format (`.obj`), which contains the coordinates x , y , and z of the vertices of the reference mesh T , and the three vertices of each triangular face. See middle column in Figure 19.1. From the load of this file, the PrisMystic editor automatically generates the reference mesh P adequate for each model. See right column in Figure 19.1.

Table 19.1 summarizes the parameters of the model and the reference meshes used for each organism.

	<i>C. elegans</i>	<i>D. anser</i>	<i>L. olor</i>	<i>Starfish</i>
<i>Number of vertices on M</i>	10425	18967	10425	10242
<i>Number of faces on M</i>	20830	37930	20830	20480
<i>Number of vertices of T</i>	8	9	10	11
<i>Number of faces of T</i>	6	7	8	10
<i>Number of edges of T</i>	13	15	17	20
<i>Number of shared edges of T</i>	5	6	7	10
<i>Number of constraints on $\phi(T)$</i>	25	30	35	50
<i>Number of control points of $\phi(T)$</i>	96	111	126	151

Table 19.1: Parameters of the model mesh M and the reference mesh T for the organisms used in the tests.

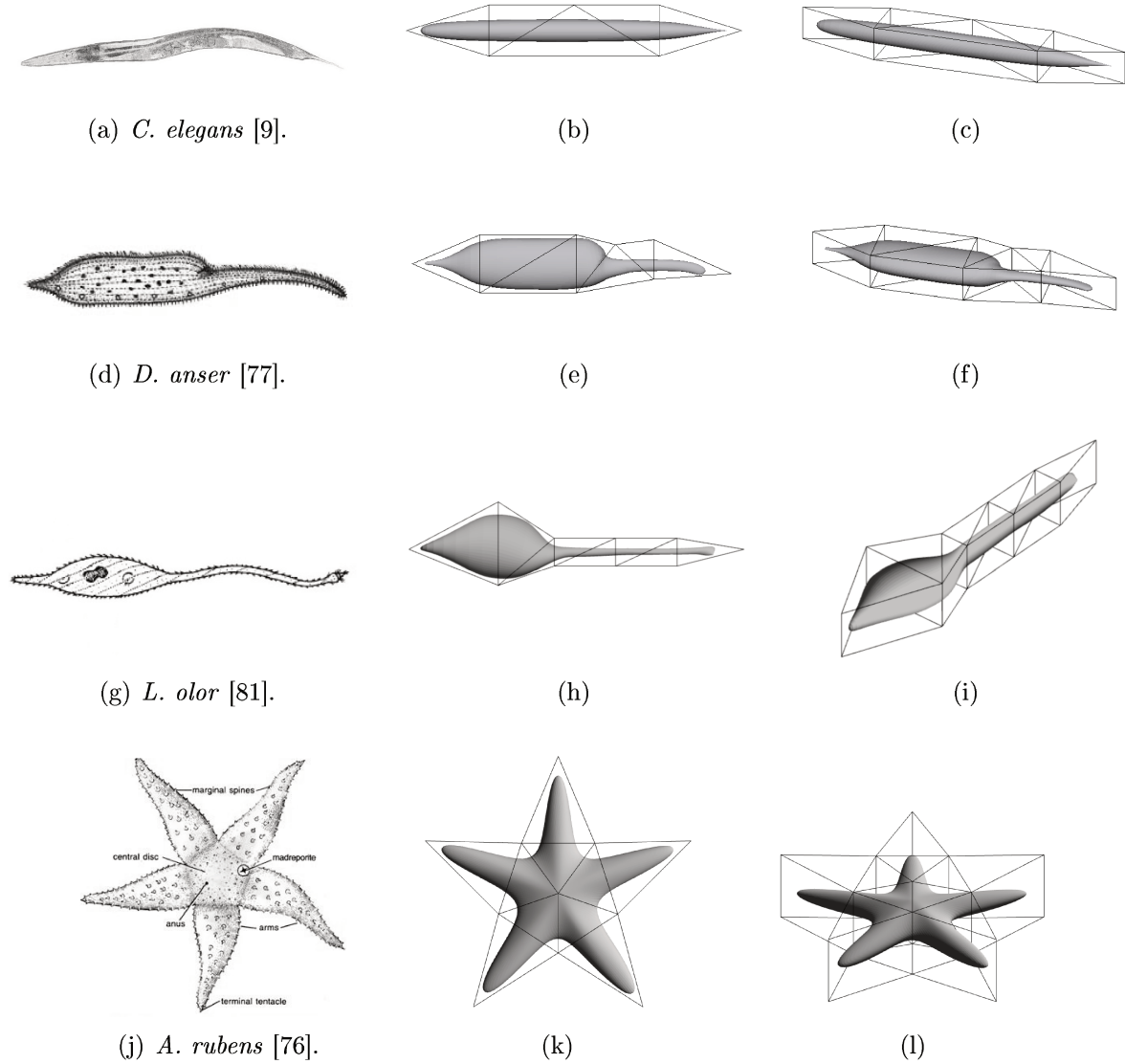


Figure 19.1: Morphology of the organism (left). The 2D view (middle) and 3D view (right) of the reference mesh, and the organism models in a typical resting shape.

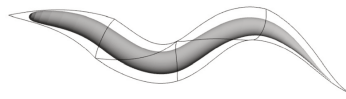
19.1.1 Results

In this section, we present some actual images of the considered organisms, in various poses and deformations. These images were obtained in the Internet.

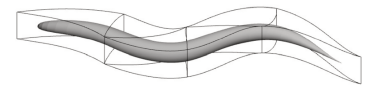
We used the PrisMystic editor to interactively deform the 3D model of each organism in order to match it with the images. The actual images, the 2D and 3D views of the obtained results are shown in Figures 19.2 to 19.5. After acquiring some experience with the editor, editing each example could last no more than 10 minutes [22].



(a) [32].



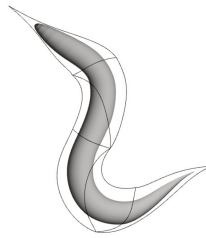
(b)



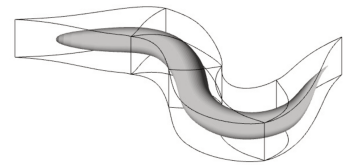
(c)



(d) [46].



