Improvements to Divisor Class Arithmetic on Hyperelliptic Curves

Lindner, Sebastian A

UNIVERSITY OF CALGARY

Improvements to Divisor Class Arithmetic on Hyperelliptic Curves

by

Sebastian Anton Lindner

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

NOVEMBER, 2020

Abstract

The divisor class group of a hyperelliptic curve over a field is a finite abelian group at the center of many open questions in algebraic geometry and number theory. Sutherland [37] surveys some of these, including the computation of the associated L-functions and zeta functions used in his investigation of Sato-Tate distributions [36]. Many of these problems lend themselves to empirical investigation, and as emphasized by Sutherland, fast arithmetic in the divisor class group is crucial for their efficiency. Indeed, implementations of these fundamental operations are at the core of the algebraic geometry packages of widely-used computer algebra systems such as Magma and Sage.

This thesis provides contributions to improve the efficiency of divisor class arithmetic on hyperelliptic curves, with special attention to the often ignored split model cases and to genus 2 and 3. There are two main contributions: the introduction of "Balanced NUCOMP" for improved arithmetic on curves given by split models of arbitrary genus, and improved explicit formulas for genus 2 (ramified and split models) and genus 3 (split models) based on Balanced NUCOMP for split models and NUCOMP for ramified models. Empirical analysis, using a complete Magma implementation and testing suite, is conducted with all contributed algorithms to provide proof of correctness and comparisons to previous best. Our results show that Balanced NUCOMP does offer improvements to split model arithmetic, further narrowing the performance gap between split and ramified models. Our explicit formulas require fewer field operations, most notably significantly fewer additions, than previous best for computing arithmetic on genus 2 (ramified and split models), and genus 3 (split models). Our empirical results demonstrate that our formulas yield the fastest general-purpose arithmetic for these cases.

Preface

This thesis is original, unpublished, independent work by the author, Sebastian, A, Lindner.

Acknowledgments

I would like to thank my supervisor, Dr. Michael J. Jacobson, for his consistent support and guidance throughout all stages of my academic growth and this dissertation. Furthermore I would like to acknowledge and thank Dr. Renate Scheidler, Dr. Laurent Imbert, Randy Yee, and Parthasarathi Das for their support and for meaningful conversations that contributed to my academic growth and the completion of this work. Finally, thank you to my wife for her seemingly endless patience, support and encouragement.

Table of Contents

Abstract . . . . **ii**

Preface . . . . **iii**

Acknowledgments . . . . **iv**

Table of Contents . . . . **iv**

List of Tables . . . . **viii**

List of Algorithms . . . . **ix**

1 Introduction **1**

1.1 Motivation 1

1.2 Summary of Research Contributions 3

1.2.1 Balanced NUCOMP 3

1.2.2 Explicit Formulas for Genus 2 and 3 Arithmetic 5

1.3 Organization of Thesis 7

2 Mathematical Background **9**

2.1 Hyperelliptic Curves 9

2.1.1 Function Field of a Hyperelliptic Curve 12

2.1.2 Points on a Hyperelliptic Curve 13

2.1.3 Orders of Rational Functions 16

2.1.4 Practical Considerations 19

2.2 Divisor Class Group 24

2.3 Affine Divisor Arithmetic 29

2.3.1 Mumford Representation of Affine Divisors 29

2.3.2 Composition and Reduction of Divisors in Mumford Representation 33

2.4 Divisor Class Arithmetic 37

2.4.1 Divisor Class Arithmetic on Ramified Models 38

2.4.2 Divisor Class Arithmetic on Split Models 40

3 Balanced NUCOMP 51

3.1 Improvements to Addition Algorithms 53

3.1.1 General Techniques 53

3.1.2 Improved Divisor Class Arithmetic for Ramified Models 59

3.1.3 Improved Balanced Arithmetic 61

3.2 NUCOMP 69

3.2.1 Continued Fraction Expansions for Divisor Class Arithmetic 69

3.2.2 NUCOMP Algorithm 76

3.3 Balanced NUCOMP 76

3.3.1 Normalization With Negative Reduced Basis 79

3.3.2 Adapting NUCOMP to the Balanced Setting 83

3.3.3 Eliminating an Adjustment for Some Non-Generic Cases 86

3.3.4 Empirical Analysis 86

4 Towards Explicit Formulas 92

4.1 Generic Methods 93

4.1.1 Harley’s Improvements in the Frequent Case 93

4.1.2 Linear Algebra Techniques in the Frequent Case 95

4.1.3 NUCOMP as Basis for Explicit Formulas 98

4.2 Explicit Formulations of Divisor Class Arithmetic 107

4.3 Techniques to Improve Explicit Formulas 108

4.3.1 Field Operation Trade Threshold 109

4.3.2 Efficient Exact Division 109

4.3.3 Karatsuba Multiplication 110

4.3.4 Karatsuba Modular Reduction 111

4.3.5 Using ‘Almost Inverse’ 112

4.3.6 Alternate Approach to Computing s 115

4.3.7 Montgomery's Trick of Simultaneous Inversion . . . 118
4.3.8 Reordering of the Normalization Step . . . 119
4.3.9 Equating Coefficients . . . 119
4.4 Novel Techniques . . . 120
4.4.1 Omitting l' and the Alternate Computation of v'' . . . 120
4.4.2 Alternate computation of k . . . 121

5 Explicit Formulas for Genus 2 Arithmetic . . . 122
5.1 Improved Genus 2 Ramified Model Explicit Formulas . . . 122
5.1.1 Curve Simplifications . . . 123
5.1.2 Prior Work . . . 125
5.1.3 Contributions Summary . . . 126
5.1.4 Explicit Formulas for Doubling . . . 127
5.1.5 Explicit Formulas for Addition . . . 135
5.1.6 Field Operation Costs and Comparisons . . . 143
5.2 Improved Genus 2 Split Model Explicit Formulas . . . 146
5.2.1 Curve Simplifications . . . 147
5.2.2 Prior Work . . . 148
5.2.3 Summary of Contributions . . . 149
5.2.4 Explicit Formulations . . . 152
5.2.5 Precomputations . . . 152
5.2.6 Explicit Formulas for Doubling . . . 153
5.2.7 Explicit Formulas for Addition . . . 161
5.2.8 Field Operation Costs and Comparisons . . . 173
5.3 Empirical Analysis . . . 176
5.3.1 Explicit Special Cases . . . 179
5.3.2 Ramified and Split Model Comparisons . . . 180
5.3.3 Summary . . . 183

6 Explicit Formulas for Genus 3 Balanced Arithmetic 185

6.1 Improved Genus 3 Split Model Explicit Formulas 185

6.1.1 Curve Simplifications 187

6.1.2 Prior Work 187

6.1.3 Summary of Contributions 188

6.1.4 Precomputations 190

6.1.5 Explicit Doubling 192

6.1.6 Explicit Addition 209

6.1.7 Field Operation Costs and Comparisons 238

6.2 Empirical Analysis 242

6.2.1 Explicit Special Cases 244

6.2.2 Split Model Comparisons 244

7 Conclusion 247

7.1 Future Work 248

Bibliography 250

List of Tables

5.1 Frequent case field operation counts of all one inversion operations required for complete divisor class group arithmetic over genus 2 hyperelliptic curve ramified models. 143

5.2 Field operation comparisons for frequent case divisor class group operations on genus 2 hyperelliptic curve ramified models. 144

5.3 Frequent case field operation counts of all one inversion operations required for complete divisor class group arithmetic over genus 2 hyperelliptic curve split models. 174

5.4 Field operation comparisons for frequent case divisor class group operations on genus 2 hyperelliptic curve split models. Operations from [9] that are combinations of two operations requiring two inversions are denoted with (2I). 175

6.1 Frequent case field operation counts of all one inversion doubling operations required for complete divisor class group arithmetic over genus 3 hyperelliptic curve split models. 239

6.2 Frequent case field operation counts of all one inversion addition operations required for complete divisor class group arithmetic over genus 3 split hyperelliptic curve split models. 240

6.3 Field operation comparisons for frequent case divisor class group operations on genus 3 hyperelliptic curve split models. 241

List of Algorithms

1 Compute V^+ . . . 23
2 Cantor's Composition . . . 34
3 Cantor's Reduction Step . . . 36
4 Cantor's Addition . . . 39
5 Balanced Composition . . . 43
6 Balanced Reduction Step . . . 44
7 Balanced Adjust . . . 48
8 Balanced Addition . . . 49
9 Composition Portion of Cantor's Addition . . . 54
10 Composition optimized for frequently-occurring input . . . 55
11 Improved Cantor's Addition . . . 60
12 Improved Cantor's Double . . . 61
13 Convert Adapted to Reduced Basis . . . 63
14 Convert Reduced to Adapted Basis . . . 63
15 Improved Balanced Add . . . 65
16 Improved Balanced Double . . . 66
17 Improved Balanced Adjust (Negative Reduced) . . . 67
18 Improved Balanced Adjust (Positive Reduced) . . . 68
19 NUCOMP . . . 77
20 NUDUPL . . . 78
21 Balanced NUCOMP . . . 80

22 Balanced NUDUPL 81
23 Genus 2 Harley Addition 95
24 Genus 2 Harley Doubling 95
25 Harley's Method Continued for Genus 3 95
26 Genus 2 Ramified Model Double (NUDUPL) 99
27 Genus 2 Ramified Model Addition (NUCOMP) 100
28 Genus 2 Split Model Double (Positive Reduced Balanced NUDUPL) 101
29 Genus 2 Split Model Addition (Positive Reduced Balanced NUCOMP) 102
30 Genus 3 Ramified Model Double 103
31 Genus 3 Ramified Model Addition 104
32 Genus 3 Split Model Double (Negative Reduced Balanced NUDUPL) 105
33 Genus 3 Split Model Addition (Negative Reduced Balanced NUCOMP) 106
34 Genus 2 Ramified Model Degree 2 Doubling 129
35 Genus 2 Ramified Model Degree 2 Doubling ($d = 0$) 133
36 Genus 2 Ramified Model Degree 1 Doubling 134
37 Genus 2 Ramified Model Degree 2 Addition 136
38 Genus 2 Ramified Model Degree 2 Addition ($d = 0$) 139
39 Genus 2 Ramified Model Degree 1 and 2 Addition 141
40 Genus 2 Ramified Model Degree 1 and 2 Addition ($d = 0$) 142
41 Genus 2 Ramified Model Degree 1 Addition 142
42 Genus 2 Split Model Precomputation 153
43 Genus 2 Split Model Degree 2 Doubling 155
44 Genus 2 Split Model Degree 2 Doubling ($d = 0$) 158
45 Genus 2 Split Model Degree 1 Doubling with Down Adjustment 159
46 Genus 2 Split Model Degree 1 Doubling with Up Adjustment 159
47 Genus 2 Split Model Degree 2 Addition 163
48 Genus 2 Split Model Degree 2 Addition $d = 0$ 165
49 Genus 2 Split Model Degree 1 and 2 Addition 167

50 Genus 2 Split Model Degree 1 and 2 Addition with Up Adjust 167
51 Genus 2 Split Model Degree 1 and 2 Addition ($d = 0$) 168
52 Genus 2 Split Model Degree 1 Addition 169
53 Genus 2 Split Model Degree 1 Addition ($d = 0$) 169
54 Genus 2 Split Model Degree 1 Addition with Down Adjust 170
55 Genus 2 Split Model Degree 1 Addition with Up Adjust 171
56 Genus 2 Split Model Degree 0 Add with Down Adjust 172
57 Genus 2 Split Model Degree 0 Add with Up Adjust 172
58 Precomputation 191
59 Genus 3 Split Model Degree 3 Doubling 195
60 Genus 3 Split Model Degree 3 Doubling ($s'_2 = 0$) 200
61 Genus 3 Split Model Degree 3 Doubling ($r_1 = 0$) 201
62 Genus 3 Split Model Degree 3 Doubling ($d = 0$) 202
63 Genus 3 Split Model Degree 2 Doubling 204
64 Genus 3 Split Model Degree 2 Doubling with two Up Adjusts 205
65 Genus 3 Split Model Degree 2 Doubling with two Up Adjusts ($d = 0$) 206
66 Genus 3 Split Model Degree 1 Doubling 207
67 Genus 3 Split Model Degree 1 Doubling with Down Adjust 208
68 Genus 3 Split Model Degree 1 Doubling with Two Up Adjusts 209
69 Genus 3 Split Model Degree 3 Addition 213
70 Genus 3 Split Model Degree 3 Addition $s'_2 = 0$ 217
71 Genus 3 Split Model Degree 3 Addition $r_1 = 0$ 218
72 Genus 3 Split Model Degree 2 and 3 Addition 220
73 Genus 3 Split Model Degree 2 and 3 Addition with Up Adjust 221
74 Genus 3 Split Model Degree 2 Addition 224
75 Genus 3 Split Model Degree 2 Addition with Up Adjust 225
76 Genus 3 Split Model Degree 2 Addition with Two Up Adjusts 226
77 Genus 3 Split Model Degree 1 and 3 Addition 228

78 Genus 3 Split Model Degree 1 and 3 Addition with Up Adjust 229
79 Genus 3 Split Model Degree 1 and 3 Addition with Two Up Adjusts 230
80 Genus 3 Split Model Degree 1 and 2 Addition 232
81 Genus 3 Split Model Degree 1 Addition 233
82 Genus 3 Split Model Degree 1 Addition with Down Adjust 234
83 Genus 3 Split Model Degree 1 Addition with Up Adjust 235
84 Genus 3 Split Model Degree 1 Addition with Two Up Adjusts 236
85 Genus 3 Split Model Degree 0 Add with Down Adjust 237
86 Genus 3 Split Model Degree 0 Add with Up Adjust 237
87 Genus 3 Split Model Degree 0 Add with Two Up Adjusts 238

Chapter 1

Introduction

1.1 Motivation

The divisor class group of a hyperelliptic curve defined over a finite field is a finite abelian group at the center of a number of important open questions in algebraic geometry and number theory. Sutherland [37] surveys some of these, including the computation of the associated L-functions and zeta functions used in his investigation of Sato-Tate distributions [36]. Many of these problems lend themselves to empirical investigation, and as emphasized by Sutherland, fast arithmetic in the divisor class group is crucial for their efficiency. Indeed, implementations of these fundamental operations are at the core of the algebraic geometry packages of widely-used computer algebra systems such as Magma and Sage.

All hyperelliptic curves are represented as models that are categorized as either ramified (imaginary), split (real), or inert according to their number of points at infinity defined over the base field. Ramified curves have one point at infinity, whereas split curves have two. Inert (also called unusual) curves have no infinite points defined over the base field and are usually avoided in practice as they have cumbersome divisor class group arithmetic and can be transformed to a split model over a quadratic extension of the base field.

Divisor class group arithmetic differs on ramified and split models. Many efficient algorithms have been proposed for the ramified setting, notably due to its extensive use in cryptographic applications. The split scenario is more complicated. As a result, optimizing divisor arithmetic on split hyperelliptic curves has received less attention from the research community. However, split models have many interesting properties; most importantly, they exist for a large array of hyperelliptic curves that cannot be described with a ramified model. Thus, exhaustive computations such as those in [36] require working on split models by necessity.

Arithmetic in the divisor class group of a hyperelliptic curve can be described algebraically using an algorithm due to Cantor [3], and expressed in terms of polynomial arithmetic. Various improvements and extensions to Cantor's algorithm have been proposed for ramified model curves, including an adaptation of Shank's NUCOMP algorithm [34] for composing binary quadratic forms [19]. The main idea behind NUCOMP is that instead of composing two divisors directly and then reducing to find an equivalent reduced divisor, a type of reduction is applied part way through the composition, so that when the composition is finished the result is almost always reduced. The effect is that the sizes of the intermediate operands are reduced and intermediate divisor class representations need not be computed, resulting in better performance in most cases. Improvements to NUCOMP have been proposed, most recently the work of [18], where best practices for computing Cantor's algorithm and NUCOMP are empirically investigated.

NUCOMP has also been proposed for arithmetic in the so-called infrastructure of a split model curve [20]. However, as shown by Galbraith et. al. [11, 28], arithmetic on split model hyperelliptic curves is most efficiently realized via a divisor arithmetic framework referred to as balanced. Although the balanced and the infrastructure frameworks are similar, NUCOMP had yet to be applied explicitly to the former.

Several optimized algorithms, known as explicit formulas, are available for divisor class arithmetic in the divisor class group of genus 2 and 3 hyperelliptic curves. These algorithms, which describe divisor class arithmetic in terms of as few field operations as possible, are typically highly specialized for certain families of curves with efficient computational properties. This is too restrictive, as curves used for number theoretic applications are generic and cannot be chosen with computationally efficient properties. These algorithms also only compute the most common divisor input cases under the assumption that the arithmetic is computed over cryptographically large finite fields of size 128-bits [2], and the uncommon cases are instead computed using polynomial arithmetic. Furthermore, field additions are often not even considered. This is not ideal for number theoretic applications, as the finite field sizes are relatively much smaller.

1.2 Summary of Research Contributions

This thesis provides contributions to improve the efficiency of divisor class arithmetic on hyperelliptic curves, with special attention to the often ignored split model cases and to genus 2 and 3. There are two main contributions; the introduction of "Balanced NUCOMP" for improved arithmetic on curves given by split models of arbitrary genus, and improved explicit formulas for genus 2 (ramified and split models) and genus 3 (split models). Empirical analysis is conducted for all contributed algorithms to provide proof of correctness and comparisons to previous best. Details for both main contributions are described next.

1.2.1 Balanced NUCOMP

The first contribution of this work is a generalization of NUCOMP for divisor class arithmetic to the balanced setting of a divisor class group over split model hyperelliptic curves. All best practices from previous works in the ramified model setting are incorporated and new balanced setting-specific improvements that further enhance practical performance are introduced. Specifically, this novel version of NUCOMP includes various improvements over its infrastructure counterpart [20]:

- Describes for the first time exactly how to use NUCOMP in the framework of balanced divisors, including explicit computations of the required balancing coefficients;
- Introduces a novel normalization of divisors in order to eliminate the extra adjustment step required in [20] for typical inputs when the genus of the hyperelliptic curve is odd, so that in all cases typical inputs require no extra reduction nor adjustment steps;
- Uses certain aspects of NUCOMP to compute one adjustment step almost for free in some cases.

In part, the efficiency of Balanced NUCOMP is achieved via a new normalization of the Mumford v polynomial dubbed "negative reduced basis", that eliminates the need for extra adjustment steps over odd genus split model curves introduced in prior work on arithmetic in the infrastructure of split model curves [20]. The use of the negative reduced basis does not affect the efficiency of arithmetic over even genus split model curves, and closes the gap between the two cases discussed in [20]. The new basis takes advantage of the inequality between the number of the two types of adjustments required in divisor class addition, that is inherently part of the definition of a balanced divisor class. Over even genus split model curves, the inequality does not exist, and the change in normalization to v has no effect.

The development of Balanced NUCOMP also involved incorporating and accounting for the balancing coefficient n from the balanced setting over split model curves. A proof of correctness is provided, relating Cantor's algorithm applied to balanced divisor arithmetic to Balanced NUCOMP. In contrast to the NUCOMP algorithm proposed in [20], this work applies NUCOMP techniques to special low input degree divisor classes in order to combine addition and adjustments steps together using NUCOMP techniques.

Finally, implementation of the work in Magma was developed as proof of concept and for empirical timing purposes. Thorough empirical timing analysis comparing the different reduced bases considered, as well as comparing Balanced NUCOMP to previous best, are provided as support for the algorithm design choices made. The analysis demonstrates the efficiency gains realized from the novel version of NUCOMP as compared with the previous best balanced divisor class group arithmetic based on Cantor's algorithm and the arithmetic implemented in Magma, showing that NUCOMP is the method of choice for all but the smallest genera. In such cases overhead introduced by NUCOMP negates the benefits, where the cross-over point is implementation dependent.

With these improvements, NUCOMP is more efficient than Cantor's algorithm for genus as low as 5, compared to 7 using the version in [20]. The implementation developed in this work is faster than Magma's built-in arithmetic for $g \geq 7$, and the gap increases with the genus. Magma is closed source software and therefore an analysis of Magma's implementation was not possible. Magma does document that built-in arithmetic takes advantage of C implementations, and in comparison any implementations written in Magma require passing through Magma's interface, consequently incurring additional computational overhead in comparison.

1.2.2 Explicit Formulas for Genus 2 and 3 Arithmetic

The second contribution is novel explicit formulas for divisor addition and doubling on generic hyperelliptic curves in Weierstrass form for both ramified and split models of genus 2 and split models of genus 3. These formulas include several contributions:

- New streamlined explicit formulas for computing divisor class arithmetic over genus 2 and 3 split model curves using the balanced framework are developed based on the novel Balanced NUCOMP algorithm introduced in this thesis. The split model formulas in this work are the first to completely encapsulate divisor class arithmetic in the balanced framework, whereas previous split model formulas were developed for computing arithmetic in the infrastructure [9, 32].

- All previous literature on computing genus 2 and 3 divisor doubling and addition was surveyed, and through combining the best aspects of previous works and novel techniques introduced here, the number of field operations in almost all cases over genus 2 ramified and genus 2 and 3 split curves were reduced. The reduction in field additions is especially significant. The relative cost of a field addition is non trivial when compared to a field multiplication; for example three additions are considered equivalent in cost to one multiplication in the work of Sutherland [37]. Ramified model genus 3 formulas were omitted as these are work in progress by a student supervised by the author of this work.

- All genus 2 ramified and genus 2 and 3 split model formulas developed are complete in the sense that all possible cases are explicitly computed, and every case requires exactly one inversion. Previous work [9, 32, 37] omitted explicit computation of most non-typical input cases.

First, the general approaches on which explicit formulas are based in previous work were considered and a complete survey of all prior explicit techniques was conducted during the development process. NUCOMP and Balanced NUCOMP were considered as possibilities, although these were not part of any previous work on explicit formulas.

NUCOMP and Balanced NUCOMP based approaches specialized to genus 2 and Balanced NUCOMP specialized to genus 3 for ramified and split model curves produced the simplest formulas requiring fewer operation counts than any prior work. NUCOMP based approaches efficiently describe special cases computations; taking advantage of this resulted in explicit formulas that are complete, and require only one inversion for any computation path. The split model formulas in this thesis are the first to completely encapsulate divisor class arithmetic in the balanced setting, whereas previous split model formulas were developed for computing arithmetic in the infrastructure [9, 32] or considered only the most frequently occurring divisor class input [37] cases.

Implementations of all formulas were developed in Magma as proof of concept. Explicit formulas are very complex and prone to errors. Both black-box and white-box testing programs were utilized to provide thorough evidence that the formulas as they appear in the Magma code are all correct. Black-box testing computes numerous additions of randomly chosen divisor classes over randomly chosen curve models and compares the output of each addition to that obtained using Cantor's Algorithm. White-box testing uses specially-selected divisor class additions that are guaranteed to use every possible computation path within each formula, and similarly compares the output to Cantor's Algorithm. Moreover, empirical timing tests were developed in Magma, including implementations of previous work, to compare the work of this thesis with prior work on the same platform.

Following [37], the formulas use an affine model where each operation requires one field inversion. In the context of computational number theoretic applications like those in [22], affine formulas are superior to the inversion-free formulas obtained using projective coordinates because group order algorithms such as baby-step giant-step require frequent equality tests. As representations of group elements using projective coordinates are not unique, their use would incur a non-negligible computational cost per equality test, and precludes the use of more efficient searchable data structures for the baby steps such as hash tables. Furthermore, as described in [37], the inversions can often be combined using a trick due to Montgomery, allowing a batch of inversions to be computed with only one field inversion and a small number of field multiplications. Thus, in this work, as cryptographic applications are not our motivation, only affine formulas are developed. If projective formulas are required for some other application, our formulas can readily be converted to that setting using standard practices.

Although the applications mentioned above involve hyperelliptic curves defined over finite fields, versions of our formulas that work over any field, as well as versions specialized to fields of characteristic not equal to two and fields of characteristic two are included. For the latter two cases, the formulas take advantage of simplifications that result from canonical forms of the hyperelliptic curve equation in which several terms are zero. Magma implementations are given for all of the formulas in this thesis and are available at https://github.com/salindne/divisorArithmetic.

1.3 Organization of Thesis

In Chapter 2, the mathematical background of hyperelliptic curves limited to the material essential to understanding this thesis is introduced. Basic definitions and properties of hyperelliptic curves, divisors, the divisor class group and the group law arithmetic over the divisor class group are provided. The most efficient divisor class group setting dubbed “balanced setting" is introduced for split model curves, using balanced representations of divisor classes. Most definitions, theorems and algorithms are followed by examples for better understanding.

The purpose of Chapter 3 is the introduction of a novel adaptation of NUCOMP to the balanced setting of the divisor class group over split model curves. A general algorithm called Balanced NUCOMP for divisor class arithmetic over all genus is introduced, with discussion of the development process and algorithm design choices. Empirical analysis producing evidence for the improvements Balanced NUCOMP brings over previous best. The chapter begins with a variety of improvements to generic genus polynomial-based divisor class addition algorithms from previous works, along with background on the NUCOMP algorithm over ramified model curves.

Chapter 4 introduces generic approaches for, basic definition of, and techniques for developing explicit formulas from previous works. A novel approach based on NUCOMP for ramified curves, and Balanced NUCOMP for split curves is also described for genus 2 and 3 curves, where the development of all explicit formulas, including special cases, is unified for each setting respectively. All novel explicit formula techniques introduced in this thesis are provided at the end of the chapter.

Chapters 5 and 6 introduce the novel formulas developed in this thesis for genus 2 and 3 respectively. Chapter 5 is split into major sections, genus 2 ramified model formulas in Section 5.1, and genus 2 split model formulas in Section 5.2. Both sections and Chapter 6 have similar structure where first curve simplifications via isomorphisms are introduced and prior work in each setting is discussed. The basic formulations of doubling and addition algorithms required for complete doubling and addition are then presented, followed by a discussion and comparison of the field operations costs compared to literature. Both Chapters 5 and 6 end with an empirical timing analysis of the formulas compared to previous best computed in Magma using the same platform. The empirical results provide evidence supporting the assertions made in the operation cost comparisons.

A summary of the work done in this thesis and suggestions for potential research for the future are given in Chapter 7. All novel implementations were solely developed by the author of this thesis. Some previous best explicit formulas implemented in Magma were available from the respective authors of previous literature, all others were implemented here, based on the respective papers.

Chapter 2

Mathematical Background

The goal of this chapter is to provide the basic mathematical background required for the work of this thesis. The chapter is organized by starting with the necessary theory on hyperelliptic curves, then divisor class groups, divisor class arithmetic and followed with standard algorithmic improvements for computing the arithmetic. The material in this chapter is primarily taken from [29, 12].

2.1 Hyperelliptic Curves

In this section, theory and properties of hyperelliptic curves relevant to the work are presented. First the definition of hyperelliptic curves is introduced, where the three models of these curves (ramified, split and inert) are defined. Next, necessary definitions and properties of the function field of a hyperelliptic curve and the set of points of hyperelliptic curves (for describing divisor class arithmetic in later sections) are presented. Then, orders of rational functions in the function field of a hyperelliptic curve are defined using the notion of uniformizing parameters. Finally, practical considerations for the generic definitions of a hyperelliptic curve relative to this work are discussed and a crucial polynomial used in split model arithmetic is introduced.

Throughout, suppose that k is a perfect field, $k[x]$ is the univariate polynomial ring over k, $k(x)$ is the field of rational functions over k and $\overline{k}$ is a fixed algebraic closure of k. The leading coefficient of a polynomial t is denoted $\mathrm{lcf}(t)$.

We begin by defining a hyperelliptic curve and the different models that can be used to represent it. In order to simplify the exposition of the practical implementations throughout this thesis, hyperelliptic curves are defined as affine plane curves in the following. The theoretical algebraic geometry approach generally describes hyperelliptic curves as projective plane curves and such descriptions can be found in [12, 4].

Definition 2.1.1. *[29, Definition 12.4.1] A* hyperelliptic equation *over a field k is an equation of the form*

$$C : y^2 + h(x)y = f(x) \tag{2.1.1}$$

with $f(x), h(x) \in k[x]$ and $f(x) \neq 0$.

Remark 2.1.2. *[29, Remark 12.4.2] The curve defined by a hyperelliptic equation (2.1.1) is non-singular if and only if for no point $(x_0, y_0) \in \overline{k} \times \overline{k}$ on the curve that satisfies the hyperelliptic equation (2.1.1), both partial derivatives vanish, i.e.,*

$$2y_0 + h(x_0) = 0 \textit{ and } h'(x_0)y_0 = f'(x_0).$$

Definition 2.1.3. *[29, Definition 12.4.3] A* hyperelliptic curve *C of genus g defined over a field k is represented by a hyperelliptic equation over k that is irreducible in $k(x, y)$, non-singular, and satisfies one of the following three conditions:*

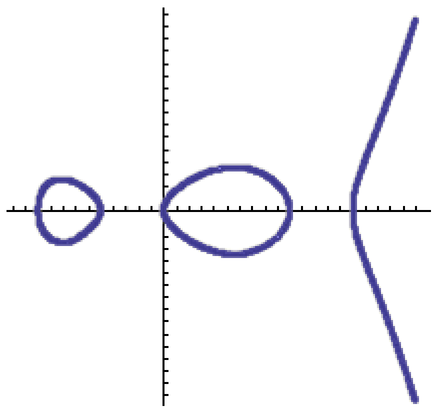
1. $\deg(f(x)) = 2g + 1$ *and* $\deg(h(x)) \leq g$*;*
2. $\deg(f(x)) \leq 2g + 1$ *and $h(x)$ is monic of degree $g + 1$, or* $\deg(f(x)) = 2g + 2$ *and*
   a) *either k has characteristic different from 2,* $\deg(h(x)) \leq g$ *and* $\mathrm{lcf}(f(x))$ *is a square in k,*
   b) *or k has characteristic 2, or $h(x)$ is monic of degree $g + 1$ and* $\mathrm{lcf}(f(x))$ *is of the form $s^2 + s$ for some $s \in k^*$;*
3. $\deg(f(x)) = 2g + 2$ *and*

a) either k has characteristic different from 2, $\deg(h(x)) \leq g$ *and* $\operatorname{lcf}(f(x))$ *is not a square in k,*

b) or k has characteristic 2, or $h(x)$ is monic of degree $g+1$ and $\operatorname{lcf}(f(x))$ *is not of the form $s^2 + s$ for some $s \in k^*$.*

A curve C is ramified *(or imaginary) in case (1),* split *(or real) in case (2), and* inert *(or unusual) in case (3).*

The terminology ramified, split, or inert comes from the behavior of the point at infinity, which will be described later in Section 2.1.2. The alternate terminology imaginary or real comes from analogies to quadratic number fields. The function field of a curve, defined in the next section, has similar properties in terms of its ideal class group and unit group to imaginary quadratic fields if the curve model is ramified and real quadratic fields if it is split. The alternate terminology for inert curve models (unusual) refers to the fact that there is no number field analogue for these models. For a detailed explanation see [29], for example.



Example 2.1.4. *The ramified hyperelliptic curve $C : y^2 = x^5 - 2x^4 - 7x^3 + 8x^2 + 12x$ with genus $g = 2$ over $\mathbb{R}$. The curve C is ramified because the degree of* $\deg(f(x)) = 2g + 1 = 5$ *and* $\deg(h(x)) \leq 2$.

2.1.1 Function Field of a Hyperelliptic Curve

Important notions such as infinite points on a hyperelliptic curve and the ideal class group of a hyperelliptic curve require the machinery presented in this section. In the following, the coordinate ring in which the ideals of a hyperelliptic curve live, and its members called polynomial functions, are defined. Using the coordinate ring, function fields of a hyperelliptic curve and their members, called rational functions, are defined. The section concludes by defining conjugates of polynomials and rational functions, as these are required to describe unique representations of both, and used when evaluating rational functions on points at infinity. For more details, the reader is referred to [27].

Definition 2.1.5. *[27, Definition 8] The* coordinate ring *of C over k, denoted $k[C]$, is the quotient ring*

$$k[C] = k[x, y]/(y^2 + h(x)y - f(x)),$$

where $(y^2+h(x)y-f(x))$ denotes the ideal in $k[x, y]$ generated by the polynomial $y^2+h(x)y-f(x)$. Similarly, the coordinate ring of C over $\overline{k}$ is defined as

$$\overline{k}[C] = \overline{k}[x, y]/(y^2 + h(x)y - f(x)).$$

Elements of $k[C]$ and $\overline{k}[C]$ are called polynomial functions *on C.*

Every polynomial function $G(x, y) \in \overline{k}[C]$ is represented as $G(x, y) = a(x) - b(x)y$, where $a(x), b(x) \in \overline{k}[x]$, are unique. The unique representation is obtained by replacing y^2 by $f(x)-h(x)y$ repeatedly wherever y^2 occurs.

Definition 2.1.6. *[27, Definition 13] The field of fractions of $k[C]$ is called the* function field *of C over k and is denoted by $k(C)$. Similarly, the function field $\overline{k}(C)$ of C over $\overline{k}$ is the field of fractions of $\overline{k}[C]$. Elements of $k(C)$ and $\overline{k}(C)$ are called* rational functions *on C.*

Definitions relating to unique representations of rational functions and their conjugates are presented in the following.

Definition 2.1.7. *[27, Adapted from Definition 10] Let $G(x, y) = a(x) - b(x)y$ be a polynomial function in $\overline{k}[C]$. The conjugate of $G(x, y)$ is defined to be the polynomial function $\overline{G}(x, y) = a(x) + b(x)(h(x) + y)$.*

Lemma 2.1.8. *[27, Adapted from Lemma 12] For a polynomial function $G \in \overline{k}[C]$, we have $G\overline{G} \in \overline{k}[x]$.*

Proof. Let $G = a(x) - b(x)y$, then $\overline{G}(x, y) = a(x) + b(x)(h(x) + y)$ and

$$G\overline{G} = a^2(x) + a(x)b(x)h(x) - b^2(x)(y^2 + yh(x)) = a^2(x) - a(x)b(x)h(x) - b^2(x)f(x) \in \overline{k}[x].$$

□

Lemma 2.1.9. *A rational function $R \in \overline{k}(C)$ can be written as $R = s(x) - t(x)y$ where $s, t \in \overline{k}(x)$.*

Proof. Let $R = G/H$, for $G, H \in \overline{k}[C]$. Let $c(x) = H\overline{H}$ by Lemma 2.1.8. Suppose that $G\overline{H} = a(x) - b(x)y$ with $a(x), b(x) \in \overline{k}[x]$, then $R = s(x) + t(x)y$ with $s(x) = a(x)/c(x) \in \overline{k}(x)$ and $t(x) = b(x)/c(x) \in \overline{k}(x)$. □

Definition 2.1.10. *For a rational function $R = s(x) - t(x)y$ with $s(x), t(x) \in \overline{k}(x)$, the conjugate of R is defined as the rational function $\overline{R} = s(x) + t(x)(h(x) + y)$, and $R, \overline{R} \in \overline{k}(C)$.*

2.1.2 Points on a Hyperelliptic Curve

In Section 2.2 the ideal class group of a hyperelliptic curve C is defined using properties of the set of points on C. In this section finite and infinite points along with their sets are defined, then the hyperelliptic involution and its properties relevant to the work is discussed.

Definition 2.1.11. *[29, Adapted from Definition 12.4.8] Let C be a hyperelliptic curve over the field k. For any extension field L of k, the set of* finite points *on C defined over L is the set of solutions $(x_0, y_0) \in L \times L$ to the hyperelliptic equation (2.1.1).*

Definition 2.1.12. *[27, Definition 14] Let $R \in \overline{k}(C)$, and let P be a finite point on C. Then the rational function R is defined at P if there exist polynomial functions $G, H \in \overline{k}[C]$ such that $R = G/H$ and $H(P) \neq 0$; otherwise, R is not defined at P. If R is defined at P, the value of R at P is defined to be $R(P) = G(P)/H(P)$.*

Definition 2.1.13. *[27, Adapted from Definition 18] Let $R \in \overline{k}(C)^*$ be a rational function, and let P be a point on C. The point P is said to be a* zero *of R if $R(P) = 0$. The function R is said to have a* pole *at P if R is not defined at P.*

Definition 2.1.14. *[29, Adapted from Definition 12.4.7] Let C be a hyperelliptic curve over a field k with hyperelliptic equation as described in Definition 2.1.1. The poles of the rational function $x \in \overline{k}(C)^*$ are called the points at infinity on C.*

Definition 2.1.15. *[12, Adapted from Definition 10.1.16] Points at infinity on the associated hyperelliptic curve lie on the projective plane $P^2(k)$ and are denoted $\infty^+ = (1 : \alpha^+ : 0)$ and $\infty^- = (1 : \alpha^- : 0)$ in weighted projective coordinates, where α^+ and α^- are particular constants in k depending on the hyperelliptic equation used to represent the curve. Conditions on α^+ and α^- for each model go as follows:*

- *$\alpha^+, \alpha^- \in k$ for split models,*
- *$\alpha = \alpha^+ = \alpha^-$, $\alpha \in k$ for ramified models and,*
- *$\alpha^+, \alpha^- \notin k$ but α^+, α^- in a quadratic extension of k for inert models.*

Explicit values for α^+, α^- are given in Definition 2.1.40.

Definition 2.1.16. *[29, Adapted from Definition 12.4.8] Let C be a hyperelliptic curve over the field k. For any extension field L of k, the* set of infinite points *on C defined over L is the set*

$$S = \begin{cases} \{\infty = \infty^+ = \infty^-\} & \text{if } C/L \text{ is ramified,} \\ \{\infty^+, \infty^-\} & \text{if } C/L \text{ is split,} \\ \emptyset & \text{if } C/L \text{ is inert.} \end{cases} \tag{2.1.2}$$

Definition 2.1.17. *[29, Adapted from Definition 12.4.8] The set of finite points defined in Definition 2.1.11, and set of infinite points defined in Definition 2.1.16 together form the set of* points of C defined over L *or the set of* L-rational points of C*, denoted by $C(L)$.*

Example 2.1.18. *Let $C : y^2 + (3x^2 + 1)y = x^5 + 4x + 1$ be hyperelliptic curve over $\mathbb{F}_5$. Then by Definition 2.1.16 C is a ramified and $C(\mathbb{F}_5) = \{(\infty), (0, 2), (1, 3), (4, 3)\}$. All finite points produce an equality within the hyperelliptic equation with $(a, b) \to (x, y)$. For the point at infinity, this can be seen via a projective coordinate representation $(x, y, z) = (1, 0, 0) = \infty$, where the weighted projective hyperelliptic equation is $C := y^2 + (3x^2z + z^3)y = zx^5 + 4xz^5 + z^6$ and finite points are represented as $(x, y, 1)$.*

Example 2.1.19. *Let $C : y^2 = 2x^6 + 2x^4 + 2x^2 + 2$ be a hyperelliptic curve of genus 2 defined over the finite field $\mathbb{F}_3$, then $C(\mathbb{F}_3) = \emptyset$ so C has no points over $\mathbb{F}_3$ and is inert by Definition 2.1.17.*

In the following, properties of hyperelliptic points are discussed, starting with the definition of the hyperelliptic involution, an important morphism on the set of $\overline{k}$-rational points of a hyperelliptic curve.

Definition 2.1.20. *[29, Adapted from Definition 12.4.9] The* hyperelliptic involution *of a hyperelliptic curve C defined over k is the map $\iota : C(\overline{k}) \to C(\overline{k})$ that sends a finite point $P = (x_0, y_0)$ of C to the point $\overline{P} = (x_0, -y_0 - h(x_0))$ of C. If C is ramified, then $\iota(\infty) = \infty$. If C is split, then $\iota(\infty^+) = \infty^-$ and $\iota(\infty^-) = \infty^+$.*

Remark 2.1.21. *Conjugates of polynomial and rational functions defined in Definitions 2.1.7 and 2.1.10 are extensions of the hyperelliptic involution to the set of polynomial and rational functions over C, respectively.*

Definition 2.1.22. *[29, Adapted from Remark 12.4.10] A point P on a hyperelliptic curve C is ramified (or special) if $\iota(P) = P$, i.e, $P = \overline{P}$, and unramified (or ordinary) otherwise.*

Example 2.1.23. *By Definition 2.1.22, the point at infinity ∞ on a ramified model hyperelliptic curve is ramified since $\iota(\infty) = \infty$. The points at infinity ∞^+, ∞^- on a split model hyperelliptic curve are unramified because $\iota(\infty^+) = \infty^-$ and vice versa.*

Example 2.1.24. *By Definition 2.1.22, all points P in Example 2.1.18 have the property that $\iota(P) = P$ and are therefore ramified.*

2.1.3 Orders of Rational Functions

In Section 2.2, the definition of a principal divisor as the divisor defined by a rational function requires the order (or valuation) of that rational function at the $\overline{k}$-rational points. In this section, the order of rational functions at a rational point is defined using the notion of uniformizing parameters. The degree of rational functions used to define the order requires discussion of Puiseux expansions and are presented first.

Definition 2.1.25. *[20, Adapted from Section 2] A (non-zero)* Puiseux series *in an indeterminate t over a field k is an infinite series of the form $\sum_{i=-\infty}^{d} a_i t^i$ where $d \in \mathbb{Z}$, $a_i \in k$ for $i \leq d$, and $a_d \neq 0$. The non-zero Puiseux series in t over k together with 0, form a field denoted by $k\langle t^{-1}\rangle$.*

Remark 2.1.26. *[20, Adapted from Section 2] Consider the hyperelliptic curve $C : y^2 + h(x)y = f(x)$ defined over k.*

- *If C is ramified, then y and $y + h$ are Puiseux series in $x^{-1/2}$ with coefficients in k.*

- *If C is split, then y and $y+h$ are Puiseux series in x^{-1} with coefficients in k.*
- *If C is inert, then y and $y+h$ are Puiseux series in x^{-1} with coefficients in a quadratic extension of k, but not in k.*

Example 2.1.27. *Consider a hyperelliptic curve $C : y^2 = f(x)$ over k. Then $y = \sqrt{f(x)}$ and there are two roots, $\sqrt{f(x)}$ and $-\sqrt{f(x)}$. If a square root y of $f(x)$ is explicitly extracted, then y has a Puiseux expansion over k in $x^{-1/2}$ when C is ramified, and in x^{-1} when C is split.*

The existence of a Puiseux series for y implies that every element of the coordinate ring $k[C]$, and hence the function field $k(C)$, has a Puiseux expansion over k in $x^{-1/2}$ for ramified models, and two in x^{-1} for split models depending on the embedding of $k(C)$ in $k < x^{-1} >$. Fixing such an embedding is discussed in Lemma 2.1.35.

Definition 2.1.28. *[20, Adapted from Section 3] Let $R \in k(C)$. The degree of R is the largest power of x in R considered as a Puiseux series in x^{-1} for both split models and inert models, and in $x^{-1/2}$ for ramified models. The degree of a rational function R, denoted $\deg(R)$, is a half integer for ramified models and an integer for split models.*

Definition 2.1.12 describes what it means for a rational function to be defined at a finite point on a hyperelliptic curve C. The following completes the picture for infinite points on C.

Definition 2.1.29. *[27, Adapted from Definition 17] Let C be a hyperelliptic curve over k and $R \in k(C)$ be a rational function. If C is ramified then*

1. *$R(\infty) = 0$ if $\deg(R) < 0$.*
2. *R is undefined at ∞ if $\deg(R) > 0$.*
3. *$R(\infty)$ is the constant coefficient of R if $\deg(R) = 0$.*

If C is split, then $R(\infty^+)$ is defined analogously to $R(\infty)$ in ramified models and $R(\infty^-) = \overline{R}(\infty^+)$, where $\overline{R}$ is the conjugate of R.

In the rest of this section, orders of rational functions at points and their properties are defined using uniformizing parameters. Intuitively the order of a function $R \in \overline{k}(C)$ at a point P is the measure of the multiplicity of the intersection of the curve C with R. First uniformizing parameters are defined, then the order of a polynomial function at a point and its properties, followed by the order of a rational function at point and explicit unifying parameters for each possible type of point.

Definition 2.1.30. *[27, Adapted from Definitions 23 and 24] Let P be a $\overline{k}$-rational point on the hyperelliptic curve C. Then there exists a function $U \in \overline{k}(C)$ with $U(P) = 0$ such that the following property holds: for each function R in $\overline{k}(C)^*$, there exists an integer d and a rational function $S \in \overline{k}(C)$ such that P is neither a zero nor a pole of S, and $R = U^d S$. Furthermore, the number d does not depend on the choice of U. The function U is called a* uniformizing parameter *for P. The integer d is called the order of R at P and is denoted by $d = \mathrm{ord}_P(R)$.*

Lemma 2.1.31. *[27, Adapted from Lemmas 25 and 28] Let $G_1, G_2 \in \overline{k}[C]^*$ and $P \in C$, and let $\mathrm{ord}_P(G_1) = r_1$, $\mathrm{ord}_P(G_2) = r_1$ then,*

1. $\mathrm{ord}_P(G_1G_2) = \mathrm{ord}_P(G_1) + \mathrm{ord}_P(G_2)$.
2. *Suppose that $G_1 \neq -G_2$. If $r_1 \neq r_2$ then $\mathrm{ord}_P(G_1 + G_2) = \min(r_1, r_2)$. If $r_1 = r_2$ then $\mathrm{ord}_P(G_1 + G_2) \geq \min(r_1, r_2)$.*
3. $\mathrm{ord}_P(G_1) = \mathrm{ord}_{\overline{P}}(\overline{G_1})$.

Theorem 2.1.32. *[27, Theorem 29] Let $G \in \overline{k}[C]^*$. Then G has a finite number of zeros and poles. Moreover,*

$$\sum_{P \in C} \mathrm{ord}_P(G) = 0.$$

Definition 2.1.33. *[27, Definition 30] Let $R = G/H \in \overline{k}(C)^*$ and $P \in C$. The order of R at P is defined to be $\mathrm{ord}_P(R) = \mathrm{ord}_P(G) - \mathrm{ord}_P(H)$.*

Theorem 2.1.34. *[12, Adapted from Theorems 7.7.1 and 7.7.11] Let C be a hyperelliptic curve over k and $R \in \bar{k}(C)^*$ a rational function. Then R has finitely many poles and zeros. Moreover,*

$$\sum_{P \in C} \operatorname{ord}_P(R) = 0.$$

Lemma 2.1.35. *[12, Adapted from Lemmas 10.1.21 and 10.1.22] Let $P = (x_P, y_P) \in C(k)$ be a point on the hyperelliptic curve C over k. Then the explicit uniformizing parameter for*

- *an unramified finite point $P = (x_P, y_P)$ is the function $x - x_P$ and $\operatorname{ord}_P(x - x_P) = 1$,*
- *a ramified finite point $P = (x_P, y_P)$ is the function $y - y_p$ and $\operatorname{ord}_P(x - x_P) = 2$,*
- *an unramified infinite point ∞^+ is the function $1/x$ and for rational function R, $\operatorname{ord}_{\infty^+}(R) = -\deg(R)$ and $\operatorname{ord}_{\infty^-}(R) = -\deg(\overline{R})$,*
- *a ramified infinite point ∞ is the function y/x^{g+1} and for rational function R, $\operatorname{ord}_\infty(-2\deg(R))$.*

2.1.4 Practical Considerations

Not all hyperelliptic curve models defined in Definition 2.1.3 are practical for divisor class arithmetic. Inert models are cumbersome to work with and only exist for small field sizes relative to the genus. Moreover, the hyperelliptic equation for split models as defined in Definition 2.1.3 has more than one form over characteristic 2 fields. These issues are addressed in this section.

In the following theorem, transformations of hyperelliptic curve models are discussed in order to address the problem with inert curves. Arguments for the inefficiency and scarcity of inert hyperelliptic curves are provided, with the conclusion that inert models are impractical and not necessary for most applications. Following, a unified form for the hyperelliptic equation of split hyperelliptic curve that aligns with efficient arithmetic techniques is discussed.

Theorem 2.1.36. *[29, Adapted from Theorem 12.4.12] Let C be a hyperelliptic curve of genus g over a field k as defined in Definition 2.1.1, and let $x_0, y_0 \in k$. Substituting*

$$x = t^{-1} + x_0, \text{ and } y = \frac{bz}{t^{g+1}} + a$$

into (2.1.1), with

$$a = \begin{cases} y_0 & \text{if } char(k) = 2 \\ -h(x_0)/2 & \text{otherwise,} \end{cases} \tag{2.1.3}$$

and

$$b = \begin{cases} 1 & \text{if } h(x_0) + 2y_0 = 0, \\ h(x_0) + 2y_0 & \text{otherwise,} \end{cases} \tag{2.1.4}$$

yields a hyperelliptic curve $C' : z^2 + H(t)z = F(t)$ of genus g over k where

$$H(t) = b^{-1}t^{g+1}(h(t^{-1} + x_0) + 2a), \;\; F(t) = b^{-2}t^{2g+2}(f(t^{-1} + x_0) - ah(t^{-1} + x_0) - a^2),$$

and the following conditions hold:

1. *If $P = (x_0, y_0)$ is a finite ramified point on C, then C' is a ramified hyperelliptic curve.*
2. *If $P = (x_0, y_0)$ is a finite unramified point on C, then C' is a split hyperelliptic curve.*
3. *If no finite point on C has x-coordinate x_0 then C' is an inert hyperelliptic curve.*

Recall that the model used to represent a hyperelliptic curve determines the number and type of points at infinity. A split model representation has two k-rational points at infinity, denoted ∞^+ and $\infty^- = \overline{\infty^+}$. Ramified models have a single ramified k-rational point at infinity, and inert models have none. It is sometimes possible to change the model of a curve C without modifying the field of definition k by translating other points to infinity. If C has a ramified k-rational point, one can obtain a ramified model for C via translation of this point to infinity, by Theorem 2.1.36. If C has a split k-rational point, then similarly, that point can be translated to infinity, providing two points at infinity ∞^+, ∞^- and thus C can be represented with a split model. If no k-rational points exists, including at infinity, then the hyperelliptic curve C is inert and no alternative ramified or split models are possible over k. However, hyperelliptic curves that have neither a ramified nor an unramified k-rational point are rare and only exist over fields whose cardinality is small relative to the genus. If k is a finite field of cardinality q, the Weil bound $\#C(k) \geq q + 1 - 2g\sqrt{q}$ guarantees that a genus g

hyperelliptic curve C over k has a k-rational point whenever $q > 4g^2$, and an unramified k-rational point when $q > 4g^2 + 2g + 2$ [37].

Example 2.1.37. *Let C be a hyperelliptic curve over k with $g = 2$. The Hasse-Weil bound implies that for any field k with cardinality greater than $4(2)^2 = 16$, C has at least one k-rational point, and therefore at worst admits a split model.*

Inert hyperelliptic curves have been shown to have less efficient arithmetic than the other models [33] and can easily be avoided in practice by translating to a split or ramified model when q is sufficiently large to guarantee a k-rational point, or by considering the curve as a split model over a quadratic extension of k otherwise. Thus, in this work we only consider hyperelliptic curves given by a ramified or split model.

Example 2.1.38. *Let $\mathbb{F}_{3^2} = \mathbb{F}_3[t]/(t^2+t+2)$, and let α be a root of the primitive polynomial t^2+t+2 in $\mathbb{F}_3$. The powers of α are:*

n	0	1	2	3	4	5	6	7
α^n	1	α	$2\alpha+1$	$2\alpha+2$	2	2α	$\alpha+2$	$\alpha+1$

Let $C : y^2 = 2x^6 + 2x^4 + 2x^2 + 2$ be defined as in Example 2.1.19, and recall that C is inert over $\mathbb{F}_3$ since it has no $\mathbb{F}_3$-rational points. Consider C over $\mathbb{F}_{3^2}$ instead, then C is split by Definition 2.1.3 because the leading coefficient of $f(x)$ is $2 = (2\alpha + 1)^2$, so it is a square in $\mathbb{F}_{3^2}$. Furthermore,
$C(\mathbb{F}_{3^2}) = \{\infty^+, \infty^-, (0, \alpha^2), (0, \alpha^6), (1, \alpha^2), (1, \alpha^6), (\alpha, 0), (\alpha^2, 0), (\alpha^3, 0),$
$(2, \alpha^2), (2, \alpha^6), (\alpha^5, 0), (\alpha^6, 0), (\alpha^7, 0)\}$.

In split hyperelliptic curve arithmetic, it is practical to only consider one form of the hyperelliptic equation of a curve and design the algorithms based on properties of that equation. Over fields of characteristic other than 2, the f polynomial of the hyperelliptic equation (2.1.1) has degree $2g + 2$. The following theorem shows that any split hyperelliptic curve over characteristic 2 fields can be transformed into a curve with the degree of $\deg(f) = 2g + 2$ as well.

Theorem 2.1.39. *Let $C : y^2 + h(x)y = f(x)$ be a split hyperelliptic curve over a perfect field k of characteristic 2 and cardinality greater than 2 with $\deg(f) < 2g + 2$ and $h(x)$ monic with degree $g + 1$ as defined in (2.1.3). Then C can be transformed to a split hyperelliptic curve $C' : Y^2 + h(x)Y = F(x)$ where $\deg(F(x)) = 2g+2$ and the leading coefficient of $F(x)$ is of the form $e^2 + e$ with $e \in k^*$.*

Proof. Let $e \in k$ where $e^2 + e \neq 0$, so $e \neq 1$. Let $Y = y + ex^{g+1}$. Then,

$$\begin{aligned} y^2 + h(x)y - f(x) &= (Y + ex^{g+1})^2 + (Y + ex^{g+1})h(x) - f(x) \\ &= Y^2 + h(x)y + e^2x^{2g+2} + eh(x)x^{g+1} - f(x) \\ &= Y^2 + h(x)y - F(x), \end{aligned}$$

for $F(x) = e^2x^{2g+2} + eh(x)x^{g+1} - f(x)$. Notice $F(x) \in k[x]$, $\deg(F(x)) = 2g + 2$, and the leading coefficient of $F(x)$ is $e^2 + e$ since $\deg(f(x)) < 2g + 2$, $\deg(h(x)) = g + 1$ and $h(x)$ is monic. Therefore, $C' : Y^2 + h(x)Y = F(x)$ is a split hyperelliptic curve with the desired form. □

Going forward, only split models with $\deg(f(x)) = 2g + 2$ will be considered. Next, a polynomial $V^+(x)$ crucial for arithmetic on hyperelliptic curves with a split model is described. The polynomial $V^+(x)$ is essentially a function that cancels the pole of y at ∞^+ for a hyperelliptic curve $C : y^2 + h(x)y = f(x)$. Its conjugate $V^-(x) = -V^+(x) - h(x)$ similarly cancels the pole of y at ∞^-. Recall from Definition 2.1.15 that $\infty^+ = (1 : \alpha^+ : 0)$ and $\infty^- = (1 : \alpha^- : 0)$. The values α^+ and α^- are explicitly defined in Definition 2.1.40 and the polynomials V^+ and V^- are defined in Definition 2.1.42.

Definition 2.1.40. *[12, Adapted from Excercise 10.1.28] Let $C : y^2 + h(x)y = f(x)$ be a split model hyperelliptic curve over k of genus g. Define $\alpha^+, \alpha^- \in k$ to be the roots of $Y^2 + h_{g+1}Y - f_{2g+2}$ where h_{g+1} and f_{2g+2} are the coefficients of x^{g+1} and x^{2g+2} of $h(x)$ and $f(x)$ respectively.*

Example 2.1.41. *Let C be defined as in Example 2.1.38. The roots of $Y^2 + h_{g+1}Y - f_{2g+2} = Y^2 - 2$ are α^2 and α^6. Then by Definition 2.1.40, the points at infinity defined over F_{3^2} are explicitly* *defined as* $\infty^+ = (1 : \alpha^2 : 0)$ *and* $\infty^- = (1 : \alpha^6 : 0)$ *satisfying the weighted projective model* $C_W : y^2 = 2x^6 + 2x^4z^2 + 2x^2z^4 + 2z^6$ *since* $2 = \alpha^4$*, so both* $(\alpha^2)^2 = \alpha^4 = 2$ *and* $(\alpha^6)^2 = \alpha^{36} = \alpha^4 = 2$*.*

Definition 2.1.42. *[12, Adapted from Excercise 10.1.28] Let* $C : y^2 + h(x)y = f(x)$ *be a split model hyperelliptic curve over* k *of genus* g*. Let* $\alpha^+, \alpha^- \in k$ *as described in Definition 2.1.40, then there exists a polynomial* $V^+(x) = \alpha^+ x^{g+1} + \cdots \in k[x]$ *such that* $\deg(V^+(x)^2 + h(x)V^+(x) - f(x)) \leq g$*. Similarly, there exists a polynomial* $V^-(x) = \alpha^- x^{g+1} + \cdots \in k[x]$ *such that* $\deg(V^-(x)^2 + h(x)V^-(x) - f(x)) \leq g$*. Furthermore,* $V^-(x) = -V^+(x) - h(x)$*.*

The computation of $V^+(x)$ is described in Algorithm 1, where $V^-(x)$ can be computed similarly starting with α^- or as $V^-(x) = -V^+(x) - h(x)$. A brief explanation follows, since the choice of leading coefficient already implies that $\deg(V^+(x)^2 + h(x)V^+(x) - f(x)) \leq 2g + 1$, the idea is to solve linear equations in the coefficients $V^+_g, V^+_{g-1}, V^+_{g-2}, \ldots, V^+_0$ iteratively, with $\deg(V^+(x)^2 + h(x)V^+(x) - f(x))$ decreasing by one in every iteration.

Algorithm 1 Compute V^+

Input: f, h, g.

Output: V^+ polynomial such that $\deg(f - V^+(V^+ + h)) \leq g$ for genus g.

```
 1: Let $h_{g+1}$ be the $g+1$ coefficient of $h$.                    // can be zero
 2: Let $f_{2g+2}$ be the $2g+2$ coefficient of $f$.
 3: Let $V^+_{g+1}$ be the positive solution $\alpha^+$ to the equation $f_{2g+2} - x(x + h_{g+1})$.
 4: $V^+ = V^+_{g+1} x^{g+1}$.
 5: $d = (2V^+_{g+1} + h_{g+1})^{-1}$.
 6: Let $i = g$.
 7: while $i \geq 0$ do
 8:     $V^+_{g+1+i} = g + 1 + i$ coefficient of $f - V^+(V^+ + h)$.
 9:     $V^+ = V^+ + d(V^+_{g+1+i} x^i)$
10:     $i = i - 1$.
11: return $V^+$.
```

2.2 Divisor Class Group

Unlike the case of elliptic curves, the points on a hyperelliptic curve defined over any extension field of k do not form an abelian group. Instead one needs to use divisors, where a divisor is a formal sum of points. The set of all divisors over a hyperelliptic curve forms an abelian group under addition, in which important subgroups can be used to form a quotient group called the divisor class group. The divisor class group of a hyperelliptic curve has many practical uses, and its definition is the main goal of this section. For more information on divisors and the divisor class group, the reader is referred to [12, Section 7.6]. In the following, a series of definitions and properties related to divisors are presented, finishing with a definition of the divisor class group.

Definition 2.2.1. *[12, Adapted from Definition 7.6.1] Let C be a hyperelliptic curve over k. A* divisor *on C is a formal sum*

$$D = \sum_{P \in C(\overline{k})} n_P(P) \tag{2.2.1}$$

where $n_P \in \mathbb{Z}$ and only finitely many $n_P \neq 0$. The integer n_P is called the order of D at P, written as $\mathrm{ord}_P(D) = n_P$.

Definition 2.2.2. *[12, Adapted from Definition 7.6.1] The* support *of the divisor D in equation (2.2.1) is* $\mathrm{supp}(D) = \{P \in C(\overline{k}) : n_P \neq 0\}$.

Example 2.2.3. *Let $C : y^2 = x^6 + 5x^5 + 3x^4 + x + 2$ be a hyperelliptic curve over $\mathbb{F}_7$, then $C(\mathbb{F}_7) = \{\infty^+, \infty^-, (0,4), (0,3), (3,1), (3,6), (4,1), (4,6), (5,1), (5,6), (6,0)\}$. The following are divisors on C:*

- $D_1 = \infty^+ + \infty^-$ *with* $\mathrm{supp}(D_1) = \{\infty^+, \infty^-\}$.
- $D_2 = 2(0,4) - 2(0,3)$ *with* $\mathrm{supp}(D_2) = \{(0,4), (0,3)\}$.
- $D_3 = (5,6) + (3,1) + 5(4,1)$ *with* $\mathrm{supp}(D_3) = \{(5,6), (3,1), (4,1)\}$.
- $D_4 = (4,6) - \infty^+$ *with* $\mathrm{supp}(D_4) = \{(4,6), \infty^+\}$.

- $D_5 = (6,0)$ *with* $\operatorname{supp}(D_5) = \{(6,0)\}$.

Definition 2.2.4. *[12, Adapted from Definition 10.3.1] Let C be a hyperelliptic curve over k. Let S be the set of infinite points on C, i.e., $S = \{\infty\}$ if C is ramified and $S = \{\infty^+, \infty^-\}$ if C is split. A divisor D is an* affine divisor *on C if* $\operatorname{supp}(D) \cap S = \emptyset$.

Example 2.2.5. *Consider the divisors D_i for $1 \le i \le 5$ from Example 2.2.3. By Definition 2.2.4 only the divisors D_2, D_3 and D_5 are affine since there are no infinite points in the supports.*

The order of a divisor D at P is similar to the order of a function $R \in C(\overline{k})$. It is the number of instances a point P appears in the formal sum of the D. The support of a divisor is not a multi-set, only keeping track of the distinct points in a divisor.

Next the group of all divisors on a hyperelliptic curve C over k is defined.

Definition 2.2.6. *[12, Adapted from Definition 7.6.1] Let* $\operatorname{Div}_{\overline{k}}(C)$ *be the set of all divisors on C. The set* $\operatorname{Div}_{\overline{k}}(C)$ *forms an abelian group under the addition rule:*

$$D + D' = \sum_{P \in C(\overline{k})} n_P(P) + \sum_{P \in C(\overline{k})} n'_P(P) = \sum_{P \in C(\overline{k})} (n_P + n'_P)(P).$$

The following are defined for divisors in $\operatorname{Div}_{\overline{k}}(C)$

- *the negative of D is $-D = \sum_P (-n_P)(P)$,*
- *the zero divisor 0 is the divisor with $n_P = 0$ for all $P \in C(\overline{k})$,*
- *$D \ge D'$ if $n_P \ge n'_P$ for all $P \in \overline{k}$, and*
- *$D \ge 0$ if $n_P \ge 0$ for all P (such a divisor is called* effective.*)*

Example 2.2.7. *The divisor with $n_P = 0$ for all $P \in C(\overline{k})$, written as $D = 0$, is affine by Definition 2.2.4 and has support* $\operatorname{supp}(D) = \emptyset$ *by Definition 2.2.2 and is an effective divisor by Definition 2.2.6.*

Example 2.2.8. *Consider the divisors* $D_1 = \infty^+ + \infty^-$ *and* $D_2 = 2(0,4) - 2(0,3)$ *from Example 2.2.3.* D_1 *has all* $n_p \geq 0$ *and therefore is an effective divisor (but not affine), and* D_2 *has* $n_p < 0$ *for the point* $(0,3)$*, and is not an effective divisor.*

In the following, an important subgroup of $\mathrm{Div}_k(C)$ for defining the divisor class group is presented that requires the notion of the degree of a divisor, defined in Definition 2.2.9.

Definition 2.2.9. *[12, Adapted from Definition 7.6.3] The* degree *of a divisor D as defined by (2.2.1) is the integer*

$$\deg(D) = \sum_{P \in C(\overline{k})} n_P.$$

Lemma 2.2.10. *[12, Adapted from Lemma 7.6.4] The set* $\mathrm{Div}^0_{\overline{k}}(C) = \{D \in \mathrm{Div}_{\overline{k}}(C) : \deg(D) = 0\}$ *of all degree zero divisors is a subgroup of* $\mathrm{Div}_{\overline{k}}(C)$.

Example 2.2.11. *Consider the divisors* D_i *for* $1 \leq i \leq 5$ *from Example 2.2.3, by Definition 2.2.9,* $\deg(D_1) = 2$*,* $\deg(D_2) = 0$*,* $\deg(D_3) = 7$*,* $\deg(D_4) = 0$*,* $\deg(D_5) = 1$*. Both* D_2 *and* D_4 *are in* $\mathrm{Div}^0_{\overline{k}}(C)$ *by Lemma 2.2.10.*

Example 2.2.12. *The divisor* 0 *from Example 2.2.7, has* $\deg(0) = 0$ *and* $0 \in \mathrm{Div}^0_{\overline{k}}(C)$*. Moreover,* 0 *is the neutral element under addition of both groups* $\mathrm{Div}^0_{\overline{k}}(C)$ *and* $\mathrm{Div}_{\overline{k}}(C)$*.*

Next, the notion of a divisor of a function R in the function field of a hyperelliptic curve $\overline{k}(C)^*$, and the associated subgroup are presented in Definition 2.2.13 and Lemma 2.2.15.

Definition 2.2.13. *[12, Adapted from Definition 7.7.2] Let* $R \in \overline{k}(C)^*$ *and define the divisor of a function*

$$\mathrm{div}(R) = \sum_{P \in C(\overline{k})} \mathrm{ord}_P(R) P.$$

By Theorem 2.1.34, $\mathrm{div}(R)$ *is a divisor, i.e. a formal finite sum, and* $\deg(\mathrm{div}(R)) = 0$*. Moreover, if* $R, S \in \overline{k}(C)^*$ *are two non-zero rational functions, then*

$$\mathrm{div}(RS) = \mathrm{div}(R) + \mathrm{div}(S).$$

The divisor of a function is also called a principal divisor.