(e)



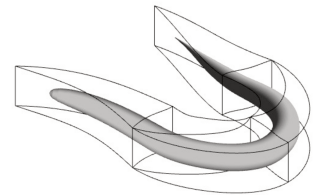
(f)



(g) [59].



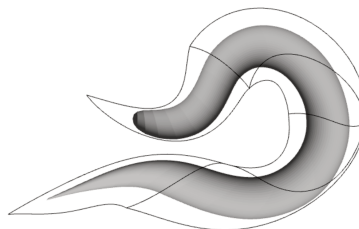
(h)



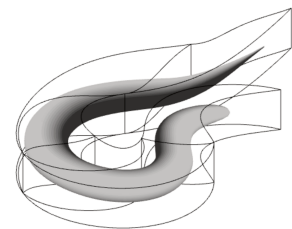
(i)



(j) [7].



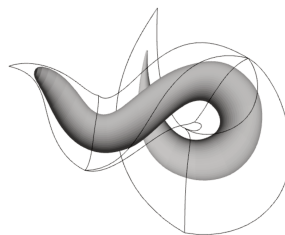
(k)



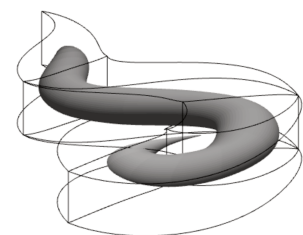
(l)



(m) [35].

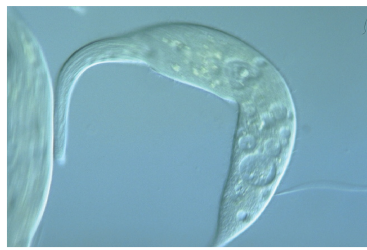


(n)

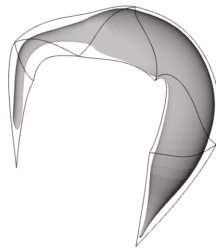


(o)

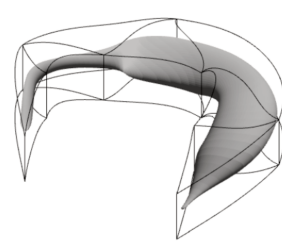
Figure 19.2: Actual microscope images of the nematode *C. elegans* (left); 2D view (middle); and 3D view (right) of the deformed models.



(a) [85].



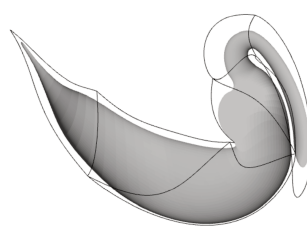
(b)



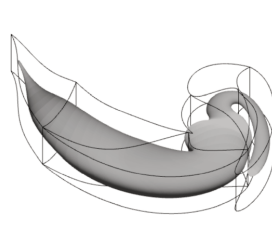
(c)



(d) [85].



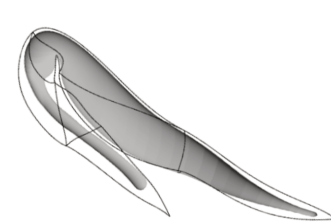
(e)



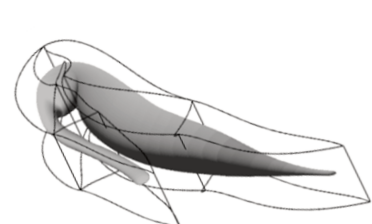
(f)



(g) [85].



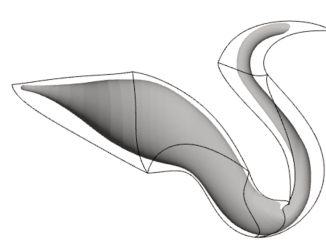
(h)



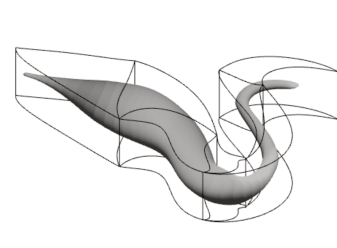
(i)



(j) [28].



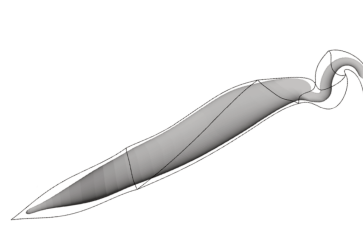
(k)



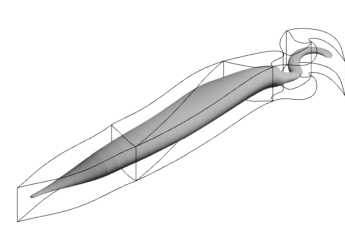
(l)



(m) [85].



(n)

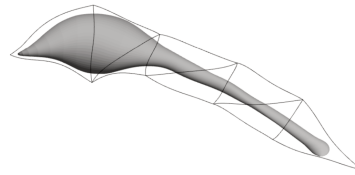


(o)

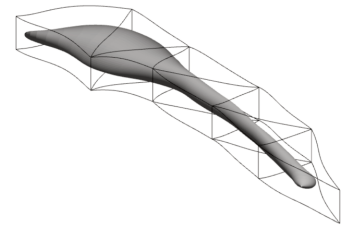
Figure 19.3: Actual microscope images of the protozoan *Dileptus anser* (left); 2D view (middle); and 3D view (right) of the deformed models.



(a) [86].



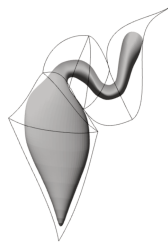
(b)



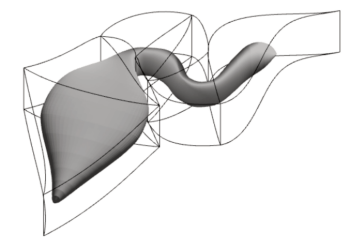
(c)



(d) [86].



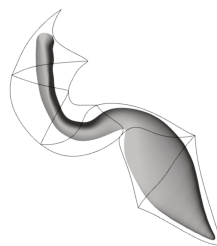
(e)



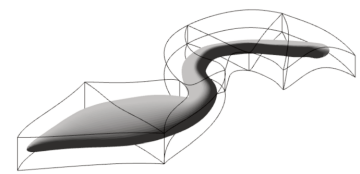
(f)



(g) [86].



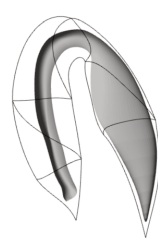
(h)



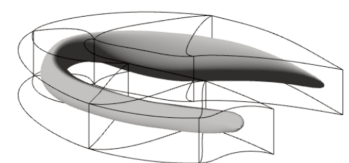
(i)



(j) [86].



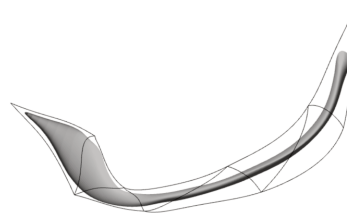
(k)



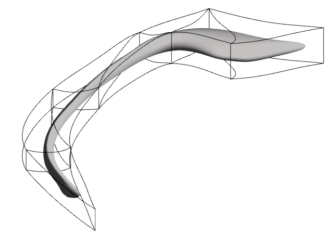
(l)



(m) [86].



(n)

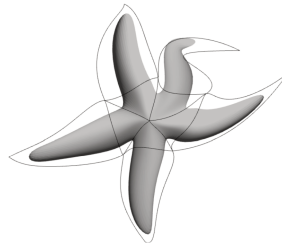


(o)

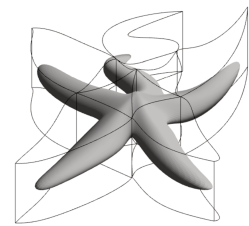
Figure 19.4: Actual microscope images of the *Lacrymaria olor* (left); 2D view (middle); and 3D view (right) of the deformed models.



(a) [14]



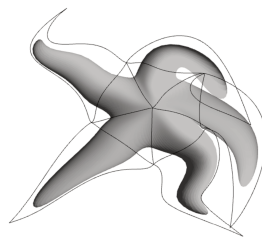
(b)



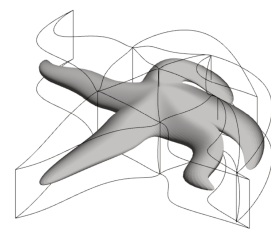
(c)



(d) [90]



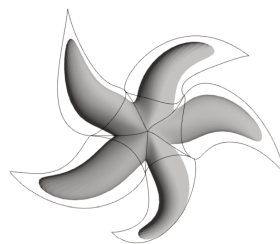
(e)



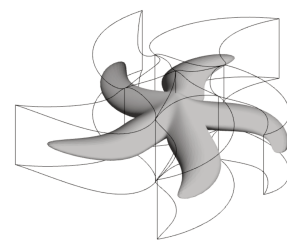
(f)



(g) [75]



(h)



(i)

Figure 19.5: Actual images of the starfish *Asterias rubens* (left); 2D view (middle); and 3D view (right) of the deformed models.

19.2 Deformation of terrain models

The PrisMystic can also be used to create and deform digital terrains. An interesting feature of PrisMystic is that it allows displacing features such as hills and rivers horizontally as well as displacing the surface vertically. It can be useful therefore to edit terrains to be used as displacement maps in 3D rendering or relief patterns in numerically controlled machining and 3D printing.

We apply 2.5D deformations in terrain surfaces to z -deformations provided by our editor. See Figure 19.6. The Blender editor was used to help visualizing the deformed terrain. See Figure 19.7. Table 19.2 summarizes the parameters of the model and the reference meshes used in this test.

	<i>Terrain</i>
<i>Number of vertices on M</i>	1089
<i>Number of faces on M</i>	2048
<i>Number of vertices of T</i>	9
<i>Number of faces of T</i>	8
<i>Number of edges of T</i>	16
<i>Number of shared edges of T</i>	8
<i>Number of constraints on $\phi(T)$</i>	121
<i>Number of control points of $\phi(T)$</i>	40

Table 19.2: Parameters of the model mesh M and the reference mesh T used in the test.

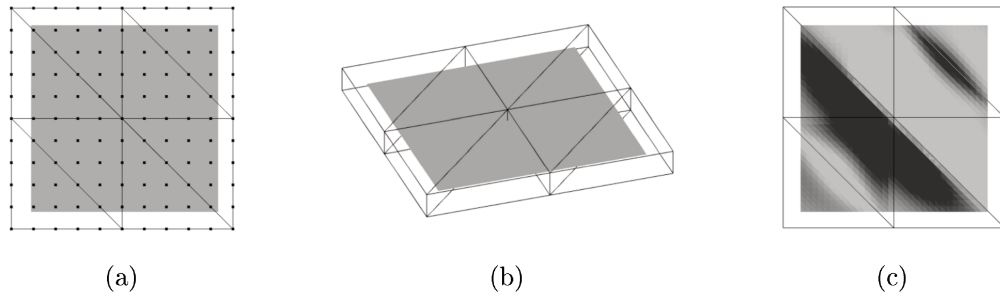


Figure 19.6: (a) 2D view and (b) 3D view of the model of a digital terrain and its reference mesh; and (c) 2D view of the deformed model using the PrisMystic editor.

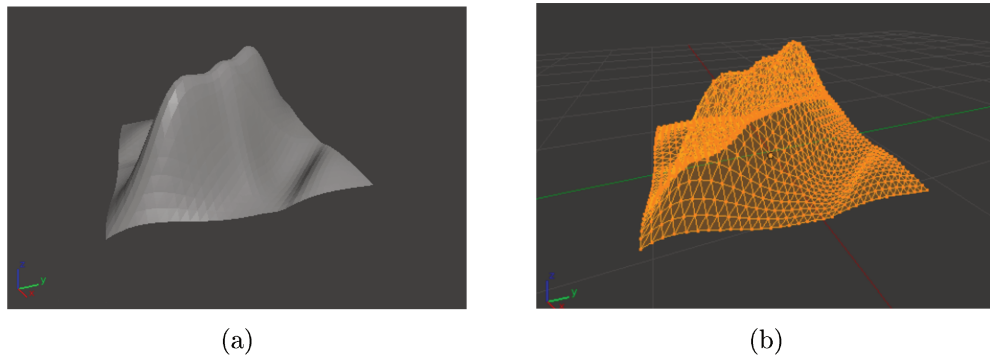


Figure 19.7: 3D view in Blender editor [83] of the deformed model shown in Figure 19.6.