Example 2.2.14. *Consider the divisor* $D_4 = (4,6) - \infty^+$ *from Example 2.2.3. Let* $R = x - 4$*, then by Lemma 2.1.35* $\mathrm{ord}_{(4,6)}(R) = 1$ *because the point* $(4,6)$ *is unramified where* $\iota(4,6) = (4,1)$ *and* $\mathrm{ord}_{\infty^+}(R) = -1$*. So* $D_4 = (4,6) - \infty^+ = \mathrm{div}(R)$*. Since* $\mathrm{ord}_P(R) = 0$ *for all other points* P *on* C*,* D_4 *is a principal divisor by Definition 2.2.13.*

Lemma 2.2.15. *[12, Adapted from Lemma 7.7.6] With notation as above, let*

$$\mathrm{Prin}_{\overline{k}}(C) = \{\mathrm{div}(R) : R \in \overline{k}(C)^*\},$$

then $\mathrm{Prin}_{\overline{k}}(C)$ *is a subgroup of* $\mathrm{Div}_{\overline{k}}(C)$*. Furthermore, by Theorem 2.1.34,* $\mathrm{Prin}_{\overline{k}}(C)$ *is a subgroup of* $\mathrm{Div}^0_{\overline{k}}(C)$*.*

Example 2.2.16. *Let* $R = c$*, for* $c \in k$*, be a constant function in* $\overline{k}(C)^*$*. The divisor* $\mathrm{div}(R) = 0$ *is a principal divisor and* $\mathrm{div}(R) = 0 \in \mathrm{Prin}_{\overline{k}}(C)$*. Furthermore,* 0 *is the neutral element of the group* $\mathrm{Prin}_{\overline{k}}(C)$ *under addition.*

The divisor class group relies on sets of divisors defined over k rather than its closure. What it means for a divisor to be defined over k is described in Definition 2.2.17.

Definition 2.2.17. *[12, Adapted from Definition 7.6.6] Let* C *be a curve over* k *and let* $D = \sum_{P \in C(\overline{k})} n_P(P)$ *be a divisor on* C*. For* $\sigma \in Gal(\overline{k}/k)$ *define* $\sigma(D) = \sum_{P \in C(\overline{k})} n_P \sigma(P)$*. Then* D *is* defined over k *if* $\sigma(D) = D$ *for all* $\sigma \in Gal(\overline{k}/k)$*. Write* $\mathrm{Div}_k(C)$ *for the set of divisors on* C *that are defined over* k*.*

Example 2.2.18. *Let $C : y^2 + xy = x^5 + 2x + 1$ be a ramified hyperelliptic curve defined over $\mathbb{F}_3$. The polynomial $x^2 + 2x + 1$ is irreducible over $\mathbb{F}_3$, and has roots $< 1 + 2\sqrt{2} >, < 1 + \sqrt{2} >$, that are elements of $\mathbb{F}_3(\sqrt{2})$. These roots provide 4 points in $C(\mathbb{F}_3(\sqrt{2}))$, given as*

$$P_1 = (1 + 2\sqrt{2}, 1 + \sqrt{2}), \quad P_2 = (1 + 2\sqrt{2}, 1), \quad P_3 = (1 + \sqrt{2}, 1), \text{ and } P_4 = (1 + \sqrt{2}, 1 + 2\sqrt{2}).$$

The points P_1 and P_2, as well as P_3 and P_4, are opposites under the hyperelliptic involution. Let $D = P_1 + P_4$. The only automorphism in $\mathrm{Gal}(\mathbb{F}_3(\sqrt{2})/\mathbb{F}_3)$ that acts non-trivially on D is σ defined by $\sigma(\sqrt{2}) = 2\sqrt{2}$. Observe that

$$\sigma(P_1) = P_4 \text{ and } \sigma(P_4) = P_1.$$

Therefore, as described in Definition 2.2.17, $\sigma(D) = D$ and D is defined over F_3.

Definition 2.2.19. *All groups defined above can be restricted to divisors defined over k. For a hyperelliptic curve C defined over k*

- *The set of all divisors defined over k on C is $\mathrm{Div}_k(C)$.*
- *The set of all degree zero divisors defined over k on C, $\mathrm{Div}_k^0(C)$ is a subgroup of $\mathrm{Div}_k(C)$.*
- *The set of all principal divisors defined over k on C, $\mathrm{Prin}_k(C) = \{\mathrm{div}(R) : R \in k(C)^*\}$, is a subgroup of $\mathrm{Div}_k^0(C)$.*

Since all the divisor groups considered in Definition 2.2.19 are Abelian, one can define the quotient group of $\mathrm{Div}_k^0(C)$ over $\mathrm{Prin}_k(C)$ called the divisor class group. The notation Pic is used for the divisor class group since the divisor class group of a hyperelliptic curve is isomorphic to its Picard group (even though the Picard group is usually defined geometrically in terms of line bundles).

Definition 2.2.20. *[12, Adapted from Definition 7.8.1] The (degree zero)* divisor class group *of a curve C over k is*

$$\mathrm{Pic}_k^0(C) = \mathrm{Div}_k^0(C)/\mathrm{Prin}_k(C).$$

Two divisors $D_1, D_2 \in \mathrm{Div}_k^0(C)$ are called linearly equivalent *and written as $D_1 \equiv D_2$ if $D_1 - D_2 \in \mathrm{Prin}_k(C)$. The equivalence class (called a* divisor class*) of a divisor $D \in \mathrm{Div}_k^0(C)$ under linear equivalence is denoted $[D]$.*

2.3 Affine Divisor Arithmetic

In Section 2.4, representations and arithmetic of divisor classes in $\mathrm{Pic}_k^0(C)$ are described over ramified and split models. The representations are described as the sum of affine divisors with points at infinity. Divisor class arithmetic depends almost entirely on the arithmetic of the affine part of the divisor in the representation. The goal of this section is to describe the basic arithmetic of affine divisors using a representation of affine divisors via two polynomials, followed by an exposition of Cantor's algorithms for computing affine divisor composition and reduction.

2.3.1 Mumford Representation of Affine Divisors

In this section, a representation of affine divisors that is useful for efficiently performing affine divisor arithmetic is presented. There exists a convenient way to represent affine divisors using two polynomials called Mumford representation, but is only possible if the divisor's formal sum has a compact form called semi-reduced. Semi-reduced divisors are defined in Definition 2.3.1.

Definition 2.3.1. *[12, Adapted from Definition 10.3.1] Let C be a hyperelliptic curve over k. A divisor D on C is* semi-reduced *if it is an effective affine divisor and for all $P \in \mathrm{supp}(D)$ the following properties hold:*

1. *If P is a ramified point, ($P = \iota(P)$), then $n_P = 1$.*
2. *If P is an unramified point, ($P \neq \iota(P)$), then $n_p > 0$ implies $n_{\iota(P)} = 0$.*

Definition 2.3.1 states that the multiplicity of a ramified point in a semi-reduced divisor is always one, and that if an unramified point with any multiplicity is in the support of a semi-reduced divisor, then its opposite under the hyperelliptic involution is not. Removing these points yields a linearly equivalent divisor. Lemma 2.3.2 states that every affine divisor on C is linearly equivalent to a semi-reduced divisor.

Lemma 2.3.2. *[12, Adapted from Lemma 10.3.3] Let C be a hyperelliptic curve. Every affine divisor on C is equivalent to a semi-reduced divisor. Equivalence in this case is as defined in Definition 2.2.20, where for two affine divisors D, D', $D \equiv D'$ whenever a principal divisor* $\mathrm{div}(f)$ *exists such that $D = D' + \mathrm{div}_a(f)$, where $\mathrm{div}_a(f)$ is the affine portion of* $\mathrm{div}(f)$*, i.e., only the affine points in the formal sum representation of* $\mathrm{div}(f)$.

Example 2.3.3. *Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be unramified points on a hyperelliptic curve C such that $x_1 \neq x_2$. Let $D = 2P_1 - \iota(P_1) + P_2 + \iota(P_2)$, then D is not semi-reduced. Note that* $\mathrm{div}(x - x_i) = P_i + \iota(P_i)$*. One has*

$$D + \mathrm{div}(x - x_1) = 2P_1 - \iota(P_1) + P_2 + \iota(P_2) + P_1 + \iota(P_1) = 3P_1 + P_2 + \iota(P_2),$$

which is effective, but still not semi-reduced. Subtracting $\mathrm{div}(x - x_2)$ *from the above, equivalent to adding the conjugate of $x - x_2$, gives*

$$D - \mathrm{div}(x - x_1) = 3P_1 + P_2 + \iota(P_2) - P_2 - \iota(P_2) = 3P_1,$$

which is semi-reduced.

Mumford [30] introduced a representation for semi-reduced divisors that uses two polynomials to represent the points on a divisor, where the first polynomial keeps track of the x-coordinates of the points, and the second polynomial interpolates the points. This representation contains all of the information required to describe the points of a divisor on a hyperelliptic curve. The semi-reduced property is critical in this representation, as if both a point and its hyperelliptic involution appear in the support of a divisor, then no polynomial can interpolate both points.

Definition 2.3.4. *[12, Adapted from Lemma 10.3.5 and Definition 10.3.6] Let $D = \sum_{i=1}^{l} n_i(x_i, y_i)$ be a semi-reduced divisor on a hyperelliptic curve $C : y^2 + h(x)y = f(x)$ defined over k (hence D*

is affine and effective). Define

$$u(x) = \prod_{i=1}^{l} (x - x_i)^{n_i} \in k[x],$$

where $u(x)$ is monic. Then there is a unique polynomial $v(x) \in k[x]$ such that $\deg(v(x)) < \deg(u(x))$*, $v(x_i) = y_i$ for all $1 \le i \le l$, and*

$$v(x)^2 + h(x)v(x) - f(x) \equiv 0 \pmod{u(x)}. \tag{2.3.1}$$

In particular,$v(x) = 0$ if and only if $u(x)|f(x)$.

The polynomials $\langle u, v\rangle$ are the Mumford representation *of D. Moreover, if $D = 0$ then $u = 1$ and $v = 0$.*

Example 2.3.5. *Let $C : y^2 + xy = x^5 + 2x + 1$ be a ramified hyperelliptic curve over $\mathbb{F}_3$. Then* $\mathrm{Pic}_k^0(C)$ *is given below where $[D]$ with $D = \langle x^2 + 2x + 2, 1\rangle$ is a generator of* $\mathrm{Pic}_k^0(C)$. $\mathrm{Pic}_k^0(C)$ *has 10 elements with the representation of each nD for $1 \le n \le 10$.*

nD	D	$2D$	$3D$	$4D$	$5D$	$6D$	$7D$	$8D$	$9D$	$10D$
$u(x)$	x^2+2x+2	x^2	x^2+x+2	x	x^2+1	x	x^2+x+2	x^2	x^2+2x+2	1
$v(x)$	1	$2x+1$	1	2	x	1	$2x+2$	2	$2x+2$	0

Definition 2.3.4 states that if a divisor is semi-reduced, then it has a Mumford representation. The converse is also true, one can recover an affine divisor D from its Mumford representation $\langle u, v\rangle$ as described in Lemma 2.3.6.

Lemma 2.3.6. *Let $u(x)$ and $v(x)$ be polynomials over $k[x]$ having the properties in Definition 2.3.4, and recall that $u(x) = \prod_{i=1}^{l}(x - x_i)^{n_i}$. Then*

$$D = \sum_{i=1}^{l} n_i(x_i, v(x_i)),$$

is a divisor on C defined over k.

Although the coefficients of the polynomials in Mumford representation are in k, the corresponding points in the summation of the divisor being represented need not be k-rational.

Remark 2.3.7. *Let C be a hyperelliptic curve over k. Let $u(x), v(x) \in k[x]$ be the Mumford representation of a semi-reduced divisor $D = \sum_{i=1}^{l} n_i(x_i, y_i)$ on C. Let $u(x) = \prod u_i(x)$ be the irreducible factorization of $u(x)$ in $k[x]$. If $\deg(u_i) \geq 2$ for any i, then the points in D corresponding to the u_i as described in Lemma 2.3.6 are not k-rational but are rational over a $\deg(u_i)$ extension of k.*

Example 2.3.8. *Let $C : y^2 + xy = x^5 + 2x + 1$ be a ramified hyperelliptic curve over $\mathbb{F}_3$. Consider the Mumford representations from Example 2.3.5. The table below shows the relationship between Mumford and formal sum representations of divisors where the formal sums consist of $\mathbb{F}_3$-rational points.*

Mumford	$\langle x^2, 2x+1\rangle$	$\langle x, 2\rangle$	$\langle x, 1\rangle$	$\langle x^2, 2\rangle$	$\langle 1, 0\rangle$
Point Sum	$2(0,1)$	$(0,2)$	$(0,1)$	$2(0,2)$	0

All other divisors from Example 2.3.5 have formal sum representations composed of $\mathbb{F}_{3^2}$-rational points. Let C be defined over $\mathbb{F}_{3^2}$ where $C(\mathbb{F}_{3^2}) = \{\infty, (0,1), (0,2), (1,\alpha^5), (1,\alpha^7), (\alpha,1), (\alpha,\alpha^6), (\alpha^2,\alpha^2), (\alpha^3,1), (\alpha^3,\alpha^2), (2,\alpha), (2,\alpha^3), (\alpha^5,1), (\alpha^5,\alpha^7), (\alpha^6,\alpha^6), (\alpha^7,1), (\alpha^7,\alpha^5)\}$ with α a square root of the irreducible polynomial $x^2 + 2x + 2$. Consider the Mumford representation $D = \langle x^2 - 1, x\rangle$. The polynomial $x^2 - 1$ is irreducible over $\mathbb{F}_3$, but $x^2 - 1 = (x - \alpha^2)(x - \alpha^6)$ in $\mathbb{F}_{3^2}$ where $\alpha^2 = 2\alpha + 2$ and $\alpha^6 = \alpha + 1$. Since the second polynomial in the Mumford representation of D is $v = x$, the corresponding summation is $D = (\alpha^2, \alpha^2) + (\alpha^6, \alpha^6)$ by Lemma 2.3.6.

The Mumford representation of a semi-reduced divisor D can also be defined in terms of principal divisors. The relationship is described in Lemma 2.3.10, but requires the following definition.

Definition 2.3.9. *[12, Adapted from Example 10.3.8] Let C be a hyperelliptic curve over k. Let $D = \sum_{P \in C(k)} n_P P$ and $D' = \sum_{P \in C(k)} n'_P P$ be divisors over C then*

$$\gcd(D, D') = \sum_{P \in C(k)} \min\{n_P, n'_P\}(P)$$

is the greatest common divisor *of D and D'.*

Lemma 2.3.10. *[12, Adapted from Example 10.3.8] Let $u, v \in k[x]$, be such that equation (2.3.1) holds. Let D be the semi-reduced divisor with Mumford representation $\langle u, v \rangle$. Then*

$$D = gcd(\mathrm{div}(u), \mathrm{div}(y - v)) = \sum_{P \in C(k)} \min\{\mathrm{ord}_P(u), \mathrm{ord}_P(y - v)\}(P).$$

Furthermore, the divisor D written as $\mathrm{div}(u, \mathrm{div}(y - v))$ *corresponds to the $k[x, y]$-ideal, $(u, y - v)$, in the coordinate ring (Definition 2.1.5) of C.*

2.3.2 Composition and Reduction of Divisors in Mumford Representation

In this section, Cantor's algorithms [3] for composition and reduction of semi-reduced divisors (which are affine and effective) on a hyperelliptic curve are presented. The natural interpretation of Cantor's algorithms is multiplication of ideals in $k[x, y] \subset k(C)$, but to stay consistent with the above exposition, will be viewed geometrically.

Two affine divisors D_1 and D_2 can be added by combining the points appearing in the formal sums $D_3 = D_1 + D_2$. D_3 can have divisor degree as large as $\deg(D_1) + \deg(D_2)$. One can find an equivalent divisor with smaller degree by interpolating all the points in D_3, intersecting the interpolating polynomial with the curve equation, and taking the hyperelliptic involution of the resulting points not already in the summation of D_3. This process, called a *reduction step* can be applied repeatedly to produce a divisor with as small degree as possible; details about reduction steps and what a reduced divisor is are described later. First, obtaining the Mumford representation of the sum D_3 via Cantor's Composition algorithm will be described. Then computing a reduction step via Cantor's Reduction Step algorithm, and the effect on the degree, are described.

Given two semi-reduced divisors D_1 and D_2 defined over $C : y^2 + hy = f$ with Mumford representation $\langle u_1(x), v_1(x)\rangle$ and $\langle u_2(x), v_2(x)\rangle$, the goal is to compute the Mumford representation $\langle u_3(x), v_3(x)\rangle$ of $D_3 = D_1 + D_2$. Cantor's Composition algorithm computes $\langle u_3(x), v_3(x)\rangle$ by combining the points appearing in the formal sums of both input divisors, accounting for multiplicity, to get $u_3(x)$, and computes a new polynomial $v(x)$ that interpolates all points in the new sum. Combining the supports with multiplicity is trivial other than in the case where for P in $\mathrm{supp}(D_1)$, $\iota(P) \in \mathrm{supp}(D_2)$ (including when $2P = 0$ for P a ramified point). In this case notice that, $(x-x_P)|u_1(x)$, $(x-x_P) \mid u_2(x)$ and $v_1(x_P) = -v_2(x_P) - h(x)$ and so $(x-x_P) \mid (v_1(x)+v_2(x)+h(x))$. For more details see [12, Section 10.3.2]. Cantor's Composition Algorithm is given in Algorithm 2.

Algorithm 2 Cantor's Composition

```
Input: $\langle u_1, v_1\rangle$, $\langle u_2, v_2\rangle$, $f$, $h$.
Output: $\langle u_3, v_3\rangle = \langle u_1, v_1\rangle + \langle u_2, v_2\rangle$.
1: Compute $S, s_1, s_2, s_3$ such that $S = u_1 s_1 + u_2 s_3 + s_3(v_1 + v_2 + h)$.
2: $u_3 = (u_1 u_2)/S^2$.
3: $v_3 = (s_1 u_1 v_2 + s_2 u_2 v_1 + s_3(v_1 v_2 + f))/S \pmod{u_3}$.
4: return $\langle u_3, v_3\rangle$.
```

Example 2.3.11. *Consider the hyperelliptic curve C from Example 2.3.5. Take the semi-reduced divisors $\langle x^2 + 2x + 2, 1\rangle$ and $\langle x^2, 2\rangle$ and compute Cantor's Composition (Algorithm 2) on $\langle x^2 + 2x + 2, 1\rangle + \langle x^2, 2\rangle$. By Step 1 of Algorithm 2,* $\gcd(x^2 + 2x + 2, x^2) = 1$ *and* $1 = (x + 2)(x^2 + 2x + 2) + (2x + 2)(x^2) + 0(1 + 2 + x)$ *so* $S = 1, s_1 = x + 2, s_2 = 2x + 2, s_3 = 0$. *Then, by Steps 2 and 3 of Algorithm 2*

$$u_3 = (x^2 + 2x + 2)(x^2) = x^4 + 2x^3 + 2x^2,$$

and

$$v_3 = (x + 2)(x^2 + 2x + 2)(2) + (2x + 2)(x^2)(1) \equiv x^3 + x^2 + 2 \pmod{x^4 + 2x^3},$$

resulting in the output semi-reduced divisor $\langle x^4 + 2x^3, x^3 + x^2 + 2\rangle$.

Example 2.3.12. *Consider the hyperelliptic curve C from Example 2.3.5. Take the semi-reduced Mumford divisors $\langle x, 2\rangle$ and $\langle x, 1\rangle$ and compute Cantor's Composition (Algorithm 2) on $\langle x, 2\rangle + \langle x, 1\rangle$.*

By Step 1 of Algorithm 2, $\gcd(x, x) = x$ *and* $x = 1x + 0x + 0x$ *so* $S = x, s_1 = 1, s_2 = s_3 = 0$. *Then, by Steps 2 and 3 of Algorithm 2*

$$u_3 = x^2/x^2 = 1, \quad v_3 = x/x = 1 \equiv 0 \pmod{1},$$

resulting in the output semi-reduced divisor $\langle 1, 0 \rangle$.

Given a semi-reduced divisor D over $C : y^2 + hy = f$ of genus g with Mumford representation $\langle u, v \rangle$, it is beneficial to obtain an equivalent divisor (where equivalence is defined as in Lemma 2.3.2) whose Mumford representation has $\deg(u)$ of lower degree if possible. Mumford representations for which the u and v have as low as possible degree are considered *reduced* representations, details of which are discussed in Section 2.4.1 over ramified models and Section 2.4.2 for split models.

Recall by Definition 2.3.4 that the polynomial v interpolates all points in the semi-reduced divisor D. If the divisor has more than g points in the support, i.e. $\deg(u) > g$, then an equivalent divisor with fewer points in the support can be found by equating $y = v$ in the equation describing C, producing a polynomial $f - v(v + h)$ whose zeros correspond to all intersections of C and v. The polynomial u divides the polynomial $f - v(v + h)$ because the points, counted with multiplicity, in the support of D represented by u are contained in the set of all intersection points of v and C. Thus, to compute the new Mumford polynomial u' of the equivalent divisor D', divide $f - v(v + h)$ by u, resulting in a u' corresponding to all other points that v intersects the curve C. Step 1 in Cantor's Reduction Step (Algorithm 3) encapsulates this process. In order to uphold equivalence as defined in Lemma 2.3.2, the hyperelliptic involution of points associated to u' is applied via conjugation of v and the new interpolating Mumford polynomial v' is computed via modular reduction by u', removing all points originally in u from the interpolation. This process corresponds with Step 2 in Cantor's Reduction Step (Algorithm 3).

Lemma 2.3.13. *[12, Adapted from Lemma 10.3.17] Let D be a semi-reduced divisor on a hyperelliptic curve C with Mumford representation $\langle u, v \rangle$. Define u' and v' as the output of Algorithm 3. Then $\langle u', v' \rangle$ is a Mumford representation of D' and $D' \equiv D$ as semi-reduced divisors where equivalence is defined as in Lemma 2.3.2.*

Algorithm 3 Cantor's Reduction Step

```
Input: $\langle u, v\rangle$, $f$, $h$.
Output: $\langle u', v'\rangle \equiv \langle u, v\rangle$.
1: $u' = (f - v(v+h))/u$, made monic.
2: $v' = -v - h \pmod{u'}$.
3: return $\langle u', v'\rangle$.
```

Definition 2.3.14. *[12, Adapted from Definition 10.3.22] Let $C : y^2 + hy = f$ of genus g. A semi-reduced divisor on C is* reduced *if its degree is at most g.*

Example 2.3.15. *Consider the hyperelliptic curve C from Example 2.3.5 where $f = x^5 + 2x + 1$ and $h = x$, and let $\langle x^4 + 2x^3, x^3 + x^2 + 2\rangle$ be the the semi-reduced divisor from Example 2.3.11. Apply Cantor's Reduction Step (Algorithm 3) to $\langle x^4 + 2x^3 + 2x^2, x^3 + x^2 + 2\rangle$. By Step 1 of Algorithm 3*

$$\frac{x^5 + 2x + 1 - (x^3 + x^2 + 2)(x^3 + x^2 + x + 2)}{x^4 + 2x^3 + 2x^2} = \frac{2x^6 + 2x^5 + x^4 + x^3 + 2x^2}{x^4 + 2x^3 + 2x^2} = x^2 + 2x + 2.$$

Then by step 2 of Algorithm 3,

$$2(x^3 + x^2 + 2) + 2x = 2x^3 + 2x^2 + 2x + 1 \equiv 2x + 2 \pmod{x^2 + 2x + 2}.$$

The output of of Algorithm 3 is the reduced divisor $\langle x^2 + 2x + 2, 2x + 2\rangle$.

At this point, representation and arithmetic of (affine) semi-reduced divisors have been discussed. The rest of this section presents properties required in Section 2.4 for defining unique representations of divisor classes over both ramified and split models. The important property that every semi-reduced divisor is equivalent to a reduced divisor over ramified curves is given in Theorem 2.3.17, an explicit Riemann-Roch theorem for hyperelliptic curves given by a ramified model. First, a useful description of properties of reduced divisors is given in Lemma 2.3.16.

Lemma 2.3.16. *[12, Adapted from Lemma 10.3.20] Let $C : y^2 + hy = f$ of genus g and let $\langle u, v\rangle$ be the Mumford representation of a semi-reduced divisor D. Let $\langle u', v'\rangle$ be the polynomials arising from a Cantor reduction step.*

1. *If* $\deg(v) \geq g + 1$, *then* $\deg(u') \leq \deg(u) - 2$.

2. *If* $\deg(f) \leq 2g + 1$, *and* $\deg(u) \geq g + 1 > \deg(v(x))$, *then* $\deg(u') \leq g$.

3. *If* $\deg(f) = 2g + 2$ *and* $\deg(u) > g + 1 > \deg(v)$ *then* $\deg(u') \leq g$.

Theorem 2.3.17. *[12, Adapted from Theorem 10.3.21] Let $C : y^2 + hy = f$ of genus g a hyperelliptic curve.*

- *If C is ramified, then every semi-reduced divisor D is equivalent to a divisor D' where* $\deg(D') \leq g$*, i.e. a reduced divisor.*

- *If C is split, then every semi-reduced divisor D is equivalent to a divisor D' where* $\deg(D') \leq g + 1$.

Proof. Perform Cantor Reduction Steps repeatedly. By Algorithm 3 and Lemma 2.3.16 the desired condition will eventually hold. □

Lemma 2.3.18 below shows that non-uniqueness of an (affine) reduced divisor can only arise with split models and will be important in describing unique divisor class representations in Section 2.4.

Lemma 2.3.18. *[12, Adapted from Lemma 10.3.24] Let $C : y^2 + hy = f$ be a hyperelliptic curve of genus g. Let D_1 and D_2 be reduced divisors. Assume that $D_1 \neq D_2$ but $D_1 \equiv D_2$. Then* $\deg(f) = 2g + 2$*, or* $\deg(h)) = g + 1$.

2.4 Divisor Class Arithmetic

The goal of this section is to describe the representation and arithmetic of divisor classes in $\mathrm{Pic}^0_k(C)$ over ramified and split models. The representation of a divisor class is described as the composition of an affine divisor with points at infinity, where the divisor class arithmetic depends almost entirely on the arithmetic of the affine divisor in the representation. Cantor's composition and reduction step algorithms for (affine) semi-reduced divisors can be used to perform arithmetic on divisor classes in the divisor class group, where additional algorithms needed over split model curves are addressed in Section 2.4.2.

In order to define arithmetic in the divisor class group of a hyperelliptic curve, a unique representative for a divisor class that accounts for both finite and infinite points is needed. First a non-unique representation of a divisor class $[D]$ is given in Lemma 2.4.1 as the sum of an (affine) semi-reduced divisor with the appropriate points at infinity such that $D \in \mathrm{Div}_k^0(C)$.

Lemma 2.4.1. *By Lemma 2.3.2, every degree zero divisor class $[D]$ on a hyperelliptic curve has a representative of the form $D = D_a + n^+(\infty^+) + n^-(\infty^-)$ where D_a is a semi-reduced (in other words, affine and effective) divisor, n^+, n^- are integers and $\deg(D_a) + n^+ + n^- = 0$, where $\infty^+ = \infty^- = \infty$ over ramified curves.*

The semi-reduced divisor D_a used in Lemma 2.4.1 is often called the *affine portion* of the representation, where the bulk of the computational work for computing divisor class arithmetic is done via Mumford representation and Cantor's algorithms. One can do computationally better than the representation in Lemma 2.4.1 by ensuring that the affine portion is reduced. Unique representatives, and algorithms for arithmetic in the divisor class group over ramified and split models are given in Sections 2.4.1 and 2.4.2 respectively.

2.4.1 Divisor Class Arithmetic on Ramified Models

In this section, unique representations of divisor classes, and their arithmetic, are described for ramified curves. On a hyperelliptic curve given by a ramified model, there is only one point at infinity. In this case, arithmetic in the divisor class group can be performed using only the affine portion of the divisor class representative. First, the existence of a unique divisor class representative with a reduced affine portion is described. This is followed by the presentation of Cantor's Addition algorithm for divisor classes for ramified curves.

Theorem 2.4.2. *Let $C : y^2 + hy = f$ of genus g with $f, h \in k[x]$ be a ramified model of a hyperelliptic curve. Let $\langle u, v \rangle$ be the Mumford representation of a semi-reduced divisor D_a. Then*

$[D] = [u, v]$ *denotes a degree zero divisor class representation where*

$$D = D_a - \deg(u)\infty.$$

If $0 \leq \deg(u) \leq g$, *then* $[u, v]$ *is a* reduced representation *of* D. *Every divisor class in* $\mathrm{Pic}^0_k(C)$ *is uniquely represented by a reduced representation* $[D] = [u, v]$ *as defined above.*

Proof. By Lemma 2.4.1, every degree zero divisor class $[D]$ on a ramified hyperelliptic curve has a representative of the form $D = D_a - n(\infty)$ where D_a is a semi-reduced divisor, n is a positive integer and $\deg(D_a) - n = 0$. Put $n = \deg(u)$, then the result follows from Theorem 2.3.17, which states the existence of an equivalent reduced divisor for any semi-reduced divisor. □

For two divisor classes $[D_1] = [u_1, v_1]$, $[D_2] = [u_2, v_2]$, the group law can be computed on the affine portion of the input divisors using Mumford representation by applying Cantor's Composition (Algorithm 2) once, then repeatedly applying Cantor's Reduction Step (Algorithm 3) until the affine portion is reduced. The group law, denoted Cantor's Addition, is presented in Algorithm 4. Note that by Lemma 2.3.16, every application of Cantor's Reduction Step (Algorithm 3) reduces the degree of the affine portion by at least 2. It is essential that the inputs are reduced, so that the addition produces a non-reduced divisor representation with degree at most $2g$. So at most $\lceil g/2 \rceil$ applications are required in order to produced a reduce divisor class representative.

Algorithm 4 Cantor's Addition

```
Input: $[u_1, v_1]$, $[u_2, v_2]$, $f$, $h$.
Output: $[u, v] \equiv [u_1, v_1] + [u_2, v_2]$.
1: Compute $S, s_1, s_2, s_3$ such that $S = u_1 s_1 + u_2 s_3 + s_3(v_1 + v_2 + h)$.
2: $u = (u_1 u_2)/S^2$.
3: $v = (s_1 u_1 v_2 + s_2 u_2 v_1 + s_3(v_1 v_2 + f))/S \pmod{u}$.
4: while $\deg(u) > g$ do
5:     $u = (f - v(v + h))/u$, made monic.
6:     $v = -v - h \pmod{u}$.
7: return $[u, v]$.
```

Example 2.4.3. *Consider the divisor class group defined in Example 2.3.5, where* $C : y^2 + xy = x^5 + 2x + 1$*, and the reduced divisor class representatives* $[x^2 + 2x + 2, 1]$ *and* $[x^2 + 2x + 2, 1]$*. Apply Cantor's Addition (Algorithm 4). By Examples 2.3.11 and 2.3.15,*

$$[x^2 + 2x + 2, 1] + [x^2, 2] \equiv [x^2 + 2x + 2, 2x + 2],$$

coinciding with Table 2.3.5, where $[x^2 + 2x + 2, 1] = [D]$*,* $[x^2, 2] = 8[D]$ *and* $[x^2 + 2x + 2, 2x + 2] = (1 + 8)[D] = 9[D]$.

The computation of the infinite part of the divisor class representative requires no computational attention. Recall that there is only one point at infinity on ramified hyperelliptic curves and so the only possibility for the infinite part of the output divisor class representative over a ramified hyperelliptic curve, where D_a is the affine portion, is $\deg(D_a)\infty$. One could view the composition step, extended to degree zero divisors, to also combine the infinite points in the input divisor class representatives implicitly, and then remove the appropriate multiples of ∞ in the reduction steps.

2.4.2 Divisor Class Arithmetic on Split Models

The goal of this section is to define unique representations of divisor classes over curves described by a split model, and to describe arithmetic in the divisor class group. In contrast to ramified curves, Lemma 2.3.18 states that degree zero divisor classes in this case contain more than one divisor with a reduced affine portion. The points at infinity must play a role in the representation and arithmetic of a degree zero divisor class in order to achieve uniqueness. Therefore, representations of semi-reduced divisors and their arithmetic, as used on ramified curves, are extended to the what is called the *balanced setting*, accounting for the points at infinity.

Thus far, the existence of reduced degree zero divisors has only been established over ramified model curves via Cantor's Reduction Step (Algorithm 3) and Lemma 2.3.16. The general Riemann-Roch theorem implies that for a semi-reduced divisor D, there should always exist an equivalent divisor that is reduced over split model curves as well, although by Theorem 2.3.17, Cantor's Reduction Step (Algorithm 3) gets stuck if the input divisor has degree $g + 1$ [12, Section 10.4.2].

This issue is resolved via operations for composing and reducing with points at infinity described later in Lemma 2.4.15. This section is split into two subsections, first a description of the balanced setting and its properties for degree zero divisors, then the application of the balanced setting to the divisor class group of a split model curve.

Balanced Setting

In this section, the representation of a degree zero divisor in the balanced setting is defined in Definition 2.4.4. Then, extensions of Cantor's algorithms to degree zero divisors on split model curves are described.

Recall from Lemma 2.4.1 that every degree zero divisor can be represented as $D = D_a + n^+(\infty^+) + n^-(\infty^-)$, where D_a is semi-reduced. One way to represent a degree zero divisor is to use $D = D_a - \deg(D)(\infty^+)$, see [31]. This representation is inefficient because it does not utilize ∞^-, resulting in unnecessary compositions and reductions at infinity in the arithmetic. A representation for degree zero divisors balanced over both points at infinity is more natural, and given in Definition 2.4.4.

Definition 2.4.4. *[12, Adapted from Definition 10.4.6] Let $C : y^2 + hy = f$ of genus g be a split model of a hyperelliptic curve. Let $D_\infty = \lceil g/2 \rceil \infty^+ + \lfloor g/2 \rfloor \infty^-$, $\langle u, v\rangle$ be the Mumford representation of a semi-reduced divisor D_a, and $n \in \mathbb{Z}$. Then $\langle u, v, n\rangle$ denotes a degree zero divisor $D = D_0 - D_\infty$, where*

$$D_0 = D_a + n(\infty^+) + (g - \deg(u) - n)(\infty^-).$$

A divisor in the form of D is called a balanced divisor*. If $0 \le \deg(u) \le g$ and $0 \le n \le g - \deg(u)$ then D is called a* reduced balanced divisor.

Example 2.4.5. *Let $C : y^2 = x^6 + x + 2$ be a split model representation of a hyperelliptic curve of genus 2 over $\mathbb{F}_3$ and let $D_\infty = \infty^+ + \infty^-$ as in Definition 2.4.4. The following are some reduced degree zero balanced divisors over C:*

a) $\langle 1, 0, 2\rangle = \langle 1, 0\rangle + 2\infty^+ + (2 - 0 - 2)\infty^- - (\infty^+ + \infty^-) = \infty^+ - \infty^-$.

b) $\langle x+2,1,1\rangle = \langle x+2,1\rangle + 1\infty^+ + (2-1-1)\infty^- - (\infty^+ + \infty^-) = \langle x+2,1\rangle - \infty^-$.

c) $\langle x+2,1,0\rangle = \langle x+2,1\rangle + 0\infty^+ + (2-1-0)\infty^- - (\infty^+ + \infty^-) = \langle x+2,1\rangle - \infty^+$.

d) $\langle x^2+x+1, 2x+2, 0\rangle = \langle x^2+x+1, 2x+2\rangle + 0\infty^+ + (2-2-0)\infty^- - (\infty^+ + \infty^-) = \langle x^2+x+1, 2x+2, \rangle - \infty^+ - \infty^-$.

e) $\langle x^2+x+2, 2x, 0\rangle = \langle x^2+x+2, 2x\rangle + 0\infty^+ + (2-2-0)\infty^- - (\infty^+ + \infty^-) = \langle x^2+x+2, 2x, \rangle - \infty^+ - \infty^-$.

f) $\langle 1,0,1\rangle = \langle 1,0\rangle + 1\infty^+ + (2-0-1)\infty^- - (\infty^+ + \infty^-) = 0$.

The representation $\langle u, v, n\rangle$ is referred to as *balanced Mumford representation* and the integer n is called the *balancing coefficient*. The balancing coefficient keeps track of the number of copies n of ∞^+ (and implicitly ∞^-) in the divisor D_0 from Definition 2.4.4. The use of balanced representatives is efficient and mostly eliminates the need for adjusting the balancing coefficient during group law computations.

The rest of this section is devoted to balanced divisor arithmetic required for computing the group law of the divisor class group. Following, Cantor's Algorithms 2 and 3 are extended to the balanced setting as Balanced Composition Algorithm 5 and Balanced Reduction Step 6. Balanced composition is almost identical to Cantor's composition, with the addition of computing the balancing coefficient n, which is described in Lemma 2.4.6.

Lemma 2.4.6. *[12, Adapted from Lemma 10.4.10] Let C be a hyperelliptic curve over k of genus g given as a split model. Let $D_1 = \langle u_1, v_1, n_1\rangle$ and $D_2 = \langle u_2, v_2, n_2\rangle$ be balanced divisors as defined in Definition 2.4.4. Let D_{a_i}, for $i \in \{1,2\}$ be the affine semi-reduced components of the D_i, let $D_{a_3} = \langle u_3, v_3\rangle$ be semi-reduced divisor equivalent to $D_{a_1} + D_{a_2}$, and let S be defined as in Cantor's Composition (Algorithm 2) for input divisors $\langle u_1, v_1\rangle, \langle u_2, v_2\rangle$. Then*

$$\langle u_1, v_1, n_1\rangle + \langle u_2, v_2, n_2\rangle \equiv \langle u_3, v_3, n_1 + n_2 + \deg(S) - \lceil g/2\rceil\rangle.$$

Notice that since S is defined to be $\gcd(u_1, u_2, v_1+v_2+h)$, by Cantor's Composition (Algorithm 2), $D_{a_1} + D_{a_2} = D_{a_3} + \text{div}_a(S)$, where $\text{div}_a(S)$ is the affine portion of $\text{div}(S)$ and therefore $D_{a_1} + D_{a_2} \equiv$

Algorithm 5 Balanced Composition

```
Input: $\langle u_1, v_1, n_1\rangle$, $\langle u_2, v_2, n_2\rangle$, $f$, $h$.
Output: $\langle u_3, v_3, n_3\rangle = \langle u_1, v_1, n_1\rangle + \langle u_2, v_2, n_2\rangle$.
1: Compute $S, s_1, s_2, s_3$ such that $S = u_1 s_1 + u_2 s_3 + s_3(v_1 + v_2 + h)$.
2: $u_3 = (u_1 u_2)/S^2$.
3: $v_3 = (s_1 u_1 v_2 + s_2 u_2 v_1 + s_3(v_1 v_2 + f))/S \pmod{u_3}$.
4: $n_3 = n_1 + n_2 + \deg(s) - \lceil g/2 \rceil$.
5: return $\langle u_3, v_3, n_3\rangle$.
```

D_{a_3} as semi-reduced divisors. Computation of the balancing coefficient n in Balanced Composition (Algorithm 5) comes directly from Definition 2.4.4. The computation can be viewed as combining the infinite points of both input degree zero divisors, extracting the required $-D_\infty$ in the representation, and accounting for the removal of common points by S.

Example 2.4.7. *Consider the split model of a hyperelliptic curve C and balanced divisors* $\langle x+2, 1, 1\rangle$ *and* $\langle x^2 + x + 1, 2x + 2, 0\rangle$ *from Example 2.4.5. Applying Balanced Composition (Algorithm 5) to* $\langle x + 2, 1, 1\rangle + \langle x^2 + x + 1, 2x + 2, 0\rangle$ *results in an unreduced degree* $3 = g + 1$ *balanced divisor* $\langle x^3 + 2, x^2, 0\rangle$*, where* $S = 1$ *and so* $n_3 = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil = 1 + 0 - 0 - 1 = 0$.

Example 2.4.8. *Consider the split hyperelliptic curve C and balanced divisors* $\langle x^2 + x + 1, 2x + 2, 0\rangle$ *and* $\langle x^2 + x + 1, 2x + 2, 0\rangle$ *from Example 2.4.5. Applying Balanced Composition (Algorithm 5) to* $\langle x^2 + x + 1, 2x + 2, 0\rangle + \langle x^2 + x + 2, 2x, 0\rangle$ *results in an unreduced degree* 4 *balanced divisor* $\langle x^4 + 2x^3 + x^2 + 2, 2x^2 + x + 1, -1\rangle$*, where* $S = 1$ *and so* $n_3 = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil = 0 + 0 - 0 - 1 = -1$.

Given a degree zero divisor D with balanced Mumford representation $\langle u, v, n\rangle$, it is necessary in divisor class group arithmetic to use divisor representatives whose balanced Mumford representation has $\deg(u)$ as small as possible. An extension of Cantor's Reduction Step (Algorithm 3) is presented as Balanced Reduction Step in Algorithm 6 where Lemma 2.3.16 guarantees that repeated applications of Algorithm 6 reduces the affine portion of a degree zero divisor down to $g + 1$ or less. First, properties of the balancing coefficient n in a reduction step are described in Lemma 2.4.9.

Lemma 2.4.9. *[11, Adapted from Section 3] Let $C : y^2 + hy = f$ be a hyperelliptic curve over k of genus g with split model. Let V^+ and V^- be the polynomials defined in Definition 2.1.42 where the leading terms α^+,α^- are the constants in the points at infinity ∞^+, ∞^- respectively. Let $D = \langle u_1, v_1, n_1\rangle$ be a degree zero divisor as defined in Definition 2.4.4. Let u', v' be the polynomials arising from a Cantor Reduction Step (Algorithm 3). There are three cases for the computation of n':*

1) If $\deg(v) = g + 1$ and $\operatorname{lcf}(v) = \alpha^+$, then set $n' = n + \deg(u) - (g + 1)$.

2) If $\deg(v) = g + 1$ and $\operatorname{lcf}(v) = \alpha^-$, then set $n' = n + g + 1 - \deg(u')$.

3) Else set $n' = n + (\deg(u) - \deg(u'))/2$.

Then $\langle u, v, n\rangle \equiv \langle u', v', n'\rangle$.

The proof for Lemma 2.4.9 can be adapted from the proof of correctness for Algorithm 2 and 3 of [11]. Balanced Reduction Step is described in Algorithm 6.

Algorithm 6 Balanced Reduction Step

```
Input: $\langle u, v, n\rangle$, $f$, $h$, $V$ where $\deg(u) > g + 1$
Output: $\langle u', v', n'\rangle \equiv \langle u, v, n\rangle$.
1: $u' = (f - v(v + h))/u$, made monic.
2: $v' = -v - h \pmod{u'}$.
3: if the leading term of $v$ and $V^+$ are equal then
4:     $n' = n + \deg(u) - (g + 1)$.
5: else if the leading term of $v$ and $-V^+ - h$ are equal then
6:     $n' = n + g + 1 - \deg(u')$.
7: else
8:     $n' = n + (\deg(u) - \deg(u'))/2$
9: return $\langle u', v', n'\rangle$.
```

Example 2.4.10. *Consider the hyperelliptic curve with split model C and unreduced balanced divisor $\langle x^3 + 2, x^2, 0\rangle$ from Example 2.4.7. Trying to apply Balanced Reduction Step (Algorithm 6) to $\langle x^3 + 2, x^2, 0\rangle$ results in another unreduced degree $3 = g + 1$ balanced divisor $\langle x^3 + 2x + 1, 2x^2, 0\rangle$,* *where $n' = n + (\deg(u) - \deg(u'))/2 = 0$. Balanced Reduction Step does not reduce the degree of the degree $g + 1$ input divisor.*

Example 2.4.11. *Consider the hyperelliptic curve with split model C and unreduced degree 4 balanced divisor $\langle x^4 + 2x^3 + x^2 + 2, 2x^2 + x + 1, -1\rangle$ from Example 2.4.8. Applying Balanced Reduction Step (Algorithm 6) to $\langle x^4 + 2x^3 + x^2 + 2, 2x^2 + x + 1, -1\rangle$ results in a balanced divisor $\langle x^2 + x + 2, x, 0\rangle$, so $n' = n + (\deg(u) - \deg(u'))/2 = -1 + 1 = 0$. In this case, Balanced Reduction Step produces a reduced balanced divisor.*

Distinguishing between balanced reduction steps for which the leading coefficients of v and either V^- or V^+ coincide as described in Lemma 2.4.9 is required in the presentation of Balanced NUCOMP in Section 3.3 and new nomenclature is introduced below.

Definition 2.4.12. *For an input divisor $\langle u, v, n\rangle$ over a hyperelliptic curve given by a split model, a balanced reduction step is denoted as* special *if either the leading term of v and V^+ are equal or the leading term of v and $-V^+ - h$ are equal, and otherwise denoted as* normal.

Divisor Class Arithmetic

In this section, the balanced setting for degree zero divisors is adapted to the degree zero divisor class group over split models. First, existence and uniqueness of a reduced representation from Definition 2.4.4 for every divisor class in $\mathrm{Pic}^0_k(C)$ is stated in Theorem 2.4.13. Additional operations encapsulating the composition and reduction of degree zero divisors with points at infinity to help with balancing and final reductions are presented. Finally, an algorithm for the group law, denoted Balanced Addition, is described in Algorithm 8.

Theorem 2.4.13. *[12, Adapted from Theorems 10.4.16 and 10.4.19] Let C be a hyperelliptic curve over k of genus g given by a split model. Then every degree zero divisor class in $\mathrm{Pic}^0_k(C)$ contains a unique reduced balanced divisor as defined in Definition 2.4.4, and is denoted $[D]$ with balanced Mumford representation $[u, v, n]$.*

Example 2.4.14. *The following are balanced representations of divisor classes on any split model hyperelliptic curve of genus g:*

1. $[1, 0, \lceil g/2 \rceil] = [1, 0] + \lceil g/2 \rceil \infty^+ + \lfloor g/2 \rfloor \infty^- - D_\infty$ *is the unique representative of the neutral divisor class in* $\mathrm{Pic}_k^0(C)$.
2. $[1, 0, \lceil g/2 \rceil + 1] = [1, 0] + (\lceil g/2 \rceil + 1)\infty^+ + (\lfloor g/2 \rfloor - 1)\infty^- - D_\infty$ *is the unique representative of the principal divisor supported at* ∞^+ *in* $\mathrm{Pic}_k^0(C)$.
3. $[1, 0, \lceil g/2 \rceil - 1] = [1, 0] + (\lceil g/2 \rceil - 1)\infty^+ + (\lfloor g/2 \rfloor + 1)\infty^- - D_\infty$ *is the unique representative of the principal divisor supported at* ∞^- *in* $\mathrm{Pic}_k^0(C)$.

In divisor class arithmetic over split models using balanced divisors as representatives, the group law requires reduced balanced representatives as input, and therefore also as output. Recall from Definition 2.4.4 that a reduced balanced divisor (u, v, n) over a split model hyperelliptic curve of genus g requires $\deg(u) \leq g$, and $0 \leq n \leq g - \deg(u)$. At this point, applying the Balanced Composition and Reduction Step (Algorithms 5 and 6) to reduced balanced divisor representatives, similar to the ramified setting, would not guarantee a reduced balanced divisor as output. Balanced reduction may result in balanced divisors over a hyperelliptic curve of genus g that have degree $g + 1$. Moreover, there is no guarantee that the balancing coefficient is in the appropriate range to be considered balanced.

Next, operations for composing and reducing with points at infinity are described. These operations have the property that if the input divisor has degree $g + 1$ then the output has degree less than or equal to g, and are performed using the polynomials V^+, V^- from Definition 2.1.42. In terms of balanced Mumford representations of degree zero divisors using notation from Definition 2.4.4, these operations can be thought of as transferring a symbolic copy of the point ∞^+ or $\infty^- = -\infty^+$ from the infinite portion of D_0 into the affine portion D_a, keeping equivalence.

Lemma 2.4.15. *[12, Adapted from Lemma 10.4.6] Let $y^2 + hy = f$ be a hyperelliptic curve of genus g over k given by a split model. Let $D = \langle u, v, n \rangle$ be the balanced Mumford representation of a degree zero divisor D such that* $\deg(u) \leq g + 1$. *Let V^+ and V^- be as defined in Definition 2.1.42.*

Let

$$\tilde{v} = V^+ - (V^+ - v \pmod{u}) \in k[x]$$

where $v - V^+$ *is reduced modulo* u *to a polynomial of degree at most* $\deg(u) - 1 \leq g$*. Define*

$$u' = \frac{f - \tilde{v}(\tilde{v} + h)}{u} \text{ made monic,} \quad v' = (-\tilde{v} - h) \pmod{u'}, \quad n' = n + \deg(u) - (g + 1).$$

Then $\deg(u') \leq g$ *by Definition 2.1.42 and for* $D' = (u', v', n')$*,* $D \equiv D'$ *as degree zero divisors. The operation of computing* (u', v', n') *from* (u, v, n) *with* V^+ *is called a* down adjustment *or* a composition and reduction at positive infinity.

Similarly, let

$$\tilde{v} = V^- - (V^- - v \pmod{u}) \in k[x],$$

where $v - V^-$ *is reduced to a polynomial of degree at most* $\deg(u) - 1 \leq g$*. Define*

$$u' = \frac{f - \tilde{v}(\tilde{v} + h)}{u} \text{ made monic,} \quad v' = (-\tilde{v} - h) \pmod{u'}, \quad n' = n + g + 1 - \deg(u').$$

Then $\deg(u') \leq g$ *by Definition 2.1.42 and for* $D' = \langle u', v', n' \rangle$*,* $D \equiv D'$ *as degree zero divisors. The operation of computing* $\langle u', v', n' \rangle$ *from* $\langle u, v, n \rangle$ *with* V^- *is called an* up adjustment *or* composition and reduction at negative infinity.

Lemma 2.4.15 states that for a balanced divisor $\langle u, v, n \rangle$, if $\deg(u) \leq g + 1$, then a down adjustment guarantees a reduced divisor output with the balancing coefficient adjusted downwards by $g + 1 - \deg(u)$ and an up adjustment guarantees a reduced divisor output with balancing coefficient adjusted upwards by $g + 1 - \deg(u')$. Thus, either operation may be used as not only a final reduction step when the degree of the input divisor is $g + 1$ but also to adjust the balancing coefficient either up or down accordingly. Depending on the input balancing coefficient, repeated applications of either adjustment to an unreduced balanced divisor would eventually produce a reduced balanced divisor. If the input balancing coefficient $n < 0$, the up adjustment operation brings n' at least one closer to the desired range. If $n > 0$ then either u is reduced to u' and n' is no closer to the desired range, or n' is one closer as well. Every application of the operations from Lemma 2.4.15 in one direction makes progress towards a reduced balanced representation, and the total number of applications is at most $\lceil g/2 + 1 \rceil$. Algorithm 7, denoted Balanced Adjust, encapsulates this process.

Algorithm 7 Balanced Adjust

Input: $[u, v, n]$, f, h, g, V^+.

Output: $[u', v', n'] \equiv [u, v, n]$.

```
1:  if $n < 0$ then
2:      while $n < 0$ do
3:          $\tilde{v} = V^- - ((V^- - v) \pmod{u})$.
4:          $u = (f - \tilde{v}(\tilde{v} + h))/u$ (made monic, exact division).
5:          $v = (-\tilde{v} - h) \pmod{u'}$.
6:          $n = n + g + 1 - \deg(u')$.
7:  else if $n > g - \deg(u)$ then
8:      while $n > g - \deg(u)$ do
9:          $\tilde{v} = V^+ - ((V^+ - v) \pmod{u})$.
10:         $u = (f - \tilde{v}(\tilde{v} + h))/u$ (made monic, exact division).
11:         $v = (-\tilde{v} - h) \pmod{u'}$.
12:         $n = n + \deg(u) - (g + 1)$.
13: return $[u', v', n'] = [u, v, n]$.
```

Example 2.4.16. *Consider the hyperelliptic curve C given by a split model and the unreduced balanced divisor $\langle x^3 + 2, x^2, 0\rangle$ from Example 2.4.7. Now apply Balanced Adjust (Algorithm 7) to $\langle x^3+2, x^2, 0\rangle$. The if statement in Step 7 of Algorithm 7 is entered as $n = 0 > -1 = 2-3 = g-\deg(u)$, and one down adjustment computation follows via Step 8. Step 9 of Algorithm 7 computes $\tilde{v} = x^3 + x^2 + 2$, Step 10 computes $u = x^2 + 2x + 2$ and Step 11 computes $v = 2$, resulting in a reduced balanced divisor $\langle x^2 + 2x + 2, 2, 0\rangle$, where $n = n + \deg(u) - (g + 1) = 0 + 3 - 3 = 0$ by Step 12 of Algorithm 7.*

Now that all machinery required for group law computations has been discussed, divisor class arithmetic for split models is described. For two divisor classes $[D_1] = [u_1, v_1, n_1], [D_2] = [u_2, v_2, n_2]$ with reduced balanced Mumford representation on a split model hyperelliptic curve of genus g, the group law denoted Balanced Addition, and described in Algorithm 8, is computed via a two-step process. First, the degree zero divisor representations are composed by applying Balanced Composition (Algorithm 5) once, then the logic of Balanced Reduction Step (Algorithm 6) is repeatedly applied. At this point it is possible that the resulting balanced divisor is not reduced, i.e., $\deg(u) = g + 1$ or $n \leq 0$ or $g - \deg(u) \leq n$. The next step is applying a series of adjustment steps, up adjustments if n needs to be increased and down adjustments if it needs to decrease, until the balancing coefficient n satisfies $0 \leq n \leq g - \deg u$ and the balanced divisor representative is thus reduced. At least one application of an adjustment step is required if $\deg(u) = g + 1$. The process of applying adjustments is given by Balanced Adjust (Algorithm 7).

Algorithm 8 Balanced Addition

Input: Reduced $[u_1, v_1, n_1]$, Reduced $[u_2, v_2, n_2]$, f, h, g, V^+.

Output: Reduced $[u, v, n] \equiv [u_1, v_1, n_1] + [u_2, v_2, n_3]$.

```
1:  Compute $S, s_1, s_2, s_3$ such that $S = u_1 s_1 + u_2 s_3 + s_3(v_1 + v_2 + h)$.
2:  $u = (u_1 u_2)/S^2$.
3:  $v = (s_1 u_1 v_2 + s_2 u_2 v_1 + s_3(v_1 v_2 + f))/S \pmod{u}$.
4:  $n = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil$.
5:  **while** $\deg(u) > g + 1$ **do**
6:      $v_o = v$, $u_o = u$;
7:      $u = (f - v_o(v_o + h))/u_o$, (made monic, exact division).
8:      $v = -v_o - h \pmod{u}$.
9:      **if** the leadinging terms of $v_o$ and $V^+$ are equal **then**
10:         $n = n + \deg(u_o) - (g + 1)$.
11:     **else if** the leadinging terms of $v_o$ and $-V^+ - h$ are equal **then**
12:         $n = n + g + 1 - \deg(u)$.
13:     **else**
14:         $n = n + (\deg(u_o) - \deg(u))/2$
15: **return** Balanced Adjust($[u, v, n], f, h, V^+$). (Alg 7)
```

Note that by the Riemann-Roch Theorem [12, Section 8.7], every application of a balanced reduction step (Algorithm 6) reduces the degree of the affine portion by at least 2, and so at most $\lceil g/2 \rceil$ applications are required in order to produced a divisor class representative with the degree of the affine part $g + 1$ or less.

Example 2.4.17. *Let $C : y^2 = x^6 + x + 2$ be a split hyperelliptic curve over $\mathbb{F}_3$. Then $\mathrm{Pic}^0_k(C)$ is given in Table 2.4.17 where $[D] = [1, 0, 0]$ is a generator of $\mathrm{Pic}^0_k(C)$.*

$n[D]$	$[D]$	$2[D]$	$3[D]$	$4[D]$	$5[D]$	$6[D]$	$7[D]$
u	1	$x+2$	$x+2$	x^2+x+2	x^2+x+1	x^2+2x+2	x^2+2x+2
v	0	1	1	$2x$	$2x+2$	1	2
n	0	1	0	0	0	0	0
$n[D]$	$8[D]$	$9[D]$	$10[D]$	$11[D]$	$12[D]$	$13[D]$	
u	x^2+x+1	x^2+x+2	$x+2$	$x+2$	1	1	
v	$x+1$	x	2	2	0	0	
n	0	0	1	0	2	1	

Example 2.4.18. *Consider the divisor class group defined in Example 2.4.17. Apply Balanced Addition (Algorithm 8) to the two reduced divisor classes* $[x^2+x+2, 2x, 0]$ *and* $[x^2+x+1, 2x+2, 0]$*. By Examples 2.4.8 and 2.4.11,*

$$[x^2+x+2, 2x, 0] + [x^2+x+1, 2x+2, 0] \equiv [x^2+x+2, x, 0],$$

coinciding with Table 2.4.17, where $[x^2+x+1, 2x+2, 0] = 5[D]$*,* $[x^2+x+2, 2x, 0] = 4[D]$*, and* $[x^2+x+1, 2x, 0] = (5+4)[D] = 9[D]$.

Example 2.4.19. *Consider the divisor class group defined in Example 2.4.17. Apply Balanced Addition (Algorithm 8) to the two reduced divisor classes* $[x+2, 1, 1]$ *and* $[x^2+x+2, 2x, 0]$*. By Examples 2.4.7 and 2.4.16,*

$$[x+2, 1, 1] + [x^2+x+2, 2x, 0] \equiv [x^2+2x+2, 2, 0],$$

coinciding with Table 2.4.17, where $[x+2, 1, 1] = 2[D]$*,* $[x^2+x+1, 2x+2, 0] = 5[D]$ *and* $[x^2+x+1, 2x, 0] = (2+5)[D] = 7[D]$.

The biggest advantage of using balanced divisor representatives for split model divisor class arithmetic is that in the generic case, where both divisors have degree g and $n = 0$, the number of adjustment steps required is zero for even genus and one for odd genus (as a final reduction from affine portion degree $g + 1$), and the computational complexity is similar to the ramified setting. In the next chapter, improvements to Cantor's algorithms and its Balanced‘ adaptations for divisor class arithmetic in the ramified and balanced settings are discussed.

Chapter 3

Balanced NUCOMP

In Chapter 2, arithmetic in the divisor class group of a hyperelliptic curve, expressed in terms of polynomial arithmetic, was described by Cantor's Addition (Algorithm 4) for ramified models and adapted to Balanced Addition (Algorithm 8) for split models. Various improvements and extensions to Cantor's algorithm have been proposed, including an adaptation of Shank's NUCOMP algorithm [34] for composing binary quadratic forms [19]. The main idea behind NUCOMP is that instead of composing two divisors directly and then reducing to find an equivalent reduced divisor, a type of reduction is applied part way through the composition, so that when the composition is finished the result is almost always reduced. The effect is that the sizes of the intermediate operands are reduced and the intermediate Mumford representations are not computed, resulting in better performance in most cases. Improvements to NUCOMP have been proposed, most recently the work of [18], where best practices for computing Cantor's algorithm and NUCOMP for divisor arithmetic on hyperelliptic curves are empirically investigated. NUCOMP has also been proposed for arithmetic in the infrastructure of a split model curve [20]. Although the balanced divisor setting for divisor class group arithmetic on split models due to Galbraith et. al. [11, 28] is in some ways similar to the infrastructure, NUCOMP has yet to be applied explicitly to that setting.

The novel contribution of this chapter is an adaptation of NUCOMP for divisor class group arithmetic on split model hyperelliptic curves that uses the balanced divisor framework. Efficient algorithmic practices from previous work in the ramified model setting are incorporated and new balanced setting-specific improvements that further enhance practical performance are introduced. Specifically, the new version of NUCOMP includes the following improvements over [20]:

- describes for the first time exactly how to use NUCOMP in the framework of balanced divisors, including explicit computations of the required balancing coefficients; (Section 3.3),
- introduces a novel normalization of divisors in order to eliminate the extra adjustment step required in [20] for frequent inputs when the genus of the hyperelliptic curve is odd, so that in all cases frequent inputs require no extra reduction nor adjustment steps; (Section 3.3.1),
- uses certain aspects of NUCOMP to compute one adjustment step almost for free in some cases. (Section 3.3.3).

Numerical results that demonstrate the efficiency gains realized from the new version of NUCOMP as compared with the previous best balanced divisor class group arithmetic based on Cantor's algorithm and Magma's built-in arithmetic are presented, showing that NUCOMP is the method of choice for all but the smallest genera. Moreover, the novel NUCOMP algorithm for split model divisor class arithmetic introduced in this chapter reduces the complexity of explicit formulas over split model genus 2 and 3 hyperelliptic curves discussed in Chapters 5 and 6 respectively.

The rest of the chapter is organized as follows. In Section 3.1, improvements to Cantor's Addition algorithm are discussed and improved divisor class addition algorithms for ramified and split model hyperelliptic curves are presented. In Section 3.2, a NUCOMP based algorithm for computing divisor class arithmetic over ramified curves is described using all applicable improvements from Section 3.1. In Section 3.3 the main contribution of this chapter, a novel adaptation of NUCOMP, denoted *Balanced NUCOMP*, for computing divisor class arithmetic in the balanced setting of a split hyperelliptic curve is presented along with numerical results comparing the new algorithm with the previous best balanced addition algorithm.

3.1 Improvements to Addition Algorithms

In this section, formulations of Cantor's and Balanced addition algorithms are presented that include a variety of practical improvements taken from the most recent work on this topic [18]. First, techniques for improving the efficiency of both Cantor's and Balanced addition (Algorithms 4 and 8) are described. Then the improved Cantor's addition algorithm, along with a version specialized to doubling for the ramified setting is presented. Finally, a computationally beneficial normalization of the v polynomial for the balanced setting is given and the resulting improved Balanced addition and doubling algorithms described.

3.1.1 General Techniques

There are two main techniques used in the improvement of Cantor's Addition in [18]: optimization for the most frequently-occurring input and the use of Tenner's algorithm for reduction, which requires a revised Mumford representation that uses a computationally useful redundant polynomial. Both techniques apply to Cantor's Addition, but as neither affects the balancing coefficient, they both apply to Balanced Addition as well. Before optimization of divisor class arithmetic for the most frequently-occurring inputs is described, extended Euclidean and quotient remainder algorithms are discussed.

XGCD and DivRem

All the divisor class addition algorithms presented in this chapter and in Chapter 4 require applications of the extended Euclidean algorithm for polynomials. The presentation of the extended Euclidean algorithm in Cantor's Addition in many previous works, given by for example line 1 of Algorithm 4 as

```
1: Compute $S, s_1, s_2, s_3$ such that $S = u_1 s_1 + u_2 s_3 + s_3(v_1 + v_2 + h)$,
```

does not specify a unique solution set for S, s_1, s_2, s_3 without bounds on the degrees. Throughout this chapter and Chapter 4, the notation

$$(d, s, t) = \mathrm{XGCD}(a, b)$$

is used to denote the output of the extended Euclidean algorithm on two input polynomials, specifically $d = \gcd(a, b) = as + bt$ with s, t normalized so that $\deg(s) < \deg(b) - \deg(d)$ and $\deg(t) < \deg(a) - \deg(d)$. Along the same line, the notation

$$(q, r) = \mathrm{DivRem}(a, b)$$

is used to denote the quotient and remainder, respectively, obtained when dividing a by b, i.e. $a = qb + r$. For uniqueness, the remainder r satisfying $r = 0$ or $\deg(r) < \deg(b)$ is taken.

Optimization for the Most Frequently-Occurring Input

Cantor's Addition (Algorithm 4) accepts all possible inputs categorized based on the degrees of the input divisor class representations, and forces unneeded polynomial arithmetic computations most of the time. Some polynomial computations in Algorithm 4 can be omitted by distinguishing between input cases according to the properties of the input.

Consider two normally distributed polynomials defined over a field of size q. These polynomials have a linear factor in common with probability of about $1/q$. The most frequently-occurring input to Cantor's Addition Algorithm is when $\gcd(u_1, u_2, v_1 + v_2 + h) = 1$ for two divisor classes $[u_1, v_1]$ and $[u_2, v_2]$ defined over $C : y^2 + yh = f$.

Recall the composition portion of Balanced Addition (Algorithm 4). Steps 1-3 are provided in Algorithm 3.1.1 for readers convenience.

Algorithm 9 Composition Portion of Cantor's Addition

```
Input: $[u_1, v_1]$, $[u_2, v_2]$, $f$, $h$.
Output: $[u, v] \equiv [u_1, v_1] + [u_2, v_2]$.
1: Compute $S, s_1, s_2, s_3$ such that $S = u_1 s_1 + u_2 s_3 + s_3(v_1 + v_2 + h)$.
2: $u = (u_1 u_2)/S^2$.
3: $v = (s_1 u_1 v_2 + s_2 u_2 v_1 + s_3(v_1 v_2 + f))/S \pmod{u}$.
   ...
```

Following the improvements described for [18, Algorithm 1], the gcd computation in Step 1 of Cantor's (Balanced) Addition can be broken up into two gcd computations, where one of the two computations is omitted for the most frequently-occurring inputs. Moreover, some operations in Step 3 can be simplified for the most frequently-occurring inputs as one of the s_i from Step 1 of Cantor's Addition (Algorithm 4) is always zero. The resulting algorithm requires the use of two new polynomials w_1 and K, where $w_1 = (f - v_1(v_1 + h))/u_1$ is discussed further in Section 3.1.1, and the computation of K depends on which s_i is zero. The resulting portion of the Cantor' Composition algorithm is is given in Algorithm 3.1.1.