Chapter 20

Conclusion and Future Work

In Part I, we developed a general method for interactive editing of parameters subject to linear or affine constraints. In Part II, we applied this method to the specific problem of creating and editing two-dimensional spline deformations subject to smoothness constraints. In Part III, we described an editor of 2.5D space deformations for three-dimensional solid modeling using the method of Part II.

20.1 Part I

In Part I of this thesis, we described the general ECLES method for interactive editing of parameters subject to linear or affine constraints. We use exact integer arithmetic in order to detect and eliminate redundancies among constraints and avoid rounding failures.

In the ECLES algorithm, the constraints and the user editing actions are combined using weighted constrained least squares, instead of the usual finite element approach, thus providing more flexible control to the user.

One aspect that needs more discussion and tools is the conversion of parameters from floating point to rational values, rounded so as to satisfy the constraints.

20.2 Part II

In Part II of this thesis, we described a general modeling technique for interactive editing of C^1 -continuous two-dimensional deformations using triangular elements with Bézier control nets. The method described, the 2DSD algorithm, supports splines of degree 5 or higher and allows convenient editing of the deformation while preserving the C^1 continuity of the surface.

We use the integer-based ECLES general method, described in Part I, to combine the user editing actions and the continuity constraints in a reliable and efficient way, avoiding the fatal failures that could arise from floating-point rounding errors.

One direction for future work is the consideration of new affine geometric and physical constraints such as C^2 continuity. Another direction for future work is the use of rational rotation matrices for soft rotation. This would allow replacing the strong solvability requirements by weak solvability, so that multiple anchors can be rotated.

20.3 Part III

In Part III of this thesis, we described the PrisMystic editor, an interactive editor for the deformation of 3D models. This editor uses a 2.5D space deformation technique and is an improved version of the editor previously described [67]. One of the main improvements is the use of the algorithm 2DSD, described in Part II, as part of the 2.5D space deformation method.

The PrisMystic editor can also be used to create and deform digital terrains. In this direction, the user interface can be improved to provide a better editing and viewing of the terrain surface.

The PrisMystic editor can be improved in many ways: one easy improvement would be the addition of colors to the model meshes. With this improvement, one could use it for image morphing by converting the original image into a fine mesh of colored triangles all on the xy plane.

A more ambitious project would be to develop an editor with similar interface for 3D deformations, using 3D simplicial splines defined on a mesh of tetrahedra, with C^1 continuity constraints. This editor would use the same general ECLES algorithm, but would require a 3DSD module analogous to 2DSD.

Bibliography

- [1] Alexis Angelidis, Marie-Paule Cani, Geoff Wyvill, and Scott King. Swirling-sweepers: Constant-volume modeling. *Graph. Models*, 68(4):324–332, 2006.
- [2] Alexis Angelidis, Geoff Wyvill, and Marie-Paule Cani. Sweepers: Swept user-defined tools for modeling by deformation. In *SMI '04: Proceedings of the Shape Modeling International 2004*, pages 63–73, Washington, DC, USA, 2004. IEEE Computer Society.
- [3] Fabrice Aubert and Dominique Bechmann. Volume-preserving space deformation. *Comput. Graph.*, 21(5):625–639, 1997.
- [4] Erwin H. Bareiss. Sylvester’s Identity and Multistep Integer-preserving Gaussian Elimination. *Mathematics of Computation*, 22:565–565, 1968.
- [5] Alan H. Barr. Global and local deformations of solid primitives. *SIGGRAPH Comput. Graph.*, 18(3):21–30, January 1984.
- [6] Cagatay Basdogan and Chih-Hao Ho. *Force reflecting deformable objects for virtual environments*. SIGGRAPH’99 Course Notes n. 38, SIGGRAPH-ACM publication, 1999.
- [7] Michael Bastiani. Howard Hughes Medical Institute. Available at: <http://www.hhmi.org/content/jorgensen-abstract-slide-show>. Accessed on Dec 10, 2016.
- [8] D. Bechmann, Y. Bertrand, and S. Thery. Continuous free form deformation. In *COMPU-GRAPHICS '96: Proceedings of the fifth international conference on computational graphics and visualization techniques on Visualization and graphics on the World Wide Web*, pages 1715–1725, New York, NY, USA, 1997. Elsevier Science Inc.
- [9] Mark Blaxter. Mark Blaxter’s teaching pages. Available at: <http://www.nematodes.org/teaching/tutorials/Caenorhabditis/caenorhabditis.shtml>. Accessed on Dec 10, 2016.
- [10] Mario Botsch and Leif Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph.*, 23(3):630–634, August 2004.
- [11] Mario Botsch and Leif Kobbelt. Real-time shape editing using radial basis functions. In *Computer Graphics Forum*, pages 611–621, 2005.
- [12] Mario Botsch, Mark Pauly, Markus Gross, and Leif Kobbelt. PriMo: Coupled Prisms for Intuitive Surface Modeling. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, SGP '06, pages 11–20, Aire-la-Ville, Suíça, Suíça, 2006. Eurographics Association.
- [13] Mario Botsch and Olga Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):213–230, 2008.

- [14] Pam Brophy. Starfish: Caswell Bay. Available at: <http://www.geograph.org.uk/photo/409413>. Accessed on Dec 10, 2016.
- [15] Sabine Coquillart. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 187–196, New York, NY, USA, 1990. ACM.
- [16] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.
- [17] Pieter Coulier and Eric Darve. Efficient Mesh Deformation Based on Radial Basis Function Interpolation by Means of the Inverse Fast Multipole Method. *Computer Methods in Applied Mechanics and Engineering*, 308:286–309, 2016.
- [18] Yuanmin Cui and Jieqing Feng. Technical section: Real-time b-spline free-form deformation via gpu acceleration. *Comput. Graph.*, 37(1-2):1–11, February 2013.
- [19] V. P. Dardengo, F.L. Schmidt, and M. C. Almeida. Identificação de medidas e conjuntos críticos usando matriz de gram e técnicas de fatoração com aritmética inteira. In *Anais do V Simpósio Brasileiro de Sistemas Elétricos (SBSE 2014)*, pages 1–6, Foz do Iguaçu, PR, Brasil, 2014.
- [20] A. de Boer, M.S. van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & Structures*, 85(11–14):784–795, 2007. Fourth MIT Conference on Computational Fluid and Solid Mechanics.
- [21] Carl de Boor. Splines as linear combinations of B-splines. A Survey, 1976.
- [22] Elisa de Cássia Silva Rodrigues. Mathematical and computational methods for modeling and editing deformations. Available at: <http://www.ic.unicamp.br/~erodrigues>. Accessed on Mar 28, 2017.
- [23] Philippe Decaudin. Geometric deformation by merging a 3D-object with a simple shape. In *GI '96: Proceedings of the conference on Graphics interface '96*, pages 55–60, Toronto, Ont., Canada, 1996. Canadian Information Processing Society.
- [24] David Dureisseix. Generalized Fraction-free LU Factorization for Singular Systems with Kernel Extraction. *Linear Algebra and its Applications*, July 2011.
- [25] Gerald Farin. *Curves and surfaces for CAGD: A practical guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2002.
- [26] Jieqing Feng, Lizhuang Ma, and Qunsheng Peng. A new free-form deformation through the control of parametric surfaces. *Computers and Graphics*, 20:531–539, 1996.
- [27] Jieqing Feng, Jin Shao, Xiaogang Jin, Qunsheng Peng, and A. Robin Forrest. Multiresolution free-form deformation with subdivision surface of arbitrary topology. *The Visual Computer*, 22(1):28–42, 2006.
- [28] Donald L. Ferry. Vidcaps from the Lake near Mud Lake. Available at: <http://wolfbat359.com/6crp142.jpg>. Accessed on Dec 10, 2016.
- [29] Michael S. Floater. Mean value coordinates. *Comput. Aided Geom. Des.*, 20:19–27, March 2003.

- [30] Michael S. Floater, Géza Kós, and Martin Reimers. Mean value coordinates in 3d. *Comput. Aided Geom. Des.*, 22:623–631, October 2005.
- [31] James Gain and Dominique Bechmann. A survey of spatial deformation from a user-centered perspective. *ACM Trans. Graph.*, 27(4):1–21, 2008.
- [32] Maria Gallegos. C. elegans Nomarski Image. Available at: <http://wormclassroom.org/image/c-elegans-nomarski-image>. Accessed on Dec 10, 2016.
- [33] Keith O. Geddes, Stephen R. Czapor, and George Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, Norwell, MA, USA, 1992.
- [34] Stefanie Hahmann and Georges-Pierre Bonneau. Polynomial surfaces interpolating arbitrary triangulations. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):99–109, January 2003.
- [35] John Alex Halderman. Worm Patterns. Available at: <http://www.youtube.com/watch?v=7W0xyVvMp8s>. Accessed on Dec 10, 2016.
- [36] W. Hart, F. Johansson, and S. Pancratz. FLINT: Fast Library for Number Theory, 2013. Version 2.4.0, <http://flintlib.org>.
- [37] Ying He, Xianfeng Gu, and Hong Qin. Fairing triangular B-splines of arbitrary topology. In *In Proceedings of Pacific Graphics '05*, pages 153 – 156, 2005.
- [38] Allan Heydon and Greg Nelson. The Juno-2 Constraint-Based Drawing Editor. In *Technical Report 131a, Digital Systems Research*, 1994.
- [39] Gentaro Hirota, Renee Maheshwari, and Ming C. Lin. Fast volume-preserving free form deformation using multi-level optimization. In *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, pages 234–245, New York, NY, USA, 1999. ACM.
- [40] Kai Hormann and Michael S. Floater. Mean value coordinates for arbitrary planar polygons. *ACM Transactions on Graphics*, 25:1424–1441, 2006.
- [41] Jin Huang, Lu Chen, Xinguo Liu, and Hujun Bao. Efficient mesh deformation using tetrahedron control mesh. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 241–247, New York, NY, USA, 2008. ACM.
- [42] Md. Baharul Islam, Md. Tukhrejul Inam, and Balaji Kaliyaperumal. Overview and challenges of different image morphing algorithms. *International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)*, 2(4), 2013.
- [43] D. J. Jeffrey. LU Factoring of Non-invertible Matrices. *ACM Commun. Comput. Algebra*, 44(1/2):1–8, July 2010.
- [44] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Trans. Graph.*, 26, July 2007.
- [45] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24:561–566, July 2005.