Algorithm 10 Composition optimized for frequently-occurring input

```
Input: $[u_1, v_1], [u_2, v_2], f, h$.
Output: $[u, v] \equiv [u_1, v_1] + [u_2, v_2]$.
1:  $(S, a_1, b_1) = \text{XGCD}(u_1, u_2)$.   $(S, a_1$ only)
2:  $K = a_1(v_2 - v_1) \pmod{u_2}$.
3:  if $S \neq 1$ then
4:      $(S', a_2, b_2) = \text{XGCD}(S, v_1 + v_2 + h)$.
5:      $w_1 = (f - v_1(v_1 + h))/u_1$.
6:      $K = a_2K + b_2w_1$.
7:      if $S' \neq 1$ then
8:          $u_1 = u_1/S'$.
9:          $u_2 = u_2/S'$.
10:         $w_1 = w_1S'$.
11:     $K = K \pmod{u_2}$.
12:     $S = S'$.
13: $u = u_1u_2$.
14: $v = v_1 + u_1K \pmod{u}$.
    ...
```

Example 3.1.1. *Let* $C : y^2 = x^7 + 6x^4 + 2x + 1$ *be a ramified hyperelliptic curve with genus* $g = 3$. *Consider divisor classes* $[u_1 = x^3 + 4x + 4, v_1 = 6x]$ *and* $[u_2 = x^3 + 4x + 1, v_2 = 4x^2 + 3x + 4]$. *Then, by Cantor's composition portion (Algorithm 3.1.1),* $\gcd(u_1, u_2, v_1 + v_2 + h) = 1$ *and for* $s_1 = 5$, $s_2 = 2$ *and* $s_3 = 0$, *the equality* $1 = s_1u_1 + s_2u_2 + s_3(v_2 + v_1 + h)$ *holds and*

$$\begin{aligned} v &= (s_1u_1v_2 + s_2u_2v_1 + s_3(v_1v_2 + f))/S \pmod{u} \\ &= 5(x^3 + 4x + 4)(4x^2 + 3x + 4) + 2(x^3 + 4x + 1)(6x) \end{aligned}$$

$$\begin{aligned} &= (6x^5 + x^4 + 2x^3 + 3) + (5x^4 + 6x^2 + 5x) \\ &= 6x^5 + 6x^4 + 2x^3 + 6x^2 + 5x + 3. \end{aligned}$$

The computation requires two gcd *computations, a degree 0 with degree 1 polynomial multiplication, a degree 0 with degree 2 polynomial multiplication, a degree 3 with 2 polynomial multiplication, a degree 3 with 1 polynomial multiplication and one polynomial addition.*

Since $\gcd(u_1, u_2) = 1$*, Algorithm 3.1.1 instead skips the computation of the second* gcd *with* $v_1 + v_2 + h$*, and only requires one degree 0 with degree 2 polynomial multiplication, one degree 3 with 2 polynomial multiplication and two polynomial additions as described below:*

$$\begin{aligned} K &= a_1(v_2 - v_1) \pmod{u_2} \\ &= 5(4x^2 + 3x + 4 - 6x) \pmod{x^3 + 4x + 1} \\ &= 6x^2 + 6x + 6, \end{aligned}$$

and

$$\begin{aligned} v &= v_1 + u_1 K \pmod{u} \\ &= 6x + (x^3 + 4x + 4)(6x^2 + 6x + 6) \pmod{x^6 + 5x^3 + 2x^2 + 6x + 4} \\ &= 6x^5 + 6x^4 + 2x^3 + 6x^2 + 5x + 3. \end{aligned}$$

Revised Mumford Representations and Tenner's Technique

One standard optimization for arithmetic with ideals of quadratic number fields is to represent the ideal as a binary quadratic form, a representation that includes a third redundant coefficient that is useful computationally. In the context of divisor class arithmetic, this means adding the polynomial $w = (f - v(v+h))/u$ to the Mumford representation. Ramified divisor class representations in this case have three coordinates, $[u, v, w]$ and balanced divisor class representations have four, $[u, v, w, n]$.

Definition 3.1.2. *Let* $C : y^2 + yh = f$ *be a hyperelliptic curve. Let* u, v *be the polynomials related to the (balanced) Mumford representations of a divisor class* $[D]$ *in the divisor class group of* C*.*

Let the polynomial w be defined as

$$w = \frac{(f - v(v+h)}{u}.$$

Then over ramified models, $[u, v, w]$ *is the* extended Mumford representation *of* $[D]$ *and over split models* $[u, v, w, n]$ *is the* balanced extended Mumford representation *of* $[D]$.

Example 3.1.3. *Let* $C : y^2 = x^7 + 6x^4 + 2x + 1$ *be a ramified hyperelliptic curve with genus* $g = 3$. *Consider the divisor class* $[D_1] = [u_1 = x^3 + 4x + 4, v_1 = 6x]$ *on* C. *Then,*

$$\begin{aligned} w &= \frac{f - v_1(v_1 + h)}{u_1} \\ &= \frac{x^7 + 6x^4 + 2x + 1 - (6x)^2}{x^3 + 4x + 4} \\ &= x^4 + 3x^2 + 2x + 2, \end{aligned}$$

and the extended Mumford representation for $[D_1]$ *is* $[x^3 + 4x + 4, 6x, x^4 + 3x^2 + 2x + 2]$.

The polynomial w as described in Definition 3.1.2 is computed in every portion of Cantor's and Balanced Addition (Algorithms 4 and 8). Having the polynomial w available as part of the divisor representation results in some savings in the divisor composition portion, and allows for the use of what is known in literature as Tenner's technique in the reduction portion of Cantor's (Balanced) Addition and in Balanced Adjust (Algorithm 7).

The main advantage of using Tenner's algorithm in sequences of reduction or adjustment steps is that one can replace expensive computations of the u polynomial for cheaper ones. The basic idea of Tenner's technique goes as follows. Recall that computing the remainder of two polynomials also inherently computes the quotient. The reduction and adjustment portions of Balanced Addition require a reduction modulo u in order to compute the v in every iteration. Tenner's technique takes advantage of this and the extended Mumford representation by computing both quotient and remainder, and then reusing the quotient to compute the new w' polynomial. The exact computations differ slightly between applications of Tenner's technique to reduction and balanced adjustment steps. Applying Tenner's technique to reduction steps is presented in the following, but application to Balanced Adjust (Algorithm 7) requires other techniques that only apply to the balanced setting and will be described in Section 3.1.3.

Let u, v, w be the extended Mumford polynomials of an unreduced divisor class, for example the resulting divisor class representative after the composition portion of Cantor's Addition, with $\deg(u) > g$ and $w = (f - v(v+h))/u$. Originally, Cantor's reduction steps compute

```
   ...
1: while $\deg(u) > g$ do
2:     $u = (f - v(v+h))/u$, (made monic, exact division).
3:     $v = -v - h \pmod{u}$.
   ...
```

The reduction steps can instead be computed using Tenner's Technique, requiring only one field inversion at the end, as

```
   ...
1: while $\deg(u) > g$ do
2:     $u' = w$.
3:     $(q, r) = \mathrm{DivRem}(-v - h, w)$.
4:     $w = u + q(r - v)$.
5:     $v = r, u = u'$.
6: $w = \mathrm{lcf}(u) w$.
7: $u = u$, (made monic).
   ...
```

The main advantage of Tenner's technique is that it is cheaper to compute $w = u + q(r - v)$ than $w = (f - v(v+h))/u$ because smaller operands are used and only a multiplication by q, which typically has degree 1, is required as opposed to an exact division by u.

Aside from Tenner's technique, extended Mumford representation can also be utilized in the composition portion of Cantors and Balanced Addition. Given the resulting intermediate divisor class representation after composition u', v', an updated w' is required to take advantage of Tenner's technique in reduction portion. Updating w' to reflect the composed divisor class representation still requires an exact division, but takes advantage of precomputed values. The computation for updating w' is given in Lemma 3.1.4.

Lemma 3.1.4. *For input divisors $[u_1, v_1, w_1]$, $[u_2, v_2, w_2]$ to the composition portion optimized for frequently-occurring input (Algorithm 3.1.1), let K, u, v be as computed in Algorithm 3.1.1, then*

$$w = \frac{w_1 - K(v_1 + h + v)}{u_2} \quad \textit{(exact division).}$$

Proof. Recall that $v = v_1 + u_1 k$, $u = u_1 u_2$ and $w_1 = (f - v_1(v_1 + h))/u_1)$. It suffices to show

$$\frac{w_1 - K(v_1 + h + v)}{u_2} = \frac{f - v(v + h)}{u}.$$

Then starting from the left side

$$\begin{aligned}
\frac{w_1 - K(v_1 + h + v)}{u_2} &= \frac{\frac{f - v_1(v_1 + h)}{u_1} - K(v_1 + h + v_1 + u_1 K)}{u_2} \\
&= \frac{f - (v_1(v_1 + h) + u_1 K(v_1 + h + v_1 + u_1 K))}{u_1 u_2} \\
&= \frac{f - (v_1^2 + 2v_1 u_1 K + (u_1 K)^2 + h v_1 + h u_1 K)}{u_1 u_2} \\
&= \frac{f - ((v_1 + u_1 K)^2 + h(v_1 + u_1 K))}{u_1 u_2} \\
&= \frac{f - v(v + h)}{u}.
\end{aligned}$$

□

3.1.2 Improved Divisor Class Arithmetic for Ramified Models

Improved Cantor's Addition, described in Algorithm 11, for adding divisor classes over ramified models, is optimized for the frequently-occurring inputs where $\gcd(u_1, u_2) = 1$, described in Section 3.1.1, and makes use of Tenner's technique, described in Section 3.1.1. Combining the two techniques, along with removing any computations that appear twice results in Algorithm 11.

A more efficient doubling formulation of Algorithm 11 can be obtained by optimizing for the case where the input divisor classes are equal. In this case, the greatest common divisor of the input u polynomials is never one, but the most frequently occurring divisor class input for doubling does take advantage of $\gcd(u, 2v + h) = 1$. Improved Cantor's doubling is described in Algorithm 12.

Algorithm 11 Improved Cantor's Addition

Input: $[u_1, v_1, w_1]$, $[u_2, v_2, w_2]$, f, h.

Output: $[u, v, w] \equiv [u_1, v_1, w_1] + [u_2, v_2, w_2]$.

```
1:  $t_1 = v_1 + h$.
2:  $(S, a_1, b_1) = \mathrm{XGCD}(u_1, u_2)$.   $(S, a_1$ only$)$
3:  $K = a_1(v_2 - v_1) \pmod{u_2}$.
4:  **if** $S \neq 1$ **then**
5:      $(S', a_2, b_2) = \mathrm{XGCD}(S, v_2 + t_1)$.
6:      $K = a_2 K + b_2 w$.
7:      **if** $S' \neq 1$ **then**
8:          $u_1 = u_1/S'$.
9:          $u_2 = u_2/S'$.   (exact division)
10:         $w_1 = w_1 S'$.
11:     $K = K \pmod{u_2}$.
12:     $S = S'$.
13: $u = u_1 u_2$.
14: $v = v_1 + u_1 K$.
15: $w = (w_1 - K(t_1 + v)/u_2$   (exact division).
16: **if** $\deg(u) < g$ **then**
17:     **if** $\deg(v) \geq \deg(u)$ **then**
18:         $(q, r) = \mathrm{DivRem}(v, u)$.
19:         $w = w + q(v + r + h)$.
20:         $v = r$.
21: **else**
22:     **while** $\deg(u) > g$ **do**
23:         $u' = w$.
24:         $(q, v') = \mathrm{DivRem}(-v - h, w)$.
25:         $w = u + q(v' - v)$.
26:         $u = u'$
27:         $v = v'$.
28:     $w = \mathrm{lcf}(u) w$
29:     $u = \mathrm{monic}(u)$.
30: **return** $[u, v, w]$.
```

Algorithm 12 Improved Cantor's Double

Input: $[u_1, v_1, w_1]$, $[u_2, v_2, w_2]$, f, h.
Output: $[u, v, w] \equiv [u_1, v_1, w_1] + [u_2, v_2, w_2]$.

```
1:  $t_1 = 2v_1 + h$.
2:  $(S, a_1, b_1) = \mathrm{XGCD}(u_1, t_1)$.
3:  $K = b_1 w$.
4:  if $S \neq 1$ then
5:      $u_1 = u_1/S$.   (exact division)
6:      $w_1 = w_1 S$.
7:  $K = K \pmod{u_1}$.
8:  $T = u_1 K$.
9:  $u = u_1^2$.
10: $v = v_1 + T$.
11: $w = (w_1 - K(t_1 + T))/u_1$   (exact division).
12: if $\deg(u) < g$ then
13:     if $\deg(v) \geq \deg(u)$ then
14:         $(q, r) = \mathrm{DivRem}(v, u)$.
15:         $w = w + q(v + r + h)$.
16:         $v = r$.
17: else
18:     while $\deg(u) > g$ do
19:         $u' = w$.
20:         $(q, v') = \mathrm{DivRem}(-v - h, w)$.
21:         $w = u + q(v' - v)$.
22:         $u = u'$.
23:         $v = v'$.
24:     $w = \mathrm{lcf}(u) w$.
25:     $u = \mathrm{monic}(u)$.
26: return $[u, v, w]$.
```

3.1.3 Improved Balanced Arithmetic

In this section, improved Balanced Addition and Adjust algorithms taking advantage of an alternate normalization of the v polynomial in Mumford representation called a *reduced basis*, in addition to all techniques from Section 3.1.1, is presented.

Divisor Representation using Reduced Bases

The standard Mumford representation of a divisor $[u, v]$ has v reduced modulo u, but any other polynomial equivalent to v modulo u can be used. In split models, an alternate representation of

v called the *reduced basis*, which has degree $g + 1$, turns out to be computationally superior in practice. Reduced bases are defined in terms of the unique polynomial V^+ or V^- (Definition 2.1.42) the polynomial (or principal) part of $y(y + h(x)) - f(x) = 0$ for which $\deg(f - V^+(V^+ + h)) < g$.

Definition 3.1.5. *A representation of the affine semi-reduced divisor* $[u, v]$ *given by* $[u, \tilde{v}]$ *is in* reduced basis *or* positive reduced basis *if* $\tilde{v} = V^+ - [(V^+ - v) \pmod u]$ *and in* negative reduced basis *if* $\tilde{v} = V^- - [(V^- - v)) \pmod u]$. *The standard Mumford representation of v in which v is taken modulo u is referred to as* adapted basis.

Although the degree of v in reduced basis ($\deg(v) = g + 1$) is higher than in adapted basis ($\deg(v) \leq g - 1$), convenient properties of the polynomials V^+ and V^- provide cancellations in the computation of the w polynomial from the extended Mumford representation, resulting in more efficient divisor addition. Recall the redundant polynomial $w = f - v(v + h)/u$ from the extended Mumford representation. A consequence of Lemma 3.1.6 is that $\deg(w) \leq g$ for v in either reduced basis, as opposed to $\deg(w) \leq g + 2$ in adapted basis.

Lemma 3.1.6. *Let* $C : y^2 + hy = f$ *be a split model for a hyperelliptic curve of genus g. Let V^+ and V^- be the the principal parts of* $y(y + h) - f = 0$ *as described in Definition 2.1.42. Let u and v be the Mumford polynomials of a divisor class over C and* $\tilde{v} = V^+ - [(V^+ - v) \pmod u]$ *or* $\tilde{v} = V^- - [(V^- - v) \pmod u]$. *Then*

$$\deg\left(f - \tilde{v}(\tilde{v} + h)\right) \leq \deg\left(f - v(v + h)\right) - 2.$$

Proof. Recall that $\deg(f) = 2g + 2$ and $\deg(h) \leq g + 1$. Let $\tilde{v} = V^+ - [(V^+ - v) \pmod u]$, proof is identical for $\tilde{v} = V^- - [(V^- - v) \pmod u]$. Then $\deg(f - v(v + h)) = 2g + 2$ because $\deg(v) < g$ by Definition 2.3.4. It suffices to show that $\deg(f - \tilde{v}(\tilde{v} + h)) \leq 2g$. By the construction of V^+ (Definition 2.1.42), V^+ has the property that, starting with the highest term coefficient V^+_{g+1} where $f_{2g+2} - V^+_{g+1}(V^+_{g+1} + h_{g+1}) = 0$, the first $g + 2$ coefficients of $f - V^+(V^+ + h)$ are zero. The result follows from equality of the first two leading coefficients of $\tilde{v}$ and V^+. □

Recall that $\deg(w) = \deg(f - \tilde{v}(\tilde{v} + h) - \deg(u)$, therefore by Lemma 3.1.6 $\deg(w) \leq \deg\left(f - v(v + h)\right) - 2 - \deg(u) \leq g$.

One can efficiently convert a divisor in balanced extended Mumford representation $[u, v, w, n]$ into either reduced basis $[u, v', w', n]$ using Tenner's technique. The conversion is described in Algorithm 13, where $V^{\pm}$ is either V^{+} for positive reduced or V^{-} for negative. Algorithm 14 describes how to convert back.

Algorithm 13 Convert Adapted to Reduced Basis

```
Input: $[u, v, w, n]$, $f$, $h$, $V^{\pm}$.
Output: $[u, v', w', n]$, where $v = v' \pmod{u}$.
1: $(q, r) = \mathrm{DivRem}(V^{\pm} - v, u)$.
2: $v' = V^{\pm} - r$.
3: $w' = w - q(v + h + v')$.
4: return $[u, v', w', n]$.
```

Algorithm 14 Convert Reduced to Adapted Basis

```
Input: $[u, v', w', n]$, $f$, $h$.
Output: $[u, v, w, n]$, where $v' = v \pmod{u}$.
1: $(q, r) = \mathrm{DivRem}(v', u)$.
2: $w = w' - q(v' + h + r)$.
3: $v = r$.
4: return $[u, v, w, n]$.
```

Divisor class composition and reduction are not affected by normalizing v in reduced basis; both produce equivalent intermediate divisor class representations and require the same computations, other than the normalization of v. Notice though, that the computation for the composition portion of composition and reduction at positive or negative infinity in Balanced Adjust (Algorithm 7 lines 9 and 3) is identical to the computation for normalizing v in positive or negative reduced basis. Therefore, depending on the reduced basis, one of two lines can be omitted. In even genus split models, no adjustments are required for the most frequently-occurring inputs, so either type of reduced basis will do. However, for odd genus one up adjustment is always required for the most frequently-occurring inputs. A divisor class representation with v normalized in negative reduced basis ensures that base changes are not required before computing this adjustment step. Therefore, in this work the negative reduced basis is used to represent balanced divisors for all generic divisor class arithmetic over a hyperelliptic curve with a split model.

Improved Balanced Addition and Doubling

The improved Balanced Addition (Algorithm 15) for adding divisor classes over split models closely follows an optimized version of Cantor's Addition (Algorithm 11), and additionally keeps v normalized in negative reduced basis (Section 3.1.3), keeps track of the balancing coefficient n, and applies adjustment steps at the end, similar to Algorithm 8.

Similar to the improved Cantor's doubling (Algorithm 12), a more efficient doubling algorithm can be obtained by specializing to the case where $D_2 = D_1$ and simplifying, described in Algorithm 16.

Improved Balanced Adjust

The improved Balanced Adjust algorithm utilizes Tenner's technique (Section 3.1.1) and a normalization of v in a reduced basis (Section 3.1.3). To make the implementation as efficient as possible, two algorithms are presented, taking advantage of either reduced basis as described in Section 3.1.3. First, the improved Balanced Adjust for a negative reduced basis is given in Algorithm 17, taking advantage of the negative reduced basis by skipping the normalization step in the up-adjustment clause. Tenner's technique as described in Section 3.1.1 is applied in both algorithms, but with the reduced basis normalization of v.

Although keeping divisor class representations in negative reduced basis works best for odd genus curves and equally well for even genus curves, positive reduced basis versions of all algorithms presented in this section are implemented for empirical analysis in Section 3.3.4. Therefore, an alternate version of Balanced Adjust using a positive reduced basis is given in Algorithm 18 where a mirrored implementation of Algorithm 17 is used.

Algorithm 15 Improved Balanced Add

Input: $[u_1, v_1, w_1, n_1]$, $[u_2, v_2, w_2, n_2]$, f, h, V^-.

Output: $[u, v, w, n] \equiv [u_1, v_1, w_1, n_1] + [u_2, v_2, w_2, n_2]$.

```
1:  $t_1 = v_1 + h$.
2:  $(S, a_1, b_1) = \text{XGCD}(u_1, u_2)$.   $(S, a_1$ only)
3:  $K = a_1(v_2 - v_1) \pmod{u_2}$.
4:  **if** $S \neq 1$ **then**
5:      $(S', a_2, b_2) = \text{XGCD}(S, v_2 + t_1)$.
6:      $K = a_2K + b_2w_1$.
7:      **if** $S' \neq 1$ **then**
8:          $u_1 = u_1/S'$.   (exact division)
9:          $u_2 = u_2/S'$.   (exact division)
10:         $w_1 = w_1S'$.
11:         $S = S'$.
12:     $K = K \pmod{u_2}$.
13: $u = u_1u_2$.
14: $v = v_1 + u_1K$.
15: $w = (w_1 - K(t_1 + v))/u_2$.   (exact division)
16: $n = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil$.
17: **if** $\deg(u) \leq g$ **then**
18:     **if** $\deg(v) \geq \deg(u)$ **then**
19:         $(q, r) = \text{DivRem}(V^- - v, u)$.
20:         $tv = V^- - r$.
21:         $w = w - q(v + h + tv)$.
22:         $v = tv$.
23: **else**
24:     **while** $\deg(u) > g + 1$ **do**
25:         **if** $\deg(v) = g + 1$ and $\text{lc}(v) = \text{lc}(-V^- - h)$ **then**
26:             $n = n + \deg(u) - g - 1$.
27:         **else if** $\deg(v) = g + 1$ and $\text{lc}(v) = \text{lc}(V^-)$ **then**
28:             $n = n + g + 1 - \deg(w)$.
29:         **else**
30:             $n + (\deg(u) - \deg(w))/2$.
31:         $u_o = u$.
32:         $u = w$.
33:         $(q, r) = \text{DivRem}(V^- + v + h, u)$.
34:         $v_t = V^- - r$.
35:         $w = u_o - q(v_t - v)$.
36:         $v = v_t$.
37:     $w = \text{lcf}(u)w$.
38:     $u = u$ (made monic).
39: **return** Balanced Adjust($[u, v, w, n], f, h, V^-$).
```

Algorithm 16 Improved Balanced Double

Input: $[u_1, v_1, w_1, n_1]$, f, h, V^-.
Output: $[u, v, w, n] \equiv 2[u_1, v_1, w_1, n_1]$.

```
1: $t_1 = 2v_1 + h$.
2: Compute $(S, a_1, b_1) = \text{XGCD}(u_1, t_1)$.
3: $K = b_1 w_1$.
4: **if** $S \neq 1$ **then**
5:     $u_1 = u_1/S$.    (exact division)
6:     $w_1 = w_1 S$.
7: $K = K \pmod{u_1}$.
8: $T = u_1 K$.
9: $u = u_1^2$.
10: $v = v_1 + T$.
11: $w = (w_1 - K(t_1 + T))/u_1$.    (exact division)
12: $n = 2n_1 + \deg(S) - \lceil g/2 \rceil$.
13: **if** $\deg(u) \leq g$ **then**
14:     **if** $\deg(v) \geq \deg(u)$ **then**
15:         $(q, r) = \text{DivRem}(V^- - v, u)$.
16:         $tv = V^- - r$.
17:         $w = w - q(v + h + tv)$.
18:         $v = tv$.
19: **else**
20:     **while** $\deg(u) > g + 1$ **do**
21:         **if** $\deg(v) = g + 1$ and $\text{lc}(v) = \text{lc}(-V^- - h)$ **then**
22:             $n = n + \deg(u) - g - 1$.
23:         **else if** $\deg(v) = g + 1$ and $\text{lc}(v) = \text{lc}(V^-)$ **then**
24:             $n = n + g + 1 - \deg(w)$.
25:         **else** $n + (\deg(u) - \deg(w))/2$.
26:         $u_o = u$.
27:         $u = w$.
28:         $(q, r) = \text{DivRem}(V^- + v + h, u)$.
29:         $v_t = V^- - r$
30:         $w = u_o - q(v_t - v)$.
31:         $v = v_t$.
32:     $w = \text{lcf}(u) w$.
33:     $u = u$ (made monic).
34: **return** Balanced Adjust($[u, v, w, n]$, $f$, $h$, $V^-$).
```

Algorithm 17 Improved Balanced Adjust (Negative Reduced)

Input: $[u_a, v_a, w_a, n_a]$, f, h, V^-, where $\deg(u_a) \leq g + 1$.

Output: $[u, v, w, n] = [u_a, v_a, w_a, n_a]$, where $\deg(u) \leq g$ and $0 \leq n \leq \deg(u) - g$.

```
1:  $u = u_a, \ v = v_a, \ w = w_a, \ n = n_a,$
2:  if $n < 0$ then
3:      while $n < 0$ do
4:          $u_o = u, \ u = w.$
5:          $(q, r) = \text{DivRem}(V^- + v + h, u).$
6:          $v_t = V^- - r.$
7:          $w = u_o - q(v_t - v), \ v = v_t.$
8:          $n = n + g + 1 - \deg(u).$
9:      $w = \text{lcf}(u)w.$
10:     $u = u$ (made monic).
11: else if $n > g - \deg(u)$ then
12:     $V^+ == V^- - h.$
13:     $t = V^+ - V^-.$
14:     $(q, r) = \text{DivRem}(t, u).$
15:     $v_t = v + t - r.$
16:     $w = w - q(v + h + v_t), \ v = v_t.$
17:     while $n > g - \deg(u) + 1$ do
18:         $n = n + \deg(u) - g - 1.$
19:         $u_o = u, \ u = w.$
20:         $(q, r) = \text{DivRem}(V^+ + v + h, u).$
21:         $v_t = t - r.$
22:         $w = u_o - q(v_t - v), \ v = v_t.$
23:     if $n > g - \deg(u)$ then
24:         $n = n + \deg(u) - g - 1.$
25:         $u_o = u, \ u = w.$
26:         $(q, r) = \text{DivRem}(V^- + v + h, u).$
27:         $v_t = V^- - r.$
28:         $w = u_o - q(v_t - v), \ v = v_t.$
29:     else
30:         $t = V^- - V^+.$
31:         $(q, r) = \text{DivRem}(t, u).$
32:         $v_t = v + t - r.$
33:         $w = w - q(v + v_t), \ v = v_t.$
34:     $w = \text{lcf}(u)w.$
35:     $u = u$ (made monic).
36: return $[u, v, w, n].$
```

Algorithm 18 Improved Balanced Adjust (Positive Reduced)

Input: $[u_a, v_a, w_a, n_a]$, f, h, V^+, where $\deg(u_a) \leq g+1$.

Output: $[u, v, w, n] = [u_a, v_a, w_a, n_a]$, where $\deg(u) \leq g$ and $0 \leq n \leq \deg(u) - g$.

```
1: $u = u_a,\ \ v = v_a,\ \ w = w_a,\ \ n = n_a,$
2: **if** $n > g - \deg(u)$ **then**
3:     **while** $n > g - \deg(u)$ **do**
4:         $n = n + \deg(u) - (g+1).$
5:         $u_o = u,\ \ u = w.$
6:         $(q, r) = \mathrm{DivRem}(V^+ + v + h, u).$
7:         $v_t = V^+ - r.$
8:         $w = u_o - q(v_t - v),\ \ v = v_t.$
9:     $w = \mathrm{lcf}(u) w.$
10:    $u = u$ (made monic).
11: **else if** $n < 0$ **then**
12:    $V^- = -V^+ - h.$
13:    $t = V^- - V^+.$
14:    $(q, r) = \mathrm{DivRem}(t, u).$
15:    $v_t = v + t - r.$
16:    $w = w - q(v + h + v_t),\ \ v = v_t.$
17:    **while** $n < -1$ **do**
18:        $u_o = u,\ \ u = w.$
19:        $(q, r) = \mathrm{DivRem}(V^- + v + h, u).$
20:        $v_t = V^- - r.$
21:        $w = u_o - q(v_t - v),\ \ v = v_t.$
22:        $n = n + g + 1 - \deg(u).$
23:    **if** $n < 0$ **then**
24:        $u_o = u,\ \ u = w.$
25:        $(q, r) = \mathrm{DivRem}(V^+ + v + h, u).$
26:        $v_t = V^+ - r.$
27:        $w = u_o - q(v_t - v),\ \ v = v_t.$
28:        $n = n + g + 1 - \deg(u).$
29:    **else**
30:        $t = V^- - V^+.$
31:        $(q, r) = \mathrm{DivRem}(t, u).$
32:        $v_t = v + t - r.$
33:        $w = w - q(v + v_t), v = v_t.$
34:    $w = \mathrm{lcf}(u) w.$
35:    $u = u$ (made monic).
36: **return** $[u, v, w, n].$
```

3.2 NUCOMP

Cantor's algorithm is closely related to Gauss' composition and reduction of binary quadratic forms. In 1988, Shanks [34] described an alternative algorithm for composition and reduction of binary quadratic forms called NUCOMP. Instead of composing and then reducing, which results in a non-reduced intermediate quadratic form with comparatively large coefficients, the idea of NUCOMP is to start the composition process and to apply an intermediate reduction of the operands using a simple continued fraction expansion that approximates the continued fraction expansion of the quadratic irrationality in regular reduction, before completing the composition. The result is that the intermediate operands are smaller, and at the end the resulting quadratic form is in most cases reduced without having to apply any additional reduction steps. Jacobson and van der Poorten [19] showed how to apply the ideas of NUCOMP to divisor class group arithmetic for ramified models, obtaining analogous reductions in the degrees of the intermediate polynomial operands. Later work by Jacobson, Scheidler and Stein [20] further relates the NUCOMP to continued fraction expansions over ramified and split models.

In this section, background on continued fraction expansions and their connection to divisor class arithmetic are described. Based on this foundation, NUCOMP for divisor class arithmetic in the ramified setting is presented utilizing all applicable improvements from Section 3.1.

3.2.1 Continued Fraction Expansions for Divisor Class Arithmetic

Consider the settings in Improved Cantor's and Balanced Addition (Algorithm 11 and 15). Cantor's Algorithm first computes the non-reduced divisor, and subsequently applies a reduction algorithm. The reduction process can instead be expressed in terms of expanding the simple continued fraction of the quadratic irrationality $(v + y)/u$. The main idea of NUCOMP is that this can be approximated by a continued fraction expansion of the rational function u_2/K. The first several partial quotients of the simple continued fraction expansion of the rational approximation u_2/K are the same as those of $(v + y)/u$, and these can be computed without having to first compute the Mumford representation $\langle u, v\rangle$ of the non-reduced divisor class. Given those partial quotients, the final reduced divisor can be computed via expressions involving them and other low-degree operands, again without having to first compute the non-reduced divisor $\langle u, v\rangle$.

In this section, simple continued fraction expansions, quadratic irrationalities and equivalence of the approximation for $(v + y)/u$ via u_2/K are discussed. All definitions and theorems in this section are adapted from [17], starting with continued fraction expansions of a irrational functions.

Continued Fraction Expansions

One of the main ideas underlying NUCOMP is to approximate the regular continued fraction expansion of a Puiseux series (irrational function) by that of a rational function $G/H \in k(C)$ for $G, H \in k[C]$. First, regular continued fraction expansions of Puiseux series and properties about convergence with rational functions are defined. The theory here is presented generally for split and ramified models.

By Remark 2.1.26, every $R \in k(C)$ has a Puiseux expansion over k in t^{-1} where $t = \sqrt{x}$ for ramified C and $t = x$ for split C. Recall that the completion of the function field $k(C)$ is the Puiseux field $k\langle t^{-1}\rangle$, and every non-zero element is a Puiseux series of the form $\sum_{i=-\infty}^{d} a_i t^i$ where $d \in \mathbb{Z}$, $a_i \in k$ for $i \geq d$ and $a_d \neq 0$.

Definition 3.2.1. *[17, Adapted from Section 2.1] Let $\rho = \sum_{i=-\infty}^{d} a_i t^i$ and put*

$$\lfloor \rho \rfloor = \sum_{i=0}^{d} a_i t^i, \quad \rho_0 = \rho, \quad q_0 = \lfloor \rho_0 \rfloor, \quad \rho_{i+1} = \frac{1}{\rho_i - q_i}, \quad q_{i+1} = \lfloor \rho_{i+1} \rfloor, \tag{3.2.1}$$

then the regular continued fraction expansion *of ρ in the Puiseux field of C is*

$$\rho = q_0 + \cfrac{1}{q_1 + \cfrac{1}{\ddots\, q_n + \cfrac{1}{\rho_{n+1}}}} = [q_0, q_1, \ldots, q_n, \rho_{n-1}],$$

with partial quotients $q_0, q_1, \ldots, q_n$.

Next, useful properties for approximating a regular continued fraction expansion are discussed.

Definition 3.2.2. *[17, Adapted from Section 2.1] Consider the setup of Definition 3.2.1. Set* $A_{-2} = 0,\ A_{-1} = 1,\ B_{-2} = 1,\ B_{-1} = 0$ *and define*

$$\begin{aligned} A_i &= q_1 A_{i-1} + A_{i-2}, \\ B_i &= q_1 B_{i-1} + B_{i-2}, \end{aligned}$$

for $0 \le i \le n$. *It can be shown by induction that*

$$A_i B_{i-1} - B_i A_{i-1} = (-1)^{i-1}. \tag{3.2.2}$$

The rational function $A_i/B_i = [q_0, q_1, ..., q_i]$ *for* $0 \le i \le n-1$ *is the* i-th convergent *of the irrational Puiseux series* ρ.

Definition 3.2.3. *[17, Adapted from Section 2.1] If instead* $\rho = G/H$, *for* $G, H \in k[x]$, *then the regular continued fraction expansion of R as defined in Definition 3.2.2 is a* simple continued fraction expansion *of* ρ. *Given two non-zero polynomials* $G, H \in k[C]$, *with* $\deg(G) > \deg(H)$. *Consider the simple continued fraction expansion of the rational function* $G/H = [q_0, q_1, ..., q_m]$, *where* $m \ge 0$. *Similarly as in Definition 3.2.1, let* $\rho_0 = G/H$ *and* $\rho_{i+1} = (\rho_i - q_i)^{-1}$, *so* $q_i = \lfloor \rho_i \rfloor$ *for i* ≥ 0. *This continued fraction expansion corresponds to the Euclidean algorithm applied to G and H, and can be computed as follows.*

Set $R_{-2} = G$, $R_{-1} = H$, *and define for* $i = 0, 1, ..., n-1$

$$R_i = R_{i-2} - q_i R_{i-1}, \quad \textit{where } q_i = \lfloor R_{i-2}/R_{i-1} \rfloor.$$

Three sequences related to Definition 3.2.3 required for describing how to recover the divisor class representative from continued fraction steps in NUCOMP later are described next. It can be shown by induction that

$$(-1)^{i+1} R_i = H A_i - G B_i \tag{3.2.3}$$

and

$$B_i R_{i-1} + B_{i-1} R_i = H. \tag{3.2.4}$$

Moreover, let the sequence $\{C_i\}_{-2\le i\le n}$ be defined via $C_i = (-1)^{i+1}B_i$. Notice that

$$C_i = C_{i-2} - q_iC_{i-1}, \tag{3.2.5}$$

so the C_i can be computed directly during the computation of R_i and q_i.

Semi-Reduced Divisor Arithmetic Using Continued Fraction Expansions

The goal of this section is to show the connection between computing the simple continued fraction expansion of u_2/K (with u_2, K from Cantor's and Balanced Addition) and the reduction of the unreduced Mumford polynomials u_0, v_0 after compositions. Note that the discussion in Section 3.2.2 can be applied to split models via Mumford polynomials, but does not give a complete description of how to perform arithmetic with balanced divisors. An explicit description of how to adapt NUCOMP to perform complete arithmetic with balanced divisors is novel work in this thesis described in Section 3.3. Therefore, the Mumford polynomials of a divisor class are referred to rather than divisor classes. All of this section is adapted from [18].

First, the connection between computing partial quotients of the irrational function $(y+v)/u$ and the reduction of the divisor represented with Mumford polynomials u, v is discussed.

Definition 3.2.4. *[18, Section 2.1] Consider the Mumford polynomials u_0, v_0 of a divisor class representative over a hyperelliptic curve C. Let $\mathfrak{a} = (u_0, v_0 + y)$ be the primitive $K[C]$-ideal generated by u and $v + y$ as a $k[x]$- module. Put $\rho_0 = (v_0 + y)/u_0$, which is irrational in $k(C)$ and let $q_0, q_1, \ldots$ be any sequence of polynomials in $k[x]$. Define*

$$v_{i+1} = q_iu_i - h - v_i, \quad u_{i+1} = \frac{f - v_{i+1}(v_{i+1} + h)}{u_i}, \tag{3.2.6}$$

for $i \ge 0$. Similar to Definition 3.2.1, set $\rho_i = (v_i + y)/u_i$ and $\rho_{i+1} = (\rho_i - q_i)^{-1}$, then for all $i \ge 0$, $\rho_0 = [q_0, q_1, \ldots, q_i, \rho_{i+1}]$. Then, Equation (3.2.6) determines a continued fraction expansion of ρ_0 in $K\langle t\rangle$, the completion field of $k(C)$ and moreover defines a sequence $\mathfrak{a}_i = (u_{i-1}, v_{i-1})$ of equivalent primitive ideals.

The sequence of primitive ideals defined in Definition 3.2.4, corresponds to a sequence of divisor class representatives that are all equivalent as semi-reduced divisors. Moreover, the sequence produces a reduced divisor class representative equivalent to the semi-reduced representative u_0, v_0 after at most $\lceil (\deg(u_0) - g)/2 \rceil$ steps. For more details see [20, Section 5].

There are two main ingredients to computing the group law of two divisor classes using continued fraction expansions, resulting in the NUCOMP algorithm. Consider the unreduced Mumford polynomials u_0, v_0 representing a divisor class after composition. The first is to recover the same partial quotients q_i that lead to a reduced divisor class representation without having to explicitly compute u_0, v_0. This is attained by computing the simple continued fraction expansion of u_2/K, where $\deg(u_2)$ and $\deg(K)$ are smaller than $\deg(u_0)$ from Cantor's Addition algorithm. The second is to derive the partial quotients q_i from the computation of of the simple continued fraction expansion of u_2/K that do not involve computing the intermediate Mumford polynomials of the reduction process, and recover the reduced Mumford polynomials at the end.

Now, the process of recovering the Mumford polynomials of a divisor class representation after one or more reduction steps from partial quotients via continued fraction expansion steps is described. First, note that

$$\rho_0 = \frac{\rho_{i+1} A_i + A_{i-1}}{\rho_{i+1} B_i + B_{i-1}} \text{ and } \rho_{i+1} = \frac{\rho_0 B_{i-1} - A_{i-1}}{\rho_0 B_i - A_i}, \tag{3.2.7}$$

with ρ, A_i, B_i from Definition 3.2.2. The first identity can be shown by repeatedly substituting $\rho_{i+1} = (\rho_i - q_i)^{-1}$ for $\rho_{i+1}, \rho_i, \ldots, \rho_1$ and simplifying. The second can be obtained by algebraic manipulations to the first.

Next, define

$$L_i = u_0 A_i - v_0 B_i. \tag{3.2.8}$$

It can be shown that

$$(-1)^{i-1} u_0 = L_i B_{i-1} - B_i L_{i-1} \tag{3.2.9}$$

by substituting Equation (3.2.8) for L_i and L_{i-1}, then applying Equation (3.2.2). By Definition 3.2.3, where $G = u_0$ and $H = v_0 + y$ and Equation (3.2.7), it follows that

$$\rho_{i+1} = -\frac{L_{i-1} - y B_{i-1}}{L_i - y B_i}. \tag{3.2.10}$$

The fact that $\rho_{i+1} = (\rho_i - q_i)^{-1}$ (Definition 3.2.1) results in

$$\rho_i = \frac{v_i + y}{u_i} \tag{3.2.11}$$

via induction, where v_i and u_i are obtained from the continued fraction expansion of ρ_0 as in Equation (3.2.6).

By using Equations (3.2.10), (3.2.11) and (3.2.9), it can be shown that

$$\begin{aligned} (-1)^{i+1} u_{i+1} &= (L_i^2 - hL_iB_i - fB_i^2)/u_0 & (3.2.12) \\ (-1)^{i+1} v_{i+1} &= (fB_iB_{i-1} + hB_iL_{i-1} - L_iL_{i-1})/u_0, & (3.2.13) \end{aligned}$$

providing a means to compute v_{i+1} and u_{i+1}, given only the partial quotients q_j, and without computing any intermediate Mumford polynomials u_j, v_j for $0 \le j \le i$. Moreover, Equations (3.2.12) and (3.2.13) also imply that

$$L_i = v_{i+1}B_i + u_{i+1}B_{i-1} + hB_i, \tag{3.2.14}$$

providing a means to recover one of u_{i+1} or v_{i+1} given the other.

At this point, a theorem that succinctly connects computing reduction steps with a simple continued fraction expansion is stated. The proof is almost identical to the proof of Theorem 4.1 in [17], the only difference being alignment of notation with this work.

Theorem 3.2.5. *[17, Adapted from Theorem 4.1] Consider the setting from Improved Cantor's Addition Algorithm 11, considering the divisors as affine semi-reduced divisors instead of divisor classes in the divisor class group. Let u_1, v_1, u_2, v_2 be the input Mumford polynomials to Cantor's Addition where u_1 and u_2 have all common factors divided out via the exact divisions by S. Let u_0, v_0 be the unreduced Mumford polynomials of the intermediate divisor after composition. Then*

$$v_0 = v_1 + u_1K, \quad u_0 = u_1u_2, \quad \textit{and} \quad v_2 \equiv v_0 \pmod{u_2}.$$

If $K/u_2 = [q_0, q_1, \ldots, q_n]$ and

$$\frac{v_0 + y}{u_0} = \left[q_0, q_1, \ldots, q_i, \frac{v_{i+1} + y}{u_{i+1}}\right] \quad (0 \le i \le n),$$

then

$$\begin{aligned} u_{i+1} &= (-1)^{i-1}(R_iM_1 - C_iM_2), \\ v_{i+1} &= (u_1R_i + u_{i+1}C_{i-1})/C_i - v_1 - h \end{aligned}$$

for $-1 \leq i < n$ *where*

$$\begin{aligned} M_1 &= (u_1R_i + (v_2 - v_1)C_i)/u_2, \\ M_2 &= (R_i(v_1 + v_2 + h) + WC_i)/u_2 \text{ with } W = (f - v_1(v_1 + h))/u_1. \end{aligned}$$

Recall that over ramified models, one only needs to consider affine semi-reduced divisor arithmetic to realise arithmetic in the divisor class. Theorem 3.2.5 shows that the simple continued fraction of the rational function K/u_2 can be computed instead of Cantor's reduction step in Cantor's Addition Algorithm 11. There are two things to note, first that

$$\frac{v_0 + y}{u_0} = \frac{v_1 + u_1K + y}{u_1u_2} = \frac{v_1 + y}{u_1u_2} + \frac{K}{u_2},$$

makes K/u_2 a close rational approximation of $(v + y/u)$. Second, considering Definition 3.2.3, the remainder sequence of the Euclidean algorithm, through the computation of the simple continued fraction expansion, applied to K/u_2 is the same as u_2/K with the first $q_i = 0$, since $\deg(u_2) > \deg(K)$. Thus, it is more practical to start with the rational function K/u_2 to approximate $(v + y)/u$ instead. The full algorithm using these results is called NUCOMP and is presented in Section 3.2 as an algorithm for divisor class arithmetic over ramified models. Over split models, more work is required in order to adapted NUCOMP to the balanced setting which is described in Section 3.3.

It remains to determine the index i from Theorem 3.2.5 at which the NUCOMP algorithm should terminate in order to ensure that the resulting divisor class Mumford polynomials u_{i+1} and v_{i+1} correspond to a reduced divisor class representative. The bound satisfying this property is given in Theorem 3.2.6. The proof of Theorem 3.2.6 is almost identical to the proof of Theorem 4.2 in [17], again the only difference being alignment of notation with this work.

Theorem 3.2.6. *[17, Adapted from Theorem 4.2] Let* u_1, v_1, u_2, v_2 *be the input Mumford polynomials to Cantor's Addition where* u_1' *and* u_2' *have all common factors divided out via the exact divisions by*

S. Let u_0, v_0 be the unreduced Mumford polynomials of the intermediate divisor after composition, then

$$v_0 = v_1 + u_1' K, \quad u_0 = u_1' u_2', \quad \textit{and} \quad v_2' \equiv v_0 \pmod{u_2'}.$$

Compute the Mumford polynomials u_{i+1} and v_{i+1} as in Theorem 3.2.5. If i is chosen such that

$$\deg(R_i) \leq \frac{(\deg(u_2) - \deg(u_1) + g)}{2} < \deg(R_{i+1}),$$

then the Mumford polynomials u_{i+1} and v_{i+1} correspond to a reduced divisor class representative.

3.2.2 NUCOMP Algorithm

The most recent work on NUCOMP [18] provides current best optimizations and numerical results demonstrating that it outperforms Cantor's algorithm for hyperelliptic curves of genus as small as 7, and that the relative performance improves as the genus increases. An enhanced version of NUCOMP for adding and reducing divisors without balancing that works for curves defined over arbitrary fields and incorporates all the optimizations described in Section 3.1 from [18] is presented in Algorithm 19. A more efficient doubling formulation, denoted NUDUPL, can be obtained by specializing to the case where $D_2 = D_1$ and simplifying. NUDUPL is described in Algorithm 20.

Although the versions of NUCOMP (Algorithm 19) and NUDUPL (Algorithm 20) presented here are intended for divisor class group addition on ramified models, these will also work for adding reduced affine divisors and producing a reduced output over a split model curve. In [20], the authors describe how to use this in order to perform arithmetic in the infrastructure of a split model hyperelliptic curve, but not in the divisor class group. It was shown that for split models the output of NUCOMP is always reduced for even genus curves, but that for odd genus at least one extra reduction step is required.

3.3 Balanced NUCOMP

In this section, an adaptation of NUCOMP for performing divisor class group arithmetic on a split model hyperelliptic curve using balanced divisor arithmetic is presented. This is novel work where

Algorithm 19 NUCOMP

```
Input: $[u_1, v_1, w_1]$, $[u_2, v_2, w_2]$, $f$, $h$.
Output: $[u, v, w]$ with $[u, v, w] = [u_1, v_1, w_1] + [u_2, v_2, w_2]$.
1:  if $\deg(u_1) < \deg(u_2)$ then
2:      $[u_t, v_t, w_t] = [u_2, v_2, w_2]$, $[u_2, v_2, w_2] = [u_1, v_1, w_1]$.
3:      $[u_1, v_1, w_1] = [u_t, v_t, w_t]$.
4:  $t_1 = v_1 + h$,  $t_2 = v_2 - v_1$.
5:  $(S, a_1, b_1) = \mathrm{XGCD}(u_1, u_2)$.  $(S, a_1$ only)
6:  $K = a_1 t_2 \pmod{u_2}$.
7:  if $S \neq 1$ then
8:      $(S', a_2, b_2) = \mathrm{XGCD}(u_1, v_2 + t_1)$.
9:      $K = a_2 K + b_2 w_1$.
10:     if $S' \neq 1$ then
11:         $u_1 = u_1/S'$,  $u_2 = u_2/S'$.   (exact division)
12:         $w_1 = w_1 S'$.
13:         $S = S'$.
14:     $K = K \pmod{u_2}$.
15: if $\deg(u_2) + \deg(u_1) \leq g$ then
16:     $u = u_2 u_1$,  $v = v_1 + u_1 K$.
17:     $w = (w_1 - K(t_1 + v))/u_2$.   (exact division)
18:     if $\deg(v) \geq \deg(u)$ then
19:         $(q, r) = \mathrm{DivRem}(v, u)$.
20:         $w = w + q(v + h + r)$,  $v = r$.
21: else
22:     Set $r = K$,  $r' = u_2$,  $c' = 0$,  $c = -1$,  $l = -1$.
23:     while $\deg(r) > (\deg(u_2) - \deg(u_1) + g)/2$ do
24:         $(q, r_n) = \mathrm{DivRem}(r', r)$.
25:         Set $r' = r$,  $r = r_n$,  $c_n = c' - qc$,  $c = c_n$,  $c' = c$,  $l = -l$.
26:     $t_3 = u_1 r$.
27:     $M_1 = (t_3 + t_2 c)/u_2$.
28:     $M_2 = (r(v_2 + t_1) + w_1 c)/u_2$.   (exact division)
29:     $u' = l(rM_1 - cM_2)$.
30:     $z = (t_3 + c'u')/c$.   (exact division)
31:     $v = (z - t_1) \pmod{u'}$.
32:     $u = u'$ (made monic).
33:     $w = (f - v(v + h))/u$.   (exact division)
34: return $[u, v, w]$.
```

Algorithm 20 NUDUPL

Input: $[u_1, v_1, w_1]$, f, h.

Output: $[u, v, w]$ with $[u, v, w] \equiv 2[u_1, v_1, w_1]$.

```
1:  $t_1 = v_1 + h, \ \ t_2 = v_1 + t_1$.
2:  $(S, a_1, b_1) = \mathrm{XGCD}(u_1, t_2)$.
3:  $K = b_1 w_1$.
4:  if $S \neq 1$ then
5:      $u_1 = u_1/S$,    (exact division)
6:      $w_1 = w_1 S$.
7:  $K = K \pmod{u_1}$.
8:  if $2\deg(u_1) \leq g$ then
9:      $u = u_1^2, \ \ v = v_1 + u_1 K$.
10:     $w = (w_1 - K(t_1 + v))/u_1$.    (exact division)
11:     if $\deg(v) \geq \deg(u)$ then
12:         $(q, r) = \mathrm{DivRem}(v, u)$.
13:         $w = w + q(v + h + r), \ \ v = r$.
14: else
15:     Set $r = K, \ \ r' = u_1, \ \ c' = 0, \ \ c = -1, \ \ l = -1$.
16:     while $\deg(r) > g/2$ do
17:         $(q, r_n) = \mathrm{DivRem}(r', r)$.
18:         Set $r' = r, \ \ r = r_n, \ \ c_n = c' - qc, \ \ c' = c, \ \ c = c_n, \ \ l = -l$.
19:     $M_2 = (rt_2 + w_1 c)/u_1$.    (exact division)
20:     $u' = l(r^2 - cM_2)$.
21:     $z = (u_1 r + c'u')/c$.    (exact division)
22:     $v = (z - t_1) \pmod{u'}$.
23:     $u = u'$ (made monic).
24:     $w = (f - v(v + h))/u$.    (exact division)
25: return $[u, v, w]$.
```

the additions and improvements to NUCOMP (Algorithm 19) over its infrastructure counterpart [20] are summarized as follows,

4.1 it describes for the first time exactly how to use NUCOMP in the framework of balanced divisors, including explicit computations of the required balancing coefficients,

4.2 it introduces the use of a novel normalization of divisors denoted *negative reduced* (described in Section 3.1.3) in order to eliminate the extra adjustment step required in [20] for frequent inputs when the genus of the hyperelliptic curve is odd, so that in all cases frequent inputs require no extra reduction nor adjustment steps, 4.3 it uses simple continued fraction steps of NUCOMP to eliminate an adjustment step for certain non-frequent cases where the degree of the output divisor is small.

The novel Balanced NUCOMP is described in Algorithm 21, and a more efficient algorithm specialized for the case where $D_2 = D_1$ and simplified, denoted Balanced NUDUPL, is described in Algorithm 22. Numerical results are presented in Section 3.3.4 that demonstrate the efficiency gains realized from this new version of NUCOMP as compared with the previous best balanced divisor class group arithmetic based on Cantor's algorithm and the arithmetic implemented in Magma, showing that NUCOMP is the method of choice for all but the smallest genera. With the improvements of this work, NUCOMP is more efficient than Cantor's algorithm for genus as low as 5, compared to 7 using the version in [20]. The implementation developed for this work is faster than Magma's built-in arithmetic for $g \geq 7$, and the gap increases with the genus.

In the following subsections, more details and justification for each of the above modifications is discussed, and numerical results comparing previous best to this novel work are presented.

3.3.1 Normalization With Negative Reduced Basis

In this section, justification for choosing to implement Balanced NUCOMP (Algorithm 21) with a negative reduced basis is discussed. Let $[u_1, v_1, w_1, n_1]$ and $[u_2, v_2, w_2, n_2]$ be the input for Balanced NUCOMP. For split models, the simple continued fraction portion of NUCOMP can absorb at most one adjustment step while still ensuring that the output divisor is reduced, by setting the bound for the simple continued fraction expansion in Step 22 appropriately. The bound for ramified models described in Theorem 3.2.6 is altered to

$$\deg(R_i) \leq \frac{(\deg(u_2) - \deg(u_1) + g + 1}{2} < \deg(R_{i+1}),$$

where $g + 1$ is used instead of g. The use of a reduced basis in the balanced setting where $\deg(v_1) = \deg(v_2) = g + 1$ requires this condition; for more details see [20, Section 7]. If the bound is set any lower than in Algorithm 21, then the resulting u polynomial ends up having degree greater than g, meaning that the divisor is not reduced.

Algorithm 21 Balanced NUCOMP

Input: $[u_1, v_1, w_1, n_1]$, $[u_2, v_2, w_2, n_2]$, f, h, V^-.
Output: $[u, v, w, n]$ with $[u, v, w, n] = [u_1, v_1, w_1, n_1] + [u_2, v_2, w_2, n_2]$.

```
 1: **if** $\deg(u_1) < \deg(u_2)$ **then**
 2:     $[u_t, v_t, w_t, n_t] = [u_2, v_2, w_2, n_2]$, $[u_2, v_2, w_2, n_2] = [u_1, v_1, w_1, n_1]$.
 3:     $[u_1, v_1, w_1, n_1] = [u_t, v_t, w_t, n_t]$.
 4: $t_1 = v_1 + h$,  $t_2 = v_2 - v_1$.
 5: $(S, a_1, b_1) = \text{XGCD}(u_1, u_2)$.   ($S, a_1$ only)
 6: $K = a_1 t_2 \pmod{u_2}$.
 7: **if** $S \neq 1$ **then**
 8:     $(S', a_2, b_2) = \text{XGCD}(S, v_2 + t_1)$.
 9:     $K = a_2 K + b_2 w_1$.
10:     **if** $S \neq 1$ **then**
11:         $u_1 = u_1 / S'$,  $u_2 = u_2 / S'$.   (exact divisions)
12:         $w_1 = w_1 S'$.
13:         $S = S'$.
14:     $K = K \pmod{u_2}$.
15: $D = \deg(u_2) + \deg(u_1)$.
16: $n = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil$.
17: **if** $D \leq g$ and $((n \geq 0$ and $n \leq g - D)$ or $(\deg(w_1) - \deg(u_2) > g))$ **then**
18:     $u = u_2 u_1$,  $v = v_1 + u_1 K$.
19:     $w = (w_1 - K(t_1 + v))/u_2$.   (exact division)
20:     **if** $\deg(v) \geq \deg(u)$ **then**
21:         $(q, r) = \text{DivRem}(V^- - v, u)$.
22:         $tv = V^- - r$,  $w = w - q(v + h + tv)$,  $v = tv$.
23: **else**
24:     Set $r = K$,  $r' = u_2$,  $c' = 0$,  $c = -1$,  $l = -1$.
25:     **while** $\deg(r) \geq (\deg(u_2) - \deg(u_1) + g + 1)/2$ **do**
26:         $(q, r_n) = \text{DivRem}(r', r)$.
27:         Set $r' = r$,  $r = r_n$,  $c_n = c' - qc$,  $c = c_n$,  $c' = c$,  $l = -l$.
28:     $t_3 = u_1 r$.
29:     $M_1 = (t_3 + c t_2)/u_2$.   (exact division)
30:     $M_2 = (r(v_2 + t_1) + w_1 c)/u_2$.   (exact division)
31:     $u = l(r M_1 - c M_2)$.
32:     $z = (t_3 + c' u)/c$.   (exact division)
33:     $v = V^- - [(t_1 - z + V^-) \pmod{u}]$.
34:     $u = u$ (made monic),  $w = (f - v(v + h))/u$.
35:     **if** $\deg(z) < g + 1$ **then**
36:         $n = n + \deg(u_2) - \deg(r') + g + 1 - \deg(u)$.
37:     **else**
38:         $n = n + \deg(u_2) - \deg(r)$.
39: **return** Balanced Adjust($[u, v, w, n]$, $f$, $h$, $V^-$). (Alg 17)
```

Algorithm 22 Balanced NUDUPL

Input: $[u_1, v_1, w_1, n_1]$ f, h, V^-.

Output: $[u, v, w, n]$ with $[u, v, w, n] = 2[u_1, v_1, w_1, n_1]$.

```
1: $t_1 = v_1 + h$, $t_2 = t_1 + v_1$.
2: $(S, a_1, b_1) = \mathrm{XGCD}(u_1, t_2)$.
3: $K = b_1 w_1$.
4: **if** $S \neq 1$ **then**
5:     $u_1 = u_1/S$.    (exact division)
6:     $w_1 = w_1 S$.
7: $K = K \pmod{u_1}$.
8: $D = 2\deg(u_1)$.
9: $n = 2n_1 + \deg(S) - \lceil g/2 \rceil$.
10: **if** $D \leq g$ and $((n \geq 0$ and $n \leq g - D)$ or $(\deg(w_1) - \deg(u_1) > g))$ **then**
11:     $T = u_1 K$, $u = u_1^2$, $v = v_1 + T$.
12:     $w = (w_1 - K(t_2 + T))/u_1$.    (exact division)
13:     **if** $\deg(v) \geq \deg(u)$ **then**
14:         $(q, r) = \mathrm{DivRem}(V^- - v, u)$.
15:         $tv = V^- - r$,   $w = w - q(v + h + tv)$,   $v = tv$.
16: **else**
17:     Set $r = K$, $r' = u_1$, $c' = 0$, $c = -1$, $l = -1$.
18:     **while** $\deg(r) \geq (g+1)/2$ **do**
19:         $(q, r_n) = \mathrm{DivRem}(r', r)$.
20:         Set $r' = r$, $r = r_n$, $c_n = c' - qc$, $c = c_n$, $c' = c$, $l = -l$.
21:     $M_2 = (rt_2 + w_1 c)/u_1$.    (exact division)
22:     $u = l(r^2 - cM_2)$.
23:     $z = (u_1 r + c' u)/c$.    (exact division)
24:     $v = V^- - [(t_1 - z + V^-) \pmod{u}]$.
25:     $u = u$ (made monic).
26:     $w = (f - v(v + h))/u$.    (exact division)
27:     **if** $\deg(z) < g + 1$ **then**
28:         $n = n + \deg(u_1) - \deg(r') + g + 1 - \deg(u)$.
29:     **else**
30:         $n = n + \deg(u_1) - \deg(r)$.
31: **return** Balanced Adjust($[u, v, w, n]$, $f, h, V^-$). (Alg 17)
```

The choice to normalize all divisor class representatives using the negative reduced basis is made for the following reasons:

1. For odd genus, the frequent case for divisor class arithmetic requires an up adjustment. Ensuring that divisor class representatives are normalized using a negative reduced basis allows for the computation of an adjustment step via an extra step in the NUCOMP simple continued fraction part, so that after NUCOMP the output for the frequent case is both reduced and balanced without any further steps.

2. For even genus, the frequent case requires no adjustments, so either positive reduced or negative reduced basis works equally well.

3. Non-frequent cases of divisor class addition over even and odd genus require either up adjustments, down adjustments or no adjustment. In even genus, out of all cases that require adjustments, exactly half are down and half are up. In odd genus, far more non-frequent cases require an up adjustment than down. This can be seen by analyzing the computation of n in the composition portion of Algorithm 15. Line 12 states $n = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil$, where the ceiling function increases the cases for which $n < 0$ for odd genus curves.

In the Balanced NUCOMP algorithm, the choice of reduced basis dictates the direction of the absorbed adjustment. In order to accurately determine the required direction of an adjustment, one first needs to compute the S polynomial as done in lines 5 and 8 of Balanced NUCOMP (Algorithm 21), then apply the basis change if needed, and finally re-compute S again in the right basis. However, applying a change of basis already requires roughly the same amount of computation as one adjustment step in the right direction relative to the basis. Instead, Balanced NUCOMP is designed to always compute output divisor classes with v normalized in negative reduced basis, forcing the computation of two extra adjustments via Balanced Adjust (Algorithm 17) when a down adjustment is required; one to cancel the wrong direction and one to compute an adjustment in the right direction. Computing the polynomial S and then a basis change is more costly than computing two adjustments, supporting this choice.

3.3.2 Adapting NUCOMP to the Balanced Setting

Most of the logic for updating the balancing coefficient n is the same as in Cantor's algorithm as presented in Algorithm 15. The main difference is that NUCOMP does not require reduction steps, as the output is already reduced due to the simple continued fraction reduction of coefficients in lines 24–26 (Algorithm 21). However, it is necessary to determine how these NUCOMP continued fraction steps affect the resulting balancing coefficient n.

The computation of the simple continued fraction expansion in NUCOMP implicitly keeps track of a principal divisor D_δ, such that for input divisors D_1, D_2 and the reduced output divisor D_3, $D_1 + D_2 = D_3 + D_\delta$, and knowledge of D_δ provides the information needed to update n. Some of this is described in the version of NUCOMP from [20], but this version does not account for special cases of the last reduction step (Definition 2.4.12), nor the use of a negative reduced basis. Both are accounted for in the following analysis, aligning with the special cases from the reduction portion of Balanced Addition (Algorithm 15) and from Balanced Adjust (Algorithm 17).

All but the last continued fraction steps in NUCOMP coincide with normal reduction steps (Definition 2.4.12), where the balancing coefficient n is updated by subtracting $\deg(r)$, described below. The last continued fraction step may either be a normal reduction step, a special reduction step or an adjustment step. Special reductions steps can be viewed as reductions that encounter cancellation with either ∞^+ or ∞^-. The cancellation effectively mimics a composition with ∞^+ or ∞^-, thus requiring the same accounting of the balancing coefficient n as an adjustment step. If the last step is an adjustment step, Balanced NUCOMP attempts a reduction, but a reduced basis effectively already applies composition at infinity, so the attempted reduction completes the adjustment. In both cases, the choice of either positive or negative reduced basis solely dictates the direction of the adjustment.

Definition 3.3.1. *The last continued fraction step is* special *if either an adjustment step or a special reduction step (as described in Definition 2.4.12) is computed; otherwise the last continued fraction step is* normal.

In the following, we describe how to test for special last continued fraction steps. The last step is special exactly when $\deg(z) < g + 1$ as in line 34 of the Balanced NUCOMP algorithm, where z is given in line 31. To see this, first recall that, as described in Section 3.2.1, each continued fraction step of NUCOMP corresponds to a divisor equivalent to the sum of the input divisors D_1 and D_2. Next, an important connection between NUCOMP and Cantor's algorithms is established in Lemma 3.3.2.

Lemma 3.3.2. *Consider Balanced NUCOMP (Algorithm 21) and Balanced Addition (Algorithm 15). Consider the same input divisor classes $[u_1, v_1, w_1, b_1]$ and $[u_2, v_2, w_2, n_2]$ to both algorithms where at least one reduction or adjustment step is required in the divisor class addition. Let u', v' denote the Mumford polynomials of the divisor corresponding to the second-last reduction or adjustment step in either algorithm, and let u, v be the reduced output. Then $v' = v_1 - z$ for $z = (t_3 + c'u)/c$, the quantity from line 31 of Balanced NUCOMP.*

Proof. Let the equation of the output Mumford polynomial from a reduction or adjustment step (Algorithms 6 and 7) in negative reduced basis be

$$v = V^- - [(v' + h - V^-) \pmod{u}].$$

In Balanced NUCOMP, the equation for v is

$$v = V^- - [(t_1 - z + V^-) \pmod{u}],$$

by line 32 of Balanced NUCOMP. The v polynomials are congruent modulo u by Theorem 3.2.5. Since $t_1 = v_1 + h$ by line 4 of Balanced NUCOMP, $v' = v_1 - z$. □

Considering the setup in Lemma 3.3.2, the last continued fraction step is a special step whenever $\deg(v') = g + 1$ and the leading coefficient of v' is the same as that of V^- (or V^+ if positive reduced basis is being used, because in that case cancellations in the leading coefficients of $V^- - v'$ (or $V^+ - v'$) in the computation of v cause the degree of u to be less than g, implying that the last step is special.

Lemma 3.3.2 compares the computation of v in line 32 of Balanced NUCOMP with the computation of v in the the reduction step of Balanced Addition (Algorithm 8), and also in any case of Balanced Adjust (Algorithm 7), concluding that $v' = v_1 - z$. Thus, the conditions for the last continued fraction step being special are satisfied when $\deg(z) < g + 1$, because this implies that the degree and leading coefficients of v' and v_1 are the same. Note that $\deg(v_1) = g + 1$ and the leading coefficient of v_1 is the same as that of V^+ (or V^-) because the input divisor $[u_1, v_1, w_1, n_1]$ is given in negative (or positive) reduced basis.

Next, computation of the balancing coefficient n, depending on the type of basis and whether the last step is special or normal, is discussed. There are four possible cases for the computation of n dictated by the choice of positive or negative reduced basis and either normal or special last steps. Note that the cases that arise with positive reduced basis are not included in Algorithm 21 due to the choice of working exclusively with the negative reduced basis. The computation of n is described for this case below, too, as both versions were implemented. Performance comparisons are presented in the next Section.

Let t be the number of copies of ∞^+, accumulated over all continued fraction steps of Algorithm 21. The change in t from any iteration of the while loop is $\delta = \deg(q) = \deg(r') - \deg(r)$ where a reduction of degree in r corresponds to a reduction of 2 in degree of the divisor. Then $t = \sum \delta$ is the accumulated distance over all steps, as every normal reduction step reduces the n value by one. (Algorithm 6.)