- [46] Judith Kimble. Microscopic worm *Caenorhabditis elegans*. Available at: <http://news.wisc.edu/newsphotos/kimble.html>. Accessed on Dec 10, 2016.
- [47] Donald E. Knuth. *Tex and Metafont, New Directions in Typesetting*. American Mathematical Society and Digital Press, Stanford, 1979.
- [48] Kazuya G. Kobayashi and Katsutoshi Ootsubo. t-ffd: Free-form deformation by using triangular mesh. In *Proceedings of the eighth ACM Symposium on Solid Modeling and Applications*, SM '03, pages 226–234, New York, NY, USA, 2003. ACM.
- [49] Nikita Kojekine, Vladimir Savchenko, Mikhail Senin, and Ichiro Hagiwara. Real-time 3D deformations by means of compactly supported radial basis functions. In *Proceedings of Eurographics 2002 (Short Papers)*, pages 35–43, 2002.
- [50] Ming-Jun Lai and Larry L. Schumaker. *Spline Functions On Triangulations*. Cambridge University Press, New York, NY, USA, 2007.
- [51] Henry J. Lamousin and Warren N. Waggenspack Jr. NURBS-based free-form deformations. *IEEE Comput. Graph. Appl.*, 14(6):59–65, 1994.
- [52] Torsten Langer, Alexander Belyaev, and Hans-Peter Seidel. Spherical barycentric coordinates. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, SGP '06, pages 81–88, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.
- [53] Seung-Yong Lee, Kyung-Yong Chwa, and Sung Yong Shin. Image metamorphosis using snakes and free-form deformations. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 439–448, New York, NY, USA, 1995. ACM.
- [54] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Trans. Graph.*, 27:78:1–78:10, August 2008.
- [55] Ignacio Llamas, Alexander Powell, Jarek Rossignac, and Chris D. Shaw. Bender: A virtual ribbon for deforming 3D shapes in biomedical and styling applications. In *Proceedings of the 2005 ACM symposium on Solid and physical modeling*, SPM '05, pages 89–99, New York, NY, USA, 2005. ACM.
- [56] Ron MacCracken and Kenneth I. Joy. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 181–188, New York, NY, USA, 1996. ACM.
- [57] H. Masuda, Y. Yoshioka, and Y. Furukawa. Interactive mesh deformation using equality-constrained least squares. *Comput. Graph.*, 30(6):936–946, December 2006.
- [58] J. Middeke, A. Almohaimeed, and D. J. Jeffrey. Common Factors in Fraction-Free Matrix Reduction. In *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 76–80, Sept 2013.
- [59] Moe. The Immortal Worm of Cain and His Illuminated Race of Worms. Available at: <http://gnosticwarrior.com/worm-of-cain.html>. Accessed on Dec 10, 2016.
- [60] George C. Nakos, Peter R. Turner, and Robert M. Williams. Fraction-free Algorithms for Linear and Polynomial Equations. *SIGSAM Bull.*, 31(3):11–19, September 1997.

- [61] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. *Comput. Graph. Forum*, 25(4):809–836, 2006.
- [62] Greg Nelson. Juno, a Constraint-based Graphics System. *SIGGRAPH Comput. Graph.*, 19(3):235–243, July 1985.
- [63] Tomoyuki Nishita, Toshihisa Fujii, and Eihachiro Nakamae. Metamorphosis using bézier clipping, 1993.
- [64] Carl E. Pearson. *Handbook of applied mathematics: Selected results and methods*. Van Nostrand Reinhold, New York, NY, US, 2th edition, 1990.
- [65] R. Piziak and P.L. Odell. *Matrix Theory: From Generalized Inverses to Jordan Form*. Chapman and Hall/CRC Pure and Applied Mathematics Series. Chapman & Hall/CRC, 2007.
- [66] William H. Press. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Nova Iorque, NY, EUA, 3 edition, set 2007.
- [67] Elisa de Cássia Silva Rodrigues. *Modelagem de Deformação do Espaço 2.5D para Estruturas Biológicas*. MSc em Ciência da Computação, Instituto de Computação, Universidade Estadual de Campinas, 2011.
- [68] Elisa de Cássia Silva Rodrigues, Anamaria Gomide, and Jorge Stolfi. A User-editable C^1 -Continuous 2.5D Space Deformation Method For 3D Models. *Electronic Notes in Theoretical Computer Science*, 281:159 – 173, 2011. Artigo selecionado da XXXVII Conferencia Latinoamericana de Informática (CLEI 2011).
- [69] Elisa de Cássia Silva Rodrigues, Anamaria Gomide, and Jorge Stolfi. A User-editable C^1 -Continuous 2.5D Space Deformation Method For 3D Models. In *CLEI '11: Proceedings of Simposio Latinoamericano sobre Computación Gráfica, Realidad Virtual y Procesamiento de Imágenes*, pages 1–16, 2011.
- [70] Elisa de Cássia Silva Rodrigues, Anamaria Gomide, and Jorge Stolfi. Editing C^1 -Continuous 2D Spline Deformations by Constrained Least Squares. Technical Report IC-14-05, Instituto de Computação, Universidade Estadual de Campinas, Fevereiro 2014.
- [71] Elisa de Cássia Silva Rodrigues and Jorge Stolfi. ECLeS: A Flexible and General Method for Local Editing of Parameters with Linear Constraints. In *SIBGRAPI '15: Proceedings of Workshop of Works in Progress*, pages 1–4, Salvador, BA, Brazil, August 2015.
- [72] Elisa de Cássia Silva Rodrigues and Jorge Stolfi. Interactive Editing of C^1 -Continuous 2D Spline Deformations Using ECLES Method. In *Technical Papers of the 29th Conference on Graphics, Patterns and Images (SIBGRAPI'16)*, São José dos Campos, SP, Brazil, October 2016.
- [73] D. Rueckert, L. I. Sonoda, C. Hayes, D. L G Hill, M. O. Leach, and D.J. Hawkes. Nonrigid registration using free-form deformations: application to breast mr images. *Medical Imaging, IEEE Transactions on*, 18(8):712–721, 1999.
- [74] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph.*, 20(4):151–160, 1986.

- [75] shadowshador. Starfish (*Asterias rubens*). Available at: <https://www.flickr.com/photos/29287337@N02/5526667479/sizes/1>. Accessed on Dec 10, 2016.
- [76] Richa Shah. *Asterias (Starfish): History, Habitat and Development*. Available at: <http://www.biologydiscussion.com/invertebrate-zoology/starfish/asterias-starfish-history-habitat-and-development/27860>. Accessed on Dec 10, 2016.
- [77] Zhang Z Shen YF. Modern biomonitoring techniques using freshwater microbiota. Available at: <http://starcentral.mbl.edu/microscope>. Accessed on Dec 10, 2016.
- [78] Daniele Silva, Iago Berndt, Rafael Torchelsen, and Anderson Maciel. A Levels-of-Precision Approach for Simulating Multiple Physics-based Soft Tissues. In L. A. F. Fernandes D. G. Aliaga, L. S. Davis and W. R. Schwartz, editors, *Technical Papers of the 29th Conference on Graphics, Patterns and Images (SIBGRAPI'16)*, São José dos Campos, SP, Brazil, october 2016.
- [79] Olga Sorkine and Daniel Cohen-Or. Least-squares meshes. In *Proceedings of the Shape Modeling International 2004, SMI '04*, pages 191–199, Washington, DC, USA, 2004. IEEE Computer Society.
- [80] A. Sotiras, C. Davatzikos, and N. Paragios. Deformable medical image registration: A survey. *Medical Imaging, IEEE Transactions on*, 32(7):1153–1190, 2013.
- [81] Heidelberg Spektrum Akademischer Verlag. *Lacrymaria*. Available at: <http://www.spektrum.de/lexikon/biologie/lacrymaria/37873>. Accessed on Dec 10, 2016.
- [82] Stanford. The Stanford 3D Scanning Repository. Available at: <http://graphics.stanford.edu/data/3Dscanrep/>. Accessed on Dec 10, 2016.
- [83] The Blender Foundation. Blender. Available at: <http://www.blender.org>. Accessed on Dec 10, 2016.
- [84] The Qt Company. Qt. Available at: <http://www.qt.io>. Accessed on Mar 28, 2017.
- [85] Y. Tsukii. Protist Images: *Dileptus anser*. Available at: <http://protist.i.hosei.ac.jp/pdb/images/Ciliophora/Dileptus>. Accessed on Dec 10, 2016.
- [86] Y. Tsukii. Protist Images: *Lacrymaria olor*. Available at: <http://protist.i.hosei.ac.jp/pdb/images/Ciliophora/Lacrymaria>. Accessed on Dec 10, 2016.
- [87] P. R. Turner. A Simplified Fraction-free Integer Gauss Elimination Algorithm. Technical report, 1995.
- [88] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125, 2006.
- [89] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Explicit control of vector field based shape deformations. In *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 291–300, Washington, DC, USA, 2007. IEEE Computer Society.
- [90] Jessica Winder. Starfish. Available at: <https://natureinfocus.files.wordpress.com/2010/03/p1020684asnapshot.jpg>. Accessed on Dec 10, 2016.

- [91] George Wolberg. Image morphing: a survey. *The Visual Computer*, 14(8-9):360–372, 1998.
- [92] Tian Xia, Binbin Liao, and Yizhou Yu. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Trans. Graph.*, 28(5):115:1–115:10, December 2009.
- [93] Dianna Xu. *Incremental algorithms for the design of triangular-based spline surfaces*. PhD in computer and information science, Faculties of the University of Pennsylvania, Philadelphia, PA, USA, 2002.
- [94] Wei-Wei Xu and Kun Zhou. Gradient domain mesh deformation: A survey. *J. Comput. Sci. Technol.*, 24(1):6–18, 2009.
- [95] Yong Zhao, Xinguo Liu, Chunxia Xiao, and Qunsheng Peng. A unified shape editing framework based on tetrahedral control mesh. *Comput. Animat. Virtual Worlds*, 20(2-3):301–310, 2009.
- [96] Wenqin Zhou and DavidJ. Jeffrey. Fraction-free Matrix Factors: New Forms for LU and QR Factors. *Frontiers of Computer Science in China*, 2(1):67–80, 2008.
- [97] Barbara Zitová and Jan Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977 – 1000, 2003.

Appendix A

Implementation

In this appendix, we describe the structure of the implemented libraries: LinSys (Linear System), LSQ (Least Squares), ECLES (Editing by Constrained Least Squares), and 2DSD (2D Spline Deformation).

We use the programming language C/C++ to implement the libraries and the Prismo editor. The interface of Prismo was implemented using the graphics libraries Qt GUI and Qt OpenGL of the framework Qt [84]. We also use functions and data structures of the library FLINT (Fast Library for Number Theory) [36] to work with integer and rational arithmetic.

The implementation and instructions for running the Prismo editor, as well as the files of the libraries are available at <http://www.ic.unicamp.br/~erodrigues>.

A.1 The LinSys library

LinSys is a C/C++ library of functions for solving linear systems using integer and rational arithmetic. This library uses the following record type:

```
typedef struct factoring_t {
    int m;
    int n;
    int r;
    int Pr[ ];
    fmpz_mat_t L;
    fmpz_mat_t D;
    fmpz_mat_t U;
    int Pc[ ];
} factoring_t
```

The components of this record are the factors of a matrix A ($m \times n$) of rank r , as defined in Chapter 4. Namely, the row permutation matrix Π_R ($m \times m$), represented as a vector of m indices; the lower triangular factor L ($m \times r$); the diagonal matrix D ($r \times r$); the upper triangular factor U ($r \times n$); and the column permutation matrix Π_C ($n \times n$), represented as a vector of n indices. The L , D , and U fields are FLINT integer matrices (`fmpz_mat_t`). Some computations use FLINT rational matrices (`fmpq_mat_t`).

A.1.1 Fraction-free matrix factoring

The fraction-free LDU factoring is obtained through the `LinSys.LDUFactor` procedure (Algorithm 6), described in Chapter 6. This algorithm is implemented by the `LDUFactor()` procedure of the library `LinSys.h`:

```
factoring_t LDUFactor(int m, int n, fmpz_mat_t A, int method)
```

which receives the $m \times n$ original matrix A to be factored, of the system $Ax = B$; and the simplification method (GCD or Turner) that must be used, as described in Chapter 7. The procedure returns a record of type `factoring_t` with the factoring of the coefficient matrix A .

A.1.2 Solving linear system

The exact linear system solution is obtained through the `LinSys.Solve` procedure (Algorithm 14), described in Chapter 8. This algorithm is implemented by the `solve()` procedure of the library `LinSys.h`:

```
fmpq_mat_t *solve(int t, factoring_t fact, fmpz_mat_t B)
```

which receives the constraints $AP_D'' = B$ in the form of the record `fact` with the factoring of the coefficient matrix A , and the right-hand-side matrix B ($n \times t$). The procedure returns the computed matrix P_D'' ($n \times t$) which is the solution of the linear system.

A.2 The LSQ library

LSQ is a C/C++ library of functions for solving the linear systems using the least squares criterion. The implementation uses the `LSQ.Solve` procedure (Algorithm 15), described in Chapter 9, except for one further optimizations. The weight matrix $M = 2*W$ is omitted since W is an identity matrix in `PrisMystic` editor. This algorithm is implemented by the `solveLSQ()` procedure of the library `LSQ.h`:

```
fmpq_mat_t *solveLSQ(int t, factoring_t fact, fmpz_mat_t B, fmpq_mat_t P1)
```

which receives the constraints $AP_D'' = B$ in the form of the record `fact` with the factoring of the coefficient matrix A , the right-hand-side matrix B ($n \times t$), and the hints matrix P_D' ($n \times t$). The procedure returns the computed matrix P_D'' ($n \times t$) which is the solution of the least squares linear system.