In the up adjustment case (the only case that arises in Algorithm 21 since a negative reduced basis is used), the total difference in n from composition without the last step is $t' = \deg(u_2) - \deg(r')$. If the last step is special then $t = t' + \delta' = \deg(u_2) - \deg(r') + g + 1 - \deg(u)$. If the last step is normal, then $t = t' + \delta' = \deg(u_2) - \deg(r') + (\deg(r') - \deg(r)) = \deg(u_2) - \deg(r)$.

In the down adjustment case, which would arise if positive reduced basis is used, the total difference in n from composition without the last step is $t' = D - t - \deg(u) = \deg(u_1) + \deg(r') - \deg(u)$. If the last step is special then $t = t' + \delta' = \deg(u_1) + \deg(r') - \deg(u) - ((2g - 2 - \deg(u)) - g - 1) = \deg(u_1) + \deg(r') - g - 1$ where the degree of the u polynomial in the previous step is $2g - 2 - \deg(u)$. If the last step is normal, then $t = t' + \delta' = \deg(u_1) + \deg(r') - \deg(u) - (\deg(r') -$

$\deg(r)) = \deg(u_1) + \deg(r) - \deg(u)$.

As described earlier, the simple continued fraction steps of Algorithm 21 (lines 24–26) can only incorporate up to one adjustment step in addition to all the required reduction steps. Thus, a final call to Algorithm 17 is required in order to ensure that the output divisor is both reduced and balanced. However, extra adjustment steps are only required for non-frequent cases. In particular, for sufficiently large fields, with high probability, every application of Balanced NUCOMP yields a reduced balanced output with no extra steps.

3.3.3 Eliminating an Adjustment for Some Non-Generic Cases

The non-balanced version of NUCOMP presented at the beginning of this section (Algorithm 19, lines 14–19) makes use of the observation that if $D = \deg(u_1/S) + \deg(u_2/S) \leq g$, then completing the composition using Cantor's algorithm will produce a divisor that is reduced without having to do any subsequent reduction steps. In the balanced setting, the corresponding balancing coefficient is $n = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil$. If this divisor is not balanced, i.e. $n < 0$ or $n > g - D$, then one may apply NUCOMP's simple continued fraction-based reduction in order to compute one adjustment step, saving one of the more expensive standard adjustment steps. However, this is only beneficial if $\deg(w_1) - \deg(u_2) \leq g$, because otherwise the resulting output divisor will not be reduced due to the fact that $\deg(u)$ depends on the degree of $w_1 S/(u_2/S) = w_1/u_2$. Thus, the composition with Cantor's algorithm (lines 16–21 of Algorithm 21) is only applied if the resulting divisor is reduced and balanced, or if it is reduced and not balanced but performing a NUCOMP reduction step would result in an non-reduced divisor.

3.3.4 Empirical Analysis

In this section, empirical data is presented that illustrates the relative performance of the addition and doubling algorithms presented in this chapter over both ramified and split models using positive and negative reduced basis representations. All algorithms for addition and doubling are implemented in Magma as a proof of concept. Therefore, the absolute timings are not of great interest. The reader should rather focus on the relative cost between the various algorithms and models. The experiments were performed on a workstation with an Intel Xeon 7550 processor that has 64 cores, each of which is 64-bit and runs at 2.00 GHz.

As a preliminary benchmark, doubling and addition algorithms were compared separately. For addition, which includes Cantor and NUCOMP over ramified and split models, a Fibonacci-like sequence of divisors using $D_{i+1} = D_i + D_{i-1}$ was computed, starting from two random divisors. Similar experiments were performed for the doubling algorithms (Cantor's algorithm and NUDUPL, which is NUCOMP specialized to doubling a divisor) over ramified and split models, by computing a series of thousands of additions of a divisor class with itself. Timings for all genera ranging from 2 to 50 and prime fields of sizes 2, 4, 8, 16, 32, 64, 128, 256, 512 and 1024 bits were collected. All timings were run over random hyperelliptic curves with $h = 0$, using implementations of our algorithms that were specialized to exclude any computations with h (see Section 5.1.1 for an explanation of why this does not sacrifice generality in these cases.)

The algorithms used in the ramified setting are:

- Improved Cantor's Addition (Algorithm 11),
- Improved Cantor's Doubling (Algorithm 12),
- NUCOMP (Algorithm 19),
- NUDUPL (Algorithm 20).

The algorithms used in the balanced setting are:

- Improved Balanced Addition (Algorithm 15),
- Improved Balanced Doubling (Algorithm 16),
- Balanced NUCOMP (Algorithm 21),
- Balanced NUDUPL (Algorithm 22).

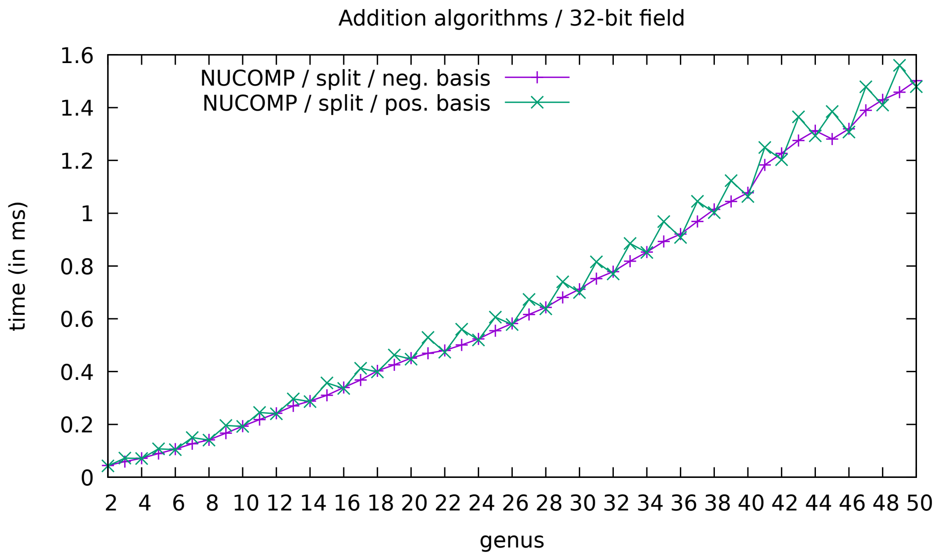
Magma's built-in arithmetic was also timed for comparison. For the balanced setting, positive reduced basis algorithms were compared as well, and are based on Algorithms 15 and 21 but with divisors normalized via $V^+ - (V^+ - v) \pmod{u}$ as opposed to a negative reduced basis. Moreover, the positive reduced version of Balanced Adjust (Algorithm 18) replaces the negative reduced version (Algorithm 17).

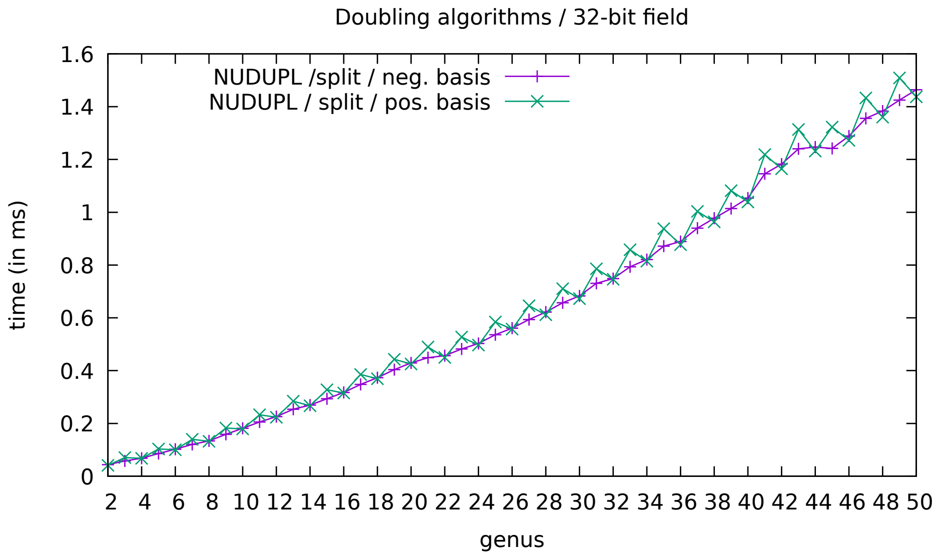
Based on observations, and apart from the absolute timings, the relative performance of the various algorithms does not depend on the field size. In the next figures, comparison for 32-bit fields only are presented, as these results are representative of the other field sizes. The following conclusions are drawn from these plots:

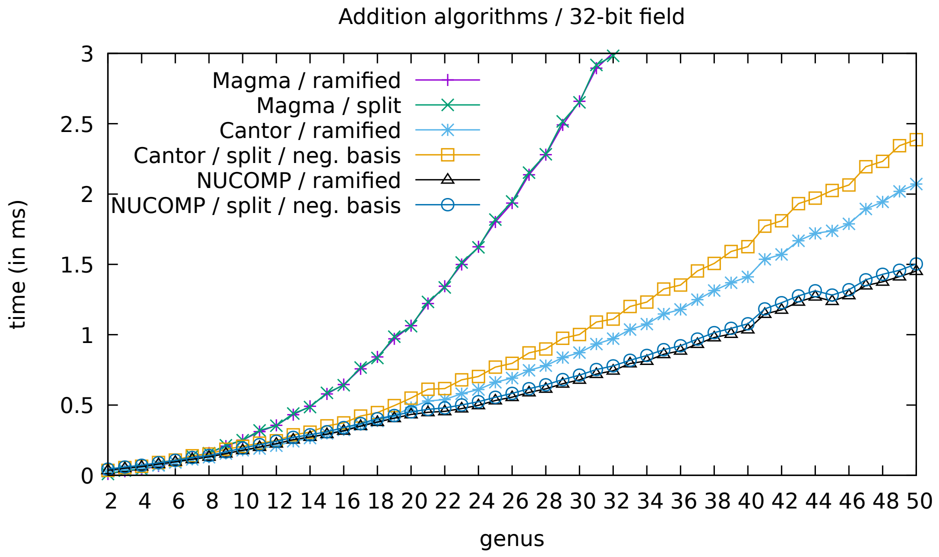
- For split models, as illustrated in the first graph, negative reduced and positive reduced basis perform about the same for even genus. As expected, negative reduced basis is slightly better for odd genus due to the fact that frequent cases require no adjustments steps in negative reduced basis as opposed to one adjustment step in positive reduced basis.

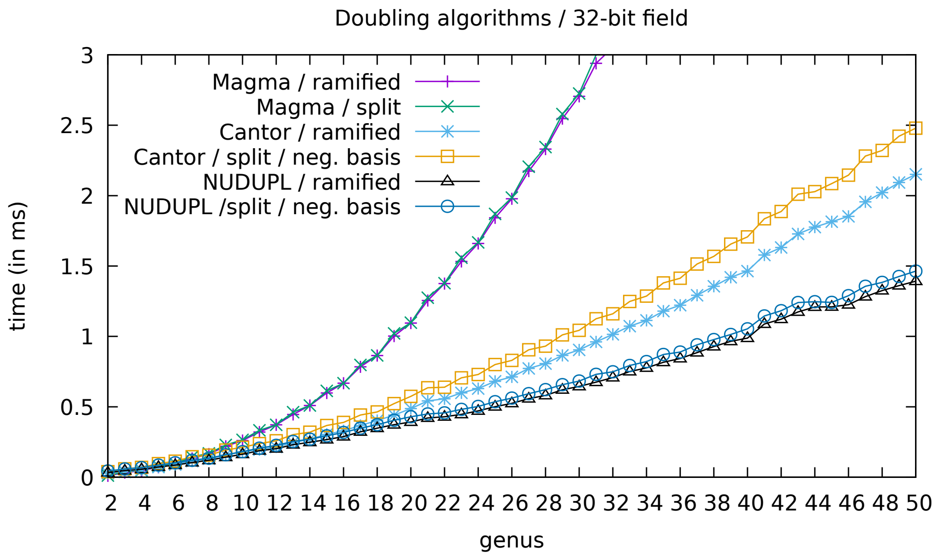
- The second graph shows that, for split models, the implementation of balanced NUCOMP rapidly becomes faster than Cantor as g grows. It also shows that, when using balanced NUCOMP, the difference between the best algorithms for split and ramified models is negligible for all genera. Furthermore, all of the implementations are considerably faster than Magma's built in arithmetic as the genus grows. The graph does not include timings for Magma for $g > 32$ so that the comparisons between the other algorithms are easier to see. We note that the best split model algorithm is about five times faster than Magma's built-in arithmetic at genus 50.

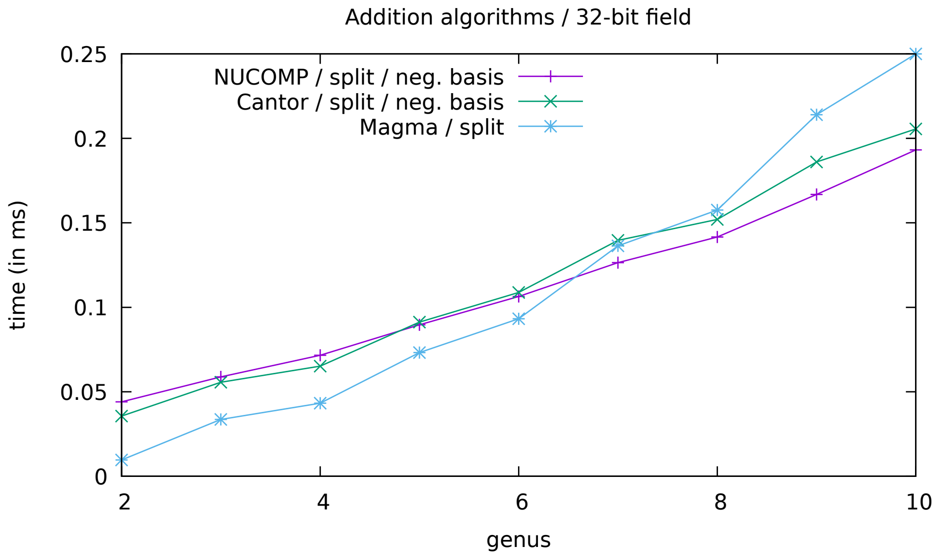
- Not surprisingly, as shown in the last graph, for small genus ($g < 5$), Cantor's algorithms are slightly faster than the NUCOMP algorithms. Magma's built-in arithmetic is also faster for $g < 7$. This is suspectedly due to Magma's implementation having access to faster internal primitives, while the implementations in this work have to use the generic polynomial ring setting. Even so, the implementation of NUCOMP in this work is the fastest option for $g \geq 7$.

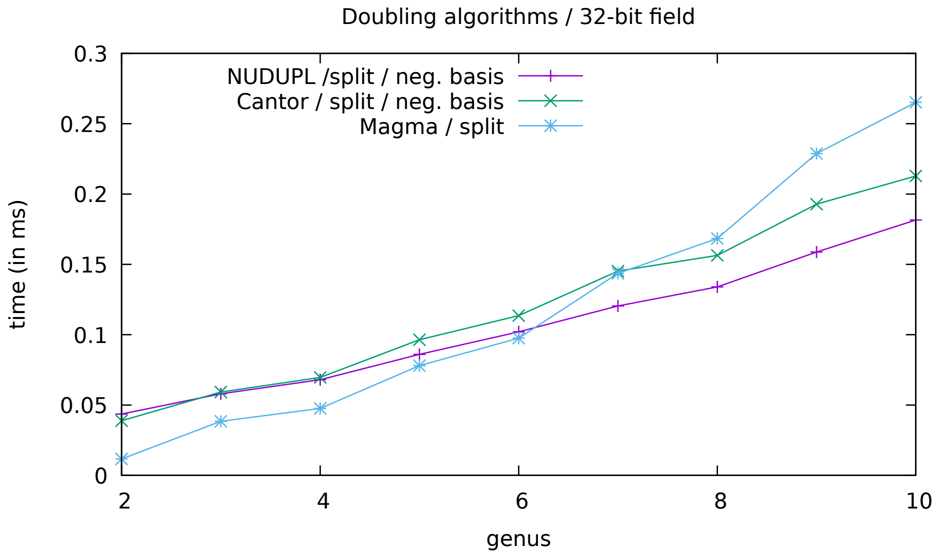

Addition algorithms / 32-bit field
NUCOMP / split / neg. basis
NUCOMP / split / pos. basis
time (in ms)
genus
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50


Doubling algorithms / 32-bit field
NUDUPL /split / neg. basis
NUDUPL / split / pos. basis
time (in ms)
genus
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50


Addition algorithms / 32-bit field
Magma / ramified
Magma / split
Cantor / ramified
Cantor / split / neg. basis
NUCOMP / ramified
NUCOMP / split / neg. basis
time (in ms)
genus
0
0.5
1
1.5
2
2.5
3
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50


Doubling algorithms / 32-bit field
Magma / ramified
Magma / split
Cantor / ramified
Cantor / split / neg. basis
NUDUPL / ramified
NUDUPL /split / neg. basis
time (in ms)
genus
0
0.5
1
1.5
2
2.5
3
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50


Addition algorithms / 32-bit field
NUCOMP / split / neg. basis
Cantor / split / neg. basis
Magma / split
time (in ms)
genus
0
0.05
0.1
0.15
0.2
0.25
2
4
6
8
10


Doubling algorithms / 32-bit field
NUDUPL /split / neg. basis
Cantor / split / neg. basis
Magma / split
time (in ms)
genus
0
0.05
0.1
0.15
0.2
0.25
0.3
2
4
6
8
10

The empirical results indicate that Balanced NUCOMP provides an improvement for computing balanced divisor class arithmetic on hyperelliptic curves given by a split model with a cross-over as low as genus 5. As expected, our choice of normalizing v in negative reduced basis and therefore incorporating up-adjustments into NUCOMP performs equally well when compared to positive reduced over even genus, and slightly better over odd. Furthermore, Balanced NUCOMP performs almost as well and sometimes better than ramified curve NUCOMP and closes the performance gap between ramified model and split model divisor arithmetic.

Improved explicit formulations for divisor arithmetic over split models in genus 2 and 3 utilize the novel Balanced NUCOMP, and are discussed in Chapters 5 and 6. Moreover, integrating algorithms from this chapter directly into Magma's built-in arithmetic, or implementing the algorithms directly in C/C++ so that they do not suffer from the overhead associated to working in a high-level system like Magma, might reduce the relative performance, either lowering or elimination any cross-over points between the algorithms and Magma's arithmetic.

Chapter 4

Towards Explicit Formulas

The goal of this chapter is to align improvements and specializations to generic polynomial-based algorithms for divisor class addition from previous works with those introduced in Chapter 3, and to introduce the main techniques used in developing explicit formulas for computing divisor class arithmetic on genus 2 and 3 hyperelliptic curves.

A common featured from previous methods is that generic methods for improving Cantor's Addition algorithms are adapted to the most frequently occurring inputs where the input divisor classes have degree g, and $\gcd(u_1, u_2, v_1 + v_2 + h) = 1$ with input divisor classes $[u_1, v_1]$, $[u_2, v_2]$ (Section 3.1.1). These cases are denoted as the *frequent case* for addition and doubling. If the field k is sufficiently large, then with high probability [24], input divisor classes are in the frequent case. All other cases are denoted as *special* in this work, as these cases occur infrequently. The main focus of this chapter is introducing a generic framework that combines both special and frequent cases, as most special cases have previously not been made explicit.

To align with previous work [24, 6, 10], explicit techniques are described for ramified models in this chapter. Almost all techniques also apply to split models, the main difference being $\deg f = 6$ not 5 and $\deg h = 3$ not 2 for genus 2 curves, and $\deg f = 8$ not 7 and $\deg h = 4$ not 3 for genus 3 curves. These differences complicate some explicit formula computations, and any discrepancies or additional techniques for split models are addressed in Chapters 5 for genus 2 and Chapter 6 for genus 3.

First, descriptions of the main generic methods from explicit formula literature for frequent case divisor class addition and doubling are presented. Comparisons are made with specializations of NUCOMP from Chapter 3 to genus 2 and 3, concluding that the NUCOMP-based methods are better suited than the generic methods for basing explicit formulas on. Explicit formulas are then introduced and previous well known explicitization techniques, as well as some novel techniques, are presented.

4.1 Generic Methods

There are two main generic methods in previous literature; Harley's improvements to Cantor's algorithm [15] and the linear algebra method due to Costello and Lauter [6]. Harley's method is a specialization of Cantor's algorithm for the frequent cases originally described for genus 2 curves, where the computations were broken up into certain sub-expressions allowing for reuse of those sub-expressions, saving on operations overall. On the other hand, the linear algebra method, described for the frequent cases over all genera, invokes linear algebra techniques to compute the polynomial that interpolates the support of the composed divisor class representative, ignoring Cantor's Composition (Algorithm 2), but utilizing Cantor's Reduction (Algorithm 3).

4.1.1 Harley's Improvements in the Frequent Case

Harley reduced the cost of the group operations for Cantor's algorithm for genus 2 curves described by a ramified model by specializing to the most frequently-occurring cases of the input divisors [15], omitting the explicit computation of the unreduced divisor class representation after composition and reusing computations from the composition portion in the reduction. This approach is precisely the approach borrowed from the binary quadratic form setting used to improve generic Cantor's Algorithm 11 in Chapter 3, but only applied to frequent cases in genus 2.

Consider a reduced divisor class in Mumford representation $[D_1] = [u_1, v_1]$ in the genus 2 frequent case for doubling. The resulting polynomials from Cantor's composition Algorithm 4 for doubling are $u' = u_1^2$ and v' of degree ≤ 3 satisfying $u' \mid f - v'(v' + h)$. Harley noticed that v_1

is a square root of f modulo u_1, and suggested that one can double the multiplicity of all points in the support of $[u_1, v_1]$ by using a Newton iteration to compute a square root modulo u_1^2. The general approach to doubling a divisor class is to use Newton iteration by setting $u' = u_1^2$, and $v' = v_1 - [(f - v_1(v_1 + h))/(h + 2v_1)] \pmod{u'}$ to create an unreduced divisor class $[D'] = [u', v']$.

Harley's method for divisor class addition is similar to doubling but the Newton iteration is replaced by a Chinese remainder calculation. In the genus 2 frequent case for addition, the two divisor classes $[D_1] = [u_1, v_1]$, and $[D_2] = [u_2, v_2]$ to be added consist of four points different from one another and from one another's negatives. The resulting polynomials from Cantor's composition algorithm are $u' = u_1u_2$ and a polynomial v' of degree ≤ 3 satisfying $u' \mid f - v'(v' + h)$. Notice the inputs u_1 and u_2 also have the properties $u_1 \mid f - v_1(v_1 + h)$ and $u_2 \mid f - v_2(v_2 + h)$, providing the relations $f - v'(v' + h) = u_1u_2k$, $f - v_1(v_1 + h) = u_1m$ and $f - v_2(v_2 + h) = u_2n$ for some polynomials k, m, n. The polynomial v' can be obtained by using Garner's Algorithm for the Chinese Remainder through the equations

$$v' \equiv v_1 \pmod{u_1},$$
$$v' \equiv v_2 \pmod{u_2},$$

to create $[D'] = [u', v']$ where $u' = u_1u_2$.

At this point $[D'] = [u', v']$ corresponds to a representation of $[D] = 2[D_1]$ or $[D] = [D_1] + [D_2]$ that is unreduced in both the doubling and addition steps presented. A reduction step is necessary to acquire the proper reduced Mumford representation of $[D]$. Cantor's reduction step goes as follows

$$u = \frac{f - v'(v' + h)}{u'} \text{ made monic,} \tag{4.1.1}$$

$$v = (-h - v') \pmod{u}, \tag{4.1.2}$$

but this is not followed literally by Harley. By using techniques such as Garner's Algorithm for the Chinese Remainder Theorem in the addition case and Newton iteration in the doubling case, values computed in the composition portion are reused in the reduction portion by polynomial manipulation tricks, resulting in Algorithms 23 and 24.

Algorithm 23 Genus 2 Harley Addition

```
Input: $[u_1, v_1]$, $[u_2, v_2]$
Output: $[u, v] = [u_1, v_1] + [u_2, v_2]$
1: $k = \frac{(f-hv_1-v_1^2)}{u_1}$
2: $s \equiv \frac{(v_2-v_1)}{u_1} \pmod{u_2}$
3: $l = su_1$
4: $u = \frac{(k-s(h+l+2v_1))}{u_2}$ made monic
5: $v \equiv -h - (l + v_1) \pmod{u}$
```

Algorithm 24 Genus 2 Harley Doubling

```
Input: $[u_1, v_1]$
Output: $[u, v] = 2[u_1, v_1]$
1: $k = \frac{(f-hv_1-v_1^2)}{u_1}$
2: $s \equiv \frac{k}{(h+2v_1)} \pmod{u_1}$
3: $l = su_1$
4: $u = s^2 - \frac{s(h+2v_1)-k}{u_1}$ made monic
5: $v \equiv -h - (l + v_1) \pmod{u}$
```

In more recent expositions of Harley's method, s is made monic at its creation rather than u made monic. Making s monic as soon as it is computed forces l and u to be monic at their creation and overall reduces the number of field operations required to compute monic u [24]. Note that $v' = l + v$ is the unreduced v' polynomial of the composed divisor class representative before reduction that interpolates the points of $u = u_1u_2$.

For genus 3 curves, an additional reduction step is necessary to achieve a reduced divisor class representation. Therefore, extending Harley's method to genus 3 curves requires two additional steps, identical in both addition and doubling.

Algorithm 25 Harley's Method Continued for Genus 3

```
  ...
6: $u = (f - v(v + h))/u$    (exact division).
7: $v = (-h - v) \pmod{u}$.
```

4.1.2 Linear Algebra Techniques in the Frequent Case

Recall that analogous to the elliptic curve chord and tangent method, divisor addition corresponds to the points in the support of the two input divisors, (counting multiplicity), being interpolated and intersected with the curve, resulting in new points corresponding to the support of the composed divisor. This is indeed true for Cantor's and Harley's methods if one considers the geometric meaning behind the Mumford representation as described in Section 2.3.1.

In the composition portion of Cantor's algorithm an alternate method that uses linear algebra i.e., solving a system of linear equations, can be used to compute the unreduced interpolating polynomial v'. This method has been presented for frequent case addition and doubling for arbitrary genus hyperelliptic curves described by a ramified model, where explicit formulas were developed based on this method for the genus 2 case [6]. In this section, a brief overview of the method in genus 2 is given. Explicit techniques in genus 2 utilized by this method are presented in Section 4.3.

Consider the input divisor classes represented geometrically as $[D_1] = (x_1, y_1) + (x_2, y_2)$ and $[D_2] = (x_3, y_3) + (x_4, y_4)$, where (x_i, y_i) are points in the respective supports on the ramified model hyperelliptic curve C. In the frequent case for addition, all four points are distinct as mentioned before. These four points uniquely determine a cubic polynomial v' that interpolates them. The new unreduced divisor class in Mumford representation is $[D'] = [u', v']$ where $u' = (x - x_1) \cdot (x - x_2) \cdot (x - x_3) \cdot (x - x_4)$. In the frequent case for doubling i.e.; $[D] = 2[D_1]$, the two points are not equal to each other's negative, and tangents of multiplicity two at the points are used to produce the cubic interpolating polynomial v'. The unreduced divisor class is $[D'] = [u', v']$ where $u' = (x - x_1)^2 \cdot (x - x_2)^2$.

Just as in Harley's method, $[D']$ is computed first through the "composition" step using linear algebra, and then Cantor's reduction step is used to compute the reduced representation $[D] \equiv [D']$. The composition step in both addition and doubling involves building a $2g$ by $2g$ linear system of equations, where the equations relate coefficients of v'. The system is then solved, producing the coefficients of v'. The highlight of the linear algebra method is that in the frequent cases the linear system of equations can be reduced to a g by g system. We give a brief overview on how to find the necessary equations in genus 2; more details can be found in [6].

For addition, let $v' = v'_3x^3 + v'_2x^2 + v'_1x + v'_0$ be the desired polynomial that interpolates the four finite points in the supports of $[D_1] = [x^2 + u_{11}x + u_{10}, v_{11}x + v_{10}]$ and $[D_2] = [x^2 + u_{21}x + u_{20}, v_{21}x + v_{20}]$ in Mumford representation. The identity

$$\begin{aligned} 0 &\equiv (v' - v) \pmod{u} \\ 0 &\equiv (v'_3x^3 + v'_2x^2 + v'_1x + v'_0 - v_{11}x - v_{10}) \pmod{x^2 + u_{11}x + u_{10}}, \qquad (4.1.3) \\ &= a_1x + a_0, \end{aligned}$$

where

$$a_1 = (v'_3(u_{11}^2 - u_{01}) - v'_2u_{11} + v'_1 - v_{11})$$

$$a_0 = (v'_3 u_{11} u_{10} - v'_2 u_{10} + v'_0 - v_{10})$$

provides two relations, ($a_1 = 0$ and $a_0 = 0$) relating the four coefficients of v' linearly. This identity comes from geometrically viewing both v' and v_1 as interpolating polynomials for the two points in the support of $[D_1]$. A similar construction is used with $[D_2]$ to produce two more linear equations. The resulting four by four linear system of equations is

$$\begin{pmatrix} 1 & 0 & -u_{10} & u_{11}u_{10} \\ 0 & 1 & -u_{11} & u_{11}^2 - u_{10} \\ 1 & 0 & -u_{20} & u_{21}u_{20} \\ 0 & 1 & -u_{21} & u_{21}^2 - u_{20} \end{pmatrix} \times \begin{pmatrix} v'_0 \\ v'_1 \\ v'_2 \\ v'_3 \end{pmatrix} = \begin{pmatrix} v_{10} \\ v_{11} \\ v_{20} \\ v_{21} \end{pmatrix}$$

Note that in the frequent input cases, the 4 by 4 system is reduced to a 2 by 2 system by subtracting the first and second row from the third and fourth row respectively. Only v'_2 and v'_3 are solved for in this manner, and v'_0 and v'_1 are instead found by equating coefficients of equation (4.1.3) above.

For doubling a divisor class $[u_1, v_1]$, equation (4.1.3) immediately provides two linear relations. There are two possible approaches for obtaining the other two relations. The first is matching derivatives

$$\frac{dy}{dx} = \frac{dv'}{dx}$$

point by point on the curve. This approach ensures multiplicity is accounted for and gives rise to two linear relations. The other way is to reduce the substitution of $v' = y$ into the curve C modulo u_1^2 (reduction modulo u_1^2 instead of u_1 ensures the zeros have multiplicity two) and then linearize the coefficients using relations that arise from substituting $v_1 = y$ into C and reducing modulo u_1. This approach results in four more equations. The second approach produces the simplest equations; details can be found in [6]. The relations are:

$$\begin{aligned} a_2 = {} & v'_3(-3h_1u_{10} - 6u_{10}v_{11}) + v'_2(h_2u_{11}^2 + 2h_2u_{10} - 2u_{11}v_{11} + h_0 + 2v_{10}) \\ & + v'_1(2h_1 + 4v_{11} - h_1u_{11} - 2h_2u_{11}) + v'_0(-3h_2) \\ & + 2h_2u_{11}v_{11} - 2u_{11}^3 - h_1v_{11} + 4h_2v_{10} - 2u_{11}u_{10} - 3v_{11}^2 - f_2, \\ a_3 = {} & v'_3(h_2u_{10} - 2h_1u_{11} - 4u_{11}v_{11} + h_0 + 2v_{10}) + v'_2(h_2u_{11} + h_1 + 2v_{11}) \end{aligned}$$

$$+ v'_1(-2h_2) + 3h_2 v_{11} - 3u_{11}^2 + 2u_{10} - f_3,$$

where $a_2 = a_3 = 0$. Similar to addition, a 4 by 4 system of equations can be set up from the 4 relations to solve for the coefficients of v' directly and reduce to a 2 by 2 system.

After $[D'] = [u', v']$ is computed via addition or doubling route, Cantor's reduction step

$$u = \frac{v'^2 + hv' - f}{u'}, \text{ made monic}$$
$$v = -h - v' \pmod{u}$$

is used to reduce D' to a reduced divisor class representative D. Recall that $su_1 + v_1 = v'$ where s is a sub-expression from Harley's Method. Relating Harley's method, the linear algebra method computes v' in a different way than computing $v' = su_1 + v_1$ but after those intermediate values are found, identical techniques can be used to compute the reduced divisor class $[D] = [u, v]$.

4.1.3 NUCOMP as Basis for Explicit Formulas

In this section, general methods based on NUCOMP in genus 2 and 3 are compared to previous methods in literature and presented as a basis for the development of explicit formulas in Chapters 5 and 6. The main benefits of using NUCOMP as the generic method is that NUCOMP captures Harley's techniques for both frequent and special cases, and the organization of steps in NUCOMP offers field operation savings over previous best when combined with certain explicit techniques described later in Section 4.3. Ramified model arithmetic is based on NUCOMP (Algorithm 19) and split model arithmetic on Balanced NUCOMP (Algorithm 21). First the approach for genus 2 is described, then genus 3.

Genus 2

NUCOMP-based algorithms are presented as the basis for all explicit formulas in Chapter 5 in order to provide the best approach that reduces computational complexity in both special cases and frequent cases, as well as unifies frequent and special cases into one algorithm. Versions of NUCOMP specialized to genus 2 addition and doubling on ramified models are presented as

Algorithms 27 and 26. Specialized versions Balanced NUCOMP for addition and doubling on split models are given in Algorithms 29 and 28.

Algorithm 26 Genus 2 Ramified Model Double (NUDUPL)

```
Input: $[u_1, v_1]$
Output: $[u_n, v_n] = 2[u_1, v_1]$
1: $k = (f - v_1(v_1 + h))/u_1$.   (exact division)
2: $(S, a_1, b_1) = \text{XGCD}(u_1, 2v_1 + h)$.
3: if $S \neq 1$ then
4:     $u_1 = u_1/S$.   (exact division)
5: $s = b_1 k \pmod{u_1}$.
6: if $\deg(u_1) \leq 1$ then
7:     $u_n = u_1^2$.
8:     $v_n = (v_1 + u_1 s) \pmod{u_n}$.
9: else
10:     $M_2 = (s(2v_1 + h) - k)/u_1$   (exact division).
11:     $u_n = s^2 + M_2$,   (made monic).
12:     $v_n = (-su_1 - v_1 - h) \pmod{u_n}$.
13: return $[u_n, v_n]$.
```

For genus 2 ramified models, improved Cantor's Addition, NUCOMP and Harley's method all result in similar formulations for frequent case input divisor classes that can be manipulated to be identical. Following the exact formulation from the NUCOMP-based Algorithms 27 and 26 performs the steps in such a way that when combining some previous explicit techniques with a novel one from this work (Section 4.3), field operation savings over previous best are introduced. Notice the composition portions of both Algorithms 4 and 19 absorb one reduction, and thus in genus 2, neither the reduction nor continued fraction loop is ever entered. Moreover, both algorithms are more general than Harley's method and include improved special case computations, where improved Cantor's addition and NUCOMP compute special cases identically.

Algorithm 27 Genus 2 Ramified Model Addition (NUCOMP)

```
Input: $[u_1, v_1]$, $[u_2, v_2]$ where $[u_1, v_1] \neq [u_2, v_2]$ and $\deg(u_2) \leq \deg(u_1)$.
Output: $[u_n, v_n] = [u_1, v_1] + [u_2, v_2]$
 1: $k = (f - v_1(v_1 + h))/u_1$.   (exact division)
 2: $(S, a_1, b_1) = \text{XGCD}(u_1, u_2)$.   $(S, a_1$ only)
 3: if $S \neq 1$ then
 4:     $(S', a_2, b_2) = \text{XGCD}(S, v_2 + v_1 + h)$.
 5:     if $S' \neq 1$ then
 6:         $u_1 = u_1/S'$,  $u_2 = u_2/S'$.   (exact division)
 7:         $s = (a_2a_1(v_2 - v_1) + b_2k) \pmod{u_2}$.
 8:     else
 9:         $s = (a_2a_1(v_2 - v_1) + b_2k) \pmod{u_2}$.
10: else
11:     $s = a_1(v_2 - v_1) \pmod{u_2}$.
12: if $\deg(u_2) + \deg(u_1) \leq 2$ then
13:     $u_n = u_2u_1$.
14:     $v_n = (v_1 + u_1s) \pmod{u_n}$.
15: else
16:     $\tilde{v} = -su_1 - v_1 - h$.
17:     $u_n = (s(\tilde{v} - v_1) + k)/u_2$   (exact division, made monic).
18:     $v_n = \tilde{v} \pmod{u_n}$.
19: return $[u_n, v_n]$.
```

Over split models, Balanced NUCOMP (Algorithm 21) yields similar formulations to the improved Balanced Addition (Algorithm 15) and Harley's techniques in the frequent cases that can also be manipulated to be identical. Similarly to the ramified setting, Balanced NUCOMP and improved Balanced Addition produce identical formulations for special cases that do not require adjustments. If adjustments are required, Balanced NUCOMP yields greatly simplified formulations over improved Balanced Addition. This can be seen by recalling that for genus 2 curves described by a split model, at most one adjustment step is required, and since Balanced NUCOMP absorbs up to one adjustment, the use of Balanced Adjust (Algorithm 7) can be completely omitted. Balanced NUCOMP incorporates the adjustment without computing the composed divisor class representative first, where in contrast improved Balanced Addition first computes the composition and reduction then calls Algorithm 7 for the final adjustment, requiring two field inversions.

Algorithm 28 Genus 2 Split Model Double (Positive Reduced Balanced NUDUPL)

```
Input: $[u_1, v_1, n_1]$
Output: $[u_n, v_n, n_n] = 2[u_1, v_1, n_1]$
 1: if $\deg(u_1) \leq 1$ and $n_1 = 0$ then
 2:     $v_1 = -V^+ - h - [-(V^+ - h - v_1) \pmod{u_1}]$.
 3: $k = (f - v_1(v_1 + h))/u_1$.   (exact division)
 4: $(S, a_1, b_1) = \mathrm{XGCD}(u_1, 2v_1 + h)$.
 5: $n_n = 2n_1 + \deg(S) - 1$.
 6: if $S \neq 1$ then
 7:     $u_1 = u_1/S$.   (exact division)
 8: $s = b_1 k \pmod{u_1}$.
 9: $D = 2\deg(u_1)$.
10: if $D \leq 2$ and $(n_n \geq 0$ and $n_n \leq 2 - D)$ then
11:     $u_n = u_1^2$.
12:     $v_n = V^+ - [(V^+ - v_1 - u_1 s) \pmod{u_n}]$.
13: else
14:     $M_2 = (s(2v_1 + h) - k)/u_1$   (exact division).
15:     $u_n = s^2 + M_2$   (made monic).
16:     $z = -u_1 s$.
17:     $v_n = V^+ - [(V^+ - z + v_1 + h) \pmod{u_n}]$.
18:     if $\deg(z) < 3$ then
19:         if $\mathrm{lcf}(v_1) = \mathrm{lcf}(V^+)$ then   $n_n = n_n + D - 3$ .
20:         else                              $n_n = n_n + 3 - \deg(u_n)$.
21:     else $n_n = n_n + \deg(u_1) + \deg(s) - \deg(u_n)$.
22: return $[u_n, v_n, n_n]$.
```

Genus 3

NUCOMP-based algorithms are presented as the basis for all explicit formulas in Chapter 6 in order to provide the best approach that unifies frequent and special cases. Previous work in genus 3 ramified and split model explicit formulas did not consider NUCOMP [10, 32, 37], and therefore produced considerably slower explicit formula algorithms in all cases when compared to this work. Moreover, for genus 3 split models, the computational complexity of the frequent cases, and similarly special cases that require adjustments, greatly improve via the continued fraction step. Specialized versions of NUCOMP for genus 3 addition and doubling on ramified models are presented in Algorithms 31 and 30. Specialized versions of Balanced NUCOMP for genus 3 addition and doubling on split models are presented in Algorithms 33 and 32.

For genus 3 ramified models two reduction steps are required in the frequent cases, therefore

Algorithm 29 Genus 2 Split Model Addition (Positive Reduced Balanced NUCOMP)

```
Input: $[u_1, v_1, n_1]$, $[u_2, v_2, n_2]$ where $[u_1, v_1, n_1] \neq [u_2, v_2, n_2]\deg(u_2) \leq \deg(u_1)$.
Output: $[u_n, v_n, n_n] = [u_1, v_1, n_1] + [u_2, v_2, n_2]$
 1: if $\deg(u_2) \leq 1$ and $n_1 = n_2 = 0$ then
 2:     $v_1 = -V^+ - h - [(-V^+ - h - v_1) \pmod{u_1}]$.
 3: $(S, a_1, b_1) = \mathrm{XGCD}(u_1, u_2)$.   $(S, a_1$ only)
 4: $s = (a_1(v_2 - v_1)) \pmod{u_2}$.
 5: $k = (f - v_1(v1 + h))/u_1$.   (exact division)
 6: if $S \neq 1$ then
 7:     $(S', a_2, b_2) = \mathrm{XGCD}(S, v_2 + v_1 + h)$.
 8:     if $S' \neq 1$ then
 9:         $u_1 = u_1/S$,  $u_2 = u_2/S$.   (exact division)
10:         $s = (a_2 a_1(v_2 - v_1) + b_2 k) \pmod{u_2}$.
11:     else
12:         $s = (a_2 a_1(v_2 - v_1) + b_2 k) \pmod{u_2}$.
13: else
14:     $s = a_1(v_2 - v_1) \pmod{u_2}$.
15: $n_n = n_1 + n_2 + \deg(S) - 1$.
16: $D = \deg(u_2) + \deg(u_1)$.
17: if $D \leq 2$ and $((0 \leq n_n \leq g - D)$ then
18:     $u_n = u_2 u_1$.
19:     $v_n = V^+ - [(V^+ - v_1 - u_1 s) \pmod{u_n}]$.
20: else
21:     $z = -u_1 s$.
22:     $\tilde{v} = z - v_1 - h$.
23:     $u_n = (s(\tilde{v} - v_1) + k)/u_2$   (exact division, made monic).
24:     $v_n = V^+ - [(V^+ - \tilde{v}) \pmod{u_n}]$.
25:     if $\deg(z) < 3$ then
26:         if $\mathrm{lcf}(v_1) = \mathrm{lcf}(V^+)$ then   $n_n = n_n + D - 3$ .
27:         else                                 $n_n = n_n + 3 - \deg(u_n)$.
28:     else  $n_n = n_n + \deg(u_1) + \deg(s) - \deg(u_n)$.
29: return $[u_n, v_n, n_n]$.
```

Algorithm 30 Genus 3 Ramified Model Double

```
Input: $[u_1, v_1]$
Output: $[u_n, v_n] = 2[u_1, v_1]$.
 1: $k = (f - v_1(v_1 + h))/u_1$.   (exact division)
 2: $(S, a_1, b_1) = \mathrm{XGCD}(u_1, 2v_1 + h)$.
 3: if $S \neq 1$ then
 4:     $u_1 = u_1/S$.   (exact division)
 5: $s = b_1 k \pmod{u_1}$.
 6: if $2\deg(u_1) \leq 3$ then
 7:     $u_n = u_1^2$.
 8:     $v_n = (v_1 + u_1 s) \pmod{u_n}]$.
 9: else if $\deg(s) < 2$ then
10:     $M_2 = (s(2v_1 + h) - kS)/u_1$   (exact division).
11:     $u_n = s^2 + M_2$   (made monic).
12:     $z = -u_1 s$.
13:     $v_n = (z - v_1 - h) \pmod{u_n}$.
14: else
15:     $(q, r) = \mathrm{DivRem}(u_1, s)$.
16:     $M_2 = (r(v_1 + v_1 + h) + kq)/u_1$   (exact division).
17:     $u_n = r^2 - qM_2$.
18:     $z = (u_1 r - u_n)/q$   (exact division).
19:     $u_n = \mathrm{monic}(u_n)$.
20:     $v_n = (z - v_1 - h) \pmod{u_n}$.
21: return $[u_n, v_n]$.
```

in contrast to the genus 2 setting, the improved Cantor and NUCOMP algorithms compute one additional reduction in their reduction and continued fraction loops, respectively. Harley's method has been similarly adapted to genus 3 by computing an additional reduction step via Algorithm 25. Improved Cantor's addition and Harley's method produce equivalent frequent-case explicit formulas that can be found in previous best work [10]. On the other hand, the second reduction step is vastly different in NUCOMP, and the explicit formulas that arise from NUCOMP on genus 3 curves are simpler. Note that explicit formulas on genus 3 curves described by a ramified model enjoy similar benefits from NUCOMP, but are omitted in this thesis due to such formulas being work in progress elsewhere.

Over split models, the frequent case for both addition and doubling requires an adjustment for the final reduction from degree $g + 1$. The specialized formulation of Balanced NUCOMP absorbs this adjustment, and similarly the first of any other adjustments in special cases. Similar to the genus

Algorithm 31 Genus 3 Ramified Model Addition

```
Input: $[u_1, v_1]$, $[u_2, v_2]$ where $\deg(u_2) \leq \deg(u_1)$.
Output: $[u_n, v_n] = [u_1, v_1] + [u_2, v_2]$
 1: $k = (f - v_1(v_1 + h))/u_1$.   (exact division)
 2: $(S, a_1, b_1) = \text{XGCD}(u_1, u_2)$.   $(S, a_1$ only)
 3: if $S \neq 1$ then
 4:     $(S, a_2, b_2) = \text{XGCD}(S, v_2 + v_1 + h)$.
 5:     if $S' \neq 1$ then
 6:         $u_1 = u_1/S'$,  $u_2 = u_2/S'$.   (exact division)
 7:     $s = (a_2 a_1 (v_2 - v_1) + b_2 k) \pmod{u_2}$.
 8: else
 9:     $s = a_1(v_2 - v_1) \pmod{u_2}$.
10: if $\deg(u_1) + \deg(u_2) \leq 3$ then
11:     $u_n = u_2 u_1$.
12:     $v_n = (v_1 + u_1 s) \pmod{u_n}]$.
13: else if $\deg(s) \leq 2$ then
14:     $z := -u_1 s$,  $\tilde{v} = z - v_1 - h$.
15:     $u_n = (s(\tilde{v} - v_1) + kS)/u_2$  . (exact division, made monic)
16:     $v_n = \tilde{v} \pmod{u_n}$.
17: else
18:     $(q, r) = \text{DivRem}(u_2, s)$.
19:     $M_1 = (q(v_2 - v_1) + r u_1)/u_2$.   (exact division).
20:     $M_2 = (r(v_2 + v_1 + h) + qk)/u_2$.   (exact division).
21:     $u_n = r M_1 - q M_2$.
22:     $z = (u_1 r - u_n)/q$   (exact division).
23:     $u_n = \text{monic}(u_n)$.
24:     $v_n = (z - v_1 - h) \pmod{u_n}$.
25: return $[u_n, v_n]$
```

2 setting, much simpler explicit formulas relative to Balanced Cantor's Addition are achieved by omitting one call to Balanced Adjust.

Discussion

A few notes on the specialized genus 2 and 3 algorithms. For both ramified and split models, the extended Mumford representation used for generic genus divisor class arithmetic from Section 3.1.1 provides no benefit to the development of explicit formulas and is not used. In the addition algorithms, the assumption that the input divisor classes are not equal is made in order to align with explicit formulas in the next chapter. For the balanced setting, a positive reduced basis is used for

Algorithm 32 Genus 3 Split Model Double (Negative Reduced Balanced NUDUPL)

Input: $[u_1, v_1, n_1]$

Output: $[u_n, v_n, n_n] = 2[u_1, v_1, n_1]$.

```
1: if $\deg(u_1) \leq 2$ and $n_1 = 3 - \deg(u_1)$ then
2:     $v_1 = -V^- - h - [(-V^- - h - v_1) \pmod{u_1}]$.
3: $k = (f - v_1(v_1 + h))/u_1$.   (exact division)
4: $(S, a_1, b_1) = \mathrm{XGCD}(u_1, 2v_1 + h)$.
5: $n_n = 2n_1 + \deg(S) - 2$.
6: if $S \neq 1$ then
7:     $u_1 = u_1/S$.   (exact division)
8: $s = b_1 k \pmod{u_1}$.
9: $D = 2\deg(u_1)$.
10: if $D \leq 3$ and $(0 \leq n_n \leq 3 - D)$ then
11:     $u_n = u_1^2$.
12:     $v_n = V^- - [(V^- - v_1 - u_1 s) \pmod{u_n}]$.
13: else if $\deg(s) < 2$ then
14:     $M_2 = (s(2v_1 + h) - kS)/u_1$   (exact division).
15:     $u_n = s^2 + M_2$   (made monic).
16:     $z = -u_1 s$.
17:     $v_n = V^- - [(V^- - z + v_1 + h) \pmod{u_n}]$.
18:     if $\mathrm{lcf}(v_1) = \mathrm{lcf}(-V^- - h)$ then   $n_n = n_n + D - 4$.
19:     else
20:         if $\deg(z) < 4$ then   $n_n = n_n + 4 - \deg(u_n)$
21:         else                    $n_n = n_n + \deg(u_1) - \deg(s)$.
22:         if $n_n < 0$ then
23:             $u_n = (f - v_n(v_n + h)/u_n)$   (exact division, made monic)
24:             $v_n = V^- - [(V^- + v_n + h) \pmod{u_n}]$.
25:             $n_n = n_n + 4 - \deg(u_n)$.
26: else
27:     $(q, r) = \mathrm{DivRem}(u_1, s)$.
28:     $M_2 = (r(v_1 + v_1 + h) + kq)/u_1$   (exact division).
29:     $u_n = r^2 - qM_2$.
30:     $z = (u_1 r - u_n)/q$   (exact division).
31:     $u_n = \mathrm{monic}(u_n)$.
32:     $v_n = V^- - [(V^- - z + v_1 + h) \pmod{u_n}]$.
33:     if $\deg(z) < 4$ then   $n_n = n_n + \deg(u_1) - \deg(s) + 4 - \deg(u_n)$.
34:     else                    $n_n = n_n + \deg(u_1) - \deg(r)$.
35: return $[u_n, v_n, n_n]$.
```

Algorithm 33 Genus 3 Split Model Addition (Negative Reduced Balanced NUCOMP)

```
Input: $[u_1, v_1, n_1]$, $[u_2, v_2, n_2]$ where $\deg(u_2) \le \deg(u_1)$.
Output: $[u_n, v_n, n_n] = [u_1, v_1, n_1] + [u_2, v_2, n_2]$
 1: if $\deg(u_1) + \deg(u_2) \le 4$ and $n_1 = 3 - \deg(u_1)$ and $n_2 = 3 - \deg(u_2)$ then
 2:     $v_1 = -V^- - h - [(-V^- - h - v_1) \pmod{u_1}]$.
 3: $k = (f - v_1(v_1 + h))/u_1$.   (exact division)
 4: $(S, a_1, b_1) = \mathrm{XGCD}(u_1, u_2)$.   $(S, a_1$ only)
 5: if $S \ne 1$ then
 6:     $(S', a_2, b_2) = \mathrm{XGCD}(S, v_2 + v_1 + h)$.
 7:     if $S' \ne 1$ then
 8:         $u_1 = u_1/S'$,  $u_2 = u_2/S'$.   (exact division)
 9:     $s = (a_2 a_1(v_2 - v_1) + b_2 k) \pmod{u_2}$.
10: else
11:     $s = a_1(v_2 - v_1) \pmod{u_2}$.
12: $n_n = n_1 + n_2 + \deg(S) - 2$.
13: $D = \deg(u_2) + \deg(u_1)$.
14: if $D \le 3$ and $(0 \le n_n \le 3 - D)$ then
15:     $u_n = u_2 u_1$.
16:     $v_n = V^- - [(V^- - v_1 - u_1 s) \pmod{u_n}]$.
17: else if $\deg(s) \le 2$ then
18:     $z := -u_1 s$,  $\tilde{v} = z - v_1 - h$.
19:     $u_n = (s(\tilde{v} - v_1) + kS)/u_2$  . (exact division, made monic)
20:     $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
21:     if  $\mathrm{lcf}(v_1) = \mathrm{lcf}(-V^- - h)$ then  $n_n = n_n + D - 4$.
22:     else
23:         if $\deg(z) < 4$ then  $n_n = n_n + 4 - \deg(u_n)$
24:         else                   $n_n = n_n + \deg(u_2) - \deg(s)$.
25:         if $n_n < 0$ then
26:             $u_n = (f - v_n(v_n + h)/u_n)$   (exact division, made monic)
27:             $v_n = V^- - [(V^- + v_n + h) \pmod{u_n}]$.
28:             $n_n = n_n + 4 - \deg(u_n)$.
29: else
30:     $(q, r) = \mathrm{DivRem}(u_2, s)$.
31:     $M_1 = (q(v_2 - v_1) + r u_1)/u_2$.   (exact division).
32:     $M_2 = (r(v_2 + v_1 + h) + qk)/u_2$.   (exact division).
33:     $u_n = r M_1 - q M_2$.
34:     $z = (u_1 r - u_n)/q$   (exact division).
35:     $u_n = \mathrm{monic}(u_n)$.
36:     $v_n = V^- - [(V^- - z + v_1 + h) \pmod{u_n}]$.
37:     if $\deg(z) < 4$ then  $n_n = n_n + \deg(u_2) - \deg(s) + 4 - \deg(u_n)$.
38:     else                   $n_n = n_n + \deg(u_2) - \deg(r)$.
39: return $[u_n, v_n, n_n]$
```

developing explicit formulas in genus 2 and therefore presented here. There is no advantage to using a negative reduced basis in genus 2 and working with a positive reduced basis aligns with previous work [9]. In genus 3, a negative reduced basis is advantageous when paired with Balanced NUCOMP and therefore is used.

Recall from the exposition of Balanced NUCOMP in Section 3.3 that the choice of reduced basis dictates the adjustment direction. In some special cases, the choice to keep input and output divisor classes in negative (or positive) reduced basis unnecessarily requires extra adjustments if an adjustment in the opposite direction is required. In genus 2 and 3, no adjustments are ever required if $\deg(S) > 0$ (but this is not true for $g \geq 4$), so one can accurately check for base change special cases with out any knowledge of S. The explicit formulas in Chapter 5 and Chapter 6 take advantage of this, and efficiently determine and apply a change of basis when necessary. Split model genus 2 and 3 divisor class Addition and Doubling (Algorithms 29, 28,33 and 32) are presented to reflect this improvement.

4.2 Explicit Formulations of Divisor Class Arithmetic

Explicit formulas are algorithms for divisor class arithmetic described by field operations instead of polynomial arithmetic. Unnecessary operations inherent in the polynomial arithmetic are eliminated and techniques that further reduce the required number of field operations can be used. The first step of converting an algorithm based on polynomial arithmetic to an explicit formula is to consider polynomials as sets of their respective coefficients, i.e., elements of the base field the polynomial is defined over. Polynomial operations then can be converted to finite field element operations by considering what happens to the coefficients of the polynomial during the polynomial operation.

In practice, reduced divisor classes are represented in affine or projective coordinates. In affine coordinates, a divisor class $[D] = [u, v]$ is represented by $[u_{g-1}, ..., u_0, v_{g-1}, ..., v_0]$, where $u = x^g + u_{g-1}x^{g-1} + ... + u_0 \in k[x]$ and $v = v_{g-1}x^{g-1} + ... + v_0 \in k[x]$, with the addition of the balancing coefficient n over split models. Similarly in either positive or negative reduced basis, $[D] = [u, v]$ is represented by $[u_{g-1}, ..., u_0, v_{g+1}, ..., v_0]$, where $u = x^g + u_{g-1}x^{g-1} + ... + u_0 \in k[x]$ and

$v = v_{g+1}x^{g+1} + \ldots + v_0 \in k[x]$. The resulting addition and doubling algorithms require at least one field inversion. In projective coordinates the same divisor is represented by $[u'_{g-1}, \ldots, u'_0, v'_{g-1}, \ldots, v'_0, z]$, or by $[u'_{g-1}, \ldots, u'_0, v'_{g+1}, \ldots, v'_0, z]$ for reduced basis where $u_i = u'_i/z$, $v_i = v'_i/z$, etc. The extra coordinate z keeps track of values that would have been inverted and multiplied through in the affine setting, enabling the development of arithmetic formulas requiring no field inversions. For genus 2 curves described by a ramified model, more complicated projective coordinates requiring additional auxiliary field elements have been shown to produce efficient explicit formulas in certain cases [16].

Going forward, divisor classes represented in projective coordinates are not considered. In applications such as numerical experiments related to Sato-Tate distributions [36], a primary motivation for the genus 2 and 3 explicit formulas of this thesis, divisor class arithmetic occurs as part of a baby-steps giant-steps search in which field inversions can be easily combined by taking steps in parallel [22, § 4.1]. The cost of an inversion is reduced to about 3 field multiplications, making projective coordinate divisor class arithmetic inadequate because significantly more than 3 field multiplications are required to eliminate the inversion. Furthermore, as representations of group elements using projective coordinates are not unique, their use would incur a non-negligible computational cost per equality test, and precludes the use of more efficient searchable data structures for the baby steps such as hash tables.

4.3 Techniques to Improve Explicit Formulas

Previous works such as in [24] and [6] for genus 2 and [10] for genus 3, present improved divisor class arithmetic algorithms described in terms of explicit formulas based on the Harley and linear algebra divisor class addition. These algorithms use explicit versions of the Chinese remainder theorem, Newton iteration, Karatsuba multiplication and reduction, Montgomery's inversion trick, Cramer's rule for solving systems of linear equations, and the re-use of previously-computed sub-expressions, in order to minimize the number of field multiplications required.

The frequent case explicit formulas presented in Chapters 5 and 6 of this thesis use some of the techniques previously applied and also introduce some new techniques. In the genus 2 formulas of this work, the composition portion of addition (Algorithms 27 and 29) and doubling (Algorithms 26 and 28) computes s by solving a linear system of equations, borrowed from the linear algebra method. Then the reduction part is computed via steps produced by specializing NUCOMP to the appropriate settings. Note that applying the equating coefficients trick (described in Section 4.3.9) as done in the linear algebra method [6] produces the same explicit formulas in the frequent case, where computations from solving the linear system of equation are reused.

Special cases are generally simpler and some of the techniques for frequent case explicit formulas may not be required. A case by case analysis is provided in Chapters 5 and 6 where the basic formulations of the novel explicit formulas and techniques used in each case are described for genus 2 in Chapter 5 and for genus 3 in Chapter 6. In the following, techniques used in previous works are described, beginning with a discussion on efficient field operations trades.

4.3.1 Field Operation Trade Threshold

There are many suggestions for the relative cost of a field addition compared to a field multiplication in previous works. These suggestions generally rely on properties of specific base fields over which the hyperelliptic curve is defined. The explicit formulas in Chapters 5 and 6 are designed to work efficiently over many fields, as is required for example, in applications such as numerical investigation of Sato-Tate distributions [36]. Aligning with the work of Sutherland [37, 36], the formulas in Chapters 5 and 6 never trade 1 field multiplication for more than 3 field additions.

4.3.2 Efficient Exact Division

As described in [39] for example, the exact quotient of two polynomials (where one is known to divide the other) can be computed more efficiently by observing that computing the quotient of two polynomials of degree d_1 and d_2 with $d_1 > d_2$ only depends on the $d_1 - d_2 + 1$ highest coefficients of the dividend. So all the coefficients of the polynomials do not need to be considered. This observation is a perfect example of why explicit formulas given by field operations are more efficient than formulas based on polynomial arithmetic, as not all coefficients need to computed in every step.

In almost all input cases for genus 2 and 3 curves with ramified and split models, the polynomial $k = (f - v(v+h))/u$ with degree d_1 for Mumford coefficients u, v is computed, where u is a monic degree d_2 polynomial. Only the highest $d_1 - d_2 + 1$ coefficients of $f - v(v+h)$ are required to compute the quotient.

4.3.3 Karatsuba Multiplication

In 1963 Karatsuba introduced a novel algorithm for multiplication of polynomials [21]. Compared to the schoolbook method, the Karatsuba method saves field multiplications at the cost of extra field additions. Polynomial multiplication occurs regularly in divisor class addition using Mumford representation, so field operations are saved by taking advantage of this technique. Karatsuba multiplication can be applied, often more than once, to multiplying polynomials of any degree, and is used every time a multiplication of polynomials is required in the explicit formulas. Next, an example of multiplying two linear polynomials is provided.

Given two polynomials $A(x) = a_1x + a_0$ and $B(x) = b_1x + b_0$, let $D_0 = a_0b_0$, $D_1 = a_1b_1$, and $D_{0,1} = (a_0 + a_1)(b_0 + b_1)$. The product of $A(x)$ and $B(x)$ can be given as

$$C(x) = D_1x^2 + (D_{0,1} - D_1 - D_0)x + D_0.$$

Notice that

$$\begin{aligned}(D_{0,1} - D_1 - D_0) &= (a_0 + a_1)(b_0 + b_1) - a_1b_1 - a_0b_0 \\ &= a_0b_0 + a_0b_1 + a_1b_0 + a_1b_1 - a_1b_1 - a_0n_0 \\ &= a_0b_1 + a_1b_0,\end{aligned}$$

as required by the schoolbook method. In this case one multiplication is traded for three additions.

For genus 2, the largest polynomial degree multiplication required that is not part of an exact division is a degree 2 monic polynomial multiplied with a linear polynomial. This effectively is the same as multiplying two linear polynomials. For genus 3 curves described by a split model, the largest degree multiplication is a monic degree 3 polynomial with a degree 2 polynomial.

The linear polynomial technique can be applied recursively to higher degree multiplications. Recursive applications are not applied directly in the explicit formulas of this work and instead all instances of the field multiplication patterns

$$ac, \quad ad+bc, \quad bd,$$

for field elements a, b, c, d are identified and replaced with

$$ac, \quad bd, \quad (a+b)(c+d)-ac-bd.$$

For genus 3 with a split model, this trade can occur up to two times for one polynomial multiplication. See for example, the computation of $s' = qk \pmod{u}$ in Section 4.3.6, where both q and k are non-monic degree 2 polynomials.

4.3.4 Karatsuba Modular Reduction

The field multiplication pattern described in Section 4.3.3, where $ac, bd, ad+bc$, for field elements a, b, c, d is traded for $ac, bd, (a+b)(c+d)-ac-bd$, applies to modular reduction of a polynomial by a monic modulus with one degree difference, trading 1 multiplication for three additions.

This trade is applied in every instance of a modular reduction by a monic modulus where the difference in degree is 1 in the explicit formulas. For genus 2 curves, a polynomial of degree three modulo a monic polynomial of degree two is required in the addition and doubling formulations to obtain the reduced divisor class representative. For $v_1x+v_0 = l_3x^3+l_2x^2+l_1x+l_0 \pmod{x^2+u_1x+u_0}$, the schoolbook method results in

$$t = l_2 - u_1l_3$$
$$v_1 = l_1 - u_0l_3 - u_1t$$
$$v_0 = l_0 - u_0t.$$

Applying, Karatsuba's technique results in

$$t_1 = u_1l_3, \quad t_2 = l_2 - t_1, \quad t_3 = u_0t_2$$

$$
\begin{aligned}
v_1 &= l_1 - (u_0 + u_1)(l_3 + t_2) + t_1 + t_3 \\
v_0 &= l_0 - t_3,
\end{aligned}
$$

trading 1 multiplication for 3 additions.

For genus 3 curves, a polynomial of degree four modulo a monic polynomial of degree three is required in the addition and doubling formulations to obtain the reduced divisor class representative. For $v_2x^2 + v_1x + v_0 = l_4x^4 + l_3x^3 + l_2x^2 + l_1x + l_0 \pmod{x^3 + u_2x^2 + u_1x + u_0}$, the schoolbook method results in

$$
\begin{aligned}
t &= l_3 - u_2l_4 \\
v_2 &= l_2 - u_1l_4 - u_2t \\
v_1 &= l_1 - u_0l_4 - u_1t \\
v_0 &= l_0 - u_0t.
\end{aligned}
$$

There are two possibilities for applying the Karatsuba technique, either to u_2l_4, u_1l_4, u_2t and u_1t or to u_1l_4, u_0l_4, u_1t and u_0t. Applying Karatsuba's technique to both is not possible as one of the multiplications omitted in one choice is always required in the other. For the explicit formulas, the second choice is chosen as follows

$$
\begin{aligned}
t_0 &= l_3 - u_2l_4, \quad t_1 = u_1l_4, \quad t_2 = u_0t_0 \\
v_2 &= l_2 - t_1 - u_2t_0 \\
v_1 &= l_1 - (u_0 + u_1)(l_4 + t) + t_1 + t_2 \\
v_0 &= l_0 - t_2,
\end{aligned}
$$

trading 1 multiplication for 3 additions.

4.3.5 Using 'Almost Inverse'

The computation of the s polynomial in all divisor class addition and doubling algorithms presented in Section 4.1.3 requires a polynomial inversion modulo another polynomial.

In the case that the modulus has degree 1, (required for special cases of both genus 2 and 3 explicit formulas), the resulting s polynomial has degree zero. The modular reduction is taken first and then the inversion is computed. For example, consider degree 1 addition for genus 2 curves described by a ramified model

$$s = (v_2 - v_1)/u_1 \pmod{u_2},$$

where $[u_1, v_1]$ and $[u_2, v_2]$ are the input divisor classes with $\deg(u_1) = \deg(u_2) = 1$ and $\deg(v_1) = \deg(v_0) = 0$. The value $d = u_1 \pmod{u_2}$ is computed, followed by $w = d^{-1}$ and then $s = w(v_2 - v_1) \pmod{u_2}$.