A.3 The ECLES library

ECLES is a C/C++ library of functions for general editing of parameters subject to linear or affine constraints. This library uses the following record type:

```
typedef struct set_t {
    int a;
    int A[ ];
} set_t
```

A.3.1 Initializing

The implementation uses the `ECLES.Initialize` procedure (Algorithm 1), described in Chapter 5. This algorithm is implemented by the `initialize()` procedure of the library `ECLES.h`:

```
bool initialize(int t, set_t A[ ], set_t D[ ], int l, int c, fmpz_mat_t R,
               fmpz_mat_t Q, fmpq_mat_t P, bool strong, int method, factoring_t fact,
               int *F, int *E)
```

which receives the sets $\mathcal{A}$ and $\mathcal{D}$ of indices of the a anchors and of the n derived parameters, respectively; the coefficient matrix R ($l \times c$), and the matrix Q ($m \times t$) of independent terms of all constraints; the matrix of the current values P ($n \times t$); a boolean flag indicating if the strong solvability condition must be checked; and an integer indicating the simplification method used during the factoring. The procedure returns the record `fact` with the factoring of the coefficient matrix $R_{\mathcal{ED}}$; a pointer to the set $\mathcal{F}'$ of indices of the relevant fixed parameters; and the set $\mathcal{E}$ of indices of the non-redundant relevant constraints defined by the procedure.

A.3.2 Updating

The updating was implemented using the `ECLES.Update` procedure (Algorithm 4), described in Chapter 5. This algorithm is implemented by the `update()` procedure of the library `ECLES.h`:

```
bool *update(int t, set_t A[ ], set_t F[ ], set_t E[ ], fmpz_mat_t R,
             fmpz_mat_t Q, factoring_t fact, fmpq_mat_t P1, fmpq_mat_t P2)
```

which receives the sets $\mathcal{A}$, $\mathcal{F}'$, and $\mathcal{E}$; the coefficient matrix R ($l \times c$), and the matrix Q ($m \times t$) of independent terms of all constraints; the record `fact` with the factoring of the coefficient matrix $R_{\mathcal{ED}}$; and the hints matrix $P'_{\mathcal{D}}$ ($n \times t$). The procedure returns the computed matrix $P''_{\mathcal{D}}$ ($n \times t$).

A.4 The 2DSD library

2DSD is a C/C++ library of functions for general editing of two-dimensional spline deformations.

A.4.1 Editing the deformation

The implementation uses the `2DSD.Select` procedure (Algorithm 16), described in Chapter 14. This algorithm is implemented by the `select()` procedure of the library `2DSD.h`:

```
bool select(int t, set_t A[ ], set_t S[ ], fmpz_mat_t R, fmpz_mat_t Q,
            fmpq_mat_t P, factoring_t fact, int *D, int *F, double *T[ ])
```

which receives the sets $\mathcal{A}$ and $\mathcal{S}$; the coefficient matrix R ($l \times c$) of all constraints; the matrix Q ($m \times t$) of independent terms of all constraints; and the matrix P ($n \times t$) of coordinates of the current positions of the control points. The procedure returns the record `fact` with the factoring of the coefficient matrix $R_{\mathcal{ED}}$; a pointer to the sets $\mathcal{D}'$ and $\mathcal{F}'$ of indices of the derived and relevant fixed parameters; and a pointer to the vector T of the values θ of each control point.

A.4.2 Deforming the spline

The implementation of the translation uses the `2DSD.Translate` procedure (Algorithm 17), described in Chapter 14. Rotation and scaling also were implemented using the same idea such as described in Section 14.1.2. These algorithms are implemented by the `translate()`, the `rotate()`, and the `scale()` procedures of the library `2DSD.h`:

```
bool translate(int v[ ], int t, set_t A[ ], set_t D[ ], set_t F[ ],
double T[ ], fmpz_mat_t Q, factoring_t fact, fmpq_mat_t P,
fmpq_mat_t P2)
```

which receives the vector v of the displacement applied to the anchor points; the sets $\mathcal{A}$, $\mathcal{D}$, and $\mathcal{F}$; the vector T of the values θ of each control point; the matrix Q ($m \times t$) of independent terms of all constraints; the record `fact` with the factoring of the coefficient matrix $R_{\mathcal{ED}}$; and the matrix P ($n \times t$) of coordinates of the current positions of the control points. The procedure returns the computed matrix $P''_{\mathcal{D}}$ ($n \times t$);

```
bool rotate(double c[ ], double ang, int t, set_t A[ ], set_t D[ ],
set_t F[ ], double T[ ], fmpz_mat_t Q, factoring_t fact, fmpq_mat_t P,
fmpq_mat_t P2)
```

which receives the center point c and the angle ang of the rotation; the sets $\mathcal{A}$, $\mathcal{D}$, and $\mathcal{F}$; the vector T of the values θ of each control point; the matrix Q ($m \times t$) of independent terms of all constraints; the record `fact` with the factoring of the coefficient matrix $R_{\mathcal{ED}}$; and the matrix P ($n \times t$) of coordinates of the current positions of the control points. The procedure returns the computed matrix $P''_{\mathcal{D}}$ ($n \times t$);

```
bool scale(double c[ ], double sf, int t, set_t A[ ], set_t D[ ],
set_t F[ ], double T[ ], fmpz_mat_t Q, factoring_t fact, fmpq_mat_t P,
fmpq_mat_t P2)
```

which receives the center point c and the scale factor sf of the scaling; the sets $\mathcal{A}$, $\mathcal{D}$, and $\mathcal{F}$; the vector T of the values θ of each control point; the matrix Q ($m \times t$) of independent terms of all constraints; the record `fact` with the factoring of the coefficient matrix $R_{\mathcal{ED}}$; and the matrix P ($n \times t$) of coordinates of the current positions of the control points. The procedure returns the computed matrix $P''_{\mathcal{D}}$ ($n \times t$).