In the case that the modulus has degree 2 (or 3), the resulting s polynomial has degree one (or two) and more sophisticated techniques are required to efficiently compute the polynomial inversion. Let $A(x)$ and $B(x)$ be two polynomials. In [39], the authors note that finding $A(x)^{-1} \bmod B(x)$ can be achieved by computing the resultant $r = \mathrm{res}(A(x), B(x))$ via the determinant of a Sylvester matrix M. Once the determinant is found, Cramer's rule is used to solve for the coefficients of the polynomial $r/A(x) \pmod{B(x)}$ called the *almost inverse* of $A(x) \pmod{B(x)}$.

In previous works for genus 2 [24] and 3 [10, 37], the division by r is not computed leaving the result as $r/A(x)$. The extra copy of r is removed later on through Montgomery's inversion trick (Section 4.3.7). The other benefit of using this method is that if r is non-zero, then there are no common factors between the two polynomials as described in [39], therefore testing for $r = 0$ can be used to flag special cases.

Next, the process for computing the resultant r and $r/A(x) \pmod{B(x)}$ in genus 2 is described. Let $A(x) = a_1x + a_0$ and $B(x) = x^2 + b_1x + b_0$, where $\gcd(A(x), B(x)) = 1$. The usual way to compute the inverse of $A(x)$ modulo $B(x)$ is to use the extended Euclidean algorithm to solve

$$S(x)B(x) + T(x)A(x) = 1$$

for some polynomials $T(x) = t_1x + t_0$, $S(x) = s_0$ (where $S(x)$ is not needed, and therefore the computation omitted), then $A^{-1}(x) = T(x) \pmod{B(x)}$. Instead, to compute the almost inverse $rT(x) \pmod{B(X)}$, we first compute the resultant of $A(x)$ and $B(x)$. Let M be the Sylvester matrix of $A(x)$ and $B(x)$ whose entries are coefficients of the two polynomials with dimensions $(\deg A + \deg B) \times (\deg A + \deg B)$. If $\deg A = n$ and $\deg B = m$, then in general,

$$M = \begin{pmatrix}
b_n & & & & a_m & & & & & \\
b_{n-1} & b_n & & & a_{m-1} & a_m & & & & \\
\vdots & b_{n-1} & \ddots & & \vdots & \vdots & \ddots & & & \\
\vdots & \vdots & & b_n & a_1 & \vdots & & \ddots & & \\
\vdots & \vdots & & b_{n-1} & a_0 & \vdots & & & \ddots & \\
\vdots & \vdots & & \vdots & & a_0 & & & & a_m \\
b_0 & \vdots & & \vdots & & & \ddots & & & \vdots \\
 & b_0 & & \vdots & & & & \ddots & & \vdots \\
 & & \ddots & \vdots & & & & & \ddots & \vdots \\
 & & & b_0 & & & & & & a_0
\end{pmatrix}.$$

In our example, the Sylvester matrix of A and B is

$$M = \begin{pmatrix} 1 & a_1 & 0 \\ b_1 & a_0 & a_1 \\ b_0 & 0 & a_0 \end{pmatrix},$$

and the resultant $r = \det(M)$. Solving

$$\begin{pmatrix} 1 & a_1 & 0 \\ b_1 & a_0 & a_1 \\ b_0 & 0 & a_0 \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ t_1 \\ t_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix},$$

yields t_1, t_0, s_0, the Bezout coefficients one would acquire from using the extend Euclidean algorithm.

The most efficient way to find a solution to this system of equations is applying Cramer's rule. The values of t_1, t_0 can be found as

$$t_1 = \frac{\det M'}{\det M}, \text{ and } t_0 = \frac{\det M''}{\det M},$$

where

$$M' = \begin{pmatrix} 1 & 0 & 0 \\ b_1 & 0 & a_1 \\ b_0 & 1 & a_0 \end{pmatrix}, \text{ and } M'' = \begin{pmatrix} 1 & a_1 & 0 \\ b_1 & a_0 & 0 \\ b_0 & 0 & 1 \end{pmatrix}.$$

Instead of taking the inverse of $\det(M) = r$ and multiplying through to solve for t_1 and t_0, let

$$t_1' = \det(M') \text{ and } t_0' = \det(M'').$$

Then $t_1' = rt_1$, $t_0' = rt_0$ and $T'(x) = r/A(x) \pmod{B(x)}$, the almost inverse of $A(x) \pmod{B(x)}$. The only computations required are those involved in computing the determinant of M, as computing the determinant of M' and M'' reuses computations.

In summary, the resultant $\det(M) = r$ is computed first as

$$r = b_0a_1 + a_0(a_0 - b_1a_1).$$

Then the almost inverse of $A(x) \pmod{B(x)}$ with weight (as described in Section 4.3.7) of r is computed as

$$t_1' = t_1r = -a_1,$$
$$t_0' = t_0r = a_0 - a_1b_1,$$

resulting in $T' = t_1'x + t_0' = rT = rA(x) \pmod{B(x)}$.

For genus 3, the approach is identical and requires the use of matrices with larger dimensions. The explicit formulas in this work do not follow this approach directly, although the resulting "almost inverse" is the same. The approach used in this work is described next in Section 4.3.6.

4.3.6 Alternate Approach to Computing s

In previous work on Addition and Doubling algorithms, a resultant is computed as part of the 'almost' inverse technique (Section 4.3.5), followed by a Karatsuba multiplication resulting in the s polynomial.

In genus 2, the author of [1] instead sets up a system of equations for the coefficients of s and then uses Cramer's rule to solve for s. This approach does not save field operations on its own, but saves one multiplication when combined with the novel technique introduced in Section 4.4.1. Note that the technique is not the same as the linear algebra approach for computing the intermediate interpolating v' polynomial, nor the almost inverse technique where the resultant r that s is comprised of is computed by computing the determinant of a Sylvester matrix. Linear algebra is used to compute s, one of the sub-expressions from Addition and Doubling algorithms, directly.

The approach is identical over split models, but working with a reduced basis changes the formulations in doubling. For the genus 2 ramified setting of doubling (Algorithm 26), the computation of

$$s = b_1 k \pmod{u_1}$$

where $b_1 = \tilde{v}^{-1} \pmod{u_1}$ and $\tilde{v} = 2v_1 + h$ is required. This yields

$$s = s_1 x + s_0 = (k_1 x + k_0)/(\tilde{v}_1 x + \tilde{v}_0) \pmod{x^2 + u_{11} x + u_{10}}$$

where $\tilde{v} = 2v_1 + h \pmod{x^2 + u_{11}x + u_{10}}$. Multiplying through by $\tilde{v}$ yields the polynomial representation

$$k_1 x + k_0 = (\tilde{v}_1 x + \tilde{v}_0)(s_1 x + s_0) - s_1 \tilde{v}_1 (x^2 + u_{11} x + u_{10}).$$

By multiplying out the right hand side and equating coefficients the following equations linear in coefficients of s arise:

$$k_0 = (\tilde{v}_0) \cdot s_0 + (-\tilde{v}_1 u_{10}) \cdot s_1,$$
$$k_1 = (\tilde{v}_1) \cdot s_0 + (\tilde{v}_0 - \tilde{v}_1 u_{11}) \cdot s_1.$$

It takes two multiplications to compute the coefficients. Once the system is set up, it takes six multiplications using Cramer's rule to solve the system for s_1, s_0 and the determinant as described later in Section 5.1.4.

For the ramified setting of addition (Algorithm 27), the same technique is used but $s = s_1 x + s_0 = (\tilde{v}_1 x + \tilde{v}_0)/(x^2 + u_{11}x + u_{10}) \pmod{x^2 + u_{21}x + u_{20}}$, where $\tilde{v} = v_2 - v_1$, is computed instead. The polynomial representation is

$$\tilde{v}_1 x + \tilde{v}_0 = (s_1 x + s_0)(x^2 + u_{11}x + u_{10}) - (s_1 x + s_1(u_{11} - u_{21}) + s_0)(x^2 + u_{21}x + u_{20}),$$

and the linear equations are:

$$\tilde{v}_0 = (u_{10} - u_{20}) \cdot s_0 + (u_{20}(u_{21} - u_{10})) \cdot s_1.$$
$$\tilde{v}_1 = -(u_{21} - u_{11}) \cdot s_0 + (u_{10} - u_{20} + u_{21}(u_{21} - u_{11})) \cdot s_1.$$

The determinant d of the 2 by 2 system is exactly the resultant of the modulus and denominator polynomial. Over fields with characteristic not equal to two, an alternate technique to Cramer's rule can be applied saving one multiplication at the cost of 13 additions, and has shown to be not as efficient when working over large finite fields [2] (see Section 4.3.1 as well.)

The same idea can be applied to genus 3, but solving for the coefficients of s directly is in fact more costly than computing the resultant as in Section 4.3.5. Here, the process of computing the resultant is alternatively presented using the system of equations for s, requiring the least number of field operations compared to previous work. In the genus 3 split setting of doubling (Algorithm 32), the computation of

$$s = b_1 k \pmod{u_1}$$

where $b_1 = \tilde{v}^{-1} \pmod{u_1}$ and $\tilde{v} = 2v_1 + h$ is also required. This yields

$$s = s_2x^2 + s_1x + s_0 = (k_2x^2 + k_1x + k_0)/(\tilde{v}_2x^2 + \tilde{v}_1x + \tilde{v}_0) \pmod{x^3 + u_{12}x^2 + u_{11}x + u_{10}},$$

where $\tilde{v} = 2v_1 + h \pmod{x^3 + u_{12}x^2 + u_{11}x + u_{10}}$. Multiplying through by $\tilde{v}$ yields the polynomial representation

$$k_2x^2 + k_1x + k_0 = (\tilde{v}_2x^2 + \tilde{v}_1x + \tilde{v}_0)(s_2x^2 + s_1x + s_0) - s_2\tilde{v}_2(x^3 + u_{12}x^2 + u_{11}x + u_{10}).$$

The resulting 3x3 matrix M is connected to the Sylvester matrix (described in Section 4.3.5) in such a way that only certain coefficients of M are required to compute the polynomial inversion of $\tilde{v}$ modulo u_1 where

$$t = m_7x^2 + m_4x + m_1 = \frac{d}{\tilde{v}} \pmod{u_1}$$

without the field inversion of $d = \det(M)$. The computation of d is chosen to reuse the same coefficients. Since computing s would require dividing by d (i.e. an expensive field inversion)

$s' = ds = kt \pmod{u_1}$ is computed instead, and as usual the inversion is deferred until later in the algorithm (see Section 4.3.7.)

One can apply Karatsuba multiplication twice, or use Toom style multiplication [23, p.294] to perform the required polynomial multiplication for s'. Toom style multiplications trades one field multiplication for five field additions and two field divisions by 2, when compared to applying Karatsuba multiplication twice, and cannot be applied over fields with characteristic 2. In order to keep the explicit formulas as simple as possible, (see Section 4.3.1) Karatsuba is applied twice instead of Toom style multiplication in this work. Genus 3 addition is similar to doubling, where $s = (v_2 - v_1)/u_1 \pmod{u_2}$ for $a_1 = 1/u_1$ in Addition Algorithm 33 is setup as $su_1 = (v_2 - v_1) \pmod{u_2}$. $s = (v_2 - v_1)/u_1 \pmod{u_2}$.

4.3.7 Montgomery's Trick of Simultaneous Inversion

The idea of Montgomery is trading costly field inversions for cheaper operations, e.g., field multiplications. Instead of computing inverses as needed, a 'copy' or 'weight' of the value to be inverted is left in the intermediate operands until the computation is at a point where all inverses can be found at once. This is described in more detail using an example below.

The computation of the inverse of the determinant d of the 2 by 2 linear system of equations when solving for s in the typical genus 2 ramified model divisor addition and doubling (described in Section 4.3.6) is required. The inverse of s_1 where $s = s_1x + s_0$ is also required to make u monic when reordering the normalization step (described in Section 4.3.8). Instead of computing each inversion, a copy of d is kept in s_1 as $s'_1 = ds_1$ and s_0 as $s'_0 = ds_0$ until s_1 and s_0 are computed. Then the inversion $w = (ds'_1)^{-1}$ is computed, and $w_{s_1} = wd^2 = 1/s_1$ and $w_d = ws'_1 = 1/d$ are extracted from w. Montgomery's inversion trick is similarly applied to the computation of s in the genus 2 and 3 split settings, and in addition to the computation of the intermediate u polynomial in any case where that requires an extra adjustment.

4.3.8 Reordering of the Normalization Step

Takahashi [38] noticed that at the end of composition in Harley addition and doubling the output u polynomial needs to be normalized in order to make it monic. Reordering where the normalization happens i.e. normalizing polynomials that u is composed of before computing u has been shown to save one multiplication [38]. Notice that the highest degree coefficient of u in line 16 of Algorithm 27 or line 10 of 26 solely comes from the highest degree coefficient of the intermediate polynomial s. By normalizing s before computing u, u is guaranteed to be monic. The obvious benefits are that working with monic polynomials uses fewer field operations. This trick directly applies to most explicit formulas described in Chapters 5 and 6, where any omissions are explicitly stated in the descriptions of the explicit formulas.

4.3.9 Equating Coefficients

One technique for computing polynomial operations that is sometimes advantageous over regular operations is equating coefficients between polynomial equalities. This technique can be used when computing the output Mumford polynomial u in all doubling and addition algorithms (see [6] for the derivation of ramified model genus 2 addition and doubling formulas). The direct computation of NUCOMP methods described in Section 4.1.3 produce similar or better formulations.

For example, on genus 2 ramified models consider u the output polynomial of Cantor's reduction step within Cantor's Addition (Algorithm 4) with inputs $[u_1, v_1]$ and $[u_2, v_2]$. Let $[u' = u_1u_2 = x^4 + u'_3x^3 + u'_2x^2 + u'_1x + u'_0, v' = v'_3x^3 + v'_2x^2 + v'_1x + v'_0]$ be the unreduced intermediate Mumford polynomials to the reduction step after composition. Then the equality

$$u_nu' = u_nu_1u_2 = \frac{f - v'(v' + h)}{v'^2_3}.$$

holds. Equating coefficients of x^5 and x^4 results in

$$u_{21} + u_{11} + u_{n_1} = \frac{2v'_2v'_3 + v'_3h_2 - f_5}{v'^2_3},$$

and

$$u_{n_0} + u_{11}u_{21} + u_{n_1}(u_{21} + u_{11}) = \frac{2v'_1v'_3 + v'^2_2 + v'_2h_2 + v'_3h_1 - f_4}{v'^2_3},$$

respectively. Manipulating the equations to solve for u_1 and u_0 produce identical formulas to those in the NUCOMP methods in the genus 2 ramified setting, but often worse formulas in the other settings. Therefore the NUCOMP methods (described in Section 4.1.3) are used in the description of the basic formulations in Chapters 5 and 6 instead.

4.4 Novel Techniques

In this section, descriptions of additional novel techniques developed in this work, that have not been used in previous work, are given.

4.4.1 Omitting l' and the Alternate Computation of v''

The technique introduced here applies to formulations over genus 2 ramified models and saves one multiplication in the Addition and Doubling Algorithms 27 and 26 following the formulation of NUCOMP when specialized to genus 2. Let $l' = s_1(l'') + v_1$ be the unreduced representation of the output v where $l'' = s''u_1$, $s'' = s/s_1$ and for example, $\tilde{v} = -l' - h$ in line 15 of Algorithm 27. In [24], l'' is computed explicitly where l''_1, l''_2 are used in the computation of the output u_0 as well as reused in the computation of output v by Karatsuba Reduction (Section 4.3.4).

When using the system of equations method for computing s (Section 4.3.6), one can reuse computations from setting up the system to simplify the equation that computes the output u_{n_0} value and removing the need for computing l'_1 and l'_2. The only computation left that relies on $l'' = s''u_1$ is

$$v_n = -l' - h = (-s_1 s'' u_1 - v_1 - h) \pmod{u_n}.$$

The value t_2 is computed beforehand in the computation of the output polynomial $u_n = x^2 + u_{n_1}x + u_{n_0}$ in the frequent cases of degree 2 doubling and addition where $u_{n_1} = s''_0 - t_2$ and therefore $t_2 = s''_0 - u_{n_1}$. Then,

$$v_{n_1} = s_1(u_1 s''_0 + u_1 u_{11} - u_{n_1}^2 - s''_0 u_{11} + u_{n_0} - u_{10}) - v_{11} - h_1 + h_2 u_1$$

$$
\begin{aligned}
&= s_1((u_{n_1} - s_0'')(u_{11} - u_{n_1}) + u_{n_0} - u_{10}) - v_{11} - h_1 + h_2 u_1 \\
v_{n_0} &= s_1(u_0 s_0'' + u_{11} u_{n_0} - u_0 u_1 - s_0'' u_{10}) - v_{10} - h_0 + h_2 u_0 \\
&= s_1(s_0''(u_{n_0} - u_{10}) + u_{n_0}(u_{11} - u_{n_1})) - v_{10} - h_0 + h_2 u_0.
\end{aligned}
$$

Notice that $t_2 = u_{n_1} - s_0''$, then the explicit formulation for v_n is

$$
\begin{aligned}
t_0 &= u_{n_0} - u_{10}, \\
t_1 &= u_{11} - u_{n_1}, \\
v_{n_1} &= s_1(t_2 t_1 + t_0) - v_{11} - h_1 + h_2 u_{n_1} \\
v_{n_0} &= s_1(s_0'' t_0 + u_{n_0} t_1) - v_{10} - h_0 + h_2 u_{n_0}.
\end{aligned}
$$

The resulting representation of the equation requires seven multiplications and since the two multiplications needed to compute l'' are omitted, there is an overall savings of one multiplication.

4.4.2 Alternate computation of k

The technique introduced here applies to all doubling and most addition formulas for split models of genus 2 and 3 hyperelliptic curves where the continued fraction loop in Steps 24–26 of Balanced NUCOMP (Algorithm 21) is not entered.

In these cases, the value wc_3 for genus 2 curves and wc_4 for genus 3 curves for some w is inverted in order to make u monic and the explicit computation of $k = (f - v_1(v_1 + h))/u_1$ is required. New precomputed values are introduced in each genus case that are curve constants divided by c_3 or c_4. The multiplication of w with c_3 or c_4 before inversion is omitted and k/c_i for $i = 3, 4$ is computed instead of k.

Applying this technique reduces the number of multiplications by at least 1 in all aforementioned cases over arbitrary fields, saves several additions in those cases over fields with characteristic not equal to 2, but do not apply over characteristic 2 fields as $c_3 = c_4 = 1$ in that setting.

Chapter 5

Explicit Formulas for Genus 2 Arithmetic

In this chapter, derivations of explicit formulas for genus 2 hyperelliptic curves given by a ramified and split model are described. All explicit formulas presented in this chapter are implemented in Magma along with implementations of random input testing and automatic generation of inputs that cover every output case. Automated operation counting and latex table generation scripts were also developed to circumvent errors in the operation count and explicit formula presentations. All software developed for this thesis is available at `https://github.com/salindne/divisorArithmetic`.

5.1 Improved Genus 2 Ramified Model Explicit Formulas

In this section, novel explicit formulas for addition and doubling of divisor classes on genus 2 hyperelliptic curve ramified models are described. In general, the curve equation of a ramified model for a genus 2 hyperelliptic curve in Weierstrass form is $C : y^2 + h(x)y = f(x)$ where

$$f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0, \qquad h = h_2x^2 + h_1x + h_0$$

with $h_2 \in \{1, 0\}$. Curve coefficients are represented as vectors of field elements

$$f = [f_4, f_3, f_2, f_1, f_0], \quad h = [h_2, h_1, h_0].$$

Input and output divisors are represented as 4-coordinate vectors of field elements

$$D = [u_1, u_0, v_1, v_0],$$

for the Mumford representation $D = [u, v]$ with $u = x^2 + u_1x + u_0$ and $v = v_1x + v_0$. Only 4 coordinates are required, the least number of coordinates possible resulting in formulas that are the simplest.

The improvements in this work are obtained by working from versions of NUCOMP and NUDUPL specialized to genus 2 (Algorithms 27 and 26) and combining the best ideas from prior works. The formulas are complete in the sense that all possible input and output cases are given explicitly as opposed to falling back on Cantor's algorithm when special cases arise. In all underlying field considerations, arbitrary, $\mathrm{char}(k) = 2$ and $\mathrm{char}(k) \neq 2$ fields, this yields novel formulas requiring fewer field operations than any others to date. Moreover, all explicit formulas presented require only one inversion through any computation path, including all special cases.

This section is structured as follows. First, assumptions about curve models over certain fields are discussed, then an overview of prior work is provided. This is followed by a statement of the novel contributions in this work and then a description of the basic formulations for doubling and addition are provided. Finally, operation costs in terms of field operations, comparisons to previous work, and empirical benchmarking results are presented.

5.1.1 Curve Simplifications

The efficiency of explicit formulas somewhat depends on the equation of the hyperelliptic curve as the polynomial-based algorithms for divisor class arithmetic presented in the previous chapters involve computations with f and h. Isomorphic models of the curve with more zero coefficients in f and h allow for explicit formulas with fewer field operations. Assumptions and simplifications to the hyperelliptic equation of a curve are discussed, starting with curve isomorphisms that transform a ramified model curve into another isomorphic ramified model curve with a less complex equation [5].

Recall that a ramified model for a genus 2 hyperelliptic curve can be given as

$$C : y^2 + h(x)y = f(x)$$

where $h(x)$ is at most degree two, and $f(x)$ is at most degree five. An isomorphic transformation of C is defined as a transformation of the variables x and y (change of variables) to other expressions involving x and y; see [5]. What isomorphic transformations can be applied depends on the characteristic of the underlying field the curve is defined over. Next, the isomorphic transformations that are possible over fields of characteristic two and characteristic not equal to two are described; for more details see [24].

char(k) ≠ 2

If char$(k) \neq 2$, then applying the substitution $y \to y - h(x)/2$ invokes the transformation

$$\begin{aligned} y^2 + h(x)y = f(x) &\to (y - h(x)/2)^2 + h(x)(y - h(x)/2) = f(x) \\ &\implies y^2 - 2h(x)y/2 + (h(x))^2/4 + h(x)y - 2(h(x))^2/4 = f(x) \\ &\implies y^2 = f'(x) \end{aligned}$$

where $f' = f + (h(x))^2/2$. This transformation is always possible when char$(k) \neq 2$, allowing the assumption of $h = 0$ in this case. If the characteristic of the underlying field is also not five, the transformation $x \to x - f_4/5$ that takes $f(x) \to f(x)'$ to get

$$f(x)' = x^5 + f_3'x^3 + f_2'x^2 + f_1'x + f_0'$$

can be applied, allowing one to assume that $f_4 = 0$ in that case. Operation counts in Section 5.1.6 assume $h = 0$ whenever char$(k) \neq 2$, but make no assumption about f_4.

char(k) = 2

If char$(k) = 2$, then division by two is not possible and the transformation

$$y \to y - \frac{h(x)}{2}$$

cannot be made. In the genus 2 setting h non-constant is guaranteed because otherwise the curve model is singular. The following isomorphism transforms a curve to one for which several coefficients of h and f are zero or one:

$$y \to h_2^5 y + f_3 h_2 x + \frac{f_3(f_3 + h_1 h_2 + f_4 h_2^2 + f_2 h_2^2)}{h_2^3}, \quad x \to h_2^2 x + f_4.$$

Dividing the equation by h_2^{10} results in $h_2 = 1$ and $f_4 = f_3 = f_2 = 0$. This equation of the curve is assumed when counting operations in Section 5.1.6 over base field k with char$(k) = 2$.

5.1.2 Prior Work

Most prior work mentioned in this section only applies to divisor class arithmetic over fields with $\text{char}(k) \neq 2$, where isomorphisms can be applied to ensure that $h(x) = 0$. The work of this thesis not only improves on divisor class arithmetic in that setting, but also extends to arbitrary base fields where no assumptions about the equation of the curve are made, as well as specializations to the case $\text{char}(k) = 2$.

In [24], Lange presented affine explicit formulas over fields with $\text{char}(k) = 2$ and $\text{char}(k) \neq 2$ using 4 coordinate representation (u_1, u_0, v_1, v_0). Each coordinate in the representation is a coefficient of a polynomial in Mumford representation (Section 2.3.1) for generic genus 2 curves with a ramified model in Weierstrass form. Frequent cases and some special cases where the output degree in the frequent case is less than the genus, as well as adding degree one with degree two divisors are presented. The novel work of this thesis directly improves on this prior work.

The author also gave a description for computing all other special cases. Some suggestions do not work in all settings, for example suggestion 2b)ii) on page 7 of [24, Section 4.1] results in an infinite loop if the divisor class being composed has 3 torsion. Most of the suggestions required several inversions as presented. More details are provided in Sections 5.1.4 and 5.1.5.

In [6], the authors presented 6-coordinate affine formulas for adding and doubling divisor classes in the frequent cases only based on the linear algebra method (Section 4.1.2) that applies a system of linear equations in the composition portion of Cantor's Algorithm. The extra coordinates are $U_0 = u_0u_1$ and $U_1 = u_1^2$, which are computed at the end of each operation, but also required in the beginning of the next operation. To avoid computing the same values twice in chains of operations, U_0 and U_1 are passed as auxiliary coordinates. Numerical testing in Magma on a consistent platform presented in Section 5.3 suggests that although the field multiplication or squaring counts are smaller, as shown in Table 5.1, a surplus of field additions results in explicit formulas that perform worse than not only the formulas of this work but in some cases also those of [24].

5.1.3 Contributions Summary

This work provides complete explicit formulas, in the sense that all computation paths are explicitly computed, requiring only one inversion in all cases that take reduced divisor class representatives as input, and output a reduced divisor class representative. Not only is every special case explicitly computed, this work also improves on previous best in the frequent cases of addition and doubling. All formulas are implemented in Magma including a testing suite (described in Section 5.3) to ensure their correctness. The implemented formulas and testing scripts are available at `https://github.com/salindne/divisorArithmetic`.

The formulations of divisor class addition and doubling are based on NUCOMP specialized to genus 2 addition and doubling that includes efficient formulations of special cases. The formulation of steps in Algorithms 27 and 26, coupled with certain explicit techniques produce the fastest explicit formulas for frequent case addition and doubling to date. The formulas incorporate different techniques borrowed from previous works and also introduce a new technique presented in Section 4.4.1.

A summary of the novel techniques that differ from previous work is as follows:

T1 Use a system of equations [1, 6] to solve for $s = s_1x + s_0$ rather than computing the resultant as done by Harley and others (see Section 4.3.6). In contrast, the linear algebra method uses a system of equations to directly solve for $V = su_1 + v_1$.

T2 Compute the coefficients of the output u and v polynomials using a NUCOMP based polynomial formulation specialized to genus 2. This technique allows for the reuse of computations from solving for s through the system of equations.

T3 Compute v differently by omitting the direct computation of su_1 and use s, u_1 directly as described in Section 4.4.1.

T4 Include explicit one inversion formulas via the specialized NUCOMP formulations (Algorithms 27 and 26) for all special cases with $\gcd(u_1, u_2, v_1 + v_2 + h) \neq 1$ producing vastly more efficient formulas for special cases when compared to previous suggestions [24].

Note that in this work, the equating coefficient technique for computing the output Mumford polynomial u from [6] (described in Section 4.3.9) produced identical formulations for computing u when compared to a direct computation of u via NUCOMP as in Addition and Doubling Algorithms 27 and 26. Moreover, the threshold for trading field multiplications for field additions is capped at 1 multiplication for 3 additions throughout (see Section 4.3.1.)

Special Cases

All implementations of frequent case arithmetic include explicit handling of special cases as they come up naturally in the computations via Technique T4. All computation paths through special cases in the formulas require only one inversion. The formulations are based on Algorithm 27, that includes efficient handling of special cases at the level of polynomial arithmetic, see Section 4.1.3. Compared to the formulas of this work, the suggestions from previous work [24] on handling these cases produced far more cumbersome explicit formulas, between 20 and 40 percent more field operations. The reduction in the number of field operations can be attributed to the removal of algorithms requiring lower degree input polynomials with weighted inputs, as required when implementing one inversion versions of the special case suggestions from [24].

5.1.4 Explicit Formulas for Doubling

In this section, complete divisor class doubling is described and the basic formulations of the corresponding explicit formulas are provided. Doubling Algorithm 26 is further specialized to different cases depending on the input divisor class degree. The explicit techniques used are discussed in each case.

The complete doubling algorithm should test the degree of the input divisor class representative in order of statistical frequency, calling degree 2 doubling (Algorithm 34) and degree 1 doubling (Algorithm 36) accordingly. The neutral element is returned if the degree of the input divisor class representative is zero, and therefore the neutral element itself. In the following, the basic formulations required for complete doubling, organized by the degree of the input divisor are provided.

Genus 2 Ramified Model Degree 2 Doubling

Let $[u_1, v_1]$ be a reduced divisor class representative with $u_1 = x^2 + u_{11}x + u_{10}$ and $v_1 = v_{11}x + v_{10}$. Steps 1-5 in Algorithm 26 are specialized to degree 2 doubling by computing the resultant $d = \mathrm{Res}(u_1, 2v_1 + h)$ instead of $S = \mathrm{XGCD}(u_1, 2v_1 + h)$ in Step 2 of Algorithm 26, where $d = 0$ implies $S \neq 1$. In this case $b_1 = 1/(2v_1 + h) \pmod{u_1}$ so $s = k/(h + 2v_1) \pmod{u_1}$ in Step 5 of Doubling Algorithm 26. The resultant d is computed by first setting up a system of equations to solve for the coefficients of s via the equality $s(h + 2v_1) = k \pmod{u_1}$, for which the determinant of the resulting 2x2 matrix with coefficients m_i is the resultant d (see Section 4.3.6).

The computation of the k polynomial in Step 1 of Algorithm 26 is postponed until after the computation of d, and instead d is immediately checked for special cases. If $d \neq 0$, then there are no common factors between u_1 and $2v_1 + h$, and Steps 5 and 9–11 of Doubling Algorithm 26 are executed. Steps 8-9 of Doubling Algorithm 26 are combined in all output cases taking advantage of combining terms that need to be multiplied by precomputed weights to make u_n monic. If $d = 0$, then Steps 4,5,7 are performed instead, described in Subroutine 35.

The basic formulation for degree 2 doubling, adapted from Doubling Algorithm 26, is described in Algorithm 34. An explanation of how each step in Algorithm 34 is converted to an explicit formula is given in the following.

In Step 1, the polynomial representation

$$k'_1x + k'_0 = (\tilde{v}_1x + \tilde{v}_0)(s_1x + s_0) - s_1\tilde{v}_1(x^2 + u_{11}x + u_{10})$$

is used to solve for s_1, s_0 where

$$s = s_1x + s_0 = (k'_1x + k'_0)/(\tilde{v}_1x + \tilde{v}_0) \pmod{x^2 + u_{11}x + u_{10}}$$

and $\tilde{v} = 2v_1 + h \pmod{u_1}$. By equating coefficients, the following equations linear in coefficients of s arise:

$$k'_0 = (\tilde{v}_0) \cdot s_0 + (-\tilde{v}_1u_{10}) \cdot s_1,$$

Algorithm 34 Genus 2 Ramified Model Degree 2 Doubling

Input: $[D_1] = [u_1, v_1]$, f, h.

Output: $[u_n, v_n] = 2[D_1]$.

```
 1: Compute $m_i$ in system for $s(h + 2v_1) = k' \pmod{u_1}$ as $\left[\begin{smallmatrix} m_4 & -m_2 \\ -m_3 & m_1 \end{smallmatrix}\right]\left[\begin{smallmatrix} s_0 \\ s_1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} k'_0 \\ k'_1 \end{smallmatrix}\right]$.
 2: $d = m_1 m_4 - m_2 m_3$.
 3: if $d$ is 0 then Go to Subroutine 35.
 4: $k' = (f - v_1(v_1 + h))/u_1 \pmod{u_1}$.   (and $k = (f - v_1(v_1 + h))/u_1$)
 5: Solve for $s'_1 = ds_1, s'_0 = ds_0$ using the $m_i$ and Cramer's rule, omitting the inversion of $d$.
 6: if $s'_1 = 0$ then
 7:     $s_0 = s'_0/d$.
 8:     $u_n = -(s_0(2v_1 + h) - k)/u_1 - s_0^2$   (exact division).
 9:     $v_n = (-s_0 u_1 - v_1 - h) \pmod{u_n}$.
10:     return $[u_n, v_n]$.
11: $w = 1/(ds'_1)$ and precompute values $1/s_1$, $1/s_1^2$ and $s_1$.
12: Compute monic $s'' = x + s''_0$ where $s''_0 = s'_0/s'_1$.
13: $u_n = (s'')^2 + (s''(2v_1 + h) - k/s_1)/(s_1 u_1)$   (exact division, made monic).
14: $v_n = (-s_1 s'' u_1 - v_1 - h) \pmod{u_n}$.
15: return $[u_n, v_n]$.
```

$$k'_1 = (\tilde{v}_1) \cdot s_0 + (\tilde{v}_0 - \tilde{v}_1 u_{11}) \cdot s_1.$$

Consider the following 2×2 system

$$\begin{pmatrix} \tilde{v}_0 & -\tilde{v}_1 u_{10} \\ \tilde{v}_1 & \tilde{v}_0 - \tilde{v}_1 u_{11} \end{pmatrix} \times \begin{pmatrix} s_0 \\ s_1 \end{pmatrix} = \begin{pmatrix} k'_0 \\ k'_1 \end{pmatrix},$$

representing the identities above. Instead set up the system as

$$\begin{pmatrix} m_4 & -m_2 \\ -m_3 & m_1 \end{pmatrix} \times \begin{pmatrix} s_0 \\ s_1 \end{pmatrix} = \begin{pmatrix} k'_0 \\ k'_1 \end{pmatrix},$$

for $m_4 = \tilde{v}_0, m_2 = \tilde{v}_1 u_{10}, m_3 = -\tilde{v}_1, m_1 = -\tilde{v}_1 u_{11}$. Multiplying both sides by the inverse of the the matrix yields the following expressions for s_1 and s_0

$$\begin{pmatrix} m_1/d & m_2/d \\ m_3/d & m_4/d \end{pmatrix} \times \begin{pmatrix} k'_0 \\ k'_1 \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \end{pmatrix},$$

where the d is the determinant. The resulting explicit formula for the m_i in Step 1 and d in Step 2 is

$$\hat{v}_0 = v_{10} + h_0,$$

$$\begin{aligned}
\hat{v}_1 &= v_{11} + h_1,\\
m_3 &= h_2u_{11} - v_{11} - \hat{v}_1,\\
m_4 &= v_{10} + \hat{v}_0 - h_2u_{10},\\
m_1 &= m_4 + m_3u_{11},\\
m_2 &= -m_3u_{10},\\
d &= m_1m_4 - m_2m_3,
\end{aligned}$$

with $\hat{v}_1$ and $\hat{v}_0$ reused later in the computation of output polynomials u_n and v_n.

If $d \neq 0$, then in Step 4 the polynomial

$$k' = k'_1x + k'_0 = (f - hv_1 - v_1^2)/u_1 \pmod{u_1}$$

is computed. First the exact division by u_1 is computed, then the modular reduction of the result by u_1 collecting expressions for the coefficients of k'_1 and k'_0. The resulting explicit formula for Step 4 is

$$\begin{aligned}
t_0 &= u_{11}^2,\\
t_1 &= f_3 + t_0 - h_2v_{11},\\
t_2 &= u_{10} + u_{10},\\
t_3 &= f_4u_{11},\\
t_4 &= f_4u_{10},\\
t_5 &= t_0 - t_3,\\
t_6 &= t_1 - t_2,\\
k_1 &= t_5 + t_5 + t_6,\\
k_0 &= u_{11}(t_2 - t_6 + t_3) + f_2 - v_{11}\hat{v}_1 - t_4 - t_4 - h_2v_{10}.
\end{aligned}$$

In Step 5, the solution of the system of equations $s' = sd$ given by the resulting 2x2 matrix after inversion above is computed (see Section 4.3.6), where the resulting explicit formula is

$$s'_0 = k'_0m_1 + k'_1m_2$$

$$s_1' = k_0'm_3 + k_1'm_4.$$

If $s_1' = 0$ the special output case is invoked with $\deg(s') = 0$. The value $s_0 = s_0'/d$ is computed directly in Step 7, followed by $u = s^2 + (s(2v_1 + h) - k)/u_1$ and $v = (-su_1 - v_1 - h) \pmod{u}$ in Steps 8-9. Notice that since $\deg(k) = 3$, with $k = (f - v_1(v_1 + h))/u_1$, where coefficients $k_3 = 1, k_2 = f_4 - u_{11}$, the resulting u_n' (not normalized) is $u_n' = -x + s_0h_2 - f_4 + 2u_1 + s_0^2$. Normalization results in $u_n = x + u_{n_0}$ with

$$u_{n_0} = f_4 - s_0^2 - s_0h_2 - 2u_{11}.$$

The polynomial v_n is computed via a direct school book modular reduction of the polynomial

$$-s_0u_1 - v_1 - h = -(s_0 + h_2)x^2 - (s_0u_{11} + h_1)x - (s_0u_{10} + h_0)$$

by $x + u_{n_0}$, resulting in the explicit formulation

$$t_1 = s_0(u_{11} - u_{n_0}) - h_2^2u_{n_0} + \hat{v}_1,$$
$$v_{n_0} = u_{n_0}t_1 - \hat{v}_0 - s_0u_{10}.$$

The field multiplication pattern for Karatsuba reduction does not occur (Section 4.3.4).

If $s_1' \neq 0$ then in Steps 11–12 the required weights for normalizing u_n, along with s_1 and monic $s'' = x + s_0''$ for $s_0'' = s_0/s_1$ are computed. The resulting explicit formula (with the actual values given in brackets) is

$$w_1 = (ds_1')^{-1},$$
$$w_2 = w_1d, \qquad (1/(ds_1))$$
$$w_3 = w_2d, \qquad (1/s_1)$$
$$w_4 = w_3^2, \qquad (1/(s_1^2))$$
$$s_1 = w_1(s_1')^2,$$
$$s_0'' = s_0'w_2,$$

as described in Section 4.3.8.

In Step 13, the polynomial $u'_n = s^2 + (s(2v_1 + h) - k)/u_1$ requires a division by s_1^2 to be normalized, so

$$u_n = (s'')^2 + ((s''(2v_1 + h) - k/s_1)/u_1)/s_1$$

is computed, utilizing an exact division (see Section 4.3.2). The resulting explicit formula for Step 13 is

$$\begin{aligned} t_2 &= s''_0 - w_4 + h_2 w_3, \\ u_{n_1} &= s''_0 + t_2, \\ u_{n_0} &= (s''_0)^2 + w_3(h_2(s''_0 - u_{11}) + v_{11} + \hat{v}_1) + w_4(2u_{11} - f_4), \end{aligned}$$

with k as described above.

In Step 14, the polynomial $v_n = (-s_1 s'' u_1 - v_1 - h) = (-su_1 - v_1 - h) \pmod{u_n}$ is computed. School book modular reduction is applied instead of Karatsuba to take advantage of reusing sub-expressions. Applying a school book modular reduction results in

$$\begin{aligned} v_{n_1} &= s_1(u_1 s''_0 + u_1 u_{11} - u_{n_1}^2 - s''_0 u_{11} + u_{n_0} - u_{10}) - v_{11} - h_1 + h_2 u_1 \\ &= s_1((u_{n_1} - s''_0)(u_{11} - u_{n_1}) + u_{n_0} - u_{10}) - v_{11} - h_1 + h_2 u_1 \\ v_{n_0} &= s_1(u_0 s''_0 + u_{11} u_{n_0} - u_0 u_1 - s''_0 u_{10}) - v_{10} - h_0 + h_2 u_0 \\ &= s_1(s''_0(u_{n_0} - u_{10}) + u_{n_0}(u_{11} - u_{n_1})) - v_{10} - h_0 + h_2 u_0. \end{aligned}$$

Recall that in Step 13, $u_{n_1} = s''_0 + t_2$ and so $t_2 = u_{n_1} - s''_0$. This combined with the formula for Step 14 above results in the following explicit formula for Step 14:

$$\begin{aligned} t_0 &= u_{n_0} - u_{10}, \\ t_1 &= u_{11} - u_{n_1}, \\ v_{n_1} &= s_1(t_2 t_1 + t_0) - v_{11} - h_1 + h_2 u_1 \\ v_{n_0} &= s_1(s''_0 t_0 + u_{n_0} t_1) - v_{10} - h_0 + h_2 u_0, \end{aligned}$$

as described in Section 4.4.1.

For $d = 0$ and $\deg(S) = 2$, with S from Doubling Algorithm 26, the input divisor class representative $[u_1, v_1]$ has order 2, resulting in the output of the neutral divisor class. Otherwise, exactly one point with x-coordinate determined by $S = x - m_4/m_3$ (one of the two factors of u_1) has two torsion. The polynomial S is removed from u_1 via an exact division (Section 4.3.2). The basic formulation of the case $d = 0$ within degree 2 doubling is presented in Subroutine 35. An explanation of how each step in Subroutine 35 is converted to an explicit formula is given in the following.

Subroutine 35 Genus 2 Ramified Model Degree 2 Doubling ($d = 0$)

```
1: Let $d_w = (h + 2v_1) \pmod{u_1} - m_3x + m_4$.
2: if $m_3$ is 0 then return $[1, 0]$.
3: $k = (f - v_1(v_1 + h))/u_1$.
4: $b_1 = 1/\mathrm{lcf}(d_w)$   (made monic).
5: $u_1 = b_1u_1/d_w$.
6: $s = b_1k \pmod{u_1}$.
7: $u_n = u_1^2$
8: $v_n = (v_1 + u_1s) \pmod{u_n}$.
9: return $[u_n, v_n]$.
```

Step 1, reuses the computations of m_3 and m_4 from degree 2 doubling Algorithm 34, resulting in $d_w = -m_3x + m_4$. In Step 3 , an exact division by the original input u_1 polynomial is computed in $k = (f - v_1(v_1 + h))/u_1$. The resulting explicit formula for Step 3 is

$$k_2 = f_4 - u_{11},$$

$$k_1 = f_3 - v_{11}h_2 - u_{10} - u_{11}k_2,$$

$$k_0 = f_2 - v_{11}\hat{v}_1 - v_{10}h_2 - u_{10}k_2 - u_{11}k_1.$$

In Steps 4–5, the leading coefficient $-m_3$ of d_w is inverted and u_1 divided by monic d_w. The explicit formula for Steps 4–5 is

$$b_1 = -m_3^{-1}, \qquad u_{10} = u_{11} - m_4b_1.$$

In Steps 6–8, the polynomial $s = b_1k \pmod{u_1}$ is computed by a modular reduction, $u_n = u_1^2$ by a polynomial squaring and $v_n = (v_1 + u_1s_0) \pmod{u_n}$ by a modular reduction as well. Since $\deg(u_1) = 1$, the multiplication pattern required for Karatsuba reduction (Section 4.3.4) does not arise and therefore school book reductions are applied. The resulting explicit formula for Steps 6-8 is

$$\begin{aligned} s_0 &= b_1(k_0 - u_{10}(k_1 - u_{10}(k_2 - u_{10}))), \\ u_{n_1} &= u_{10} + u_{10}, \\ u_{n_0} &= u_{10}^2, \\ v_{n_1} &= v_{11} + s_0, \\ v_{n_0} &= v_{10} + s_0 u_{10}. \end{aligned}$$

Genus 2 Ramified Model Degree 1 Doubling

Let $[u_1, v_1]$ be a reduced divisor class representative with $u_1 = x + u_{10}$ and $v_1 = v_{10}$. Degree 1 doubling is computed by Steps 1-7 of Doubling Algorithm 26. The resultant $d = \mathrm{Res}(u_1, 2v_1 + h)$ is computed instead of $S = \mathrm{XGCD}(u_1, 2v_1 + h)$ in Step 2 of Algorithm 26, where $d = 0$ implies $S \neq 1$. In this case, $b_1 = 1/(2v_1 + h) \pmod{u_1}$ and so $s = k/(h + 2v_1) \pmod{u_1}$ in Step 5 of Doubling Algorithm 26. The basic formulation for degree 1 doubling is described in Algorithm 36. An explanation of how each step in Algorithm 36 is converted to an explicit formula is given in the following.

Algorithm 36 Genus 2 Ramified Model Degree 1 Doubling

Input: $[D_1] = [u_1, v_1]$, f, h.
Output: $[u_n, v_n] = 2[D_1]$.

```
1: $d = (h + 2v_1) \pmod{u_1}$.
2: **if** $d$ is 0 **then return** $[1, 0]$.
3: $k' = (f - v_1(v_1 + h))/u_1 \pmod{u_1}$.
4: $s_0 = k'/d \pmod{u_1}$   where $1/d = b_1$.
5: $u_n = u_1^2$.
6: $v_n = s_0 u_1 + v_1 \pmod{u_n}$.
7: **return** $[u_n, v_n]$.
```

In Step 1, the resultant $d = (h + 2v_1) \pmod{u_1}$ is computed by a school book modular reduction, since $\deg(u_1) = 1$. The equality $d = 0$ is immediately checked in Step 2, and if so, the appropriate special case invoked where the single x-coordinate represented by u corresponding to the affine point in the divisor class has two torsion and the neutral element is returned. Steps 3–5 are performed similarly to Steps 6–8 of Degree 2 Doubling when $d = 0$ using Subroutine 35.

5.1.5 Explicit Formulas for Addition

In this section, complete divisor class addition is described and the basic formulations of the corresponding explicit formulas are provided. Addition Algorithm 27 is further specialized to different cases depending on the input divisor class degree. The explicit techniques used are discussed in each case.

The complete addition algorithm should test the degree of the input divisor class in order of statistical frequency, calling degree 2 addition (Algorithm 37), degree 1 with 2 addition (Algorithm 39) and degree 1 addition (Algorithm 41) accordingly. The neutral element is returned if the degree of both input divisor classes is zero. In the following, the basic formulations required for complete addition, organized by the degree of the input divisor class are provided.

Genus 2 Ramified Model Degree 2 Addition

Let $[u_1, v_1]$ and $[u_2, v_2]$ be reduced divisor class representatives with $u_1 = x^2 + u_{11}x + u_{10}$, $v_1 = v_{11}x + v_{10}$ and $u_2 = x^2 + u_{21}x + u_{20}$, $v_2 = v_{21}x + v_{20}$. Steps 1-11 in Algorithm 27 are specialized to degree 2 addition by computing the resultant $d = \text{Res}(u_1, u_2)$ instead of $S = \text{XGCD}(u_1 u_2)$ in Step 2 of Algorithm 27, where $d = 0$ implies $S \neq 1$. In this case $a_1 = 1/u_1 \pmod{u_2}$ so $s = (v_2 - v_1)/u_1 \pmod{u_2}$ in Step 11 of Addition Algorithm 27. The resultant d is computed by first setting up a system of equations to solve for the coefficients of s via the equality $su_1 = (v_2 - v_1) \pmod{u_2}$, for which the determinant of the resulting 2x2 matrix with coefficients m_i is the resultant d (see Section 4.3.6).

The computation of the k polynomial in Step 1 of Algorithm 27 is postponed until after the computation of d, and instead d is immediately checked for special cases. If $d \neq 0$, then there are no common factors between u_1 and u_2, and Steps 11,15–19 of Addition Algorithm 27 are performed. Steps 15-16 of Addition Algorithm 27 are combined in all output cases taking advantage of combining terms that need to be multiplied by precomputed weights to make u_n monic. If $d = 0$, then a check for $d_w = \text{Res}(u_1, v_2 + v_1 + h) = 0$ is made, and if true, the appropriate points are removed. If $d_w \neq 0$, then s is computed by Step 9 of Addition Algorithm 27, described in Subroutine 38.

The basic formulation for degree 2 addition, adapted from Addition Algorithm 27, is described in Algorithm 37. An explanation of how each step in Algorithm 37 is converted to an explicit formula is given in the following.

Algorithm 37 Genus 2 Ramified Model Degree 2 Addition

Input: $[D_1] = [u_1, v_1]$, $[D_2] = [u_2, v_2]$, $[D1] \neq [D2]$

Output: $[u_n, v_n] = [D_1] + [D_2]$.

```
 1: Compute $m_i$ in system of equations for $su_1 \equiv \tilde{v} \pmod{u_2}$ as $\begin{bmatrix} m_4 & -m_2 \\ -m_3 & m_1 \end{bmatrix}\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} \tilde{v}_0 \\ \tilde{v}_1 \end{bmatrix}$, where
    $\tilde{v} = \tilde{v}_1 x + \tilde{v}_0 = v_2 - v_1$.
 2: $d = m_1 m_4 - m_2 m_3$.
 3: if $d = 0$ then Go to Degree 2 Addition $d = 0$ Subroutine 38.
 4: Solve for $s_1' = ds_1, s_0' = ds_0$ using the $m_i$ and Cramer's rule, omitting the inversion of $d$.
 5: if $s_1' = 0$ then
 6:     $s_0 = s_0'/d$.
 7:     $\tilde{v} = -s_0 u_1 - v_1 - h$.
 8:     $k = (f - v_1(v_1 + h))/u_1 \pmod{u_1}$.
 9:     $u_n = (s_0(\tilde{v} - v_1) + k)/u_2$ (exact division).
10:     $v_n = \tilde{v} \pmod{u_n}$.
11:     return $[u_n, v_n]$.
12: $w = 1/(ds_1')$ and precompute the values $1/s_1, 1/(s_1)^2, s_1$.
13: Compute monic $s'' = x + s_0''$ where $s_0'' = s_0'/s_1'$.
14: $k = (f - v_1(v_1 + h))/u_1$.
15: $\tilde{v} = -su_1 - v_1 - h$.
16: $u_n = (s''(\tilde{v} - v_1/s_1) + k/s_1^2)/u_2$ (exact division).
17: $v_n = \tilde{v} \pmod{u_n}$.
18: return $[u_n, v_n]$.
```

In Step 1, a similar computation to Degree 2 Doubling (Algorithm 34) for the matrix coefficients m_i and d is performed using the system of equations

$$\tilde{v}_0 = (u_{10} - u_{20}) \cdot s_0 + (u_{20}(u_{21} - u_{10})) \cdot s_1,$$

$$\tilde{v}_1 = -(u_{21} - u_{11}) \cdot s_0 + (u_{10} - u_{20} + u_{21}(u_{21} - u_{11})) \cdot s_1,$$

linear in s_1 and s_0, for $s = (v_2 - v_1)/u_1 \pmod{u_2}$.

If $d \neq 0$, then in Step 4, the solution of the system of equations $s' = sd$ is computed (Section 4.3.6), where the explicit formula for Step 4 is

$$\begin{aligned}
\tilde{v}_1 &= v_{21} - v_{11},\\
\tilde{v}_0 &= v_{20} - v_{10},\\
s'_0 &= \tilde{v}_0 m_1 + \tilde{v}_1 m_2,\\
s'_1 &= \tilde{v}_0 m_3 + \tilde{v}_1 m_4.
\end{aligned}$$

If $s'_1 = 0$ the special output case is invoked with $\deg(s') = 0$. The value $s_0 = s'_0/d$ is computed followed a direct computation of Steps 15–17 of Addition Algorithm 27 where $\tilde{v} = -su_1 - v_1 - h$ is combined into $u_n = (s(\tilde{v} - v_1) + k)/u_2$ and $v_n = \tilde{v} \pmod{u_n}$ is computed directly. Since $\deg(k) = 3$ with coefficients $k_3 = 1, k_2 = f_4 - u_{11}$, the direct compuation of output polynomial u_n is already monic and

$$u_n = x + f_4 - u_{11} - u_{21} - s_0^2 - h_2 s_0$$

by an exact division (see Section 4.3.2). The polynomial v_n is computed via a school book modular reduction of the polynomial

$$-s_0 u_1 - v_1 - h = -(s_0 + h_2)x^2 - (s_0 u_{11} + h_1)x - (s_0 u_{10} + h_0)$$

by the polynomial $x + u_{n_0}$, resulting in

$$\begin{aligned}
t_1 &= s_0(u_{11} - u_{n_0}) - h_2^2 u_{n_0} + v_{11} + h_1,\\
v_{n_0} &= u_{n_0} t_1 - \hat{v}_0 - s_0 u_{10}.
\end{aligned}$$

If $s'_1 \neq 0$ then in Steps 12–13 the required weights for normalizing u_n, along with s_1 and monic $s'' = x + s''_0$ for $s''_0 = s_0/s_1$, are computed. The resulting explicit formula for Steps 12–13 is

$$\begin{aligned}
w_1 &= (ds'_1)^{-1},\\
w_2 &= w_1 d, \qquad (1/(ds_1))\\
w_3 &= w_2 d, \qquad (1/(s_1))\\
w_4 &= w_3^2, \qquad (1/(s_1)^2)
\end{aligned}$$

$$s_1 = w_1(s_1')^2,$$

$$s_0'' = s_0' w_2,$$

as described in Section 4.3.8.

In Steps 14–16, the polynomial $u_n' = (s(-su_1 - 2v_1 - h) + k)/u_2$ requires a division by s_1^2 to be normalized, so

$$u_n = s''(\tilde{v} - v_1/s_1) + k/(s_1)^2)/u_2$$

is computed combining the three steps, utilizing an exact division (see Section 4.3.2), and taking advantage of combining terms that need to be multiplied by the precomputed weights w_i. The resulting formula for Steps 14–16 is

$$t_3 = s_0'' - m_3,$$

$$t_2 = t_3 - w_4 + h_2 w_3,$$

$$\hat{v}_1 = h_1 + v_1,$$

$$u_{n_1} = s_0'' + t_2,$$

$$u_{n_0} = s_0''(t_3 - m_3) + m_1 + w_3(h_2(s_0'' - u_{21}) + \hat{v}_1 + v_{11}) + w_4(u_{11} + u_{21} - f_4),$$

where only the degree 2 coefficient $k_2 =$ of k is used. In Step 17, computation of the polynomial $v_n = \tilde{v} \equiv (-su_1 - v_1 - h) \pmod{u_n}$ is identical to degree 2 doubling (also see Section 4.4.1.)

The basic formulation of the case $d = 0$ within degree 2 addition is presented as Subroutine 38. An explanation of how each step in Subroutine 38 is converted to an explicit formula is given in the following.

In the case $d = 0$, there exists at least one common x-coordinate among the two input divisor classes. With $d = 0$, if $\gcd(u_1, u_2) = u_1$ (equivalently $m_3 = 0$) and assuming $[D_1] \neq [D_2]$, then for Steps 3-12, there are two possible cases: either all four points being compared are opposite, or two are opposite and two are common, (since all four points being common implies a doubling scenario.) If the divisor classes contain all opposites, then $v_1 = (-v_2 - h) \pmod{u_1}$ and equivalently $d_{w_2} = (v_2 + v_1 + h) \pmod{u_1} = 0$, resulting in a return of the neutral divisor class. Otherwise the

Subroutine 38 Genus 2 Ramified Model Degree 2 Addition ($d = 0$)

```
1:  Let and $d_{w_1} = u_1 \pmod{u_2} = -m_3 x + m_4$.
2:  if $m_3$ is 0 then
3:      $d_{w_2} = (v_2 + v_1 + h) \pmod{u_1}$.
4:      if $d_{w_2}$ is 0 then return $[1, 0]$.
5:      else
6:          $k = (f - v_1(v_1 + h))/u_1 \pmod{u_1}$.
7:          $d_{w_2} = \mathrm{monic}(d_{w_2})$ with $b_2 = 1/\mathrm{lcf}(d_{w_2})$.
8:          $u_1 = u_1/d_{w_2}$
9:          $s = b_2 k \pmod{u_1}$.
10:         $u_n = u_1^2$.
11:         $v_n = (v_1 + s u_1) \pmod{u_n}$.
12:         return $[u_n, v_n]$.
13: else
14:     Compute weighted $d_{w_3} = (v_2 + v_1 + h) \pmod{d_{w_1}}$.
15:     if $d_{w_3}$ is 0 then
16:         $u_c = \mathrm{monic}(d_{w_1})$ with $a_1 = 1/\mathrm{lcf}(d_{w_1})$.
17:         $u_1 = u_1/d_{w_1}$.
18:         $s = a_1(v_2 - v_1) \pmod{u_2}$.
19:         $u_n = u_1 u_2$.
20:         $v_n = (v_1 + s u_1) \pmod{u_n}$.
21:         return $[u_n, v_n]$.
22:     else
23:         $k = (f - v_1(v_1 + h))/u_1 \pmod{u_1}$.
24:         $a_{12} = a_1(1 - b_2(v_1 + v_2 + h))/d_{w_1} \pmod{u_2}$ with weight $m_3 d_{w_3}$
25:         $s' = s'_1 x + s'_0 = d_{w_3} s_1 x + d_{w_3} s_0 = (a_{12}(v_2 - v_1) + m_3^3 k) \pmod{u_2}$, with weight $d_{w_3}$.
26:         Go to Step 5 of Degree 2 Addition Algorithm 37, with $d = d_{w_3}$.
```

opposite point is extracted through an exact division by $\mathrm{monic}(d_{w_2})$ and computed similarly as in Subroutine 35.

With $d = 0$, if $m_3 \neq 0$, and equivalently $\gcd(u_1, u_2) \neq u_1$, then only one point computed via $\gcd(u_1, u_2)$, is common or opposite. In Steps 14–15, $d_{w_3} = (v_2 + v_1 + h) \pmod{d_{w_1}}$ is computed to check for the existence of an opposite point. If $d_{w_3} = 0$, then in Steps 16–17 the opposite point is extracted through an exact division by $u_c = \mathrm{monic}(d_{w_1})$ and computed similarly to the special degree 2 doubling case, but two degree 1 divisors are added. In Step 18, the polynomial $s = a_1(v_2 - v_1) \pmod{u_2}$ is computed via a school book modular reduction and Steps 19–20 compute $u_n = u_1 u_2$ and $v_n = (v_1 + u_1 s_0) \pmod{u_n}$ via a school book polynomial multiplication and modular reduction respectively.

Otherwise for Steps 23–25, $d_{w_3} \neq 0$ and a weighted $s' = s'_1 x + '_0$ is computed as $s = (a_2 a_1(v_2 - v_1) + b_2 k) \pmod{u_2}$, omitting the inversion required for $a_1 = 1/{-m_3}$. First

$$a_{12} = -a_1 a_2 = m_3^2(h_2 u_{21)-v_{21}-t_1})$$

where $a_2 = (1 - b_2(v_1 + v_2 + h) \pmod{u_2})/\text{monic}(d_{w_1})$ is computed with weight $m_3 d_{w_3}$. Then the k_2, k_1, k_0 coefficients of $k = (f - v_1(v_1 + h)/u_1)$ are computed as described in degree 2 ramified doubling. Finally, $s = a_2 a_1(v_2 - v_1) + b_2 k) \pmod{u_2}$ is computed using the above values. The explicit formula for Steps 23–25 is

$$\begin{aligned}
a_{12} &= M_3(h_2 u_{21} - v_{21} - t_1),\\
t_2 &= k_2 - u_{21},\\
t_0 &= m_3 M_3,\\
s'_1 &= t_0(k_1 - u_{20} - u_{21} t_2) - a_{12}(v_{21} - v_{11}),\\
s'_0 &= t_0(k_0 - u_{20} t_2) - a_{12}(v_{20} - v_{10}),\\
d_1 &= m_3 d_{w_3},
\end{aligned}$$

with $t_1 = v_{11} + h_1$ and $M_3 = m_3^2$ computed earlier. The rest of the computations are identical to the frequent case of degree 2 addition starting at Step 5 of Algorithm 37 with $d = d_{w_3}$.

Genus 2 Ramified Model Degree 1 and 2 Addition

Let $[u_1, v_1]$ and $[u_2, v_2]$ be reduced divisor class representatives with $u_1 = x^2 + u_{11}x + u_{10}$, $v_1 = v_{11}x + v_{10}$ and $u_2 = x + u_{20}$, $v_2 = v_{20}$. Algorithm 27 is specialized to degree 1 and 2 addition by computing the resultant $d = \text{Res}(u_1, u_2)$ instead of $S = \text{XGCD}(u_1 u_2)$ in Step 2 of Algorithm 27, where $d = 0$ implies $S \neq 1$. In this case $a_1 = 1/u_1 \pmod{u_2}$ and so $s = (v_2 - v_1)/u_1 \pmod{u_2}$ in Step 11 of Addition Algorithm 27. The computation of k in Step 1 of Algorithm 27 is not required in all cases and is pushed further into the algorithm. The basic formulation for degree 1 and 2 addition is described in Algorithm 39. An explanation of how each step in Algorithm 39 is converted to an explicit formula is given in the following.

Algorithm 39 Genus 2 Ramified Model Degree 1 and 2 Addition

Input: $[D_1]$ = degree 2 $[u_1, v_1]$, $[D_2]$ = degree 1 $[u_2, v_2]$,

Output: $[u_n, v_n] = [D_1] + [D_2]$.

```
1: $d = u_1 \pmod{u_2}$.
2: **if** $d$ is 0 **then** Go to Degree 1 and 2 Addition $d = 0$ Subroutine 40.
3: $a_1 = d^{-1}$.
4: $s_0 = a_1(v_2 - v_1) \pmod{u_2}$.
5: $k = ((f - v_1(v_1 + h))/u_1) \pmod{u_1}$.
6: $\tilde{v} = -s_0 u_1 - v_1 - h$.
7: $u_n = (s_0(\tilde{v} - v_1) + k)/u_2$   (exact division).
8: $v_n = \tilde{v} \pmod{u_n}$.
9: **return** $[u_n, v_n]$.
```

In Step 1, the resultant $d = u_1 \pmod{u_2}$ is computed via a school book modular reduction with a degree one modulus. If $d \neq 0$, then in Steps 3–4 $s = s_0 = a_1(v_2 - v_1) \pmod{u_2}$ is computed via a school book modular reduction and an inversion $a_1 = 1/d$. In Step 5, the polynomial k is monic and has degree 3 and the polynomial $\tilde{v}$ has degree 2. Only the degree one and two coefficients of k and $\tilde{v}$ are required in the computation of u_n, where the computation of $\tilde{v}$ and u_n are combined and computed via an exact division in Steps 6–7. In this case, the output u_n polynomial computed is already monic, since k is monic. In Step 8 v_n is computed via a school book modular reduction of a degree 2 polynomial modulo a degree 2 polynomial where the Karatsuba multiplication pattern does not arise (see Section 4.3.4).

A basic formulation of the case $d = 0$ within degree 1 and 2 addition is presented in Subroutine 40. An explanation of how each step in Subroutine 40 is converted to an explicit formula is given in the following.

If d is 0, then either the point corresponding to u_2 or its opposite is in the support of $[u_1, v_1]$. This case is a subset of the Degree 2 Addition for the case $d = 0$ using Subroutine 38.

Genus 2 Ramified Model Degree 1 Addition

Let $[u_1, v_1]$ and $[u_2, v_2]$ be reduced divisor class representatives with $u_1 = x + u_{10}$, $v_1 = v_{10}$ and $u_2 = x + u_{20}$, $v_2 = v_{20}$. Steps 2–13 of Algorithm 27 are specialized to degree 1 addition by computing the resultant $d = \mathrm{Res}(u_1, u_2)$ instead of $S = \mathrm{XGCD}(u_1 u_2)$ in Step 2 of Algorithm 27,

Subroutine 40 Genus 2 Ramified Model Degree 1 and 2 Addition ($d = 0$)

```
1: $d_w = (v_1 + v_2 + h) \pmod{u_1}$.
2: if $d_w$ is 0 then
3:     $u_n = u_1/u_2$   (exact division).
4:     $v_n = v_1 \pmod{u_n}$.
5:     return $[u_n, v_n]$.
6: else
7:     $k = ((f - v_1(v_1 + h))/u_1) \pmod{u_1}$.
8:     $b_2 = d_w^{-1}$.
9:     $s_0 = b_2 k \pmod{u_2}$.
10:    Go to Step 3 of Degree 1 and 2 Addition Algorithm 39, with $d = d_w$.
```

where $d = 0$ implies $S \neq 1$. In this case $a_1 = 1/u_1 \pmod{u_2}$ and so $s = (v_2 - v_1)/u_1 \pmod{u_2}$ in Step 11 of Addition Algorithm 27. The computation of k in Step 1 of Algorithm 27 is not required at all. After the computation of s in Step 11, the output polynomials u_n and v_n are computed by Step 13 of Algorithm 27. The basic formulation for degree 1 addition is described in Algorithm 41. An explanation of how each step in Algorithm 41 is converted to an explicit formula is given in the following.

Algorithm 41 Genus 2 Ramified Model Degree 1 Addition
Input: $[D_1] = [u_1, v_1]$, $[D_2] = [u_2, v_2]$
Output: $[u_n, v_n] = [D_1] + [D_2]$.

```
1: $d = u_1 \pmod{u_2}$.
2: if $d$ is 0 then return $[1, 0]$.
3: $a_1 = d^{-1}$.
4: $s = a_1(v_2 - v_1)$.
5: $u_n = u_1 u_2$.
6: $v_n = (v_1 + s u_1) \pmod{u_n}$.
7: return $[u_n, v_n]$.
```

In Step 1, the resultant $d = u_1 \pmod{u_2}$ is computed via a school book modular reduction with a degree one modulus. If $d \neq 0$, then Steps 3–6 are computed with no techniques used. In the special case that $d = 0$, the only possibility is that the two divisor classes were inverses of each other and the neutral element $[1, 0]$ is returned.

5.1.6 Field Operation Costs and Comparisons

Field operation costs of the algorithms presented above are given in Table 5.1 and are compared to previous methods in Table 5.2. Standard practice is followed for describing the cost of the formulas in terms of field operations where M denotes a multiplication, S a squaring, C a multiplication by a constant curve coefficient and A an addition. Unlike many previous works, addition counts are included, as these have been shown to have non-trivial cost working with large fields [16], and significant cost relative to multiplications working with word sized fields [37]. Recall that based on the findings of Sutherland [36, 37], the techniques that we applied to any explicit formula never trade a multiplication for more than 3 additions based on [36].

All divisor class operations require one inversion, and the field operation counts presented are only for the frequent cases. Field operation counts can be reduced if assumptions about the characteristic of the base field are made. Three base field and curve equation cases are presented; the generic ramified curve equation over arbitrary fields, and the curve equation assumptions described in Section 5.1.1 where $h_2 = h_1 = h_0 = 0$ for fields with characteristic not equal to 2 and $h_2 = 1$, $f_4 = f_3 = f_2 = 0$ for fields with characteristic 2. Operation counts for all formulas from this section are given in Table 5.1.

Table 5.1: Frequent case field operation counts of all one inversion operations required for complete divisor class group arithmetic over genus 2 hyperelliptic curve ramified models.

	Arbitrary				**char$(k) \neq 2$**				**char$(k) = 2$**			
Operations	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**
Doubling												
Degree 1	4	1	15	3	3	1	9	1	2	2	5	2
Degree 2	22	4	42	2	21	5	25	0	21	4	25	0
Addition												
Degree 1	3	0	4	0	3	0	4	0	3	0	4	0
Degree 1 and 2	9	1	22	0	8	2	15	0	8	1	19	0
Degree 2	21	2	31	0	21	2	23	0	20	3	26	0

Comparisons of explicit formulas from this work to the previous best over genus 2 ramified models are presented in Table 5.2. Operation count comparisons for doubling degree 2 divisors

(2DBL), adding degree 2 divisors, degree 1 with 2, and degree 1 with 1 (2ADD, 12ADD, 1ADD) are presented. Although the contributions of this thesis are for the simpler 4-coordinate affine (labelled by A4 in the table), the 6-coordinate "linear algebra" setting [6] (labelled by A6 in the table) over fields with characteristic not equal to 2 is included for comparison.

Table 5.2: Field operation comparisons for frequent case divisor class group operations on genus 2 hyperelliptic curve ramified models.

Arbitrary Fields																				
	1DBL				**2DBL**				**1ADD**				**12ADD**				**2ADD**			
Previous Work	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**
A4 - Lange [24]	-	-	-	-	22	6	56	3	-	-	-	-	10	1	25	0	22	3	40	0
A4 - This work	4	1	15	3	22	4	42	2	3	0	4	0	9	1	22	0	21	2	31	0

char(k) = 2 Fields																				
	1DBL				**2DBL**				**1ADD**				**12ADD**				**2ADD**			
Previous Work	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**
A4 - Lange [24]	-	-	-	-	22	5	33	0	-	-	-	-	10	1	23	0	22	3	33	0
A4 - This work	2	2	5	2	21	4	25	0	3	0	4	0	8	1	19	0	20	3	26	0

char$(k) \neq 2$ Fields																				
	1DBL				**2DBL**				**1ADD**				**12ADD**				**2ADD**			
Previous Work	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**
A4 - Lange [24]	-	-	-	-	22	5	36	0	-	-	-	-	10	1	18	0	22	3	32	0
A6 - Costello *et al.* [6] (No Trades)	-	-	-	-	20	6	41	0	-	-	-	-	-	-	-	-	18	4	32	0
A4 - This work	3	1	9	1	21	5	25	0	3	0	4	0	8	2	15	0	21	2	23	0

Comparison Considerations

The authors of [6] use an algorithm that solves a 2×2 linear system with $5M$ instead of $6M$ at the cost of $13A$. Later publications by the same authors [16, 2] show that this trade-off and others do not result in faster divisor class operations and are omitted. These observations are supported by numerical analysis provided in Section 5.3, where the trade-off is shown to be strictly worse in practice. Therefore, the trade is omitted in the presentation of the new formulas and comparisons, so the linear algebra formula counts labelled A6 in Table 5.1 are without trades for fair comparison. Attempts to expand the linear algebra 6-coordinate setting to characteristic 2 and arbitrary fields proved to be not as efficient as the 4-coordinate affine setting. The introduction of the non-zero $h(x)$ polynomial increases the number of field operations significantly [25].

Frequent Case Comparisons

Over arbitrary fields and in the frequent cases, the new formulas improve on previous best by **2S + 1C + 14A** for doubling and **1M + 1S + 9A** for degree 2 addition. Over characteristic 2 fields, the new formulas improve on previous best by **1M + 1S + 8A** for doubling and **2M + 7A** for addition. Over characteristic not equal to 2 fields, the new doubling formula requires an additional **1M** but saves **1S** and **16A**, and addition formula requires an additional **3M** but saves **2S** and **9A** compared to the 6 coordinate linear algebra formulas. When compared to previous best 4 coordinate formulas, the formulas save **1M + 11A** and **1M + 1S + 9A** for doubling and addition respectively.

Special Case Comparisons

All implementations of frequent case arithmetic include explicit handling of special cases where the input divisor class representatives have points in common (or are opposite) as they come up naturally in the computations. Rather than making calls to Cantor's Algorithm, the special cases are computed explicitly and with only one inversion. These cases are made as fast as possible given the techniques that can be applied. These special cases are not analyzed here because there is no previous best to compare to.

Degree 1 input addition and doubling are simple cases and there is little room for advanced techniques. For degree 1 with degree 2 addition, a few additions and 1-2 multiplications were saved in all settings. For special cases where the output divisor class representative has degree less than 2, i.e. $s = s_0$, 1 to 2 field multiplications or squarings are saved, and a few field additions in all settings when compared to [24]. Note that there is a typo on page 10 of [24], under the section $s = s_0$, a multiplication is missed in Step 6' and should read $w_2 = s_0u_0 + v_{20} + h_0$, bringing the cost up to I,2S,12M.

5.2 Improved Genus 2 Split Model Explicit Formulas

In this section, novel explicit formulas for addition and doubling of divisor classes on genus 2 hyperelliptic curves represented by split models in the balanced setting are described. In general, the curve equation of a split model for a genus 2 hyperelliptic curve in Weierstrass form is $C : y^2 + h(x)y = f(x)$ where

$$f = f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0, \qquad h = h_3x^3 + h_2x^2 + h_1x + h_0,$$

where f_6 is a square in k if $\mathrm{char}(k) \neq 2$ and $f_6 = \beta^2 + \beta$, for some $\beta \in k$ otherwise. The polynomial $V^+ = V_3^+x^3 + V_2^+x^2 + V_1^+x + V_0^+$ such that $\deg(f - V^+(V^+ + h)) \leq 2$ (see Definition 2.1.42) is precomputed using Algorithm 42 that outputs a table of precomputations and curve coefficients. Input and output balanced divisors are represented as 5-coordinate vectors of four field elements and a balancing coefficient n as

$$D = [u_1, u_0, v_1, v_0, n],$$

for the Balanced Mumford representation $D = [u, v, n]$ with $u = x^2 + u_1x + u_0$ and $v = v_1x + v_0$ in positive reduced basis (Section 3.1.3).

In contrast to previous work designed for arithmetic in the infrastructure of a genus 2 split model [9], the formulas of this work are complete and are designed to be efficient for arithmetic in the divisor class group (using balanced divisor representatives) as opposed to the infrastructure. Not only do the frequent case formulas require fewer field operations in all underlying field considerations than in [9], but the framework used based on balanced divisor class representatives [11], encapsulates all adjustments in special cases, resulting in formulas that require only one inversion given any two divisor classes as input. Applying many of the same techniques as for ramified model formulas, and introducing some novel ones, complete explicit formulas are developed including all special cases based on Algorithms 29 and 28.

Recall that the main difference between the balanced arithmetic and Cantor's ramified arithmetic is that there is no analog for Balanced Adjust Algorithm 7 in Cantor's framework. For genus 2 split models, and even genus in general, non-trivial calls to adjust only occur for special cases, i.e.

adjust is never called in any frequent case. This effectively makes balanced arithmetic almost the same as Cantor's ramified arithmetic over fields with large cardinality, the main difference being the degree of the polynomials that define the curve, where $\deg f = 6$ instead of 5, and $\deg h = 3$ instead of 2. In the frequent cases, these differences affect the computation of k in Algorithm 34 as this computation is dependent on f and h, and the computation of the resulting Mumford polynomial u in both Algorithms 34, 37 directly. The discrepancy in field operations between ramified and split models comes from these conditions.

Another difference is the use of a reduced basis over split models. A reduced basis keeps the degree of the $k = f - v(v + h)$ polynomial less than or equal to 2 via cancellations, in contrast to degree 3 over ramified model curves. Computing a reduced basis for the output v polynomial does sometimes require more field operations when compared to using no reduced basis, but overall help the discrepancy in operation counts between the two settings. Moreover, the two leading coefficients of v in any reduced basis only rely on curve parameters and therefore still only 4 field elements are required to represent a divisor class.

This section is structured as follows; first, assumptions about curve models over certain fields are discussed, followed by an overview of prior work. This is followed by a statement of the novel contributions in this work and a description of the basic formulations for doubling and addition. Finally, operation costs in terms of field operations, comparisons to previous work, and empirical benchmarking results are presented.

5.2.1 Curve Simplifications

Recall from Section 5.1.1 that isomorphic transformations of curves, that transform a curve into another isomorphic curve with a less complex equation, can be applied depending on the characteristic of the underlying field of the hyperelliptic curve. Next, the isomorphic transformations that are possible over fields of characteristic two and characteristic not equal to two are described; for more details see [9, Section 3].

char(k) ≠ 2

If char(k) ≠ 2, then applying the substitution $y \to y - h(x)/2$ invokes the transformation

$$\begin{aligned} y^2 + h(x)y = f(x) &\to (y - h(x)/2)^2 + h(x)(y - h(x)/2) = f(x) \\ &\implies y^2 - 2h(x)y/2 + (h(x))^2/4 + h(x)y - 2(h(x))^2/4 = f(x) \\ &\implies y^2 = f'(x) \end{aligned}$$

where $f' = f + (h(x))^2/2$. This transformation is always possible when char(k) ≠ 2, allowing for the assumption of $h = 0$ in this case. If the characteristic of the underlying field is also not three, the transformation $x \to x - f_5/6$ that takes $f(x) \to f(x)'$ to get

$$f(x)' = x^6 + f_4'x^4 + f_3'x^3 + f_2'x^2 + f_1'x + f_0'$$

can be applied, allowing one to assume that $f_5 = 0$ in that case. Operation counts in Section 5.2.8 assume $h = 0$ whenever char(k) ≠ 2, but make no assumption about f_5.

char(k) = 2

If char(k) = 2, then division by two is not possible and the transformation

$$y \to y - \frac{h(x)}{2}$$

cannot be made. The polynomial $h(x)$ is a monic degree 3 polynomial, $\deg(f) = 6$ and $\mathrm{lcf}(f)$ is of the form $s^2 + s$ for some $s \in k^*$. The isomorphic transformation

$$x \to x + h_2, \quad y \to y + f_5x^2 + (f_4 + h_2f_5 + f_5^2 + h_2^2f_6)x + (f_3 + f_5(h_1 + h_2^2))$$

results in $h_2 = 0$ and $f_5 = f_4 = f_3 = 0$. This curve equation is assumed when counting operations in Section 5.2.8 over a base field k with char(k) = 2.

5.2.2 Prior Work

In [9] the authors present explicit formulas for divisor class arithmetic on genus 2 split models over arbitrary fields, characteristic 2 fields, and fields of characteristic not equal to 2. The formulas are not presented for divisor class arithmetic, but only the related (but different) infrastructure of a genus 2 curve split model. Adapting the formulas to the divisor class group in the balanced setting can be achieved, but requires accounting of the balancing coefficient n and invoking multiple operations to replace a single operation in this work.

The authors of [9] use a slightly different definition of Mumford representation, where $u \mid f + hv - v^2$ instead of $u \mid f - hv - v^2$. This difference does not change how the divisor class arithmetic works as long as Cantor's algorithm is adapted to reflect this change. The difference in Mumford representation does however affect the normalization of the v polynomial in positive reduced basis. An alternate reduced normalization of the v polynomial for a divisor class $[u, v]$ is used by the authors, where

$$\begin{aligned} u' &= u \\ v' &= V^+ + h - [(V^+ + h - v) \pmod{u}]. \end{aligned} \tag{5.2.1}$$

This is not the same as the positive reduced representation used in this work, where the positive reduced basis is slightly simpler. The authors also label the regular Mumford representation for v as *adapted basis* and report that in the frequent cases and almost all special cases, the use of a reduced basis produces explicit formulas that require fewer field operations over the adapted basis.

5.2.3 Summary of Contributions

Similar to the ramified setting, this work provides complete explicit formulas, in the sense that all computation paths are explicitly computed, requiring only one inversion in all cases that take reduced divisor class representatives as input, and output a reduced divisor class representative. Every special case is explicitly computed, and frequent case addition and doubling require fewer operation counts over previous best. All formulas are implemented in Magma including a testing suite to ensure their correctness. The implemented formulas and testing scripts are available at `https://github.com/salindne/divisorArithmetic`.

Formulas presented here make use of a positive reduced basis normalization for v rather than negative reduced basis used for general genus computations in Chapter 3. Complete addition and doubling have equal net costs comparing the two, where operations that include up and down adjustments mirror each other in cost between the two settings, aligning with polynomial arithmetic timing results from Section 3.3.4. Moreover, the work in [9] uses a reduced basis similar to the positive reduced basis, and so the use of a positive reduced basis facilitates clearer comparisons.

The formulations of divisor class addition and doubling are based on Balanced NUCOMP specialized to genus 2 addition and doubling using a positive reduced basis that include efficient formulations of special cases. The formulation of steps in Algorithms 29 and 28, coupled with certain explicit techniques produce the fastest explicit formulas for frequent case addition and doubling to date. The formulas incorporate different techniques borrowed from previous works and also introduce a new technique presented in Section 4.4.2.

Explicit formulas for balanced divisor class addition and doubling over split models use the novel technique T1 from Section 5.1.3 and in addition:

T5 Explicitly compute balanced divisor representations, keeping track of the balancing coefficient n (the number of points at infinity ∞_+ in the support), resulting in explicit divisor class arithmetic in the divisor class group, rather than over the infrastructure as in [9].

T6 Apply a positive reduced basis:

$$v' = V_3^+ x^3 + V_2^+ x^2 + v_1 x + v_0,$$

such that for an adapted basis v, the reduced basis of the same divisor is:

$$v' = V^+ - [(V^+ - v) \pmod{u}].$$

This reduced basis corresponds to the definition of Mumford representation in this work and simplifies precomputations when compared to [9].

T7 Use specialized genus 2 versions of Balanced NUCOMP and NUDUPL (Algorithms 29 and 28) as the basis for all explicit formulas including all special cases that require adjustments; both degree 1 doubling (Algorithms 45 and 46), both degree 1 and 2 addition (Algorithms 49 and 50), degree 1 addition with down adjustment (Algorithm 54), degree 1 addition with up adjustment (Algorithm 55), and both degree 0 with 1 or 2 addition (Algorithms 56 and 57). Applying Balanced NUCOMP instead of Balanced ADD (Algorithm 15) in all these cases but degree 0 additions avoids either a second inversion or pushes weighted values from the composition step to the adjustment step with Montgomery's inversion trick, resulting in a significant reduction in the number of field operations required. Applying Balanced NUCOMP to degree 0 additions results in similar computation costs as applying Balanced Adjust (Algorithm 7), but allows for the use of Technique T8.

T8 Apply an alternate computation of the output polynomial u in all cases that Balanced NUCOMP is applied, and the degree 2 doubling case as described in Section 4.4.2.

T9 Include explicit one inversion formulas for all special cases with $\gcd(u_1, u_2, v_1 + v_2 + h) \neq 1$ based on Algorithm 21, producing much more efficient formulas when compared to previous suggestions [9].

Recall that the threshold for trading field multiplications for field additions is capped at 1 multiplication for 3 additions throughout (see Section 4.3.1.)

Special Cases

Implementations of frequent case arithmetic include explicit handling of special cases as they come up naturally in the computations via Technique T9. All computation paths through special cases in the formulas require only one inversion and are based on Algorithms 29 and 28. Suggestions from previous work [9], (based on the ramified model suggestions of [24]), for handling these cases produced far more cumbersome explicit formulas compared to using the specialization of Balanced NUCOMP to genus 2; requiring between 20 and 40 percent more field operations. The reduction in field operations can be attributed to applying continued fraction steps from Balanced NUCOMP for adjustment computations and the omission of invoking lower polynomial degree algorithms with weighted inputs.

5.2.4 Explicit Formulations

The basic formulation of the frequent cases of degree 2 Addition and Doubling algorithms are similar to their ramified counterparts (Algorithms 37 and 34). The major differences lie in the second last step where u is computed and the last step where the reduced basis of v is computed instead of an adapted basis. We restate them here as Algorithms 47 and 43 with v in positive reduced basis as input, and note that in frequent cases, n remains unchanged ($n = 0$). Several new arithmetic cases with built-in adjustments are required in the divisor class group over split models. These special cases can be invoked based on the value n of both input divisors as described in Section 4.1.3. There are three degree 0 divisors possible, $n = 0, 1, 2$, of which only one (when $n = 1$) is the divisor class groups neutral element. Note that the results of degree 0 divisor doubling, and degree 0 with degree 0 addition are completely precomputed, more details are provide in the next section.

5.2.5 Precomputations

Since there is a need to compute and store the coefficients of V^+ in addition to the coefficients of the curve equation, and the extra complexity of split model arithmetic admits many curve constant computations including degree 0 compositions completely dependent on curve constants, a precomputation table is used to omit computing the same values multiple times. The use of a positive reduced basis as discussed in Technique T6 does require storing fewer precomputed values relative to the alternate reduced basis used in [9, Table 1], but four new precomputations related to Technique T8 are introduced.

As stated above, in order to explicitly compute complete addition and doubling, precomputed results for addition and doubling primitive divisors $[1, V^+, 2]$ and $[1, V^+, 0]$ are required. Depending on the polynomials f and h that define the curve, the output divisor from composing those divisor classes may be degree 2, 1, or 0. For genus 2 split model curves, $V^+ = V_3^+x^3 + V_2^+x^2 + V_1^+x + V_0^+$ is computed as described in Definition 2.1.42. Given f and h, V^+ can be computed explicitly, where the leading coefficient of V^+, V_3^+, is set to be a solution to the quadratic equation $V_3^+(V_3^+ + h_3) = f_6$. The precomputed c_i, and d_i are used throughout the explicit formulas, and the coefficients of the polynomials a and b are used for degree 0 doubling, and addition.

Algorithm 42 Genus 2 Split Model Precomputation

Input: $h = h_3x^3 + h_2x^2 + h_1x + h_0,\ f = f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$.

```
1: $V_3^+$ is a solution to the quadratic equation $(V^+)3)^2 + h_3V_3^+ = f_6$.
2: $c_3 = 2V_3^+ + h_3, \quad d_5 = 1/c_3$.
3: $d_4 = f_4 - V_3^+h_1, \quad d_3 = f_3 - V_3^+h_0$.
4: $V_2^+ = d_5(f_5 - V_3^+h_2)$.
5: $V_1^+ = d_5(d_4 - V_2^+(V_2^+ + h_2))$.
6: $c_1 = V_1^+ + h_1, \quad c_5 = c_1 + V_1^+$.
7: $V_0^+ = d_5(d_3 - V_2^+c_5) - V_1^+h_2)$.
8: $c_0 = V_0^+ + h_0, \quad c_2 = 2V_2^+ + h_2$.
9: $c_4 = c_2 + c_3, \quad d_2 = f_2 - V_2^+h_0 - V_0^+c_2$.
10: $d_1 = f_2 - V_2^+h_0 - V_1^+c_1, \quad d_0 = f_1 - V_1^+h_0$.
11: $d_6 = d_5c_2, \quad d_7 = d_5d_0, \quad d_8 = d_5d_1, \quad d_9 = d_5c_5$.
12: Compute $a = a_2x^2 + a_1x + a_0 = (f - V^+(V^+ + h)$ (made monic).
13: Compute $b = b_1x + b_0 = V^+ - (2V^+ - h) \pmod{a}$.
14: return $[[f_0, f_1, f_2, f_3, f_4, f_5, f_6],[h_0, h_1, h_2, h_3],[V_0^+, V_1^+, V_2^+, V_3^+],$
        $[d_0, d_1, d_2, d_3, d_4, d_5, d_6],[c_0, c_1, c_2, c_3, c_4, c_5],[a_0, a_1, a_2, \deg(a), b_0, b_1]]$
```

5.2.6 Explicit Formulas for Doubling

In this section, complete divisor class doubling is described and the basic formulations of the corresponding explicit formulas are provided. Doubling Algorithm 28 is further specialized into different cases depending on the input divisor class degree and balancing coefficient. Any explicit techniques used are discussed in each case.

The complete doubling algorithm should test the degree of the input divisor in order of statistical frequency and choose the appropriate algorithm depending on the input balancing coefficient n_1. The three doubling formulas used for complete doubling are:

- Degree 2 doubling
- Degree 1 doubling with down adjustment
- Degree 1 doubling with up adjustment.

Only precomputed values are required for degree zero doubling, and precomputations from Algorithm 42 may be used. In the frequent cases, the output balancing coefficient n is always zero, and in all special cases n is computed via lines 5,16–18 of Algorithm 28. Basic formulations of the algorithms required for complete doubling, organized by the degree of the input divisor, are described in the following.

Balancing Coefficient

Let $[u_1, v_1, n_1]$ be the input reduced divisor class representative. For all doubling formulations presented, generically $n_n = 2n_1 + \deg(S) - 1$ if no reduction or adjustment is required, and $n_n = 2n_1 + \deg(S) - 1 + \deg(z) - \deg(u_n)$ if a reduction or adjustment step is required (with S, z, u_n as described in Doubling Algorithm 28.) In the latter, an additional case for computing n_n is possible. If $\deg(z) < 3$, then Steps 17–19 are performed instead of Step 20 in Doubling Algorithm 28, where $n_n = 2n_1 + \deg(S) - 1 + 3 - \deg(u_n)$ if Step 2 of Doubling Algorithm 28 is computed normalizing the input divisor into negative reduced basis, or $n_n = 2n_1 + \deg(S) - 1 + 2\deg(u_1) - 3$ otherwise.

Genus 2 Split Model Degree 2 Doubling

Let $[u_1, v_1, n_1]$ be the input reduced divisor class representative with $u_1 = x^2 + u_{11}x + u_{10}$, $v_1 = V_3^+x^3 + V_2^+x^2 + v_{11}x + v_{10}$ and $n_1 = 0$. The basic formulation is similar to the ramified setting, but in addition computes the balancing coefficient n_n as described in Section 5.2.6). For degree 2 doubling, Steps 1–2 are always skipped and $n_1 = 0$. The basic formulation for degree 2 doubling is described in Algorithm 43. An explanation of how each step in Algorithm 43 is converted to an explicit formula is given in the following.

Degree 2 doubling in the balanced setting is similar to its ramified counterpart (Algorithm 34), so only the differences are described in the following. In Step 1, the normalization of input v_1 polynomial in positive reduced basis slightly alters the computations in the setup of the 2x2 matrix used to solve for s (see Section 4.3.6), where $\deg(2v_1 + h) = 3$ instead of 2 as in the ramified setting, changing the computation of the modular reduction by u_1 for $\tilde{v} = (2v_1 + h) \pmod{u_1}$.

Algorithm 43 Genus 2 Split Model Degree 2 Doubling

Input: $[D_1] = [u_1, v_1, 0]$

Output: $[u_n, v_n, n_n] = 2[D_1]$

```
 1: Compute $m_i$ in system for $s(h + 2v_1) \equiv z \pmod{u_1}$ as $\left[\begin{smallmatrix} m_4 & -m_2 \\ -m_3 & m_1 \end{smallmatrix}\right]\left[\begin{smallmatrix} s_0 \\ s_1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} k_0 \\ k_1 \end{smallmatrix}\right]$.
 2: $d = m_1m_4 - m_2m_3$.
 3: if $d$ is 0 then Go to Degree 2 Doubling ($d = 0$) Subroutine 44.
 4: $z = (f - v_1(v_1 + h))/(c_3u_1) \pmod{u_1}$.
 5: Solve for $s'_1 = ds_1/c_3, s'_0 = ds_0/c_3$ using the $m_i$ and Cramer's rule, omitting the inversion of $d$.
 6: if $s'_1 = 0$ then
 7:     if $s'_0 = 0$ then return $[1, V^+, 0]$.
 8:     else
 9:         Compute $w_3 = c_3/(s_0d^2) = (ds'_0)^{-1}$ and precompute values $c_3/s_0$ and $s_0/c_3$.
10:         $u_n = (s_0/c_3 + (2v_1 + h)/c_3 - (s_0/c_3)(z/c_3))/u_1)$.
11:         $v_n = V^+ - [(V^+ + c_3(s_0/c_3)u_1 + v_1 + h) \pmod{u_n}]$
12:         return $[u_n, v_n, 0]$
13: $w_1 = s'_1(s'_1 + d)$.
14: if $w_1 = 0$ then
15:     $w_0 = s'_1(v_{11} + h_1) - s'_0$.
16:     if $w_0 = 0$ then return $[1, V^+, 2]$.
17:     else
18:         Compute $w_2 = (w_1d)^{-1}$ and precompute values $c_3/(s_1(v_{11} + h_1) - s_0)$ and $s_0/c_3$.
19:         $u_n = ((s_0/c_3)(s_0/c_3 + (2v_1 + h)/c_3) - z/c_3)/u_1)$,    (made monic).
20:         $v_n = V^+ - [(V^+ + c_3(s_0/c_3)u_1 + v_1 + h) \pmod{u_n}]$
21:         return $[u_n, v_n, 1]$
22: Compute $w_2 = (dw_1)^{-1}$ and precompute value $1/(s_1^2 + s_1)$.
23: $s'' = s_1/c_3x + s_0/c_3$ with $s''_1 = s'_1/d$ and $s''_0 = s'_0/d$.
24: $u_n = (s''(s'' + (2v_1 + h)/c_3) - z/c_3)/u_1)$,    (made monic).
25: $v_n = V^+ - [(V^+ + c_3s''u_1 + v_1 + h) \pmod{u_n}]$
26: return $[u_n, v_n, 0]$.
```

With $d \neq 0$, another difference arises in Step 4. The computation of the polynomial $k = (f - v_1(v_1 + h))/u_1$ is simplified via the reduced basis of v_1, resulting in $\deg(k) = 2$ instead of 3 as in the ramified setting. In this cases, Technique T8 is applied where $z = k/c_3 = z_1x + z_0$ is computed as follows. The values z_3 and z_2 omit a field multiplication by c_3 and $d_5 = 1/c_3$, $d_6 = c_2/c_3$ are used instead of 1, and c_2 required in the computation of k. The resulting explicit formula is

$$z_3 = V_0^+ - v_{10},$$

$$z_2 = V_1^+ - v_{11},$$

$$z_1 = z_3 - z_2(2u_{11} - d_6),$$

$$z_0 = d_6 z_3 + d_5(d_2 - v_{11}(v_{11} + h_1)) - u_{11}z_1 - z_2(u_{11}^2 + 2u_{10}).$$

In Step 5, $s' = ds/c_3$ is computed identically to ramified degree 2 doubling (Algorithm 34), where the extra weight of $1/c_3$ results from previously computing $z = k/c_3$ instead of k.

If $d \neq 0$, there are two special cases where the output divisor class representative has degree less than 2. Step 6 catches the first case, where $\deg(s') \leq 1$ is computationally similar to the ramified setting, and $n_n = 0$ described in Section 5.2.6. In contrast to the ramified setting, the unnormalized u_n polynomial has leading coefficient $s_1c_3 + s_1^2$ instead of s_1^2. In Step 13, the weighted leading coefficient is computed as

$$w_1 = s_1'(s_1' + d) \qquad ((d/c_3)^2(s_1^2 + s_1)).$$

The check for $w_1 = 0$ in Step 14 corresponds to the other special case when $s_1c_3 + s_1^2 = 0$, resulting in lower degree output polynomials u_n and v_n computed similarly to the ramified setting, and balancing coefficient $n_n = 2 - \deg(u_n) \neq 0$ as described in Section 5.2.6.

If $d \neq 0$ and $w_1 \neq 0$, Steps 22–25 compute the output polynomials u_n, v_n. In contrast to the ramified setting, the s polynomial is not normalized, as computation of u_n does not benefit from such an approach. Instead Technique T8 is applied with $z = k/c_3$ as described above, resulting in $s'' - s_1/c_3x + s_0/c_3$ used in the computation of u_n. First, the required weights for normalizing u_n, and $s'' = s_1''x + s_0'' = s_1/c_3x + s_0/c_3$ are computed. The resulting explicit formula for Steps 22–23 is

$$w_2 = (dw_1')^{-1},$$

$$w_3 = w_1w_2, \qquad (1/d)$$

$$D = d^2,$$

$$w_4 = w_2Dd, \qquad (c_3^2/(s_1^2 + s_1))$$

$$s_1'' = w_3s_1', \qquad (s_1/c_3)$$

$$s_0'' = w_3s_0'. \qquad (s_0/c_3)$$

In Step 24, the required output polynomial $u_n = s^2 + M_2$ needs a division by $(s_1^2 + s_1)$ to be normalized. Therefore

$$u_n = (c_3^2/(s_1^2 + s_1))((s'')^2 + M_2/(c_3^2)) = (c_3^2/(s_1^2 + s_1))(s''(s'' + (2v_1 + h)/c_3) - z/c_3)/u_1)$$

is computed all in one step utilizing an exact division (see Section 4.3.2) for the M_2 part and taking advantage of combining terms that need to be multiplied by the precomputed weights w_i. The resulting explicit formula for Step 24 is

$$t_4 = s_0 + d_6 - u_{11},$$

$$u_{n_1} = w_4((s_0 + t_4)s_1 + s_0),$$

$$u_{n_0} = w_4(t_4 s_0 - d_5(m_3 s_1 + z_1)).$$

In Step 25, the required output polynomial $v_n = V^+ - [(V^+ + su_1 + v_1 + h) \pmod{u_n}]$ is instead computed as $v_n = V^+ - [(V^+ + c_3 s'' u_1 + v_1 + h) \pmod{u_n}]$ via Karatsuba modular reduction (see Section 4.3.4). The resulting explicit formula for Step 25 is

$$t_2 = u_{n_0} - u_{10},$$

$$t_3 = u_{n_1} - u_{11},$$

$$w_0 = t_2 s_0,$$

$$w_1 = t_3 s_1,$$

$$w_2 = d_6 - w_1 - u_{n_1},$$

$$v_{n_1} = c_3((s_0 + s_1)(t_2 + t_3) - w_0 - w_1 + u_{n_0} + w_2 u_{n_1}) - \hat{v}_1,$$

$$v_{n_0} = c_3(w_0 + w_2 u_{n_0}) - \hat{v}_0,$$

where $\hat{v}_1$ and $\hat{v}_0$ are from the setup of the 2x2 matrix above.

The basic formulation for the degree 2 doubling $d = 0$ subroutine is described in Subroutine 44. The special case where $d = 0$ is almost identical the to ramified setting, but differs in the conversion of v_n into positive reduced basis, the computation of k as k/c_3, and how the balancing coefficient n_n is updated.

Subroutine 44 Genus 2 Split Model Degree 2 Doubling ($d = 0$)

```
1: Let $dw = (h + 2v_1) \pmod{u_1} = -m_3 x + m_4$.
2: if $m_3$ is 0 then return $[1, V^+, 1]$.
3: else
4:     $k = (f - v_1(v_1 + h))/u_1$.
5:     $b_1 = 1/\mathrm{lcf}(dw)$ and $\mathrm{monic}(dw)$.
6:     $u_1 = u_1/dw$.
7:     $s = b_1 k \pmod{u_1}$.
8:     $u_n = u_1^2$.
9:     $v_n = V^+ - (V^+ - v_1 - u_1 s) \pmod{u_n}$.
10:    return $[u_n, v_n, 0]$.
```

Genus 2 Split Model Degree 1 Doubling

Let $[u_1, v_1, n_1]$ be the input reduced divisor class representative with $u_1 = x + u_{10}$, $v_1 = V_3^+ x^3 + V_2^+ x^2 + V_1^+ x + v_{10}$ and n_1 either 1 or 0. In the balanced setting, there are two degree 1 doubling algorithms required for complete divisor class arithmetic; degree 1 doubling with down adjust, and degree 1 doubling with up adjust. There is always an adjustment after degree 1 doubling. Depending on the input balancing coefficient, if n_1 is 0 or 1, the resulting $n = 2n_1 - 1$ value is -1 or 1 respectively. The degree of the divisor after degree 1 doubling is 2, so an adjustment up or down is required to bring n to 0. The composition and adjustment steps are combined in Doubling Algorithm 28 as described in T7.

Recall from Section 3.3 that the normalization of v_1 dictates whether an up or down adjustment is computed, and is reflected by Steps 1–2 of Algorithm 28. Since v_1 is generally kept in positive reduced basis, v_1 already has the required normalization for a down adjustment. The only difference between the two degree 1 doubling formulations is that for the up adjustment formulation, $v' = -V^+ - h + ((V^+ + h + v_1) \pmod{u_1})$ is computed first. In all output cases, $\deg(u_1 s) < 3$, and so n_n is updated as $n_n = 1 + 2 - 3 = 0$ for Degree 1 Doubling with Down Adjust by Step 17 of Doubling Algorithm 26, and $n_n = -1 + 3 - \deg(u_n) = 2 - \deg(u_n)$ for Degree 1 Doubling with Up Adjust by Step 18 of Doubling Algorithm 26 (see Section 5.2.6.) The basic formulation for degree 1 doubling is described in Algorithm 45 for $n_1 = 1$ and in Algorithm 46 for $n_1 = 0$. An explanation of how each step in Algorithms 45 and 46 is converted to an explicit formula is given in the following.

In Step 0 of degree 1 doubling with up adjustment Algorithm 46, a negative reduced basis is

Algorithm 45 Genus 2 Split Model Degree 1 Doubling with Down Adjustment

Input: $[D_1] = [u_1, v_1, 1]$.

Output: $[u_n, v_n, n_n] = 2[D_1]$.

```
1: Compute $d = (h + 2v_1) \pmod{u_1}$
2: **if** $d$ is 0 **then return** $[1, V^+, 2]$.
3: $z = z_1x + z_0 = (f - v_1(v_1 + h))/(c_3u_1)$.
4: Compute $s_0' = ds = z \pmod{u_1}$.
5: **if** $s_0' = 0$ **then**
6:     **if** $z_1 = 0$ **then return** $[1, V^+, 0]$.
7:     $u_n = -z/u_1$   (made monic).
8:     $v_n = V^+ - (V^+ + v_1 + h) \pmod{u_n}$.
9:     **return** $[u_n, v_n, 0]$.
10: Compute $1/s_0$ and $s = s_0$.
11: $M_2 = (s(2v_1 + h) - z)/u_1$   (exact division).
12: $u_n = s''^2 + M_2$   (made monic).
13: $v_n = V^+ - (V^+ + su_1 + v_1 + h) \pmod{u_n}$.
14: **return** $[u_n, v_n, 0]$.
```

Algorithm 46 Genus 2 Split Model Degree 1 Doubling with Up Adjustment

Input: $[D_1] = [u_1, v_1, 0]$.

Output: $[u_n, v_n, n_n] = 2[D_1]$.

```
0: $v_1 = -V^+ - h + ((V^+ + h + v_1) \pmod{u_1})$.
1: Compute $d = (h + 2v_1) \pmod{u_1}$
2: **if** $d$ is 0 **then return** $[1, V^+, 0]$.
3: $z = z_1x + z_0 = (f - v_1(v_1 + h))/(c_3u_1)$.
4: Compute $s_0' = ds = z \pmod{u_1}$.
5: **if** $s_0' = 0$ **then**
6:     **if** $z_1 = 0$ **then return** $[1, V^+, 2]$.
7:     $u_n = -z/u_1$   (made monic).
8:     $v_n = V^+ - (V^+ + v_1 + h) \pmod{u_n}$.
9:     **return** $[u_n, v_n, 1]$.
10: Compute $1/s_0$ and $s = s_0$.
11: $M_2 = (s(2v_1 + h) - z)/u_1$   (exact division).
12: $u_n = s''^2 + M_2$   (made monic).
13: $v_n = V^+ - (V^+ + su_1 + v_1 + h) \pmod{u_n}$.
14: **return** $[u_n, v_n, 0]$.
```

computed for v_1 via a school book modular reduction with u_1. The resulting explicit formula for Step 0 is

$$\begin{aligned} t_0 &= u_{10}c_3, \\ t_1 &= u_{10}(c_5 - u_{10}(c_2 - t_0)), \\ v_{10} &= v_{10} - t_1, \end{aligned}$$

with $c_2 = 2V_2^+ + h_2$, and $c_5 = 2V_1^+ + h_1$ precomputed values. All other steps in both algorithms are almost identical, differentiating in the computation of n_n only.

In Step 2, the computation of $d = (h + 2v_1) \pmod{u_1}$ is similar to Degree 1 doubling Algorithm 36, but the input v_1 is normalized with a positive reduced basis for the down variant, or negative for the up variant. In the up variant, the value t_1 from above is reused, resulting in the explicit formula

$$\begin{aligned} v_0' &= -v_{10} - h_0, \\ d &= v_{10} - v_0' + t_1. \end{aligned}$$

In Step 3, the polynomial $z = (f - v_1(v_1+h)/u_1c_3$ is computed directly as described by Technique T8 and in Step 4 $s' = ds = z \pmod{u}$ is computed via a school book modular reduction without the weight $1/c_3$.

Since $\deg(s') = 0$, computations similar in nature to the special output case for $s_1' = 0$ in Degree 2 Doubling Algorithm 43 are performed. Explicit formulas for the output polynomial v_n differ among the two degree 1 doubling algorithms by the normalization of v_1. In both down and up adjust variants, if $s_0' = 0$, the computation of u_n is solely dependent on z and u_1 where $u_n = -z/u_1$ made monic and the computation of v_n includes a school book modular reduction on $V^+ + v_1 + h$ omitting the su_1 portion.

In contrast to the ramified setting Degree 1 doubling Algorithm 36, if $d = 0$, then the degree zero divisor class representative with balancing coefficient $n = 2$ is returned for degree 1 doubling with down adjustment and $n = 0$ for the up adjustment variant, instead of the neutral element.

5.2.7 Explicit Formulas for Addition

In this section, complete divisor addition and the basic formulations of all the corresponding explicit formulas are described. Addition Algorithm 29 is further specialized into different cases depending on the input divisor class degree and balancing coefficient.

The complete addition algorithm should test the degree of the input divisors in order of statistical frequency and choose the appropriate algorithm depending on the input balancing coefficients n_1 and n_2. Ten addition formulas are required for complete addition:

- degree 2 addition,
- degree 1 and 2 addition,
- degree 1 and 2 addition with up adjustment,
- degree 1 addition,
- degree 1 addition with down adjustment,
- degree 1 addition with up adjustment,
- degree 0 and 2 addition with down adjustment,
- degree 0 and 2 addition with up adjustment,
- degree 0 and 1 addition with down adjustment,
- degree 0 and 1 addition with up adjustment.

Only precomputed values are required for degree zero addition, and precomputations from Algorithm 42 may be used. In the frequent cases, the output balancing coefficient n is always zero, and for any special cases, n is computed via lines 16, 23–26 of Algorithm 29. Basic formulations of the algorithms required in complete addition, organized by the degree of the input divisors, are described in the following.

Balancing Coefficient

Let $[u_1, v_1, n_1]$ and $[u_2, v_2, n_2]$ be input reduced divisor class representatives. For all addition formulations presented, generically $n_n = n_1 + n_2 + \deg(S) - 1$ if no reduction or adjustment is required, and $n_n = n_1 + n_2 + \deg(S) - 1 + \deg(z) - \deg(u_n)$ if a reduction or adjustment step is required (with S, z, u_n described in Addition Algorithm 29.) In the latter, an additional case for computing n_n is possible. If $\deg(z) < 3$, then Steps 23–25, are computed instead of Step 26 in Addition Algorithm 29, where $n_n = n_1 + n_2 + \deg(S) - 1 + 3 - \deg(u_n)$ if Step 2 of Addition Algorithm 29 is computed normalizing the input divisor into negative reduced basis, or $n_n = n_1 + n_2 + \deg(S) - 1 + 2\deg(u_1) - 3$ otherwise.

Genus 2 Split Model Degree 2 Addition

Let $[u_1, v_1, 0]$ and $[u_2, v_2, 0]$ be reduced divisor class representatives with $u_1 = x^2 + u_{11}x + u_{10}$, $v_1 = V_3^+x^3 + V_2^+x^2 + v_{11}x + v_{10}$ and $u_2 = x^2 + u_{21}x + u_{20}$, $v_2 = V_3^+x^3 + V_2^+x^2 + v_{21}x + v_{20}$. The basic formulation is similar to the ramified setting, with the addition of computing the balancing coefficient n_n. For degree 2 addition, Steps 1–2 are always skipped and $n_1 = n_2 = 0$. The basic formulation for degree 2 addition is described in Algorithm 47. An explanation of how each step in Algorithm 47 is converted to an explicit formula is given in the following.

Degree 2 addition in the balanced setting is identical to its ramified counterpart Algorithm 37 in the computation of the m_i, d and $s' = ds$. With $d \neq 0$, there are two special cases where the output divisor class representative has degree less than 2. The first case where $\deg(s') \leq 1$ starts at Step 5 is computationally similar to the ramified setting and sets the output balancing $n_n = 0$. In contrast to the ramified setting, the u_n polynomial without normalization, has leading coefficient $s_1c_3 + s_1^2$ instead of s_1^2. In Step 15, the weighted leading coefficient is computed as

$$w_1 = s_1'(s_1' + c_3d) \qquad ((s_1^2 + s_1c_3)d^2).$$

The other special case starting at Step 16 corresponds to $s_1c_3 + s_1^2 = 0$, resulting in lower degree output polynomials u_n and v_n, where the output balancing coefficient $n_n = 2 = \deg(u_n)$ is computed as described in Section 5.2.7.

Algorithm 47 Genus 2 Split Model Degree 2 Addition

Input: $[D_1]$ given as u_1, v_1, $[D_2]$ given as u_2, v_2, where $[D1] \neq [D2]$

Output: $[D] = [u_n, v_n, n] = [D_1] + [D_2]$.

```
1: Compute $m_i$ in system of equations for $su_1 \equiv (v_2 - v_1) \pmod{u_2}$ as $\begin{bmatrix} m_4 & -m_2 \\ -m_3 & m_1 \end{bmatrix}\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} v_{20}-v_{10} \\ v_{21}-v_{20} \end{bmatrix}$.
2: $d = m_1m_4 - m_2m_3$.
3: if $d$ is 0 then  Go to ($d = 0$) Subroutine 48.
4: Solve for $s_1' = ds_1, s_0 = ds_0$ using Cramer's rule, omitting the inversion of $d$.
5: if $s_1' = 0$ then
6:     if $s_0' = 0$ then return $[1, V^+, 0]$.
7:     else
8:         Compute $w = (ds_0')^{-1}$ and precomputation $1/s_0$.
9:         Compute $s = s_0$ with $s_0 = s_0'/d$.
10:        $k = (f - v_1(v_1 + h))/u_1$.   (exact division).
11:        $z = -u_1 s$.
12:        $\tilde{v} = z - v_1 - h$.
13:        $u_n = (s(\tilde{v} - v_1) + k)/u_2$   (exact division, made monic).
14:        $v_n = V^+ - [(V^+ - \tilde{v}) \pmod{u_n}]$,   return $[u_n, v_n, 0]$.
15: $w_1 = s_1'(s_1' + c_3 d)$.
16: if $w_1 = 0$ then
17:     $w_0 = s_0' + d(c_2 - c_3 u_{11})$.
18:     if $w_0 = 0$ then return $[1, V^+, 2]$.
19:     else
20:         Compute $w = (dw_0)^{-1}$ and precomputation $1/(s_0 + c_2 - c_3 u_{11})$
21:         Compute $s = s_1 x + s_0$ with $s_1 = s_1'/d$ and $s_0 = s_0'/d$.
22:         $k = (f - v_1(v_1 + h))/u_1$.   (exact division).
23:         $z = -u_1 s$.
24:         $\tilde{v} = z - v_1 - h$.
25:         $u_n = (s(\tilde{v} - v_1) + k)/u_2$   (exact division, made monic).
26:         $v_n = V^+ - [(V^+ - \tilde{v}) \pmod{u_n}]$,   return $[u_n, v_n, 1]$.
27: Compute $w = (dw_1)^{-1}$ and precomputation $1/(s_1(s_1 + c_3))$.
28: Compute $s = s_1 x + s_0$ with $s_1 = s_1'/d$ and $s_0 = s_0'/d$.
29: $z = -u_1 s$.
30: $\tilde{v} = z - v_1 - h$.
31: $u_n = (s(\tilde{v} - v_1) + k)/u_2$   (exact division, made monic).
32: $v_n = V^+ - [(V^+ - \tilde{v}) \pmod{u_n}]$,   return $[u_n, v_n, 0]$.
```

Similar to split model degree 2 doubling, the s polynomial is not normalized, as the computation of u_n does not benefit from such an approach. In Steps 27-28, the required weights w_i for normalizing u_n, and $s'' = s''_1 x + s''_0 = s_1/c_3 x + s_0/c_3$ are computed identically as described for Steps 22–23 of degree 2 doubling Algorithm 43. In Step 29, computation of the polynomial $z = u_1 s$ takes advantage of Karatsuba multiplication (see Section 4.3.3). The resulting explicit formula for Step 29 is

$$\begin{aligned} z_0 &= s_0 u_{10}, \\ t_1 &= s_1 u_{11}, \\ z_2 &= s_0 + t_1, \\ z_1 &= (s_0 + s_1)(u_{10} + u_{11}) - z_0 - t_1. \end{aligned}$$

In Steps 30–31, computations of the required polynomials $\tilde{v} = -z - v_1 - h$ and $u'_n = -u_n(s_1^2 + s_1 c_3) = (s(-\tilde{v} + v_1) - k))/u_2$ are combined where u'_n requires a division by $(-s_1^2 - s_1 c_3)$ to be normalized, so

$$u_n = (-1/(s_1^2 + s_1 c_3))(s(z + 2v_1 + h) - k)/u_2,$$

computed all in one step, utilizing an exact division (see Section 4.3.2) and taking advantage of combining terms to be multiplied by the precomputed weights w_i. The resulting explicit formula for Steps 30–31 is

$$\begin{aligned} \hat{v}_1 &= v_{11} + h_1, \\ t_2 &= z_2 + c_2, \\ u_{n_1} &= -w_4(s_1(t_2 + s_0) + s_0 c_3) - u_{21}, \\ u_{n_0} &= -w_4(s_0 t_2 + s_1(z_1 + v_{11} + \hat{v}_1) - c_3(V_1^+ - v_{11}) - u_{20} - u_{21} u_{n_1}. \end{aligned}$$

In Step 32, the required polynomial $v_n = V^+ - [(V^+ - \tilde{v}) \pmod{u_n}]$ is instead computed as $v_n = V^+ - [(V^+ + s u_1 + v_1 + h) \pmod{u_n}]$ applying Karatsuba modular reduction (see Section 4.3.4) and reusing $\hat{v}_1$ and t_2 from the above step. The resulting explicit formula for Step 32 is

$$\begin{aligned} t_0 &= c_3 + s_1, \\ t_1 &= u_{n_1} t_0, \end{aligned}$$

$$t_2 = t_2 - t_1,$$

$$t_3 = u_{n_0} t_2,$$

$$v_{n_1} = (s_0 + s_1)(t_2 + t_3) - w_0 - w_1 + u_{n_0} + w_2 u_{n_1}) - \hat{v}_1,$$

$$v_{n_0} = (w_0 + w_2 u_{n_0}) - \hat{v}_0.$$

The special case where $d = 0$ is given in Subroutine 48. The same logic is applied as in the ramified setting (Subroutine 38 with the addition of computing the output v_n in positive reduced basis and returning the appropriate balancing coefficient n described in Section 5.2.7.

Subroutine 48 Genus 2 Split Model Degree 2 Addition $d = 0$

```
 1: Let $d_{w_1} = u_1 \pmod{u_2} = -m_3 x + m_4$.
 2: **if** $m_3$ is 0 **then**
 3:     $d_{w_2} = (v_2 + v_1 + h) \pmod{u_1}$.
 4:     **if** $d_{w_2}$ is 0 **then return** $[1, V^+, 1]$.
 5:     **else**
 6:         $k = (f - v_1(v_1 + h))/u_1 \pmod{u_1}$.
 7:         $d_{w_2} = \text{monic}(d_{w_2})$ with $b_2 = 1/\text{lcf}(d_{w_2})$.
 8:         $u_1 = u_1 / d_{w_2}$.
 9:         $s = b_2 k \pmod{u_1}$.
10:         $u_n = u_1^2$.
11:         $v_n = V^+ - [(V^+ - v_1 - s u_1) \pmod{u_n}]$.
12:         **return** $[u_n, v_n, 0]$.
13: **else**
14:     Compute weighted $d_{w_3} = (v_2 + v_1 + h) \pmod{d_{w_1}}$.
15:     **if** $d_{w_3}$ is 0 **then**
16:         $u_c = d_{w_1}$ made monic with $a_1 = 1/\text{lcf}(d_{w_1})$.
17:         $u_1 = u_1 / d_{w_1}$
18:         $s = a_1(v_2 - v_1) \pmod{u_2}$
19:         $u_n = u_1 u_2$, $v_n = V^+ - [(V^+ - v_1 - s u_1) \pmod{u_n}]$, **return** $[u_n, v_n, 0]$.
20:     **else**
21:         $k = (f - v_1(v_1 + h))/u_1 \pmod{u_1}$.
22:         $a_{12} = (1 - b_2(v_1 + v_2 + h))/d_{w_1} \pmod{u_2}$ with weight $1/m_3^2$.
23:         $s' = s'_1 x + s'_0 = d_{w_3} s_1 x + d_{w_3} s_0 = a_{12}(v_2 - v_1) + m_3^3 k) \pmod{u_2}$, with weight $d_{w_3}$.
24:         Go to line 5 of Degree 2 Addition Algorithm 47, and let $d = d_{w_3}$.
```

Genus 2 Split Model Degree 1 and 2 Addition

Let $[u_1, v_1, n_1]$ and $[u_2, v_2, n_2]$ be the input reduced divisor class representatives with $u_1 = x^2 + u_{11}x + u_{10}$, $v_1 = V_3^+x^3 + V_2^+x^2 + v_{11}x + v_{10}$ and $u_2 = x + u_{20}$, $v_2 = V_3^+x^3 + V_2^+x^2 + V_1^+x + v_{20}$. Divisor class arithmetic in the balanced setting requires two degree 1 with 2 addition algorithms for complete divisor class arithmetic; degree 1 with 2 add, and degree 1 with 2 add with up adjust.

There are two cases for the balancing coefficient n_2 of the degree 1 divisor $[D_2]$, $n_2 = 1$ and $n_2 = 0$. If $n_2 = 1$, a down adjustment is required, if $n_2 = 0$ an up adjustment is required. The reduced basis normalization of v_1 dictates which adjustment step is computed, positive for a down adjustment and negative for an up adjustment. Since v_1 is normalized in positive reduced basis, no extra change of basis is required in the former case. In the latter case, the polynomial v_1 is normalized into a negative reduced basis to facilitate an up adjustment, resulting in an otherwise similar second algorithm.

Both Degree 1 and 2 additions result in a degree 3 divisor after the composition step, and therefore an adjustment step is required instead of a reduction step. Addition Algorithm 29 combines composition and adjustment via NUCOMP continued fraction steps as described in Technique T7. The basic formulation for degree 1 and 2 addition is described in Algorithm 49 and degree 1 and 2 addition with an up adjustment is described in Algorithm 50. An explanation of how each step in both Algorithms 49 and 50 is converted to an explicit formula is given in the following.

Step 0 of degree 1 and 2 addition with up adjust computes a negative reduced normalization of v_1 to accommodate the up adjustment. Steps 1-3 in both algorithms is identical to the ramified setting. Steps 4–12, are similar to the ramified setting other than $z = k/c_3$ is computed via Technique T8. Therefore, in Steps 13–15 the computation of the required monic $u_n = (s(\tilde{v} - v_1) - k)/u_2$ is $u_n = ((s(\tilde{v} - v_1))/c_3 - z)/u_2$ made monic where $z = (f - v_1(v_1 + h)/(c_3u_1)$ and only the x^2 term of z is computed. Note that the polynomial s' is not computed with a weight of $1/c_3$ as in Degree 2 Doubling Algorithm 43, only $z = k/c_3$. More details are given in split model degree 2 doubling and addition sections above.

Similar to Degree 1 Doubling Algorithms 45 and 46, in the special output case $s' = 0$,

Algorithm 49 Genus 2 Split Model Degree 1 and 2 Addition

Input: $[D_1] = [u_1, v_1, 1]$, degree 2 $[D_2] = [u_2, v_2, 0]$.

Output: $[u_n, v_n, n] = [D_1] + [D_2]$.

```
1: $d = u_1 \pmod{u_2}$
2: **if** $d$ is 0 **then**
3:     Go to Degree 1 and 2 Addition ($d = 0$) Subroutine 51.
4: $z = (f - v_1(v_1 + h)/(c_3 u_1)$   (exact division).
5: Compute $s_0' = ds_0 = (v_2 - v_1) \pmod{u_2}$.
6: **if** $s_0' = 0$ **then**
7:     **if** degree 2 coefficient of $z$ is 0 **then return** $[1, V^+, 0]$
8: **else**
9:     $u_n = -z/u_2$   (exact division, made monic).
10:     $v_n = V^+ - ((V^+ + v_1 + h) \pmod{u_n}.)$
11:     **return** $[u_n, v_n, 0]$
12: Compute $w = (s_0' d)^{-1}$ and precomputations $1/s_0$, $s_0 = s_0'/d$.
13: $\tilde{v} = -u_1 s - v_1 - h$.
14: $u_n = ((s(\tilde{v} - v_1))/c_3 + z)/u_2$   (exact division, made monic).
15: $v_n = V^+ - [(V^+ - \tilde{v}) \pmod{u_n}]$,
16: **return** $[u_n, v_n, 0]$
```

Algorithm 50 Genus 2 Split Model Degree 1 and 2 Addition with Up Adjust

Input: $[D_1] = [u_1, v_1, 1]$, degree 2 $[D_2] = [u_2, v_2, 0]$.

Output: $[u_n, v_n, n] = [D_1] + [D_2]$.

```
0: $v_1 = -V^+ - h - (-V^+ - h - v_1) \pmod{u_1}$.
1: $d = u_1 \pmod{u_2}$
2: **if** $d$ is 0 **then**
3:     Go to Degree 1 and 2 Addition ($d = 0$) Subroutine 51.
4: $z = (f - v_1(v_1 + h)/(c_3 u_1)$   (exact division).
5: Compute $s_0' = ds_0 = (v_2 - v_1) \pmod{u_2}$.
6: **if** $s_0' = 0$ **then**
7:     **if** degree 2 coefficient of $z$ is 0 **then return** $[1, V^+, 2]$
8: **else**
9:     $u_n = -z/u_2$   (exact division, made monic).
10:     $v_n = V^+ - ((V^+ + v_1 + h) \pmod{u_n}.)$
11:     **return** $[u_n, v_n, 1]$
12: Compute $w = (s_0' d)^{-1}$ and precomputations $1/s_0$, $s_0 = s_0'/d$.
13: $\tilde{v} = -u_1 s - v_1 - h$.
14: $u_n = ((s(\tilde{v} - v_1))/c_3 + z)/u_2$   (exact division, made monic).
15: $v_n = V^+ - [(V^+ - \tilde{v}) \pmod{u_n}]$,
16: **return** $[u_n, v_n, 0]$
```

$u = ((s(\tilde{v} - v_1))/c_3 - z)/u_2$ reduces to $u = -z/u_2$ and so computation of the polynomial su_1 is omitted resulting in $\tilde{v} = -v_1 - h_1$, where $\tilde{v}$ is not directly computed and Steps 13–14 are combined. The balancing coefficient n_n is updated as $n_n = n_2 + n_1 - 1 + \deg(u_1) + \deg(u_2) - 3 = 0$ in all output cases of degree 1 and 2 addition, and $n_n = n_2 + n_1 - 1 + 3 - \deg(u_n) = 2 - \deg(u_n)$ in all output cases of degree 1 and 2 addition with an up adjust as described in Section 5.2.7.

The special case where $d = 0$ is given in Subroutine 51. An explanation of how each step in Subroutine 48 is converted to an explicit formula is given in the following.

Subroutine 51 Genus 2 Split Model Degree 1 and 2 Addition ($d = 0$)

```
1: $d_w = v_2 - v_1 - h \pmod{u_2}$.
2: if $d_w$ is 0 then
3:     $u_n = u_1/u_2$.
4:     $v_n = V^+ - (V^+ - v_1) \pmod{u_n}$.
5:     return $[u_n, v_n, n_2]$.
6: else
7:     $k = (f - v_1(v_1 + h))/u_1$    (exact division).
8:     $s_0' = d_w s_0 = k \pmod{u_2}$.
9:     if $n_2 = 0$ then
10:        Go to line 7 of Degree 1 and 2 Addition with Up Adjust Algorithm 50 and let $d = d_w$.
11:    else
12:        Go to line 6 of Degree 1 and 2 Addition Algorithm 49 and let $d = d_w$.
```

For $d = 0$ in both algorithms, the same logic is applied as in the ramified setting of Degree 1 and 2 when $d = 0$ (Subroutine 40) with the addition of computing the output v_n in positive reduced basis and returning the appropriate balancing coefficient n described in Section 5.2.7.

Genus 2 Split Model Degree 1 Addition

Let $[u_1, v_1, n_1]$ and $[u_2, v_2, n_2]$ be the input reduced divisor class representatives with $u_1 = x + u_{10}$, $v_1 = V_3^+x^3 + V_2^+x^2 + V_1^+x + v_{10}$ and $u_2 = x + u_{20}$, $v_2 = V_3^+x^3 + V_2^+x^2 + V_1^+x + v_{20}$. Divisor class arithmetic in the balanced setting requires three degree 1 addition algorithms for complete divisor class arithmetic; degree 1 add, degree 1 add with down adjust and degree 1 add with up adjust.

In the case that $n_1 + n_2 = 0$, Degree 1 Addition (Algorithm 52) is computed almost exactly the same as in the ramified setting. The basic formulation for degree 1 addition is described in

Algorithm 52. An explanation of how each step in Algorithm 52 is converted to an explicit formula is given in the following.

Algorithm 52 Genus 2 Split Model Degree 1 Addition

Input: $[D_1] = [u_1, v_1, n_1]$, $[D_2] = [u_2, v_2, n_2]$ where $[D_1] \neq [D_2]$

Output: $[u_n, v_n] = [D_1] + [D_2]$.

```
1: $d = u_1 \pmod{u_2}$.
2: if $d$ is 0 then
3:     $d_w = (v_1 + v_2 + h) \pmod{u_1}$.
4:     if $d_w$ is 0 then
5:         return $[1, V^+, 1]$,
6:     else
7:         $k = (f - v_1(v_1 + h))/u_1$   (exact division).
8:         $s = k/d_w \pmod{u_1}$.
9:         $u_n = u_1^2$.
10:        $v_n = V^+ - [(V^+ - su_1 - v_1) \pmod{u_n}]$.
11:        return $[u_n, v_n, 0]$.
12: $s = (v_2 - v_1)/d \pmod{u_1}$
13: $u_n = u_1 u_2$.
14: $v_n = V^+ - [(V^+ - su_1 - v_1) \pmod{u_n}]$.
15: return $[u_n, v_n, 0]$.
```

The frequent case is almost identical to the ramified setting (Algorithm 41), with the addition of converting v_n into reduced basis and the computation of n_n. The special case where $d = 0$ is given in Subroutine 53. An explanation of how each step in Subroutine 48 is converted to an explicit formula is given in the following.

Algorithm 53 Genus 2 Split Model Degree 1 Addition ($d = 0$)

```
1: $d_w = (v_1 + v_2 + h) \pmod{u_1}$.
2: if $d_w$ is 0 then
3:     return $[1, V^+, 1]$,
4: else
5:     $k = (f - v_1(v_1 + h))/u_1$   (exact division).
6:     $s = k/d_w \pmod{u_1}$.
7:     $u_n = u_1^2$.
8:     $v_n = V^+ - [(V^+ - su_1 - v_1) \pmod{u_n}]$.
9:     return $[u_n, v_n, 0]$.
```

If $d = 0$, there is a new special case where $u_1 = u_2, v_1 = v_2$ but $n_1 \neq n_2$. In Steps 2–8, an explicit degree 1 double without an adjustment is performed with output balancing coefficient

$n_n = 0$ by computing $s = k/d_w \pmod{u_1}$ with $d_w = (v_2 + v_1 + h) \pmod{u_1}$, then $u_n = u_1^2$ and $V^+ - [(V^+ - su_1 - v_1) \pmod{u_n}]$.

For degree 1 additions with adjustments, the intermediate balancing coefficient after addition, $(n' = n_1 + n_2 - 1)$, is -1 or 1 depending on whether n_1 and n_2 are both 0 or both 1 respectively. Similarly to the split model degree 1 doubling algorithms, degree 1 addition algorithms with adjustments combine the composition and adjustment together by Technique T7. The basic formulation for degree 1 addition with a down adjustment in Algorithm 54 and degree 1 addition with an up adjustment in Algorithm 55. An explanation of how each step in both Algorithms is converted to an explicit formula is given in the following.

Algorithm 54 Genus 2 Split Model Degree 1 Addition with Down Adjust
Input: $[D_1] = [u_1, v_1, n_1]$, $[D_2] = [u_2, v_2, n_2]$ where $[D_1] \neq [D_2]$
Output: $[u_n, v_n] = [D_1] + [D_2]$.

```
1: Compute $d = u_1 \pmod{u_2}$.
2: if $d$ is 0 then
3:     return $[1, V^+, 2]$.
4: $s'_0 = ds_0 = (v_2 - v_1) \pmod{u_2}$.
5: $z = (f - v_1(v_1 + h))/u_1 c_3$    (exact division).
6: if $s'_0 = 0$ then
7:     if degree 2 coefficient of $z$ is 0 then
8:         return $[1, V^+, 0]$.
9:     $u_n = -z/u_1$    (exact division, made monic).
10:    $v_n = V^+ - [(V^+ + v_1 + h) \pmod{u_n}]$.
11:    return $[u_n, v_n, 0]$
12: Compute $w = (s'_0 d)^{-1}$ and precomputations $1/s_0$ and $s_0 = s'_0/d$.
13: $\tilde{v} = -su_1 - v_1 - h$.
14: $u_n = (s(\tilde{v} - v_1)/c_3 - z)/u_2$    (exact division, made monic).
15: $v_n = V^+ - [(V^+ - \tilde{v}) \pmod{u_n}]$,
16: return $[u_n, v_n, 0]$.
```

Recall that the reduced basis of v_1 dictates which direction is applied, and that v_1 is already in the form required for a down adjustment. Up and down adjustment formulations are almost identical, with the addition of normalizing v_1 into negative reduced basis before it is used in the up adjustment formulation. The resulting explicit formulas follow an identical approach to the corresponding degree 1 doubling Algorithms 45 and 46, but compute $\tilde{v} = -su_1 - v_1 - h$, monic

Algorithm 55 Genus 2 Split Model Degree 1 Addition with Up Adjust

Input: $[D_1] = [u_1, v_1, n_1]$, $[D_2] = [u_2, v_2, n_2]$ where $[D_1] \neq [D_2]$

Output: $[u_n, v_n] = [D_1] + [D_2]$.

```
1:  Compute $d = u_1 \pmod{u_2}$.
2:  **if** $d$ is 0 **then**
3:      **return** $[1, V^+, 2]$.
4:  $v_1 = -V^+ - h - ((-V^+ - h - v_1) \pmod{u_1})$.
5:  $s'_0 = ds_0 = (v_2 - v_1) \pmod{u_2}$.
6:  $z = (f - v_1(v_1 + h))/u_1 c_3$   (exact division).
7:  **if** $s'_0 = 0$ **then**
8:      **if** degree 2 coefficient of $z$ is 0 **then**
9:          **return** $[1, V^+, 2]$.
10:     $u_n = -z/u_1$   (exact division, made monic).
11:     $v_n = V^+ - [(V^+ + v_1 + h) \pmod{u_n}]$.
12:     **return** $[u_n, v_n, 1]$
13: Compute $w = (s'_0 d)^{-1}$ and precomputations $1/s_0$ and $s_0 = s'_0/d$.
14: $\tilde{v} = -su_1 - v_1 - h$.
15: $u_n = (s(\tilde{v} - v_1)/c_3 - z)/u_2$   (exact division, made monic).
16: $v_n = V^+ - [(V^+ - \tilde{v}) \pmod{u_n}]$,
17: **return** $[u_n, v_n, 0]$.
```

$u_n = (s(\tilde{v} - v_1)/c_3 - z)/u_2$, specialized to addition.

Genus 2 Split Model Degree 0 Addition

Let $[u_1, v_1, n_1]$ and $[u_2, v_2, n_2]$ be the input reduced divisor class representatives with $u_1 = x^2 + u_{11}x + u_{10}$, $v_1 = V_3^+x^3 + V_2^+x^2 + v_{11}x + v_{10}$ and $u_2 = 1$, $v_2 = V^+$. In the balanced setting, there are two main degree 0 addition algorithms; degree 0 addition with down adjust and degree 0 addition with up adjust, both of which have variants based on the input degree of the divisor. The basic formulations of the degree 1 and 2 variants are identical.

By Technique T7, degree 0 add with down adjust is formulated from Addition Algorithm 29 directly. Degree 0 with up adjust first normalizes v_1 into negative reduced basis, before performing identical computations to the down adjustment case. The basic formulation for degree 0 addition with adown adjustment is described in Algorithm 56 and degree 0 addition with an up adjustment in Algorithm 57. An explanation of how each step in both Algorithms is converted to an explicit formula is given in the following.

Algorithm 56 Genus 2 Split Model Degree 0 Add with Down Adjust

Input: $[D_1] = [u_1, v_1, n_1]$.

Output: $[u_n, v_n, n] = [D_1] + [1, V^+, 0]$.

```
1: $z' = z'_4x^4 + z'_3x^3 + z'_2x^2 + z'_1x + z'_0 = (f - v_1(v_1 + h))/c_3$.
2: **if** $z'_4$ is 0 **then**
3:     **if** $z'_3$ is 0 **then return** $[1, V^+, 0]$.
4:     **else**
5:         $u_n = -z'/u_1 \pmod{u_1}$   (exact division, made monic).
6:         $v_n = V^+ - (V^+ + v_1 + h) \pmod{u_n}$.
7:         **return** $[u_n, v_n, 0]$.
8: $u_n = z'/u_1 \pmod{u_1}$   (exact division, made monic).
9: $v_n = V^+ - (V^+ + v_1 + h) \pmod{u_n}$.
10: **return** $[u_n, v_n, 0]$.
```

Algorithm 57 Genus 2 Split Model Degree 0 Add with Up Adjust

Input: $[D_1] = [u_1, v_1, n_1]$.

Output: $[u_n, v_n, n] = [D_1] + [1, V^+, 2]$.

```
0: $v_1 = -V^+ - h - [(-V^+ - h - v_1) \pmod{u_1}.]$
1: $z' = z'_4x^4 + z'_3x^3 + z'_2x^2 + z'_1x + z'_0 = (f - v_1(v_1 + h))/c_3$.
2: **if** $z'_4$ is 0 **then**
3:     **if** $z'_3$ is 0 **then return** $[1, V^+, 2]$.
4:     **else**
5:         $u_n = z'/u_1$   (exact division, made monic).
6:         $v_n = V^+ - (V^+ + v_1 + h) \pmod{u_n}$.
7:         **return** $[u_n, v_n, 1]$.
8: $u_n = z'/u_1$   (exact division, made monic).
9: $v_n = V^+ - (V^+ + v_1 + h) \pmod{u_n}$.
10: **return** $[u_n, v_n, 0]$.
```

Technique T8 is applied to Steps 1 and 8. In Step 1, the polynomial $z' = zu_1 = ku_1/c_3$ is computed first without the exact division by u_1 to check for special output cases. The formulation of the special output cases does not change other than the value of the output balancing coefficient n_n. For degree 0 addition with down adjust $n_n = 0$ and for up adjust $n_n = 2 - \deg(u_n)$ in all cases (see Section 5.2.7.) All four variants result in explicit formulas computing the polynomial $u_n = z = (f - v_1(v_1 + h)/(u_1c_3)$ in Step 8 and $V^+ - (V^+ + v_1 + h) \pmod{u_n}$ in Step 9, and are similar to previously described addition formulas after the computation of s' with $s' = 0$.

5.2.8 Field Operation Costs and Comparisons

Field operation costs of the algorithms presented above are given in Table 5.3 and are compared to previous methods in Table 5.4. Standard protocol is followed using M to denote a multiplication, S a squaring, C a multiplication by a constant curve coefficient and A an addition. Unlike many previous works addition counts are included, as these have been shown to have non-trivial cost working with large fields [16], and significant cost relative to multiplications working with word sized fields [36]. Recall that based on the findings of Sutherland [36, 37], techniques applied to any explicit formula never trade a multiplication for more than 3 additions based on [36]. The assumption that division by two over fields with characteristic not equal to 2 is equivalent in cost to an addition [37] is applied to the analysis.

All divisor class operations require one inversion where field operation counts are presented for the frequent cases. Field operation counts can be reduced if assumptions about the characteristic of the base field are made. Three base field and curve equation cases are presented: the generic split curve equation over arbitrary fields, and the curve equation assumptions described in Section 5.2.1 where $h_3 = h_2, h_1, h_0 = 0$ for fields with characteristic not equal to 2 and $h_3 = 1$, $h_2 = f_5 = f_4 = f_3 = 0$ for fields with characteristic 2. Several new formulas that require exactly one inversion in all cases and fully encapsulate divisor class arithmetic on a genus 2 split model are introduced in this work including degree 0 additions (adjustments steps) of degree 2 and degree 1 divisors, degree 1 with 1 addition with up or down adjustments and degree 1 with 2 addition with up adjustment. Field operation counts for all formulas from this section are given in Table 5.3.

Comparisons of explicit formulas from this work to the previous best over genus 2 split models are presented in Table 5.4. Operation count comparisons for doubling degree 2 divisors (2DBL), adding degree 2 divisors, degree 1 with 2, and degree 1 with 1 (2ADD, 12ADD, 1ADD) are presented. Degree 1 doubling is omitted, as that operation always requires either a “down" or an “up" adjustment step; comparisons for those operations are listed (1DDW, 1DUP). Also included are comparisons for all other possible cases, specifically adjustment steps of degree 2 and degree 1 divisors (2DWN, 1DWN, 2UP, 1UP), degree 1 with 1 addition with up or down adjustments (1ADW,

Table 5.3: Frequent case field operation counts of all one inversion operations required for complete divisor class group arithmetic over genus 2 hyperelliptic curve split models.

	Arbitrary				**char ≠ 2**				**char = 2**			
Operations	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**
Doubling												
Degree 1 with Down Adjust	14	2	26	4	12	3	23	1	11	4	16	1
Degree 1 with Up Adjust	12	2	20	3	10	3	19	1	11	3	14	1
Degree 2	30	2	44	8	29	3	39	0	29	2	31	0
Addition												
Degree 0 and 1 with Down Adjust	4	0	13	3	3	1	9	2	3	1	8	2
Degree 0 and 1 with Up Adjust	4	0	9	3	3	1	7	2	3	1	7	2
Degree 0 and 2 with Down Adjust	5	0	17	4	4	2	12	0	5	1	11	0
Degree 0 and 2 with Up Adjust	5	0	17	4	4	2	12	0	5	1	11	0
Degree 1	3	0	5	0	3	0	5	0	3	0	4	0
Degree 1 with Down Adjust	9	2	20	3	9	2	16	1	8	3	14	1
Degree 1 with Up Adjust	11	2	19	4	9	4	17	1	9	4	15	1
Degree 1 and 2	14	2	26	3	14	2	22	0	14	2	19	0
Degree 1 and 2 with Up Adjust	15	2	30	5	15	2	26	0	14	3	22	0
Degree 2	27	1	37	3	26	2	36	0	27	1	34	0

1AUP) and degree 1 with 2 addition with up adjustment (12AUP.) Some of the formulas in this work cannot be directly compared to previous best, and instead the operations from [9] denoted with (2I) that are combinations of two operations requiring two inversions are compared to.

Comparison Summary

Baby Step and Inverse Baby Step operations in [9] are equivalent to the Degree 0 and 2 ADD with Up adjust, and Degree 0 and 2 Add with Down Adjust respectively in this work and are compared in Table 5.4. At least **1M + 2A + 1C** is saved in both Degree 0 and 1 ADD operations and at least **1M + 2A** in both Degree 0 and 2 ADD operations over arbitrary fields. Over fields with characteristic not equal to 2, **3A** in all 4 operations are saved, and over characteristic 2 fields between **3A** and **5A** are saved over [9].

Degree 1 Doubling comparisons are omitted, as no such operation exists in the balanced divisor

Table 5.4: Field operation comparisons for frequent case divisor class group operations on genus 2 hyperelliptic curve split models. Operations from [9] that are combinations of two operations requiring two inversions are denoted with (2I).

Arbitrary Fields (M S A C)								
	2DBL	2ADD	12ADD	1ADD	2DWN	1DWN	2UP	1UP
[9]	30 2 50 10	27 1 37 3	17 2 30 6	3 0 5 0	6 0 21 4	5 0 15 4	6 0 19 4	5 0 12 4
This	30 2 44 8	27 1 37 3	14 2 26 3	3 0 5 0	5 0 17 4	4 0 13 3	5 0 17 4	4 0 9 3
	1DDW	1DUP	1ADW	1AUP	12AUP			
(2I) [9]	16 1 63 9	16 1 61 9	9 0 26 4	9 0 24 4	22 2 42 10			
This	14 2 26 4	12 2 20 3	9 2 20 3	11 2 19 4	15 2 30 5			

char = 2 (M S A C)								
	2DBL	2ADD	12ADD	1ADD	2DWN	1DWN	2UP	1UP
[9]	29 2 32 0	27 1 35 0	15 3 24 0	3 0 5 0	5 1 16 0	3 1 12 2	5 1 16 0	3 1 10 2
This	29 2 31 0	27 1 34 0	14 2 19 0	3 0 4 0	5 1 11 0	3 1 8 2	5 1 11 0	3 1 7 2
	1DDW	1DUP	1ADW	1AUP	12AUP			
(2I) [9]	14 2 48 2	14 2 48 2	8 1 21 0	8 1 21 0	20 4 40 0			
This	11 4 16 1	11 3 14 1	8 3 14 1	9 4 15 1	14 3 22 0			

char ≠ 2 (M S A C)								
	2DBL	2ADD	12ADD	1ADD	2DWN	1DWN	2UP	1UP
[9]	28 4 46 0	26 2 37 0	16 3 29 0	3 0 5 0	4 2 15 0	3 1 12 2	4 2 15 0	3 1 10 2
This	29 3 39 0	26 2 36 0	14 2 22 0	3 0 5 0	4 2 12 0	3 1 9 2	4 2 12 0	3 1 7 2
	1DDW	1DUP	1ADW	1AUP	12AUP			
(2I) [9]	13 3 35 0	13 3 35 0	7 2 20 0	7 2 20 0	20 5 44 0			
This	12 3 23 1	10 3 19 1	9 2 16 1	9 4 17 1	15 2 26 0			

class group setting. Not all divisor class operations in [9] have a direct comparison, so combined divisor class operations where two inversions are required are included in Table 5.4, for example a degree 1 double followed by either a baby step or inverse baby step in [9] are the same as the degree 1 double with and adjust up or down in this work respectively. The use of Balanced NUCOMP specialized to genus 2 as the basis for the formulas (Technique T7) in all divisor class operations where adjustment steps are combined with compositions provide great improvements over directly using Balanced Addition and Adjust Algorithms 15 and 7. The worst trade-off is 5 total multiplicative field operations (M,S,C) for an inverse, and additions are always saved.

Improvements to the frequent case addition are small, saving an addition over fields of characteristic 2 and not 2. Improvements to frequent case doubling save **6A + 2C** over arbitrary fields, an addition over characteristic 2 fields, and trade **1M** for **1S + 7A** over fields with characteristic not equal to 2. For degree 1 and 2 addition, there are savings of **3M + 4A + 3C** over previous best, and slightly smaller savings over characteristic 2 and characteristic not equal to 2 fields. The savings come from viewing the degree 3 reduction as an adjustment and applying Balanced NUCOMP. For special cases where the degree of the output divisor is less than 2, i.e. $s = s_0$, there are similar savings throughout over [9].

All implementations of frequent case arithmetic include explicit handling of special cases where the input divisor class representatives have points in common (or are opposite) as they come up naturally in the computations. Rather than making calls to Cantor's Algorithm, the special cases are computed explicitly and with only one inversion. These cases are made as fast as possible given the techniques that can be applied. These special cases are not analyzed here because there is no previous best to compare to.

5.3 Empirical Analysis

In this section, empirical data are presented that illustrates the relative performance of the genus 2 addition and doubling algorithms presented in this chapter over both ramified and split models. All algorithms for addition and doubling are implemented in the high level language Magma as a proof of concept. Therefore, the absolute timings are not of great interest. The reader should rather focus on the relative cost between the various algorithms and models. The experiments were performed on a workstation with an Intel Xeon 7550 processor that has 64 cores, each of which is 64-bit and runs at 2.00 GHz.

Explicit formulas are very complex and prone to errors. Both black-box and white-box testing programs are utilized to provide thorough evidence that the formulas as they appear in the Magma code are all correct. Black-box testing computes numerous additions of randomly chosen divisor classes over randomly chosen curve models with output of each addition compared to Cantor's Algorithm. White-box testing computes a divisor class addition for every possible computation path within each formula, and similarly compares the output to Cantor's Algorithm. The sample input divisor classes and curve models for the white-box testing are automatically generated via python scripts. The python scripts invoke Magma to compute additions with randomly generated inputs, flagging the cases as found and saving the inputs. This process is repeated until every computation path is encountered by flagging return statements. Finally, the python scripts generate a Magma white box testing script that computes all sample inputs generated.

As a preliminary benchmark, doubling and addition algorithms over both ramified and split models are compared separately, using our new formulas, previous best explicit formulas, Cantor's algorithm and Magma's built-in. For addition, a Fibonacci-like sequence of divisors $D_{i+1} = D_i + D_{i-1}$ was computed, starting from two random divisors. Similar experiments were performed for the doubling algorithms over ramified and split models, by computing series of thousands of additions of a divisor class with itself. Timings taken over a range of prime fields of sizes 2, 4, 8, 16, 32, 64, 128, 256, 512 and 1024 bits were collected. All timings were run over random hyperelliptic curves with $h = 0$ defined over prime fields, using algorithms specialized for this setting.

All implementations of previous best formulas fall back on Cantor's algorithm in any special cases not considered. Adaptations of the complete addition and doubling algorithms of this work were implemented similarly for direct comparison. Only the degree 2 addition and doubling frequent cases were explicitly developed in the work of Costello-Lauter [6] over ramified models. Two versions of Costello-Lauter formulas are compared, the original addition and doubling formulas as given in [6], and the other omitting a trade of 1 field multiplication for 13 field additions. Degree 1 and 2 addition, degree 2 addition and degree 2 doubling, along with special output cases for each, were explicitly developed in the work of Lange [24] over ramified models. Frequent and special output cases for degree 2 and degree 1 doubling, as well as degree 2, degree 1 and 2, degree 1, degree 0 and 2 and degree 0 and 1 addition were considered in the work of Erickson et. al. [9] over split models. The special output cases in [9] do not apply to the balanced setting and therefore are omitted.

The algorithms used for addition in the ramified setting are:

- Complete explicit addition using formulas of this work (Section 5.1.5),
- Lange explicit addition formulas [24],
- Explicit addition but calling Cantor's algorithm in the same cases as Lange,
- Costello-Lauter explicit addition formulas [6],
- Costello-Lauter explicit addition formulas without trading 1M for 13A,
- Explicit addition but calling Cantor's algorithm in the same cases as Costello-Lauter,
- Improved Cantor's Addition (Algorithm 11).

The algorithms used for doubling in the ramified setting are:

- Complete explicit doubling using formulas of this work (Section 5.1.4),
- Lange explicit doubling formulas [24],
- Explicit doubling but calling Cantor's algorithm in the same cases as Lange,
- Costello-Lauter explicit doubling formulas [6],
- Costello-Lauter explicit doubling formulas with no trades,
- Explicit doubling but calling Cantor's algorithm in the same cases as Costello-Lauter,

- Improved Cantor's Doubling (Algorithm 12).

The algorithms used for addition in the balanced setting are:

- Complete explicit addition using formulas of this work (Section 5.2.7),
- Erickson et. al. explicit addition formulas [9],
- Explicit addition but calling Cantor's algorithm in the same cases as Erickson et. al.,
- Improved Balanced Addition (Positive Reduced Basis) (Algorithm 15).

The algorithms used for doubling in the balanced setting are:

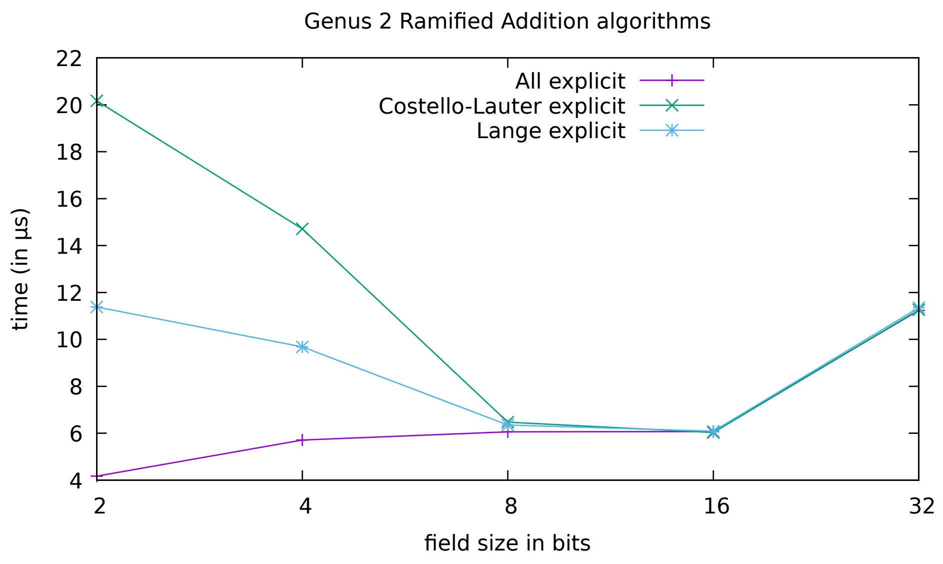
- Complete explicit doubling using formulas of this work (Section 5.2.6),
- Erickson et. al. explicit doubling formulas [9],
- Explicit doubling but calling Cantor's algorithm in the same cases as Erickson et. al.,
- Improved Balanced Double (Positive Reduced Basis) (Algorithm 16).

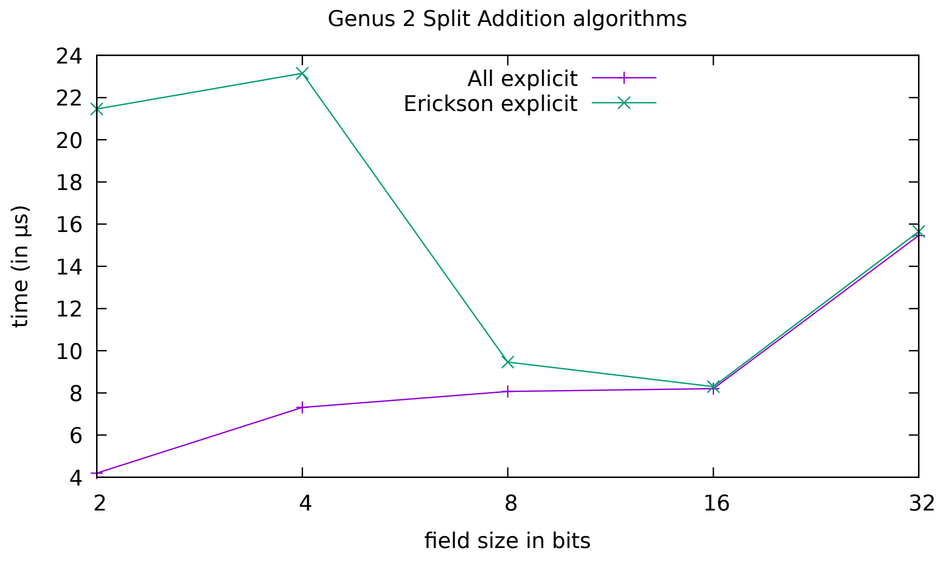
Magma's built-in arithmetic was also timed for comparison. In the following, several observations are discussed and organized into sub-sections.

5.3.1 Explicit Special Cases

In the next figures, complete explicit formulas from this work are compared to adaptations where some special cases fall back on the appropriate Cantor's algorithm. Only addition comparisons are presented, as doubling shows similar results. The following conclusions are drawn from these plots:

- For both models, our complete explicit formulas greatly outperform any variation of our formulas that rely on Cantor's algorithm for special cases over fields with small cardinality.
- Convergence in the timings occur at 16-bit fields, suggesting that the frequent case only implementations are effectively as efficient as complete explicit algorithms for 16-bit fields or larger when implemented in Magma.





5.3.2 Ramified and Split Model Comparisons

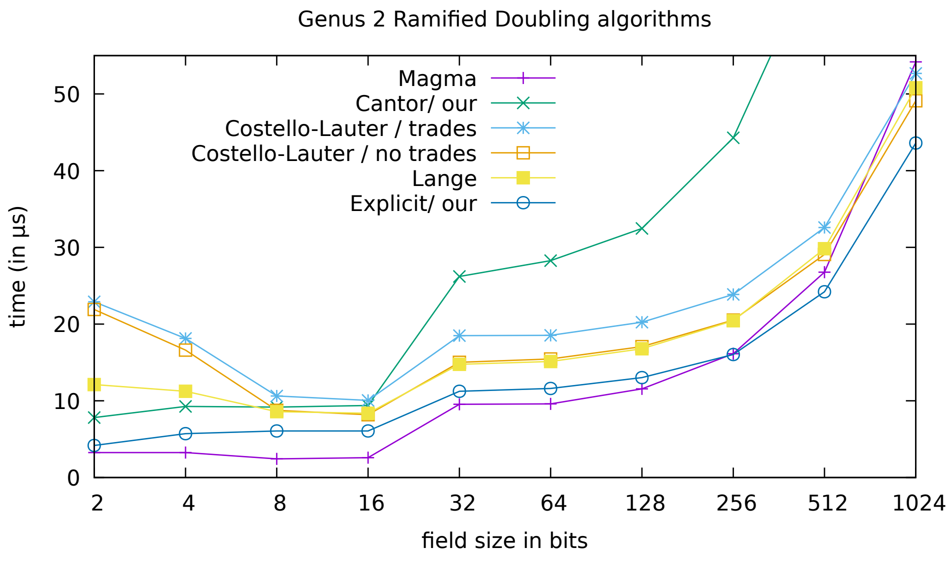
In the next figures, explicit formulas introduced in this work are compared to all relevant previous best algorithms over ramified and split models. Our split and ramified model formulas are also plotted for comparison of genus 2 arithmetic in the last figure. The following conclusions are drawn from these plots:

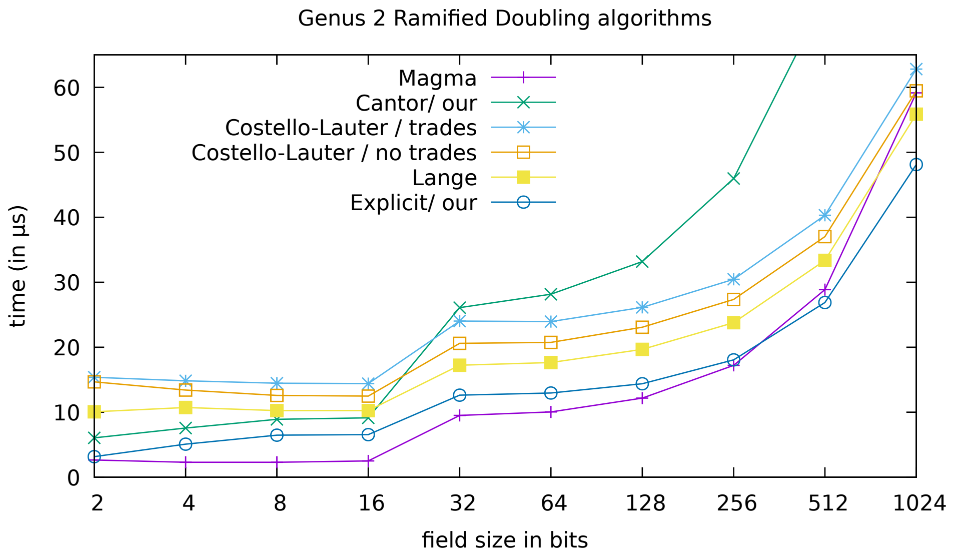
- On the same platform, our explicit formulas outperform all relevant previous algorithms, including state-of-the-art and Cantor based algorithms, over every field size.

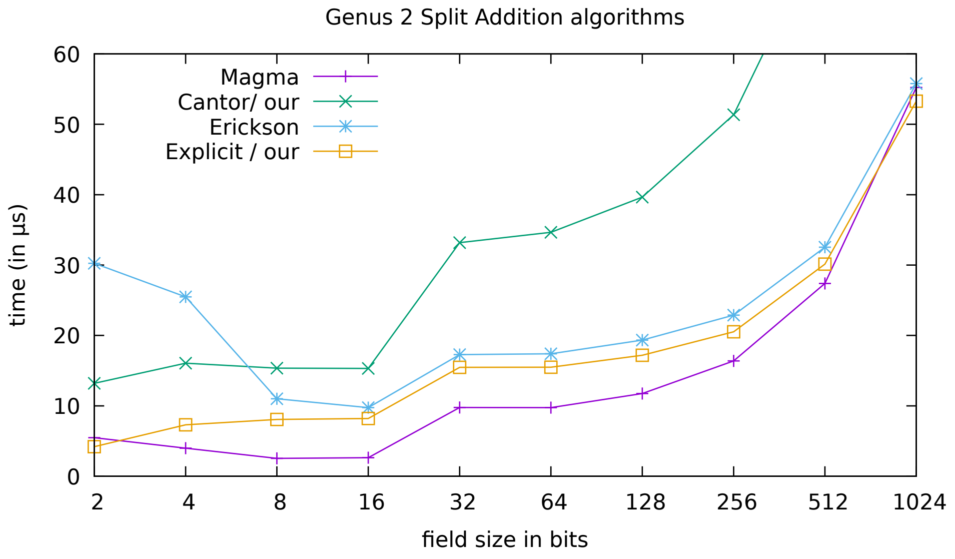
- Expensive field multiplication for addition trades, such as the trade of 13 additions for 1 multiplication proposed in [6] are detrimental for efficiency over any field size, providing evidence for the design choices of this work to cap trades at 3 additions for 1 multiplication (see Section 4.3.1).

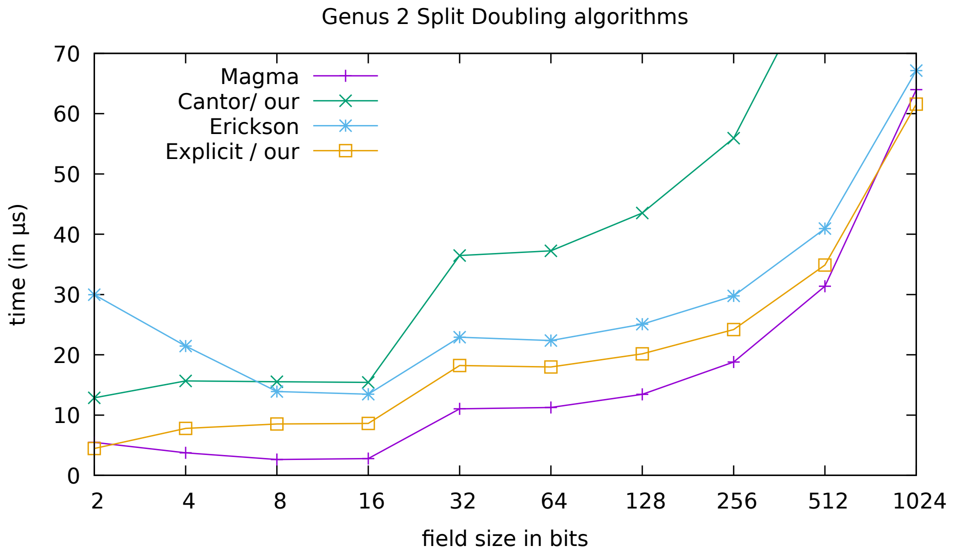
- Our explicit formulas outperform Magma's built-in arithmetic over 512-bit fields and up for ramified models and 1024-bit fields for split models. Based on the relative performance of prior state-of-the-art, Magma is likely accessing internal *C* functions that is not accessible via the interface provided.

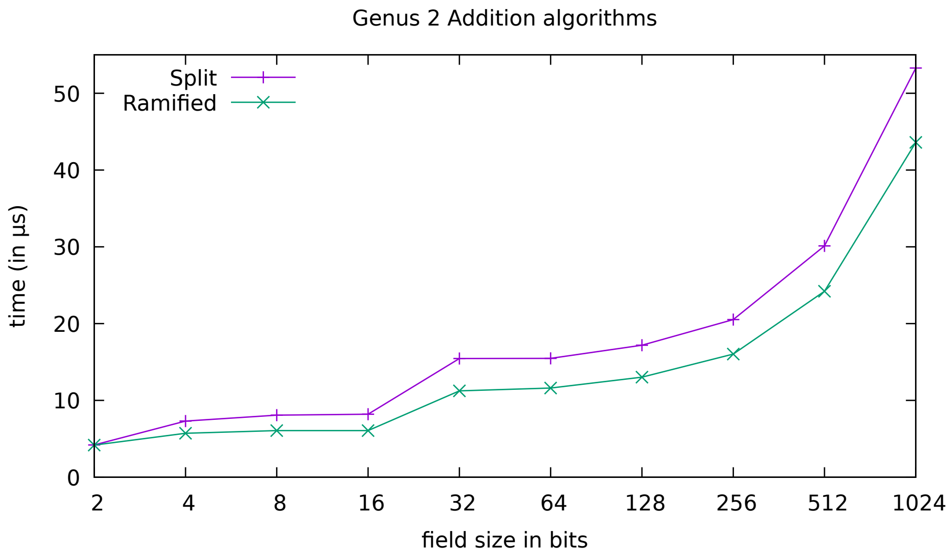
- For Magma based implementations, split model arithmetic using balanced divisor classes is about 20% slower than ramified arithmetic for genus 2 curves.

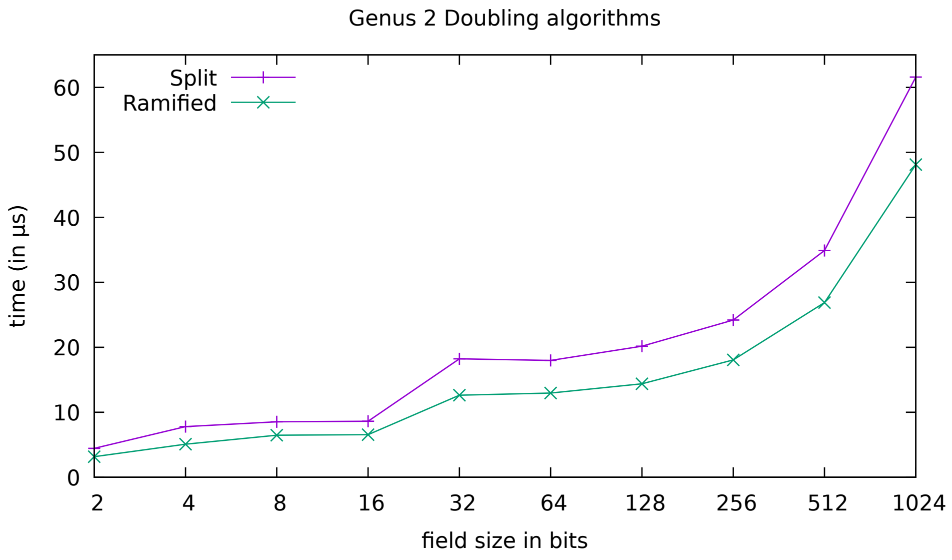



Genus 2 Ramified Doubling algorithms
Magma
Cantor/ our
Costello-Lauter / trades
Costello-Lauter / no trades
Lange
Explicit/ our
time (in µs)
field size in bits
0
10
20
30
40
50
60
2
4
8
16
32
64
128
256
512
1024


Genus 2 Split Addition algorithms
Magma
Cantor/ our
Erickson
Explicit / our
time (in µs)
field size in bits
0
10
20
30
40
50
60
2
4
8
16
32
64
128
256
512
1024


Genus 2 Split Doubling algorithms
Magma
Cantor/ our
Erickson
Explicit / our
time (in µs)
field size in bits
0
10
20
30
40
50
60
70
2
4
8
16
32
64
128
256
512
1024





5.3.3 Summary

The empirical results indicate that complete explicit formulas provide an improvement for computing ramified and split model arithmetic over small cardinality fields with a cross-over at 16-bits. As expected, the relatively fewer field operations required in the formulas of this work translate to faster implementations when using the same platform. The relative performance of Magma's built-in arithmetic and the formulas of this chapter is likely due to Magma's high-level interface. Integrating algorithms from this chapter directly into Magma's built-in arithmetic, or implementing the algorithms directly in C/C++ so that they do not suffer from the overhead associated to working in a high-level system like Magma, should reduce the relative performance to Magma's built-in arithmetic, the relative performance between genus 2 split and ramified model arithmetic and increase the field size for performance convergence between complete explicit formulas and frequent case only implementations.

Chapter 6

Explicit Formulas for Genus 3 Balanced Arithmetic

In this chapter, derivations of explicit formulas for genus 3 hyperelliptic curves given by a split model are described. All explicit formulas presented in this chapter are implemented in Magma along with implementations of random input testing and automatic generation of inputs that cover every output case. Automated operation counting and latex table generation scripts were also developed to circumvent errors in the operation count and explicit formula presentations. All software developed for this thesis is available at https://github.com/salindne/divisorArithmetic. Corresponding results for ramified models are developed by another student and will appear together with these results in a paper to be submitted later.

6.1 Improved Genus 3 Split Model Explicit Formulas

In this section, novel explicit formulas for addition and doubling of divisor classes on genus 3 hyperelliptic curves represented by split models are described. In general, the curve equation of a split model for a genus 2 hyperelliptic curve in Weierstrass form is $C : y^2 + h(x)y = f(x)$ where

$$f = f_8x^8 + f_7x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0, \qquad h = h_4x^4 + h_3x^3 + h_2x^2 + h_1x + h_0,$$

where f_8 is a square in k if $\text{char}(k) \neq 2$ and $f_8 = \beta^2 + \beta$, for some $\beta \in k$ otherwise. The polynomial $V^+ = V_4^- x^4 + V_3^- x^3 + V_2^- x^2 + V_1^- x + V_0^-$ such that $\deg(f - V^+(V^+ + h)) \leq 3$ (see Definition 2.1.42), is precomputed in Algorithm 58 along with a table of precomputations and curve coefficients. Input and output balanced divisors are represented by 7-coordinate vectors of six field elements and a balancing coefficient n as

$$D = [u_2, u_1, u_0, v_2, v_1, v_0, n],$$

corresponding to the Balanced Mumford representation $D = [u, v, n]$ with $u = x^3 + u_2 x^2 + u_1 x + u_0$ and $v = v_2 x^2 + v_1 x + v_0$ in negative reduced basis (Section 3.1.3). The two leading coefficients of v in any reduced basis only rely on curve parameters and therefore still only six base field elements are required to represent a divisor class.

In contrast to previous work designed for arithmetic in the infrastructure of a genus 3 split model curve [32], the formulas of this work are complete and are designed to be efficient for arithmetic in the divisor class group (using balanced divisor representations) as opposed to the infrastructure. The work in [37] does present explicit formulas for addition and doubling in the divisor class group for genus 3 split models using the balanced setting, but only considers frequent cases. Moreover, the frequent case formulas in this work require considerably fewer field operations than previous best [32] and [37]. The framework introduced in this work based on balanced divisor class representations [11] and the novel Balanced NUCOMP from Section 3.3 encapsulates all adjustments in special cases, resulting in formulas that require only one inversion given any two divisor classes as input. Applying many of the same techniques as for genus 2 curve formulas adapted to genus 3, and introducing some novel ones, complete explicit formulas are developed including all special cases based on Algorithms 33 and 32.

For genus 3 split models, and odd genus in general, non-trivial calls to adjust occur as part of the frequent cases reduction process i.e.; adjust is always called in the frequent case. The adjustment is not auxiliary, but replaces the final reduction step when reducing the degree of the intermediate divisor representative from degree four to three in Cantor's algorithm. As suggested in Section 3.3, a negative reduced basis is used to absorb one adjustment via the continued fraction steps of Balanced

NUCOMP resulting in removing the need for any further adjustment steps in the frequent cases. Working with a negative reduced basis also reduces the degree of the $k = f - v(v + h)$ polynomial that arises in the computations via cancellations.

This section is structured as follows; first, assumptions about curve models over certain fields are discussed, followed by an overview of prior work. This is followed by a statement of the novel contributions in this work and a description of the basic formulations for doubling and addition. Finally, operation costs in terms of field operations, comparisons to previous work, and empirical benchmarking results are presented.

6.1.1 Curve Simplifications

Recall from Section 5.2.1 that isomorphic transformations can be applied depending on the characteristic of the base field of the hyperelliptic curve, where the possible transformation for genus 2 curve split models were given. On genus 3 split models, the same isomorphic transformations can be applied over fields of characteristic 2 and not equal to 2 (Section 5.2.1). Operation counts in Section 6.1.7 assume the following:

- If $\mathrm{char}(k) \neq 2$ then $h = 0$.
- If $\mathrm{char}(k) = 2$ then $h_4 = 1$ and $h_3 = f_5 = f_4 = f_3 = 0$.

6.1.2 Prior Work

In [32] the authors present explicit formulas for frequent case addition, doubling and degree zero addition (referred to as baby steps) on genus 3 split models over characteristic 2 and characteristic not equal to 2 fields. The formulas are not presented for divisor class arithmetic, but only the related (but different) infrastructure of a genus 3 curve split model. Similar to the work in [9] the authors use a slightly different definition of Mumford representation, where $u \mid f + hv - v^2$ instead of $u \mid f - hv - v^2$ and the alternate positive reduced normalization of the v polynomial where

$$u' = u$$

$$v' = V^+ + h - [(V^+ + h - v) \mod u]. \tag{6.1.1}$$

In [37] the author presents explicit formulas for frequent case addition and doubling, with no special cases considered, over the divisor class group of a split model genus 3 hyperelliptic curve using balanced divisors. The polynomial v of the Mumford representation of a divisor class is not normalized into a reduced basis, and left exactly as-is from the definition of Mumford representation (Definition 2.3.1). Comparing to operation counts of the explicit formulas in [32], frequent case doubling requires fewer field operations, but addition requires more field operations than [32].

6.1.3 Summary of Contributions

This work provides complete explicit formulas, in the sense that all computation paths are explicitly computed, requiring only one inversion in all cases that take reduced divisor class representatives as input, and output a reduced divisor class representative. Not only is every special case explicitly computed, this work also considerably improves the field operation counts of divisor class operations over previous best in the frequent cases of addition and doubling. All formulas are implemented in Magma including a testing suite to ensure their correctness (described in Section 6.2). The implemented formulas and testing scripts are available at `https://github.com/salindne/divisorArithmetic`.

The explicit formulas for divisor class addition and doubling are based on Balanced NUCOMP specialized to genus 3 addition and doubling using a negative reduced basis (see Section 3.1.3) that include efficient formulations of special cases. The explicit formulation of steps in Algorithms 33 and 32, coupled with certain explicit techniques produce the fastest explicit formulas for frequent case addition and doubling to date. As explained in Section 4.1.3, a negative reduced basis normalization for v is beneficial to a NUCOMP based algorithm and is used in the explicit formulas in contrast to the positive reduced basis used for genus 2 arithmetic.

A summary of the novel techniques that differ from previous work for genus 3 split model explicit formulas goes as follows:

T10 Use specialized genus 3 versions of Balanced NUCOMP and NUDUPL Algorithms 33 and 32 as the basis for all explicit formulas. Using the specialized Balanced NUCOMP variants greatly reduces the field operation counts in the frequent cases when compared to previous methods. For special cases, using Balanced NUCOMP instead of Balanced ADD Algorithm 15 avoids either a second inversion or pushes weighted values from the composition step to the adjustment step with Montgomery's inversion trick, resulting in a significant reduction in the number of field operations. Applying Balanced NUCOMP to degree 0 additions results in similar formulations as Balanced Adjust Algorithm 17, but this work improves on previous best with Technique T11.

T11 Apply an alternate computation of the output k polynomial in all doubling and many addition algorithms with input divisor degrees of 2 or less as described in Section 4.4.2.

T12 Include explicit one-inversion formulas for all special cases with $\gcd(u_1, u_2, v_1 + v_2 + h) \neq 1$ based on Algorithms 33 and 32, producing much more efficient formulas when compared to adaptations of previous suggestions [9] to genus 3.

T13 Use a system of equations for $s = s_2x^2 + s_1x + s_0$, similar to T1, but instead of solving for s, compute the required inverted polynomial q with weight resultant d from the system as described in Section 4.3.6. Then compute $ds = q(v_1 - v_2) \pmod{u_2}$ applying Karatsuba multiplication (Section 4.3.3) twice instead of a Toom style multiplication as done in [37]. The net difference in operation counts for this portion of the algorithms is 5 fewer field additions and two fewer field divisions by 2, but requires one more field multiplication when compared to [37].

Recall that the threshold for trading field multiplications for field additions is capped at 1 multiplication for 3 additions through out (see Section 4.3.1.)

Special Cases

Implementations of frequent case arithmetic include explicit handling of special cases as they come up naturally in the computations via Technique T12. All computation paths through special cases in the formulas require only one inversion and are based on Algorithm 33. Suggestions from previous work [9] adapted to the genus 3 split model setting, (based on the ramified model suggestions of [24]), for handling these cases produced far more cumbersome explicit formulas compared to using the specialization of Balanced NUCOMP to genus 3. The relative reduction in finite field operations can be attributed to applying continued fraction steps from Balanced NUCOMP for adjustment computations and the omission of invoking algorithms for lower-degree divisor classes with weighted inputs.

The basic formulation of the frequent cases Degree 3 Addition and Doubling algorithms are presented as Algorithms 69 and 59 with v in negative reduced basis as input. Note that in frequent cases, n remains unchanged ($n = 0$). Several new arithmetic cases over the divisor class group of a split model curve are presented that were not considered in previous works. These special cases can be invoked based on the degree and balancing coefficient n of both input divisors as described in Section 4.1.3. There are four degree 0 divisors possible, $n = 0, 1, 2, 3$, of which only one (when $n = 2$) is the divisor class group's neutral element. Degree 0 divisor doubling, and degree 0 with degree 0 addition are completely precomputed and discussed in the next section.

6.1.4 Precomputations

Similar to the genus 2 split model setting, there is a need to compute and store the coefficients of V^- along with curve constants, and the extra complexity of split model arithmetic admits many curve constant computations including degree 0 compositions completely dependent on curve constant computations. A precomputation table is used to omit computing the same values multiple times.

As stated above, in order to explicitly compute complete addition and doubling, precomputed results for addition and doubling of divisors $[1, V^-, 0]$, $[1, V^-, 1]$ and $[1, V^-, 3]$ are required. Depending on the hyperelliptic polynomials f and h, the output divisor from composing those

Algorithm 58 Precomputation

```
Input: $h = h_4x^4 + h_3x^3 + h_2x^2 + h_1x + h_0$,
       $f = f_8x^8 + f_7x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$.
1: $V_4^+$ is a solution to the quadratic equation $(V_4^+)^2 + h_4V_4^+ = f_8$.
2: $V_4^- = -V_4^+ - h_4, \quad c_4 = -V_4^- + V_4^+, \quad c_6 = 1/c_4$.
3: $V_3^+ = c_6(f_7 - V_4^+h_3), \quad V_3^- = -V_3^+ - h_3$.
4: $V_2^+ = c_6(f_6 - V_4^+h_2 + V_3^+V_3^-), \quad V_2^- = -V_2^+ - h_2, \quad c_2 = -V_2^- + V_2^+$.
5: $V_1^+ = c_6(f_5 - V_4^+h_1 + V_3^+V_2^- + V_2^+V_3^-), \quad V_1^- = -V_1^+ - h_1, \quad c_1 = -V_1^- + V_1^+$.
6: $c_3 = -V_3^- + V_3^+, \quad c_2 = -V_2^- + V_2^+, \quad c_1 = -V_1^- + V_1^+, \quad c_{10} = V_2^+V_2^-$.
7: $V_0^+ = c_6(f_4 - V_4^+h_0 + V_3^+V_1^- + c_{10} + V_1^+V_3^-), \quad V_0^- = -V_0^+ - h_0$.
8: $c_0 = -V_0^- + V_0^+, \quad c_5 = c_4 + c_2, \quad c_7 = c_1c_6, \quad c_8 = c_2c_6, \quad c_9 = c_3c_6$.
9: $\overline{d}_5 = f_3 - h_0V_3^-, \quad \overline{d}_4 = \overline{d}_5 - h_1V_2^-, \quad \overline{d}_3 = \overline{d}_4 + c_2V_1^-$.
10: $\overline{d}_0 = f_2 - h_0V_2^-, \quad \overline{d}_2 = \overline{d}_0 + V_1^-V_1^+, \quad \overline{d}_1 = f_1 - h_0V_1^-$.
11: $\overline{d}_6 = \overline{d}_1c_6, \quad \overline{d}_7 = \overline{d}_2c_6, \quad \overline{d}_8 = \overline{d}_3c_6, \quad \overline{d}_9 = \overline{d}_4c_6, \quad \overline{d}_{10} = \overline{d}_0c_6$.
12: $d_5 = f_3 - h_0V_3^+, \quad d_4 = d_5 - h_1V_2^+, \quad d_3 = d_4 + c_2V_1^+$.
13: $d_0 = f_2 - h_0V_2^+, \quad d_2 = d_0 + V_1^+V_1^-, \quad d_1 = f_1 - h_0V_1^+$.
14: $d_6 = d_1c_6, \quad d_7 = d_2c_6, \quad d_8 = d_3c_6, \quad d_9 = d_4c_6, \quad d_{10} = d_0c_6$.
15: Compute $u = u_3x^3 + u_2x^2 + u_1x + u_0 = (f - V^-(V^- + h)$ (made monic).
16: Compute $b = b_2x^2 + b_1x + b_0 = V^- - (V^- - V^+) \pmod{u}$.
17: Compute $u' = u_3'x^3 + u_2'x^2 + u_1'x + u_0' = (f - b(b + h)$ (made monic).
18: Compute $b' = b_2'x^2 + b_1'x + b_0' = V^- - (-V^+ - b) \pmod{u'}$.
19: return $[[f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8], [h_0, h_1, h_2, h_3, h_4], [V_0^+, V_1^+, V_2^+, V_3^+, V_4^+]$,
        $[V_0^-, V_1^-, V_2^-, V_3^-, V_4^-], [c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}]$,
        $[d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}], [\overline{d}_0, \overline{d}_1, \overline{d}_2, \overline{d}_3, \overline{d}_4, \overline{d}_5, \overline{d}_6, \overline{d}_7, \overline{d}_8, \overline{d}_9, \overline{d}_{10}]$,
        $[u_0, u_1, u_2, u_3, n, b_0, b_1, b_2], [u_0', u_1', u_2', u_3', n', b_0', b_1', b_2']]$
```

divisor classes may be degree 3,2, 1, or 0. The computation of all possible cases is considered in the precomputation table. For genus 3 split models, it is beneficial to store both $V^+ = V_4^+x^4 + V_3^+x^3 + V_2^+x^2 + V_1^+x + V_0^+$ and $V^- = V_4^-x^4 + V_3^-x^3 + V_2^-x^2 + V_1^-x + V_0^-$ as V^- is often used because of the negative reduced basis. First, V^+ is computed as described in Definition 2.1.42, and then $V^- = -V^+ - h$. Given f and h, V^+ can be computed explicitly, where the leading coefficient of V^+, V_4^+, is set to be a solution to the quadratic equation $V_4^+(V_4^+ + h_4) = f_8$. The precomputed c_i correspond to coefficients of $c = 2V^+ + h$ and convenient precomputations using coefficients of c. The d_i are used throughout the explicit formulas for positive reduced basis and the $\overline{d}_i$ are used in negative reduced basis. The coefficients of the polynomials u and v and balancing coefficient n correspond to the output after adjusting input $[1, V^-, -1]$, the u' and v' and balancing coefficient

n' to the output after twice adjusting input $[1, V^-, -2]$. The polynomials u, V^- and balancing coefficient $n = 0$ are re-used as the output of adjusting input $[1, V^-, 4]$. The balancing coefficients are computed according to Steps 20,23,27 of Algorithm 33.

6.1.5 Explicit Doubling

In this section, complete divisor class doubling is described and the basic formulations of the corresponding explicit formulas are provided. Doubling Algorithm 32 is further specialized into different cases depending on the input divisor class degree and balancing coefficient. Any explicit techniques used are discussed in each case.

The complete doubling algorithm should test the degree of the input divisor in order of statistical frequency and choose the appropriate algorithm depending on the input balancing coefficient n_1. The six doubling formulas used for complete doubling are:

- Degree 3 doubling
- Degree 2 doubling
- Degree 2 doubling with two up adjustments
- Degree 1 doubling
- Degree 1 doubling with down adjustment
- Degree 1 doubling with two up adjustments.

Only precomputed values are required for degree zero doubling, and precomputations from Algorithm 58 may be used. In the frequent cases, the output balancing coefficient n is always zero, and any special cases, n is computed via Steps 5,19,23–24,31–32 of Algorithm 32. Basic formulations of the algorithms required for complete doubling, organized by the degree of the input divisor, are described in the following.

Balancing Coefficient

Let $[u_1, v_1, n_1]$ be an input reduced divisor class representative. Let S be the greatest common divisor of u_1 and $2v_1 + h$ and s be the polynomial computed by Step 8 in Doubling Algorithm 32. There are 3 main cases:

1. If no reduction or adjustment is required as checked by Step 14 of Doubling Algorithm 32 , then $n_n = 2n_1 + \deg(S) - 2$ by Step 5 of Doubling Algorithm 32.

2. If $\deg(s) < 2$, the NUCOMP continued fraction loop is not entered, but at least one reduction or adjustment step is required. There are two sub-cases, if v_1 is normalized with a positive reduced basis then $n_n = 2n_1 + \deg(S) - 2 + 2\deg(u_1) - 4$ by Step 18 of Doubling Algorithm 32. Otherwise for v_1 normalized with a negative reduced basis, $n_n = 2n_1 + \deg(S) - 2 + \deg(u_1) - \deg(s)$ by Step 24 unless $\deg(u_1 s) < 4$, then $n_n = 2n_1 + \deg(S) - 2 + 4 - \deg(u_n)$ where $n_n \geq 0$. Finally, if $n_n < 0$ in the previous case, a final adjustment up is required and one more computation for the balancing coefficient is required where $n_n = n_n + 4 - \deg(u_n)$ by Step 25 of Doubling Algorithm 32.

3. If $\deg(s) = 2$, the NUCOMP continued fraction loop is entered for one iteration, resulting in $n_n = 2n_1 + \deg(S) - 2 + \deg(u_1) - \deg(r)$ with r computed by Step 27 of Doubling Algorithm 32, unless $\deg(z) < 4$ for z computed by Step 30 of Doubling Algorithm 32, in which case $n_n = 2n_1 + \deg(S) - 2 + \deg(u_1) - \deg(s) + 4 - \deg(u_n)$.

Genus 3 Split Model Degree 3 Doubling

Let $[u_1, v_1, n_1]$ be the input reduced divisor class representative with $u_1 = x^3 + u_{12}x^2 + u_{11}x + u_{10}$, $v_1 = V_4^- x^4 + V_3^- x^3 + v_{12}x^2 + v_{11}x + v_{10}$ and $n_1 = 0$. Similar to genus 2 split model doubling Algorithm 28, Steps 1-2 of Doubling Algorithm 32 are always skipped. Steps 3–8 in Algorithm 32 are specialized to degree 3 doubling by computing the resultant $d = \operatorname{Res}(u_1, 2v_1 + h)$ instead of $S = \operatorname{XGCD}(u_1, 2v_1 + h)$ in Step 4 of Algorithm 32, where $d = 0$ implies $S \neq 1$.

To compute a degree 3 double, first create the system of equations required for the alternate computation of the s polynomial in genus 3 (Section 4.3.6). As described in Section 4.3.6 only the coefficients m_1, m_4, m_7 and the determinant $d = \mathrm{Res}(u_1, 2v_1 + h)$ are computed where m_1, m_4, m_7 correspond to $t = m_7x^2 + m_4x + m_1 = d/(2v_1 + h) \pmod{u_1}$. The resultant d is then immediately checked to be zero for special cases as soon as possible in order to minimize unnecessary computations.

The computation of the k polynomial in Step 3 of Algorithm 32 is postponed until after the computation of d, and instead d is immediately checked for special cases. If $d \neq 0$, then in contrast to the genus 2 setting, the coefficients of s are not solved for directly and instead $s = kt \pmod{u_1}$ is computed in Step 8. The computation of Steps 25–33 of Doubling Algorithm 32 follows. If $d = 0$, then Steps 7–24 are performed instead depending on $\deg(u_1)$ and $\deg(s)$, as described in Subroutine 62.

The basic formulation for degree 3 doubling is described in Algorithm 59. An explanation of how each step in Algorithm 59 is converted to an explicit formula follows. In Step 1, the polynomial representation

$$k_2'x^2+k_1'x+k_0' = (\tilde{v}_2x^2+\tilde{v}_1x+\tilde{v}_0)(s_2x^2+s_1x+s_0)-(s_2\tilde{v}_2x+\tilde{v}_2s_1+s_2(\tilde{v}_1-u_{12}\tilde{v}_2))(x^3+u_{12}x^2+u_{11}x+u_{10})$$

is used to set up a system of equations linear in the coefficients of s where

$$s = s_2x^2 + s_1x + s_0 = (k_2'x^2 + k_1'x + k_0')/(\tilde{v}_2x^2 + \tilde{v}_1x + \tilde{v}_0) \pmod{x^3 + u_{12}x^2 + u_{11}x + u_{10}},$$

and $\tilde{v} = 2v_1 + h \pmod{x^3 + u_{12}x^2 + u_{11}x + u_{10}}$. By equating coefficients of the polynomials in the above identity, the following linear system of equations given by the 3 by 3 matrix T arises:

$$\begin{pmatrix} t_1 = \tilde{v}_0 & t_2 = -u_{10}\tilde{v}_2 & t_3 = -u_{10}(\tilde{v}_1 - u_{12}\tilde{v}_2) \\ t_4 = \tilde{v}_1 & t_5 = \tilde{v}_0 - u_{11}\tilde{v}_2 & t_6 = -u_{10}\tilde{v}_2 - u_{11}(\tilde{v}_1 - u_{12}\tilde{v}_2) \\ t_7 = \tilde{v}_2 & t_8 = \tilde{v}_1 - u_{12}\tilde{v}_2 & t_9 = \tilde{v}_0 - u_{11}\tilde{v}_2 - u_{12}(\tilde{v}_1 - u_{12}\tilde{v}_2) \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} k_0 \\ k_1 \\ k_2 \end{pmatrix}.$$

The resulting explicit formula for Step 1 is

$$T_0 = u_{12}c_4,$$

Algorithm 59 Genus 3 Split Model Degree 3 Doubling

Input: $[D_1] = [u_1, v_1, 0]$

Output: $[u_n, v_n, n_n] = 2[D_1]$

```
1: Compute matrix coefficients $t_i$ representing $s(h+2v_1) \equiv k \pmod{u_1}$ as the 3x3 linear system
   $\begin{bmatrix} t_1 & t_2 & t_3 \\ t_4 & t_5 & t_6 \\ t_7 & t_8 & t_9 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} k_0 \\ k_1 \\ k_2 \end{bmatrix}$.
2: Compute coefficients $m_1 = t_5t_9,\ m_4 = t_6t_7 - t_4t_9,\ m_7 = t_4t_8 - t_5 - t_7$ of the inverse of the matrix
   and let $t = m_7x^2 + m_4x + m_1$.
3: $d = t_1m_1 + t_2m_4 + t_3m_7$.
4: if $d = 0$ then
5:     Go to Degree 3 Doubling ($d = 0$) Subroutine 62.
6: $k' = (f - v_1(v_1 + h))/u_1 \pmod{u_1}$.
7: $s' = ds = s_2'x^2 + s_1'x + s_0' = k't \pmod{u_1}$.
8: if $s_2' = 0$ then
9:     Go to Degree 3 Doubling ($s_2 = 0$) Subroutine 60.
10: Compute monic quotient $q' = x + s_2'(q_0/q_1) = x + q_0'$ of $u_1/s'$ with $1/q_1 = s_2$.
11: Compute remainder $r' = s_2'dr = r_1'x + r_0'$ of $u_1/s'$.
12: if $r_1'$ is 0 then
13:     Go to Degree 3 Doubling ($r_1' = 0$) Subroutine 61.
14: Compute values $q_1/r_1, r_1/q_1, q_0/q_1, r_0/q_1, q_0/r_1, r_0/r_1$.
15: $M_2' = (r(2v_1 + h) + qk)/u_1$   (exact division, made monic).
16: $u_n = r^2/(c_4r_1q_1) - qM_2'/q_1$.
17: $z = (u_1r - u_n(c_4r_1q_1))/q$   (exact division).
18: $v_n = V^- - [(V^- - z + v_1 + h) \pmod{u_n}]$.
19: return $[u_n, v_n, 0]$.
```

$$T_1 = c_3 - T_0,$$

$$v_{n_0} = -v_{10} - h_0,$$

$$v_{n_1} = -v_{11} - h_1,$$

$$v_{n_2} = -v_{12} - h_2,$$

$$\hat{v}_1 = v_{11} - v_{n_1},$$

$$\hat{v}_2 = v_{12} - v_{n_2},$$

$$t_1 = v_{10} - v_{n_0} + u_{10}T_1,$$

$$t_4 = \hat{v}_1 + u_{10}c_4 + u_{11}T_1,$$

$$t_7 = \hat{v}_2 + u_{11}c_4 + u_{12}T_1,$$

$$\begin{aligned}
t_2 &= -u_{10}t_7,\\
t_5 &= t_1 - u_{11}t_7,\\
t_8 &= t_4 - u_{12}t_7,\\
t_3 &= -u_{10}t_8,\\
t_6 &= t_2 - u_{11}t_8,\\
t_9 &= t_5 - u_{12}t_8.
\end{aligned}$$

In Steps 2–3, the inverse of T from Step 1 is computed omitting divisions by $\det(T)$, yielding

$$\det(T)T^{-1} = M = \begin{pmatrix} m_1 = t_5t_9 - t_8t_6 & m_2 = t_3t_8 - t_2t_9 & m_3 = t_2t_6 - t_3t_5 \\ m_4 = t_6t_7 - t_4t_9 & m_5 = t_1t_9 - t_3t_7 & m_6 = t_3t_4 - t_1t_6 \\ m_7 = t_4t_8 - t_5t_7 & m_8 = t_2t_7 - t_1t_8 & m_9 = t_1t_5 - t_2t_4 \end{pmatrix}.$$

Only the terms m_1, m_4, m_7 are required to compute the polynomial inversion of $\tilde{v}$ modulo u_1 where

$$t = m_7x^2 + m_4x + m_1 = \frac{d}{\tilde{v}} \pmod{u_1}$$

without the field inversion to divide by d. The computation of $d = \det(T)$ is chosen to reuse the same m_i. The resulting explicit formula for Steps 2–3 is

$$\begin{aligned}
m_1 &= t_5t_9 - t_6t_8,\\
m_4 &= t_6t_7 - t_4t_9,\\
m_7 &= t_4t_8 - t_5t_7,\\
d &= t_1m_1 + t_2m_4 + t_3m_7.
\end{aligned}$$

The value $d = 0$ is then checked for special cases as soon as possible in order to minimize unnecessary computations.

If $d \neq 0$, in Step 6 $k' = (f - v_1(v_1 + h))/u_1 \pmod{u_1}$ is computed by an exact division, and school book modular reduction of a degree 3 polynomial by a monic degree 3 modulus. Coefficients k_i of k without the modular reduction are saved. The explicit formula for Step 6 is

$$t_1 = v_{11} - V_1^-,$$

$$t_2 = v_{12} - V_2^-,$$

$$k_3 = c_4 t_2,$$

$$t_{10} = u_{12} k_3,$$

$$k_2 = c_3 t_2 + c_4 t_1 - t_{10},$$

$$k_2' = k_2 - t_{10},$$

$$t_{11} = u_{11} k_3,$$

$$k_1 = c_4(v_{10} - V_0^-) + c_3 t_1 + v_{21} v_{n_2} - c_{10} - u_{12} k_2 - t_{11},$$

$$k_1' = k_1 - t_{11},$$

$$t_{12} = u_{10} k_3,$$

$$k_0' = d_{n_5} + c_3 v_{10} - h_2 v_{11} - v_{12}\hat{v}_1 - u_{11} k_2 - u_{12} k_1 - t_{12} - t_{12}.$$

In Step 7, $s' = k't \pmod{u_1}$ is computed by applying Karatsuba multiplication (see Section 4.3.3), and Karatsuba modular reduction (see Section 4.3.4) of a degree 4 polynomial and monic degree 3 modulus. The explicit formula for Step 7 is

$$t_0 = k_0' m_1,$$

$$t_1 = k_1' m_4,$$

$$t_2 = k_2' m_7,$$

$$t_3 = (m_4 + m_7)(k_1' + k_2') - t_2 - t_1 - u_{12} t_2,$$

$$t_4 = u_{11} t_2,$$

$$t_5 = t_4 - t_1,$$

$$s_0' = t_0 - u_{10} t_3,$$

$$s_1' = (m_1 + m_4)(k_0' + k_1') - s_0' - (u_{10} + u_{11})(t_2 + t_3) + t_5,$$

$$s_2' = (m_1 + m_7)(k_0' + k_2') - t_0 - t_2 - t_5 - u_{12} t_3.$$

Rather than directly computing the polynomials $q = q_1x + q_0$ and $r = r_1x + r_0$, monic q with weight s'_2 and weighted r with weight s'_2d are computed as q' and r' in Steps 10–11 and note that $q_1 = 1/s_2 = d/s'_2$. The value $r'_1 = 0$ is immediately checked after creation to minimize unnecessary computations. The resulting explicit formula for Steps 10–11 (with the actual values given in brackets) is

$t_6 = s'^2_2,$	$(s_2^2d^2)$
$t_7 = ds'_2,$	(s_2d^2)
$q'_0 = u_{12}s'_2 - s'_1,$	$((q_0/q_1)s_2d)$
$r'_1 = t_6u_{11} - s'_0s'_2 - s'_1q'_0,$	$((r_1/q_1)s_2d^2)$
$r'_0 = t_6u_{10} - s'_0q'_0,$	$((r_0/q_1)s_2d^2)$

If $r'_1 \neq 0$, all necessary ratios of the coefficients of q and r that arise in Steps 15–18 for computing M'_2, u_n, v_n are computed beforehand in Step 14. A description of how these values are applied is given respectively in each corresponding step. The explicit formula for Step 14 (with the actual values given in brackets) is

$w_3 = (t_7r'_1)^{-1},$	$((q_1/r_1)(1/(s_2^2d^4)))$
$w_4 = w_3r'_1,$	$(1/(s_2d^2))$
$w_5 = w_4d,$	$(1/(s_2d))$
$w_6 = r'_1w_5^2c_4,$	(r_1c_4)
$q_{r_{11}} = w_3t_7^2,$	(q_1/r_1)
$r_{q_{11}} = r'_1w_4,$	(r_1/q_1)
$q_{q_{01}} = q'_0w_5,$	(q_0/q_1)
$r_{q_{01}} = r'_0w_4,$	(r_0/q_1)
$q_{r_{01}} = q_{q_{01}}q_{r_{11}},$	(q_0/r_1)
$r_{r_{01}} = r_{q_{01}}q_{r_{11}}.$	(r_0/r_1)

In Step 15, $M_2' = M_2/(c_4r_1) = (r(2v_1 + h) + qk)/((c_4r_1)u_1)$ the negative of M_2 monic is computed instead of M_2, with $c_6 = 1/c_4$. The polynomial $M_2 = (r(2v_1 + h) + qk)/u_1$ is directly computed and then symbolically multiplied by $1/(r_1c_4)$ where all terms in the equation for each coefficient M_{21}' and M_{20}' that have divisions by q_1 or r_1 are replaced with the precomputed ratios from Step 14. Any terms that require a division by c_4 are grouped together and multiplied by c_6. The explicit formula for Step 15 is

$$\begin{aligned} M_{21}' &= c_6(c_3 - q_{r_{11}}k_3) + r_{r_{01}} - u_{12} \\ M_{20}' &= c_6(r_{r_{01}}c_3 - 2v_{12} - h_2 - q_{r_{11}}k_2 - q_{r_{01}}k_3) - u_{11} - u_{12}M_{21}'. \end{aligned}$$

In Step 16, the output polynomial

$$u_n = r^2 - qM_2 = r^2/(c_4r_1q_1) + (qM_2')/q_1$$

is computed using the same approach as in Step 15. The explicit formula for Step 16 is:

$$\begin{aligned} u_{n_2} &= c_6r_{q_{11}} + M_{21}' + q_{q_{01}} \\ u_{n_1} &= c_6r_{q_{01}}M_{20}' + q_{q_{01}}M_{21}' \\ u_{n_0} &= c_6r_{q_{01}}r_{r_{01}} + q_{q_{01}}M_{20}'. \end{aligned}$$

The original leading coefficient of u_n is $c_4r_1q_1$. Since u_n is required to be monic, the computation of $z = (ru_1 - u_n)/q$ in Step 17 must be adjusted to $z = (ru_1 - u_n(c_4r_1q_1))/q$. The exact polynomial division by $q = q_1x + q_0$ introduces divisions in the coefficients of z where the precomputed ratios from Step 14 are used instead. Steps 17 and 18 are combined as

$$v_n = V^- - [(V^- + v_1 + h - z) \pmod{u}],$$

using a Karatsuba modular reduction (see Section 4.3.4) to perform the modular reduction. The explicit formula for Steps 17–18 is

$$\begin{aligned} t_3 &= c_3 + r_{q_{11}} - u_{n_2}c_4, \\ t_2 &= r_{q_{11}}(u_{n_2} - q_{q_{01}}) + r_{q_{01}} - w_6, \end{aligned}$$

$$t_1 = r_{q_{11}} u_{11} + r_{q_{01}} u_{12} - w_6 u_{n_2} - q_{q_{01}} t_2,$$

$$t_5 = u_{n_0} t_3,$$

$$t_4 = u_{n_1} c_4,$$

$$v_2 = -v_{12} - h_2 + t_2 - t_4 - u_{n_2} t_3,$$

$$v_1 = -v_{11} - h_1 + t_1 - (u_{n_0} + u_{n_1})(c_4 + t_3) + t_4 + t_5,$$

$$v_0 = -v_{10} - h_0 + r_{q_{11}} u_{n_0} - w_6 u_{n_1} - q_{q_{01}} t_1 - t_5.$$

Basic formulations of the special output cases $s'_2 = 0$ and $r_1 = 0$ are given in Subroutines 60 and 61. There are two types of special output cases, $\deg(s') < 2$ and $\deg(s) = 2$ but $\deg(r) < 1$. If $\deg(s') < 2$, then the computation of q, r is omitted and the output is computed similarly to degree 2 doubling Algorithm 43. In the case that $\deg(s') = 2$ but $\deg(r) < 1$, then $\deg(z) < 4$ and lower degree output divisors using same approach as the frequent case are computed with output n_n computed as described in Section 6.1.5.

Subroutine 60 Genus 3 Split Model Degree 3 Doubling ($s'_2 = 0$)

```
1:  if $s'_1$ is 0 then
2:      if $s'_0$ is 0 then return $[1, V^+, 2]$.
3:      else
4:          Compute $1/(s_0)$ and $s = s_0$.
5:          $M_2 = (s(2v_1 + h) - k)/u_1$   (exact division).
6:          $u_n = s^2 + M_2$,    (made monic).
7:          $v_n = V^- - (V^- + su_1 + v_1 + h) \pmod{u_n}$.
8:          return $[u_n, v_n, 1]$
9:  if Degree 4 term $M_{24}$ of $s(2v_1 + h) - k$ is 0 then
10:     if Degree 3 term $M_{23}$ of $s(2v_1 + h) - k$ is 0 then return $[1, V^-, 0]$
11:     else
12:         Compute $1/M_{24}$ and $s = s_0$.
13:         $M_2 = (s(2v_1 + h) - k)/u_1$   (exact division).
14:         $u_n = s^2 + M_2$,    (made monic).
15:         $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
16:         return $[u_n, v_n, 0]$
17: Compute $1/M_{25}$ degree 5 term of $s(2v_1 + h) - k$ and $s = s_1 x + s_0$.
18: $M_2 = (s(2v_1 + h) - k)/u_1$   (exact division).
19: $u_n = s^2 + M_2$,    (made monic).
20: $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
21: return $[u_n, v_n, 0]$.
```

Subroutine 61 Genus 3 Split Model Degree 3 Doubling ($r_1 = 0$)

```
 1: if $r_0$ is 0 then
 2:     $u_n = -q^2k/u_1$,    (exact division, made monic)
 3:     $v_n = V^- - (V^- - v_1 - h) \pmod{u_n}$.
 4:     return $[u_n, v_n, 2]$.
 5: else
 6:     if Degree 4 term $M_{24}$ of $r(2v_1 + h) - qk$ is 0 then
 7:         if Degree 3 term $M_{23}$ of $r(2v_1 + h) - qk$ is 0 then return $[1, V^-, 3]$
 8:         else
 9:             Compute $1/M_{24}$ and $r_0$.
10:             $M_2 = (r(2v_1 + h) + qk)/u_1$   (exact division).
11:             $u_n = r^2 - qM_2$,    (made monic).
12:             $z = (u_1r - u_n)/q$    (exact division).
13:             $v_n = V^- - [(V^- - z + v_1 + h) \pmod{u_n}]$.
14:             return $[u_n, v_n, 2]$
15:     Compute $1/q_1$, $q_0/q_1$ and $r_0/q_1$.
16:     $M_2 = (r(2v_1 + h) + qk)/u_1$   (exact division).
17:     $u_n = r^2 - qM_2$,    (made monic).
18:     $z = (u_1r - u_n(c_4r_0 + q_1k_3))/q$    (exact division).
19:     $v_n = V^- - [(V^- - z + v_1 + h) \pmod{u_n}]$.
20:     return $[u_n, v_n, 1]$
```

The basic formulation of the special case $d = 0$ is described in Subroutine 62. This case implies that $\gcd(h + 2v_1, u_1)$ has either degree 1, 2 or 3. Similarly to genus 2 split model doubling (Algorithm 44), the common factor is removed from u_1 via an exact division (Section 4.3.2), and $s, u_, v_n$ are then computed accordingly depending on the degree. The output balancing coefficient n_n is computed as described in Section 6.1.5.

Subroutine 62 Genus 3 Split Model Degree 3 Doubling ($d = 0$)

```
1:  **if** $m_7$ is 0 **then**
2:      **if** $t_7$ is 0 **then** **return** $[1, V^-, 1]$.
3:      **else** $d_w = (h + 2v_1) \pmod{u_1} = t_7x^2 + t_4x + t_1$.
4:          $b_1 = 1/\mathrm{lcf}(d_w)$ and $d_w$ made monic.
5:          $k = (f - v_1(v_1 + h))/u_1$.
6:          $u_1 = u_1/d_w$.
7:          $s = b_1k \pmod{u_1}$.
8:          $u_n = u_1^2$, $v_n = V^- - (V^- - v_1 - u_1s) \pmod{u_n}$.
9:          **return** $[u_n, v_n, 0]$
10: **else** $m_8 = t_2t_7 - t_1t_8$.
11:     $S = -m_7x + m_8$.
12:     $u'_1 = u_1/S = u'_{11}x + u'_{10}$ where $u'_{11}$ has weight $m_7$ and $u'_{10}$ weight $m_7^2$.
13:     Compute $m_i$ in system of equations for $su_1 \equiv \tilde{v} \pmod{u_2}$ as $\begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix}\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} \tilde{v}_0 \\ \tilde{v}_1 \end{bmatrix}$,
         where $\tilde{v} = \tilde{v}_1x + \tilde{v}_0 = (2v_1 + h) \pmod{u'_1}$.
14:     $d_2 = m_1m_4 - m_2m_3$.
15:     $k = (f - v_1(v_1 + h))/u'_1)$.
16:     Solve for $s_1(d_2/m_7), s_0(d_2)$ using $m_i$ and Cramer's rule, omitting the inversion of $d_2$.
17:     **if** $s_1(d_2/m_7)$ is 0 **then**
18:         **if** $s_0(d_2)$ is 0 **then**
19:             **if** $k_3 = 0$ **then** $u_n = -k/u_1$,   (made monic).
20:                 $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
21:                 **return** $[u_n, v_n, 2]$
22:             **else** $u_n = -k/u_1$,   (made monic).
23:                 $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
24:                 **return** $[u_n, v_n, 1]$
25:         **else**
26:             **if** Degree 4 term $M_{24}$ of $s(2v_1 + h) - k$ is 0 **then**
27:                 **if** Degree 3 term $M_{23}$ of $s(2v_1 + h) - k$ is 0 **then return** $[1, V^-, 3]$
28:                 **else** Compute $1/M_{24}$, and unweighted $u_1$.
29:                     $M_2 = (s(2v_1 + h) - k)/u'_1$   (exact division).
30:                     $u_n = s^2 + M_2$,   (made monic).
31:                     $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
32:                     **return** $[u_n, v_n, 2]$
33:             Compute $1/M_{24}$, $s = s_0$ and unweighted $u_1$.
34:             $M_2 = (s(2v_1 + h) - k)/u'_1$   (exact division).
35:             $u_n = s^2 + M_2$,   (made monic).
36:             $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
37:             **return** $[u_n, v_n, 1]$
38:     Compute $s = s_1x + s_0$ and unweighted $u_1$ and $k$.
39:     $M_2 = (s(2v_1 + h) - k)/u'_1$   (exact division).
40:     $u_n = s^2 + M_2$,   (made monic).
41:     $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
42:     **return** $[u_n, v_n, 0]$.
```

Genus 3 Split Model Degree 2 Doubling

Let $[u_1, v_1, n_1]$ be the input reduced divisor class representative with $u_1 = x^2 + u_{11}x + u_{10}$, $v_1 = V_4^- x^4 + V_3^- x^3 + V_2^- x^2 + v_{11}x + v_{10}$ and $n_1 = 0$ or $n_1 = 1$. There are two degree 2 doubling algorithms required for complete divisor class arithmetic; degree 2 doubling, and degree 2 doubling with two up adjusts. The balancing coefficient $n = 2n_1 - 2$ is -2 or 0, depending on whether n_1 is 0 or 1 respectively. The degree of the divisor after degree 2 doubling is 4 (where $4 = g + 1$), so an adjustment is required for a final reduction step.

If $n_1 = 1$ then a down adjustment is applied as part of the reduction by normalizing v_1 in positive reduced basis before any other computations. If $n_1 = 0$, then an up adjustment is applied as the reduction bringing the balancing coefficient to -1 and a second up adjustment is required for n to be in range. The composition and first adjustment steps are combined via continued fraction steps of Doubling Algorithm 32 as described in T10, and the second up adjustment for degree 2 doubling with two up adjusts is applied as described in Steps 21–23 of Algorithm 32 using Montgomery's inversion trick (Section 4.3.7). The basic formulation for degree 2 doubling is described in Algorithm 63. An explanation of how each step in Algorithm 63 is converted to an explicit formula is given in the following.

The formulation of degree 2 doubling in the balanced setting over genus 3 split models, including the special case where $d = 0$, is similar to degree 2 doubling over genus 2 split models (Algorithm 43). Both compute the composition related to a degree 4 intermediate divisor class representative, but in the genus 2 case, there is a reduction to a degree 2 or less divisor class representative, and in genus 3 the reduction results in a degree 3 or less divisor class representative via an adjustment step. Adjustment steps only have one possibility for a special output case, instead of two as in Algorithm 43. The alternate computation of k (Technique T8) from Algorithm 43 is adapted to the genus 3 setting where $z = k/c_4 = (f - v_1(v_1 + h))/u_1c_4$ is computed instead (Technique T11).

The basic formulation for degree 2 doubling with two up adjustments is described in Algorithm 64. An explanation of how each step in Algorithm 64 is converted to an explicit formula is given in the following.

Algorithm 63 Genus 3 Split Model Degree 2 Doubling

Input: $[D_1] = [u_1, v_1, 1]$

Output: $[u_n, v_n, n_n] = 2[D_1]$

```
1: $v_1 = V^+ - [(V^+ - v_1) \pmod{u_1}]$.
2: Compute $t_i$ in system for $s(h + 2v_1) \equiv k \pmod{u_1}$ as $\begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix}\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} k_0 \\ k_1 \end{bmatrix}$, where $(h + 2v_1) \pmod{u_1} = d_w = -m_3x + m_4$.
3: $d = m_1m_4 - m_2m_3$.
4: if $d = 0$ then
5:     if $m_3$ is 0 then return $[1, V^+, 2]$.
6:     else
7:         $k = (f - v_1(v_1 + h))/u_1$.
8:         $b_1 = 1/\mathrm{lcf}(d_w)$ and $d_w$ made monic.
9:         $u_1 = u_1/d_w$.
10:        $s = b_1k \pmod{u_1}$.
11:        $u_n = u_1^2$.
12:        $v_n = V^+ - (V^+ - v_1 - u_1s) \pmod{u_n}$.
13:        return $[u_n, v_n, 1]$.
14: $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1) \pmod{u_1}$.
15: Solve for $s'_1 = ds_1, s'_0 = ds_0$ using $m_i$ and Cramer's rule, omitting the inversion of $d$.
16: if $s'_1 = 0$ then
17:     if $s'_0 = 0$ then
18:         if $z_3 = 0$ then return $[1, V^-, 0]$.
19:         else
20:             Compute $1/z_3$.
21:             $u_n = z/u_1$   (exact division, made monic).
22:             $v_n = V^- - (V^- - v_1 - h) \pmod{u_n}$.
23:             return $[u_n, v_n, 0]$
24:     else
25:         Compute $1/s_0$ and $s = s_0$.
26:         $M_2 = (s(2v_1 + h)/c_4 - z)/u_1$   (exact division).
27:         $u_n = s^2 + M_2$,   (made monic).
28:         $v_n = V^- - (V^- + su_1 + v_1 + h) \pmod{u_n}$.
29:         return $[u_n, v_n, 0]$
30: Compute $1/(s_1)$ and $s = s_1x + s_0$.
31: $M_2 = (s(2v_1 + h)/c_4 - z)/u_1$   (exact division).
32: $u_n = s^2 + M_2$,   (made monic).
33: $v_n = V^- - (V^- + su_1 + v_1 + h) \pmod{u_n}$.
34: return $[u_n, v_n, 0]$.
```

Algorithm 64 Genus 3 Split Model Degree 2 Doubling with two Up Adjusts

Input: $[D_1] = [u_1, v_1, 0]$

Output: $[u_n, v_n, n_n] = 2[D_1]$

```
1: $v_1 = V^+ - [(V^+ - v_1) \pmod{u_1}]$.
2: Compute $t_i$ in system for $s(h + 2v_1) \equiv k \pmod{u_1}$ as $\left[\begin{smallmatrix} m_1 & m_2 \\ m_3 & m_4 \end{smallmatrix}\right]\left[\begin{smallmatrix} s_0 \\ s_1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} k_0 \\ k_1 \end{smallmatrix}\right]$, where $(h + 2v_1) \pmod{u_1} = d_w = -m_3x + m_4$.
3: $d = m_1m_4 - m_2m_3$.
4: if $d = 0$ then
5:     Go to Genus 3 Split Model Degree 2 Doubling with 2 Up Adjusts ($d = 0$) Subroutine 65.
6: $k = k_3x^3 + k_2x^2 + k_1x + k_0 = (f - v_1(v_1 + h))/u_1 \pmod{u_1}$.
7: Solve for $s_1(d), s_0(d)$ using $m_i$ and Cramer's rule, omitting the inversion of $d$.
8: if $s_1' = 0$ then
9:     if $s_0' = 0$ then
10:        if $k_3 = 0$ then return $[1, V^-, 2]$.
11:        else Compute $1/k_3$.
12:            $u_n = k/u_1$   (exact division, made monic).
13:            $v_n = V^- - (V^- - v_1 - h) \pmod{u_n}$.
14:            return $[u_n, v_n, 1]$
15:    else  Compute $1/s_0$ and $s = s_0$.
16:        $M_2 = (s(2v_1 + h)/c_4 - z)/u_1$   (exact division).
17:        $u_n = s^2 + M_2$,   (made monic).
18:        $v_n = V^- - (V^- + su_1 + v_1 + h) \pmod{u_n}$.
19:        return $[u_n, v_n, 0]$
20: $W = -s_1(d)d$,   $W_2 = W^2$.
21: $M_2 = (s(2v_1 + h) - k)/u_1$,   with weight $W$.
22: $u' = s^2 + M_2$,   with weight $W$.
23: $v' = V^- - (V^- + su_1 + v_1 + h) \pmod{u'}$, with weight $W_2$.
24: $w = w_6x^6 + w_5x^5 + w_4x^4 + w_3x^3 + w_2x^2 + w_1x + w_0 = (f - v'(v' + h))/c_4$, weighted.
25: Compute $1/W$, $1/W_2$ and unweighted $u', v'$
26: $u_n = w/u'$   (exact division, made monic).
27: $v_n = V^- - (V^- + v' + h) \pmod{u_n}$.
28: $n_n = 3 - \deg(u_n)$.
29: return $[u_n, v_n, n_n]$.
```

Degree 2 doubling with two adjusts is similar to degree 2 doubling, but does not normalize v_1 into positive reduced basis, and requires one more adjustment step at the end. The inversion in the degree 2 doubling portion is omitted using Montgomery's inversion trick (see Section 4.3.7), and is instead computed in the final adjustment. Computing $z = k/c_4$ instead of k by Technique T11 does not benefit any operation that requires 2 up adjustments and is omitted.

The basic formulation for degree 2 doubling with two adjusts is described in Subroutine 65. An explanation of how each step in Subroutine 65 is converted to an explicit formula is given in the following.

Algorithm 65 Genus 3 Split Model Degree 2 Doubling with two Up Adjusts ($d = 0$)

```
1: **if** $m_3$ is 0 **then** **return** $[1, V^+, 0]$.
2: $k = (f - v_1(v_1 + h))/u_1$.
3: $u_1(-m_3) = u_1/d_w$, with weight $-m_3$.
4: $s = k \pmod{u_1(-m_3)}$, with weight $M_3 = (-m_3)^3$.
5: $k = kdw$, with weight $M_4 = m_3^4$.
6: **if** Degree 4 term $M_{24}$ of $s(2v_1 + h) - k$ is 0 **then**
7:     **if** Degree 3 term $M_{23}$ of $s(2v_1 + h) - k$ is 0 **then**
8:         **if** Degree 2 term $M_{22}$ of $s(2v_1 + h) - k$ is 0 **then return** $[1, V^-, 3]$
9:         **else** Compute $1/m_3$, $1/M_{22}$ and $s = s_0$.
10:            $M_2 = (s(2v_1 + h) - k)/u_1$ (exact division).
11:            $u_n = s^2 + M_2$, (made monic).
12:            $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
13:            **return** $[u_n, v_n, 2]$
14:    **else** Compute $1/m_3$, $1/M_{23}$ and $s = s_0$.
15:        $M_2 = (s(2v_1 + h) - k)/u_1$ (exact division).
16:        $u_n = s^2 + M_2$, (made monic).
17:        $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
18:        **return** $[u_n, v_n, 1]$
19: Compute $1/m_3$, $1/M_{24}$ and $s = s_0$.
20: $M_2 = (s(2v_1 + h) - k)/u_1$ (exact division).
21: $u_n = s^2 + M_2$, (made monic).
22: $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
23: **return** $[u_n, v_n, 1]$
```

For $d = 0$, a second up adjustment is not required in any cases, as the degree of the common divisor $\gcd(u_1, 2v_1 + h)$ is added to the balancing coefficient n_n as described in Section 6.1.5, resulting in a similar approach to Degree 2 doubling Algorithm 46 computing the single up adjustment on a degree 1 doubling. Lower degree output divisor class representatives are compute based on cancellations in the coefficients of M_2.

Degree 1 Doubling

Let $[u_1, v_1, n_1]$ be the input reduced divisor class representative with $u_1 = x + u_{10}$, $v_1 = V_4^- x^4 + V_3^- x^3 + V_2^- x^2 + V_1^- x + v_{10}$ and $n_1 = 0$, $n_1 = 1$ or $n_1 = 2$. There are three degree 1 doubling algorithms required for complete divisor class arithmetic; degree 1 doubling, degree 1 doubling with down adjust and degree 1 doubling with two up adjusts. Depending on the input balancing coefficient, if n_1 is 0,1 or 2, the resulting $n = 2n_1 - 2$ value is -2, 0 or 2 respectively. The degree of the divisor after degree 1 doubling is 2, and so the output balancing coefficient range is $n \in [0, 1]$. If $n_1 = 0$ two up adjustments are required to produce a reduced divisor class representative, if $n_1 = 1$ then no adjustments are required, and if $n_1 = 2$ then one down adjustment is required. The formulation of degree one doubling Algorithm 66 follows Doubling Algorithm 32 closely. The basic formulation for degree 1 doubling is described in Algorithm 66. Degree 1 Doubling Algorithm 66 is similar to

Algorithm 66 Genus 3 Split Model Degree 1 Doubling

Input: $[D_1] = [u_1, v_1, 1]$.

Output: $[u_n, v_n, n_n] = 2[D_1]$.

```
1: $d = (h + 2v_1)/c_4 \pmod{u_1}$.
2: **if** $d = 0$ **then return** $[1, V^-, 1]$.
3: $z = (f - v_1(v_1 + h))/(c_4 u_1)$.
4: $s_0 = z/d \pmod{u_1}$.   (note that $1/d = b_1 \pmod{u_1}$ in Alg 32)
5: $u_n = u_1^2$.
6: $v_n = V^- - (V^- - s_0 u_1 - v_1) \pmod{u_n}$.
7: **return** $[u_n, v_n, 0]$.
```

genus 2 Degree 1 Addition ($d = 0$) Subroutine 53, where the balancing coefficient n_n is computed as described in Section 6.1.5.

For degree 1 double with down adjust, v_1 is first normalized into positive reduced basis accommodating a down adjustment as described in Section 4.1.3. The composition and adjustment steps are combined via Doubling Algorithm 32 as described in T10. If d is 0, then the degree of S from line 4 of Algorithm 32 is one, resulting in a degree 0 divisor output where the balancing coefficient is computed as $n_n = 2n_1 + \deg(S) - 2 = 3$ based on the input balancing coefficient $n_1 = 2$ (see Section 6.1.5). Special output cases corresponding to the degree of $s(2v + h)/c_4 - z$ after the adjustment step are accounted for. The basic formulation for Degree 1 Doubling with down Adjust is described by Algorithm 67. Degree one doubling with down adjustment Algorithm 67 is similar

Algorithm 67 Genus 3 Split Model Degree 1 Doubling with Down Adjust

Input: $[D_1] = [u_1, v_1, 2]$.

Output: $[u_n, v_n, n_n] = 2[D_1]$.

```
1:
2: $d = (h + 2v_1) \pmod{u_1}$.
3: **if** $d = 0$ **then return** $[1, V^-, 3]$.
4: $z = (f - v_1(v_1 + h))/(c_4 u_1)$.
5: $s'_0 = ds_0 = c_4 z \pmod{u_1}$   with weight $d$.
6: **if** $s'_0 = 0$ **then**
7:     $u_n = -z/u_n$   (exact division, made monic).
8:     $v_n = V^- - (V^- - v_1) \pmod{u_n}$.
9:     **return** $[u_n, v_n, 0]$
10: Compute $1/s_0$ and $s = s_0$.
11: $M_2 = (s(2v_1 + h)/c_4 - z)/u_n$   (exact division, made monic).
12: $u_n = s_0^2 + M_2$.
13: $v = V^- - (V^- - s_0 u_1 - v_1) \pmod{u_n}$.
14: **return** $[u_n, v, 0]$.
```

to genus 2 Degree 1 Doubling with Down Adjust Algorithm 45, with the addition of normalizing v_1 into positive reduced basis at the beginning. The balancing coefficient n_n is computed as described in Section 6.1.5.

For degree 1 double with two up adjusts, the composition and first adjustment step are combined via continued fraction steps of Doubling Algorithm 32 as described in T10, and the second up adjustment is applied as described in Steps 21–23 of Algorithm 32 using Montgomery's inversion trick (Section 4.3.7). If $d = 0$, then the degree of S from line 4 of Algorithm 32 is one, resulting in a degree 0 divisor output, where the balancing coefficient is computed as $n = 2n_1 + \deg(S) - 2 = -1$ based on the input balancing coefficient $n_1 = 0$ (see Section 6.1.5). A precomputed adjustment of $[1, V^-, -1]$ is returned (see Section 6.1.4). Both special output cases corresponding to the degree of $s(2v_1 + h) - k$ after continued fraction steps, and the degree of $f - v(v + h)$ in the extra adjustment step are accounted for. Technique T11 is not applied in this case due to the use of Montgomery's trick. The basic formulation for degree 1 doubling with two up adjustments is described by Algorithm 68.

Algorithm 68 Genus 3 Split Model Degree 1 Doubling with Two Up Adjusts

Input: $[D_1] = [u_1, v_1, 0]$.

Output: $[u_n, v_n, n_n] = 2[D_1]$.

```
1:
2: $d = (h + 2v_1) \pmod{u_1}$.
3: if $d = 0$ then return Precomputed up adjustment of $[1, V^-, -1]$.
4: $k = (f - v_1(v_1 + h))/u_1$.
5: $s'_0 = ds_0 = k \pmod{u_1}$   with weight $d$.
6: if $s'_0 = 0$ then
7:     $u_n = -k/u_n$   (exact division, made monic).
8:     $v_n = V^- - (V^- - v_1) \pmod{u_n}$.
9:     $n_n = 3 - \deg(k)$
10:    return $[u_n, v_n, n_n]$
11: $W = s'_0 d$,   $W_2 = W^2$.
12: $M_2 = (s'_0(2v_1 + h) - k)/u_1$,   with weight $W$.
13: $u' = (s'_0)^2 + M_2$,   with weight $W$.
14: $v' = V^- - (V^- + s'_0 u_1 + v_1 + h) \pmod{u'}$, with weight $W_2$.
15: $w = w_6x^6 + w_5x^5 + w_4x^4 + w_3x^3 + w_2x^2 + w_1x + w_0 = (f - v(v + h))/c_4$, weighted.
16: Compute $1/W$, $1/W_2$ and unweighted $u'$, $v'$
17: $u_n = w/u'$   (exact division, made monic).
18: $v_n = V^- - (V^- + v + h) \pmod{u_n}$.
19: $n_n = 3 - \deg(u_n)$.
20: return $[u_n, v_n, n_n]$.
```

Degree 1 doubling with two up adjustments Algorithm 68 is similar to genus 2 Degree 1 Doubling with Up Adjust Algorithm 46, for the composition and first adjustment portion. Instead of inverting at the end of the first adjustment, a second adjustment is computed with weighted polynomials u', v' similar to Degree 2 Doubling with 2 up adjustments Algorithm 64. The balancing coefficient n_n is computed as described in Section 6.1.5.

6.1.6 Explicit Addition

In this section, complete divisor addition and the basic formulations of all the corresponding explicit formulas are described. Addition Algorithm 33 is further specialized into different cases depending on the input divisor class degree and balancing coefficient.

The complete addition algorithm should test the degree of the input divisors in order of statistical frequency and choose the appropriate algorithm depending on the input balancing coefficients n_1 and n_2. 26 addition formulas are required for complete addition:

- degree 3 addition,
- degree 2 and 3 addition,
- degree 2 and 3 addition with up adjustment,
- degree 2 addition,
- degree 2 addition with up adjustment,
- degree 2 addition with two up adjustments,
- degree 1 and 3 addition,
- degree 1 and 3 addition with up adjustment,
- degree 1 and 3 addition with two up adjustments,
- degree 1 and 2 addition,
- degree 1 and 2 addition with down adjustment,
- degree 1 and 2 addition with up adjustment,
- degree 1 and 2 addition with two up adjustments,
- degree 1 addition,
- degree 1 addition with down adjustment,
- degree 1 addition with up adjustment,
- degree 1 addition with two up adjustments,
- degree 0 and 3 addition with down adjustment,

- degree 0 and 3 addition with up adjustment,
- degree 0 and 3 addition with two up adjustments,
- degree 0 and 2 addition with down adjustment,
- degree 0 and 2 addition with up adjustment,
- degree 0 and 2 addition with two up adjustments,
- degree 0 and 1 addition with down adjustment,
- degree 0 and 1 addition with up adjustment,
- degree 0 and 1 addition with two up adjustments.

Only precomputed values are required for degree zero addition, and precomputations from Algorithm 58 may be used. In the frequent cases, the output balancing coefficient n is always zero, and any special cases, n is computed via Steps 12,20–23,27,33–34, of Algorithm 33. Basic formulations of the algorithms required in complete addition, organized by the degree of the input divisors, are described next.

Balancing Coefficient

Let $[u_1, v_1, n_1]$ and $[u_2, v_2, n_2]$ be the input reduced divisor class representatives. Let S be the greatest common divisor of u_1, u_2 and $v_2 + v_1 + h$ and s be the polynomial computed by Step 9 or 11 in Addition Algorithm 33. There are 3 main cases:

1. If no reduction or adjustment is required as checked by Step 14 of Addition Algorithm 33 , then $n_n = n_1 + n_2 + \deg(S) - 2$ by Step 12 of Addition Algorithm 33.
2. Otherwise, if $\deg(s) < 2$, the NUCOMP continued fraction loop is not entered, but at least one reduction or adjustment is required. There are two sub-cases, if v_1 is normalized with a positive reduced basis then $n_n = n_1 + n_2 + \deg(S) - 2 + \deg(u_1) + \deg(u_2) - 4$ by Step 21 of Addition Algorithm 33. Otherwise for v_1 normalized with a negative reduced

basis, $n_n = n_1 + n_2 + \deg(S) - 2 + \deg(u_2) - \deg(s)$ by Step 24 unless $\deg(u_1 s) < 4$, then $n_n = n_1 + n_2 + \deg(S) - 2 + 4 - \deg(u_n)$ where $n_n \geq 0$. Finally, if $n_n < 0$ in the previous case, a final adjustment up is required and one more computation for the balancing coefficient is required where $n_n = n_n + 4 - \deg(u_n)$ by Step 28 of Addition Algorithm 33.

3. If $\deg(s) = 2$, the NUCOMP continued fraction loop is entered for one iteration, resulting in $n_n = n_1 + n_2 + \deg(S) - 2 + \deg(u_2) - \deg(r)$ with r computed by Step 30 of Addition Algorithm 33, unless $\deg(z) < 4$ for z computed by Step 34 of Addition Algorithm 33, in which case $n_n = n_1 + n_2 + \deg(S) - 2 + \deg(u_2) - \deg(s) + 4 - \deg(u_n)$.

Degree 3 Addition

Let $[u_1, v_1, 0]$ and $[u_2, v_2, 0]$ be the input reduced divisor class representatives with a $u_1 = x^3 + u_{12}x^2 + u_{11}x + u_{10}$, $v_1 = V_4^- x^4 + V_3^- x^3 + v_{12}x^2 + v_{11}x + v_{10}$ and $u_2 = x^3 + u_{12}x^2 + u_{21}x + u_{20}$, $v_2 = V_4^- x^4 + V_3^- x^3 + v_{22}x^2 + v_{21}x + v_{20}$. Similar to genus 2 split model addition Algorithm 29, Steps 1-2 of Addition Algorithm 33 are always skipped. Steps 3–8 in Algorithm 33 are specialized to degree 3 addition by computing the resultant $d = \text{Res}(u_1, u_2)$ instead of $S = \text{XGCD}(u_1, u_2)$ in Step 4 of Algorithm 33, where $d = 0$ implies $S \neq 1$.

To compute a degree 3 addition, first create the system of equations required for the alternate computation of the s polynomial in genus 3 (Section 4.3.6). Only m_1, m_4, m_7 and the determinant d are computed at first where d corresponds to the resultant of u_1 and u_2 and m_1, m_4, m_7 correspond to $t = m_7x^2 + m_4x + m_1 = d/u_1 \pmod{u_2}$. The resultant d is then immediately checked to be zero for special cases as soon as possible in order to minimize unnecessary computations.

The computation of the k polynomial in Step 3 of Algorithm 33 is postponed until after the computation of d, and instead d is immediately checked for special cases. If $d \neq 0$, then in contrast to the genus 2 setting, the coefficients of s are not solved for directly and instead $s = (v_2 - v_1)t \pmod{u_2}$ is computed in Step 11. The computation of Steps 30–38 of Addition Algorithm 33 follows, where the computation of M_1 and u_n is combined. The basic formulation for degree 3 addition is described in Algorithm 69. An explanation of how each step in Algorithm 69 is converted to an explicit formula is given in the following.

Algorithm 69 Genus 3 Split Model Degree 3 Addition

Input: $[D_1] = [u_1, v_1, 0]$, $[D_2] = [u_2, v_2, 0]$, $\deg(u_1) = \deg(u_2) = 3$.
Output: $[u_n, v_n, n_n] = [D_1] + [D_2]$

```
1: Compute matrix coefficients $t_i$ representing $su_1 \equiv (v_1 - v_2) \pmod{u_2}$ as the 3x3 linear system
   $\begin{bmatrix} t_1 & t_2 & t_3 \\ t_4 & t_5 & t_6 \\ t_7 & t_8 & t_9 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} k_0 \\ k_1 \\ k_2 \end{bmatrix}$.
2: Compute coefficients $m_1 = t_5 t_9,\ m_4 = t_6 t_7 - t_4 t_9,\ m_7 = t_4 t_8 - t_5 - t_7$ of the inverse of the matrix
   and let $t = m_7 x^2 + m_4 x + m_1$.
3: $d = t_1 m_1 + t_2 m_4 + t_3 m_7$.
4: if $d = 0$ then
5:     See description below.
6: $\tilde{v} = \tilde{v}_2 x^2 + \tilde{v}_1 x + \tilde{v}_0 = v_2 - v_1$.
7: $s' = ds = s'_2 x^2 + s'_1 x + s'_0 = \tilde{v} t \pmod{u_2}$.
8: if $s'_2$ is 0 then
9:     Go to Degree 3 Addition ($s_2 = 0$) Subroutine 70.
10: Compute monic quotient $q' = x + s'_2 (q_0/q_1) = x + q'_0$ of $u_1/s'$ with $1/q_1 = s_2$.
11: Compute remainder $r' = s'_2 dr = r'_1 x + r'_0$ of $u_1/s'$.
12: if $r'_1$ is 0 then
13:     Go to Degree 3 Addition ($r'_1 = 0$) Subroutine 71.
14: Compute values $q_1/r_1, r_1/q_1, q_0/q_1, r_0/q_1, q_0/r_1, r_0/r_1$.
15: $k = (f - v_1(v_1 + h)/u_1)$   (only degree 2 and 3 terms $k_2, k_3$).
16: $M'_2 = (r(v_2 + v_1 + h) + qk)/u_2$   (exact division, made monic).
17: $u_n = r(q(v_2 - v_1) + u_1 r)/(u_2 c_4 r_1 q_1) - qM'_2/q_1$.
18: $z = (u_1 r - u_n(c_4 r_1 q_1))/q$   (exact division).
19: $v_n = V^- - [(V^- - z + v_1 + h) \pmod{u_n}]$.
20: return $[u_n, v_n, 0]$.
```

In Step 1, similar to degree 3 doubling Algorithm 59, the polynomial representation

$$\tilde{v} = u_1 s - (s_2 x^2 + (s_2(u_{12} - u_{22}) + s_1)x + s_2(u_{11} - u_{21} + u_{22}(u_{12} - u_{22})) + s_1(u_{12} - u_{22}))(x^3 + u_{22}x^2 + u_{21}x + u_{20})$$

for $\tilde{v} = (v_{22} - v_{12})x^2 + (v_{21} - v_{11})x + (v_{20} - v_{10})$, $u_1 = x^3 + u_{12}x^2 + u_{11}x + u_{10}$ and $s = s_2 x^2 + s_1 x + s_0$ is used to set up a system of equations linear in the coefficients of s where

$$s = s_2 x^2 + s_1 x + s_0 = (\tilde{v}_2 x^2 + \tilde{v}_1 x + \tilde{v}_0)/(x^3 + u_{12}x^2 + u_{11}x + u_{10}) \pmod{x^3 + u_{22}x^2 + u_{21}x + u_{20}}.$$

Equating coefficients of the polynomials in the above identity results in a 3 by 3 matrix T with coefficients t_i. The resulting explicit formula for Step 1 is

$$\begin{aligned}
t_1 &= u_{10} - u_{20}, \\
t_4 &= u_{11} - u_{21}, \\
t_7 &= u_{12} - u_{22}, \\
t_2 &= -u_{20}t_7, \\
t_5 &= t_1 - u_{21}t_7, \\
t_8 &= t_4 - u_{22}t_7, \\
t_3 &= -u_{20}t_8, \\
t_6 &= t_2 - u_{21}t_8, \\
t_9 &= t_5 - u_{22}t_8.
\end{aligned}$$

In Steps 2–3, the inverse of T from Step 1 is computed omitting divisions by $\det(T)$, yielding

$$\det(T)T^{-1} = M = \begin{pmatrix} m_1 = t_5t_9 - t_8t_6 & m_2 = t_3t_8 - t_2t_9 & m_3 = t_2t_6 - t_3t_5 \\ m_4 = t_6t_7 - t_4t_9 & m_5 = t_1t_9 - t_3t_7 & m_6 = t_3t_4 - t_1t_6 \\ m_7 = t_4t_8 - t_5t_7 & m_8 = t_2t_7 - t_1t_8 & m_9 = t_1t_5 - t_2t_4 \end{pmatrix}.$$

Only the terms m_1, m_4, m_7 are required to compute the polynomial inversion of $\tilde{v}$ modulo u_1 where

$$t = m_7x^2 + m_4x + m_1 = \frac{d}{\tilde{v}} \pmod{u_1}$$

omitting the field inversion to divide by d. The computation of $d = \det(T)$ is chosen to reuse the same m_i. The resulting explicit formula for Steps 2–3 is

$$\begin{aligned}
m_1 &= t_5t_9 - t_6t_8, \\
m_4 &= t_6t_7 - t_4t_9, \\
m_7 &= t_4t_8 - t_5t_7, \\
d &= t_1m_1 + t_2m_4 + t_3m_7.
\end{aligned}$$

The value $d = 0$ is then checked for special cases as soon as possible in order to minimize unnecessary computations.

If $d \neq 0$, in Step 6 $\tilde{v} = v_2 - v_1$ is computed, where the degree 3 and 4 terms cancel and no modular reduction is required. The explicit formula for Step 6 is

$$\begin{aligned}
\tilde{v}_2 &= v_{22} - v_{12}, \\
\tilde{v}_1 &= v_{21} - v_{11}, \\
\tilde{v}_0 &= v_{20} - v_{10}.
\end{aligned}$$

In Step 7, $s' = \tilde{v}t \pmod{u_2}$ is computed by applying Karatsuba multiplication (see Section 4.3.3), and Karatsuba modular reduction (see Section 4.3.4) of a degree 4 polynomial and monic degree 3 modulus. The explicit formula for Step 7 is

$$\begin{aligned}
t_0 &= \tilde{v}_0 m_1, \\
t_1 &= \tilde{v}_1 m_4, \\
t_2 &= \tilde{v}_2 m_7, \\
t_3 &= (m_4 + m_7)(\tilde{v}_1 + \tilde{v}_2) - t_2 - t_1 - u_{22} t_2, \\
t_4 &= u_{21} t_2, \\
t_5 &= t_4 - t_1, \\
s'_0 &= t_0 - u_{20} t_3, \\
s'_1 &= (m_1 + m_4)(\tilde{v}_0 + \tilde{v}_1) - s'_0 - (u_{20} + u_{21})(t_2 + t_3) + t_5, \\
s'_2 &= (m_1 + m_7)(\tilde{v}_0 + \tilde{v}_2) - t_0 - t_2 - t_5 - u_{22} t_3.
\end{aligned}$$

Rather than directly computing the polynomials $q = q_1 x + q_0$ and $r = r_1 x + r_0$, monic q with weight s'_2 and weighted r with weight $s'_2 d$ are computed as q' and r' in Steps 10–11 and note that $q_1 = 1/s_2 = d/s'_2$. The value $r'_1 = 0$ is immediately checked after creation to minimize unnecessary computations. If $r'_1 \neq 0$, all necessary ratios of the coefficients of q and r that arise in Steps 16–19 for computing M'_2, u_n, v_n are computed beforehand in Step 14. A description of how these values are applied is given respectively in each corresponding step. The explicit formula for Steps 10 and 14 is identical to that used in the same steps in Degree 3 Doubling Algorithm 59.

In Step 15, only the degree 2 and 3 terms of $k = (f - v_1(v_1 + h)/u_1)$ are computed and are similar to the computation of k in degree 3 doubling Algorithm 59. The explicit formula for Step 14 is

$$\begin{aligned} t_0 &= v_{12} - V_2^-, \\ k_3 &= c_4 t_0, \\ k_2 &= c_4(v_{11} - V_1^-) + c_3 t_0 - u_{12} k_3. \end{aligned}$$

In Step 16, $M_2' = M_2/(c_4 r_1) = (r(2v_1 + h) + qk)/((c_4 r_1)u_1)$ the negative of M_2 monic is computed instead of M_2, with $c_6 = 1/c_4$. The polynomial $M_2 = (r(2v_1 + h) + qk)/u_1$ is directly computed and then symbolically multiplied by $1/(r_1 c_4)$ where all terms in the equation for each coefficient M_{21}' and M_{20}' that have divisions by q_1 or r_1 are replaced with the precomputed ratios from Step 14. Any terms that require a division by c_4 are grouped together and multiplied by c_6. The explicit formula for Step 16 is

$$\begin{aligned} M_{21}' &= c_6(c_3 - q_{r_{11}} k_3) + r_{r_{01}} - u_{22} \\ M_{20}' &= c_6(r_{r_{01}} c_3 - v_{22} - v_{12} - h_2 - q_{r_{11}} k_2 - q_{r_{01}} k_3) - u_{21} - u_{22} M_{21}'. \end{aligned}$$

In Step 17, the output polynomial $u_n = rM_1 - qM_2$ from Algorithm 33 is directly computed monic using the computations from Step 14 as

$$r(q(v_2 - v_1) + u_1 r)/(u_2 c_4 r_1 q_1) - qM_2'/q_1,$$

using the same approach as in Step 16. The explicit formula for Step 15 is:

$$\begin{aligned} u_{n_2} &= c_6 r_{q_{11}} + M_{21}' + q_{q_{01}} \\ u_{n_1} &= c_6(r_{\tilde{v}_2 + r_{q_{11}} t_7 + r_{q_0 1}} + r_{q_{01}}) + M_{20}' + q_{q_{01}} M_{21}' \\ u_{n_0} &= c_6(r_{\tilde{v}_2 r_{r_{01}} + r_{q_0 1}}(t_7 + r_{r_{01}}))q_{q_{01}} M_{20}'. \end{aligned}$$

Since u_n is required to be monic, the computation of $z = (ru_1 - u_n)/q$ in Step 17 must be adjusted to $z = (ru_1 - u_n(c_4 r_1 q_1))/q$. The exact polynomial division by $q = q_1 x + q_0$ introduces divisions in the coefficients of z where the precomputed ratios from Step 14 are used instead. Steps 17 and 18 are combined as

$$v_n = V^- - [(V^- + v_1 + h - z) \pmod{u}],$$

using a Karatsuba modular reduction (see Section 4.3.4) to perform the modular reduction. The explicit formula for Steps 17-18 is identical to that used in the same steps in Degree 3 Doubling Algorithm 59.

Basic formulations of the special output cases $s_2' = 0$ and $r_1 = 0$ are given in Subroutines 70 and 71. An explanation of how each step in both subroutines is converted to an explicit formula is given in the following.

Subroutine 70 Genus 3 Split Model Degree 3 Addition $s_2' = 0$

```
1: **if** $s_1' = 0$ **then**
2:     **if** $s_0' = 0$ **then return** $[1, V^+, 2]$.
3:     **else**
4:         Compute $1/(s_0)$ and $s = s_0$.
5:         $(s_0(s_0 u_1 + 2v_1 + h) - k)/u_2$,   (made monic);
6:         $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
7:         **return** $[u_n, v_n, 1]$
8: **if** Degree 4 term $w_4$ of $su_1 + 2v_1 + h$ is 0 **then**
9:     **if** Degree 3 term $w_3$ of $su_1 + 2v_1 + h$ is 0 **then return** $[1, V^-, 0]$
10:    **else**
11:        Compute $1/s_1 w_3$ and $s = s_1 x + s_0$.
12:        Compute degree 3 term of $k = (f - v_1(v_1 = h))/u_1$.
13:        $u_n = s(su_1 + 2v_1 + h) - k)/$,   (made monic).
14:        $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
15:        **return** $[u_n, v_n, 0]$
16: Compute $1/s_1 w_4$ and $s = s_1 x + s_0$.
17: Compute degree 3 term of $k = (f - v_1(v_1 = h))/u_1$.
18: $u_n = s(su_1 + 2v_1 + h) - k)/$,   (made monic).
19: $v_n = V^- - (V^- + su_1 - v_1 - h) \pmod{u_n}$.
20: **return** $[u_n, v_n, 0]$.
```

There are two types of special output cases, $\deg(s) < 2$ and $\deg(s) = 2$ but $\deg(r) < 1$. If $\deg(s) < 2$, then the computation of q, r is omitted and the output is computed similarly to degree

Subroutine 71 Genus 3 Split Model Degree 3 Addition $r_1 = 0$

```
1: Compute degree 2 and 3 terms $k_2$ and $k_3$ of $k = (f - v_1(v_1 + h))/u_1$.
2: if $r_0$ is 0 then
3:     $M_2 = kq/u2$   (exact division)
4:     $u_n = -qM_2$
5:     $v_n = V^- - (V^- - M_2 - v_1 - h) \pmod{u_n}$.
6:     if $k_3 = 0$ then  return $[u_n, v_n, 2]$.
7:     else               return $[u_n, v_n, 1]$.
8: else
9:     if Degree 4 term $M_{24}$ of $r(v_1 + v_2 + h) - qk$ is 0 then
10:        if Degree 3 term $M_{23}$ of $r(v_1 + v_2s + h) - qk$ is 0 then return $[1, V^-, 3]$
11:        else
12:            Compute $1/M_{23}$ and $r_0$.
13:            $M_1 = q(v_2 - v_1) + u_1r)/u_2$   (exact division).
14:            $M_2 = (r(2v_1 + h) + qk)/u_1$   (exact division).
15:            $u_n = rM_1 - qM_2$,   (made monic).
16:            $z = (u_1r - u_n(M_{23}))/q$   (exact division).
17:            $v_n = V^- - [(V^- - z + v_1 + h) \pmod{u_n}]$.
18:            return $[u_n, v_n, 2]$
19:    Compute $1/M_{24}$ and $r_0$.
20:    $M_1 = q(v_2 - v_1) + u_1r)/u_2$   (exact division).
21:    $M_2 = (r(2v_1 + h) + qk)/u_1$   (exact division).
22:    $u_n = rM_1 - qM_2$,   (made monic).
23:    $z = (u_1r - u_n(M_{24}))/q$   (exact division).
24:    $v_n = V^- - [(V^- - z + v_1 + h) \pmod{u_n}]$.
25:    return $[u_n, v_n, 1]$
```

2 addition Algorithm 47. In the case that $\deg(s') = 2$ but $\deg(r) < 1$, then $\deg(z) < 4$ and lower degree output divisors using same approach as the frequent case are computed with output n_n computed as described in Section 6.1.6.

The basic formulation of the special case $d = 0$ is omitted. The number of special sub cases for $d = 0$ grows considerably with the input degree. Instead the following high level description is provided in which all the required special cases can be derived easily. If d is zero then $w_1 = u_1 \pmod{u_2}$ is computed for $\deg(u_2) \leq \deg(u_1)$, where the degree of w_1 dictates the number of common x-coordinates between the two divisor class inputs. If $w_1 = 0$, then there are three common point x-coordinates between, u_1 and u_2, if $\deg(w_1) = 1$ then there are two common x-coordinates, and if $\deg(w_1) = 2$ then there is one common x-coordinate. In each respective case, checks are made to determine whether the common x-coordinates correspond to common or opposite points by computing $w_2 = (v_2 + v_1 + h) \pmod{w_1}$. Similar to w_1 the degree of w_2 corresponds to the number of points that are opposites in each case. Opposite points are removed via exact divisions prior to composition. Montgomery's inversion technique (Section 4.3.7) is applied in most cases where two or more inversions are required.

The same approaches for developing explicit formulas from the above high level description are used as in the genus 2 split model degree 2 addition with $d = 0$ Subroutine 48, but applied to the higher degree polynomials required in genus 3 arithmetic.

Degree 2 and 3 Addition

Let $[u_1, v_1, 0]$ and $[u_2, v_2, n_2]$ be reduced divisor class representatives with $u_1 = x^3+u_{12}x^2+u_{11}x+u_{10}$, $v_1 = V_4^- x^4 + V_3^- x^3 + v_{12}x^2 + v_{11}x + v_{10}$ and $u_2 = x^2 + u_{21}x + u_{20}$, $v_2 = V_4^- x^4 + V_3^- x^3 + V_2^- x^2 + v_{21}x + v_{20}$, and $n_2 = 1$ or $n_2 = 0$. There are two degree 2 and 3 addition algorithms required for complete divisor class arithmetic; degree 2 and 3 addition, and degree 2 and 3 addition with up adjust. Depending on the input balancing coefficient, if the degree 2 divisor class balancing coefficient n_2 is 0 or 1, the resulting $n_n = n_2 + n_1 - 2$ value is -2 or -1 respectively. The degree of the divisor after degree 2 and 3 addition is 5, so a single reduction step is required to achieve a divisor class representative in the required degree range.

Recall that a reduction step always increases the balancing coefficient by 1, so if $n_2 = 1$ then no additional adjustments are required. If $n_2 = 0$, then an additional up adjustment is required for the balancing coefficient n to be in range. The up adjustment for degree 2 and 3 addition with up adjust is applied as described in Steps 25–27 of Algorithm 33 using Montgomery's inversion trick (Section 4.3.7).

For degree 2 and 3 addition, Addition Algorithm 33 is adapted similarly to genus 2 degree 2 addition Algorithm 47, where computations are done modulo the degree 2 u_2, but it requires computation of higher degree k and output polynomials in genus 3 (see for example Algorithm 69). Computation of the output balancing coefficient n_n is described in Section 6.1.6. The basic formulation for degree 2 and 3 addition is described in Algorithm 72.

For degree 2 and 3 addition with up adjust, the basic approach is similar to Algorithm 72,

Algorithm 72 Genus 3 Split Model Degree 2 and 3 Addition

Input: $[D_1] = [u_1, v_1, n_1]$, $\deg(u_1) = 3$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_2) = 2$, $n_1 + n_2 = 1$
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
1: Compute $m_i$ in system of equations for $su_1 \equiv \tilde{v} \pmod{u_2}$ as $\left[\begin{smallmatrix} m_1 & m_2 \\ m_3 & m_4 \end{smallmatrix}\right]\left[\begin{smallmatrix} s_0 \\ s_1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} \tilde{v}_0 \\ \tilde{v}_1 \end{smallmatrix}\right]$, where
   $\tilde{v} = \tilde{v}_1 x + \tilde{v}_0 = v_2 - v_1$.
2: $d = m_1 m_4 - m_2 m_3$.
3: if $d = 0$ then
4:     See description below.
5: Solve for $s'_1 = ds_1, s'_0 = ds_0$ using Cramer's rule, omitting the inversion of $d$.
6: Compute only $k_3, k_2$ of $k = k_3x^3 + k_2x^2 + k_1x + k_0 = (f - v_1(v_1 + h))/u_1$   (exact division).
7: if $s'_1 = 0$ then
8:     if $s'_0 = 0$ then
9:         if $k_3 = 0$ then return $[1, V^+, 3]$.
10:        else Compute $1/k_3$.
11:            $u_n = k/u_2$   (exact division).
12:            $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
13:            return $[u_n, v_n, 2]$.
14:    else Compute $1/s_0$ and $s_0$.
15:        $z = -u_1 s_0$.
16:        $\tilde{v} = z - v_1 - h$.
17:        $u_n = (s(\tilde{v} - v_1) + k)/u_2$   (exact division, made monic).
18:        $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
19:        return $[u_n, v_n, 1]$.
20: $z = -u_1 s$.
21: $\tilde{v} = z - v_1 - h$.
22: Let $w = w_4x^4 + w_3x^3 + w_2x^2 + w_1x + w_0 = s(\tilde{v} - v_1) + k$.
23: if $w_4 = 0$ then
24:     if $w_3 = 0$ then
25:         if $w_2 = 0$ then return $[1, V^-, 0]$.
26:         else $W = w_2 d$.
27:     else $W = w_3 d$.
28: else $W = w_4 d$.
29: Compute $1/W$ and $s = s_1 x + s_0$.
30: $u_n = w/(Wu_2)$   (exact division, made monic).
31: $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
32: return $[u_n, v_n, 0]$.
```

Algorithm 73 Genus 3 Split Model Degree 2 and 3 Addition with Up Adjust

Input: $[D_1] = [u_1, v_1, n_1]$, $\deg(u_1) = 3$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_2) = 2$, $n_1 + n_2 = 0$
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
1: Compute $m_i$ in system of equations for $su_1 \equiv \tilde{v} \pmod{u_2}$ as $\begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix}\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} \tilde{v}_0 \\ \tilde{v}_1 \end{bmatrix}$, where $\tilde{v} = \tilde{v}_1 x + \tilde{v}_0 = v_2 - v_1$.
2: $d = m_1 m_4 - m_2 m_3$.
3: if $d = 0$ then
4:     See description above.
5: Solve for $s'_1, = ds_1, s'_0 = ds_0$ using Cramer's rule, omitting the inversion of $d$.
6: Compute only $k_3, k_2$ of $k = k_3x^3 + k_2x^2 + k_1x + k_0 = (f - v_1(v_1 + h))/u_1$   (exact division).
7: if $s'_1 = 0$ then
8:     if $s'_0 = 0$ then
9:         if $k_3 = 0$ then return $[1, V^+, 2]$.
10:        else Compute $1/k_3$.
11:            $u_n = k/u_2$   (exact division).
12:            $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
13:            return $[u_n, v_n, 1]$.
14:    else Compute $1/s_0$ and $s_0$.
15:        $l = -u_1 s_0$,  $\tilde{v} = l - v_1 - h$.
16:        $u_n = (s(\tilde{v} - v_1) + k)/u_2$   (exact division, made monic).
17:        $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
18:        return $[u_n, v_n, 0]$.
19: $z = -u_1 s_0$.
20: $\tilde{v} = z - v_1 - h$.
21: Let $w = w_4x^4 + w_3x^3 + w_2x^2 + w_1x + w_0 = (s(\tilde{v} - v_1) + k)$.
22: if $w_4 = 0$ then
23:     if $w_3 = 0$ then
24:         if $w_2 = 0$ then return Precomputed up adjustment of $[1, V^-, -1]$.
25:         else $W = w_2 d$,  $W_2 = W^4$.
26:     else $W = w_3 d$,  $W_2 = W^3$.
27: else $W = w_4 d$,  $W_2 = W^2$.
28: $u' = w/u_2$   (exact division) with weight $W$.
29: $v' = V^+ - [(V^+ - \tilde{v}) \pmod{u'}]$ with weight $W_2$.
30: $w = (f - v'(v' + h))/c_4$, weighted.
31: Compute $1/W$, $1/W_2$ and unweighted $u', v'$
32: $u_n = w/u'$   (exact division, made monic).
33: $v_n = V^- - (V^- + v' + h) \pmod{u_n}$.
34: $n_n = 3 - \deg(u_n)$.
35: return $[u_n, v_n, n_n]$.
```

with the addition of the up adjustment after reduction being applied as described in Steps 24–27 of Algorithm 33 using Montgomery's inversion trick (Section 4.3.7). Computation of the output balancing coefficient is described in Section 6.1.6. The basic formulation for degree 2 and 3 addition is described in Algorithm 73.

The basic formulation of the special case $d = 0$ is omitted. The number of special sub cases for $d = 0$ grows considerably with the input degree. Instead, as before, a high level description is provided. If d is zero then $w_1 = u_1 \pmod{u_2}$ is computed, where the degree of w_1 dictates the number of common x-coordinates between the two divisor class inputs. If $w_1 = 0$, then there are two common point x-coordinates between, u_1 and u_2, and if $\deg(w_1) = 1$ then there is one common x-coordinate. In each respective case, checks are made to determine whether the common x-coordinates correspond to common or opposite points by computing $w_2 = (v_2 + v_1 + h) \pmod{w_1}$. Similar to w_1 the degree of w_2 corresponds to the number of points that are opposites in each case. Opposite points are removed via exact divisions prior to composition. Montgomery's inversion technique (Section 4.3.7) is applied in most cases where two or more inversions are required.

The same approaches for developing explicit formulas from the above high level description are used as in the genus 2 split model degree 2 addition with $d = 0$ Subroutine 48, but applied to the higher degree polynomials required in genus 3 arithmetic.

Degree 2 Addition

Let $[u_1, v_1, n_1]$ and $[u_2, v_2, n_2]$ be the reduced divisor class representatives with $u_1 = x^2 + u_{11}x + u_{10}$, $v_1 = V_4^- x^4 + V_3^- x^3 + V_2^- x^2 + v_{11}x + v_{10}$ and $u_2 = x^2 + u_{21}x + u_{20}$, $v_2 = V_4^- x^4 + V_3^- x^3 + V_2^- x^2 + v_{21}x + v_{20}$. The balancing coefficients are $n_2 = 1, 0$ and $n_1 = 1, 0$. There are three degree 2 addition algorithms required for complete divisor class arithmetic; degree 2 addition, degree 2 addition with up adjust, and degree 2 addition with two up adjusts. The degree of the divisor after degree 2 addition is 4, so an adjustment is required for a final reduction step.

The choice of algorithm depends on the input balancing coefficients n_1 and n_2 where the possibilities for $n' = n_1 + n_2$ are 0,1,2. If $n' = 2$ then the output $n'' = n' - 2 = 0$ by Step 12 of Addition Algorithm 33, and a degree 2 addition is required. There is an inherent down adjustment that does not alter the balancing coefficient where $n_n = n'' = 0$ in all output cases. If $n' = 1$, then $n'' = n' - 2 = -1$ and degree 2 addition with up adjust is required. In this case, applying the up adjustment does increase the balancing coefficient, bringing the balancing coefficient into range. Finally, if $n' = 0$, then $n'' = n' - 2 = -2$ and degree 2 addition with two adjustment steps is applied. An up adjust is used as the reduction, where a second up adjustment is required to bring n into range if no special output case is computed in the first adjustment. The addition and reduction via adjustment are combined through the continued fraction steps of Addition Algorithm 33 as described in T10, and the second up adjustment required for degree 2 addition with two up adjusts is applied as described in Steps 25–27 of Algorithm 33 using Montgomery's inversion trick (Section 4.3.7).

For degree 2 addition, Algorithm 33 is adapted similarly to genus 2 degree 2 addition Algorithm 47. The main differences between genus 2 and genus 3 degree addition is that v_1 is normalized to positive reduced basis and an adjustment step is applied instead of a reduction step after composition in genus 3 degree 2 addition. NUCOMP does not differentiate between reductions and adjustments so a similar formulation is attained. In the genus 3 degree 2 addition case, $\deg(\tilde{v} - v_1) = 4$ and so only one special output case arises where $\deg(s') < 1$ as there is no cancellation possible when adding the degree 3 z in Step 31 of Addition Algorithm 33. Technique T11 is applied in genus 3, where $z = k/c_4$ is computed instead of k. In the case that $d = 0$, the formulations are similar to genus 2 split model degree 2 addition (see Subroutine 48). Computation of the output balancing coefficient n_n is described in Section 6.1.6. The basic formulation for degree 2 addition is described in Algorithm 74.

Algorithm 74 Genus 3 Split Model Degree 2 Addition

Input: $[D_1] = [u_1, v_1, n_1]$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_1) = \deg(u_2) = 2$, $n_1 + n_2 = 2$.
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
1: $v_1 = V^+ - (V^+ - v_1) \pmod{u_1}$.
2: Compute $m_i$ in system of equations for $su_1 \equiv \tilde{v} \pmod{u_2}$ as $\begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix}\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} \tilde{v}_0 \\ \tilde{v}_1 \end{bmatrix}$, where
   $\tilde{v} = \tilde{v}_1 x + \tilde{v}_0 = v_2 - v_1$ and $dw_1 = u_1 \pmod{u_2} = -m_3 x + m_4$.
3: $d = m_1 m_4 - m_2 m_3$.
4: if $d = 0$ then
5:     Similar to genus 2 degree 2 addition ($d = 0$) Subroutine 48.
6: Solve for $s_1' = ds_1, s_0' = ds_0$ using Cramer's rule, omitting the inversion of $d$.
7: Compute only $z_3, z_2$ of $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$   (exact division).
8: if $s_1' = 0$ then
9:     if $s_0' = 0$ then
10:        if $z_3 = 0$ then return $[1, V^+, 0]$.
11:        else Compute $1/z_3$.
12:            $u_n = z/u_2$   (exact division).
13:            $v_n = V^- - (V^- + v_n + h) \pmod{u_n}$.
14:            return $[u_n, v_n, 0]$.
15:    else Compute $1/s_0$ and $s_0$.
16:        $l = -u_1 s_0$.
17:        $\tilde{v} = l - v_1 - h$.
18:        $u_n = (s(\tilde{v} - v_1)/c_4 + z)/u_2$   (exact division, made monic).
19:        $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
20:        return $[u_n, v_n, 0]$.
21: Compute $1/s_1$ and $s = s_1 x + s_0$.
22: $l = -u_1 s$.
23: $\tilde{v} = l - v_1 - h$.
24: $u_n = (s(\tilde{v} - v_1)/c_4 + z)/u_2$   (exact division, made monic).
25: $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
26: return $[u_n, v_n, 0]$.
```

For degree 2 addition with up adjust, the basic approach is similar to Algorithm 74, but v_1 is kept in negative reduced basis and the computation of the balancing coefficient changes accordingly as described in Section 6.1.6. In the case that $d = 0$, the formulations are similar to genus 2 split model degree 2 addition (see Subroutine 48). Computation of the output balancing coefficient is described in Section 6.1.6. The basic formulation for degree 2 addition with up adjust is given in Algorithm 75.

Algorithm 75 Genus 3 Split Model Degree 2 Addition with Up Adjust

Input: $[D_1] = [u_1, v_1, n_1]$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_1) = \deg(u_2) = 2$, $n_1 + n_2 = 1$.
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
 1: Compute $m_i$ in system of equations for $su_1 \equiv \tilde{v} \pmod{u_2}$ as $\left[\begin{smallmatrix} m_1 & m_2 \\ m_3 & m_4 \end{smallmatrix}\right]\left[\begin{smallmatrix} s_0 \\ s_1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} \tilde{v}_0 \\ \tilde{v}_1 \end{smallmatrix}\right]$, where
    $\tilde{v} = \tilde{v}_1 x + \tilde{v}_0 = v_2 - v_1$ and $dw_1 = u_1 \pmod{u_2} = -m_3 x + m_4$.
 2: $d = m_1 m_4 - m_2 m_3$.
 3: if $d = 0$ then
 4:     Similar to genus 2 degree 2 addition ($d = 0$) Subroutine 48.
 5: Solve for $s_1(d), s_0(d)$ using Cramer's rule, omitting the inversion of $d$.
 6: Compute only $z_3, z_2$ of $z = z_3 x^3 + z_2 x^2 + z_1 x + z_0 = (f - v_1(v_1 + h))/(c_4 u_1)$   (exact division).
 7: if $s_1' = 0$ then
 8:     if $s_0' = 0$ then
 9:         if $z_3 = 0$ then return $[1, V^+, 3]$.
10:         else Compute $1/z_3$.
11:             $u_n = z/u_2$   (exact division).
12:             $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
13:             return $[u_n, v_n, 2]$.
14:     else Compute $1/s_0$ and $s_0$.
15:         $l = -u_1 s_0$.
16:         $\tilde{v} = l - v_1 - h$.
17:         $u_n = (s(\tilde{v} - v_1)/c_4 + z)/u_2$   (exact division, made monic).
18:         $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
19:         return $[u_n, v_n, 1]$.
20: Compute $1/s_1$ and $s = s_1 x + s_0$.
21: $l = -u_1 s$,   $\tilde{v} = l - v_1 - h$.
22: $u_n = (s(\tilde{v} - v_1)/c_4 + z)/u_2$   (exact division, made monic).
23: $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
24: return $[u_n, v_n, 0]$.
```

For degree 2 addition with two up adjusts, the basic approach is similar to Algorithm 75, with the addition of the up adjustment after reduction being applied as described in Steps 24–27 of Algorithm 33 using Montgomery's inversion trick (Section 4.3.7). Technique T11 is not applied because there is no benefit when combined with Montgomery's inversion trick. Computation of the output balancing coefficient n_n is described in Section 6.1.6. The basic formulation for degree 2 addition with two up adjusts is described in Algorithm 76.

Algorithm 76 Genus 3 Split Model Degree 2 Addition with Two Up Adjusts

Input: $[D_1] = [u_1, v_1, n_1]$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_1) = \deg(u_2) = 2$, $n_1 + n_2 = 0$.
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
1: Compute $m_i$ in system of equations for $su_1 \equiv \tilde{v} \pmod{u_2}$ as $\begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix}\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} \tilde{v}_0 \\ \tilde{v}_1 \end{bmatrix}$, where
   $\tilde{v} = \tilde{v}_1 x + \tilde{v}_0 = v_2 - v_1$ and $dw_1 = u_1 \pmod{u_2} = -m_3 x + m_4$.
2: $d = m_1 m_4 - m_2 m_3$.
3: **if** $d = 0$ **then**
4:     Similar to genus 2 degree 2 addition ($d = 0$) Subroutine 48.
5: Solve for $s_1(d), s_0(d)$ using Cramer's rule, omitting the inversion of $d$.
6: Compute only $k_3, k_2$ of $k = k_3x^3 + k_2x^2 + k_1x + k_0 = (f - v_1(v_1 + h))/u_1$   (exact division).
7: **if** $s'_1 = 0$ **then**
8:     **if** $s'_0 = 0$ **then**
9:         **if** $k_3 = 0$ **then return** $[1, V^+, 2]$.
10:        **else** Compute $1/z_3$.
11:            $u_n = k/u_2$   (exact division).
12:            $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
13:            **return** $[u_n, v_n, 1]$.
14:    **else** Compute $1/s_0$ and $s_0$.
15:        $l = -u_1 s_0$,  $\tilde{v} = l - v_1 - h$.
16:        $u_n = (s(\tilde{v} - v_1) + k)/u_2$   (exact division, made monic).
17:        $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
18:        **return** $[u_n, v_n, 0]$.
19: $W = s'_1 d$,  $W_2 = W^2$.
20: $l = -u_1 s'_0$.
21: $\tilde{v} = l - v_1 - h$.
22: $u' = (s(\tilde{v} - v_1) + k)/u_2$   (exact division) with weight $W$.
23: $v' = V^+ - [(V^+ - \tilde{v}) \pmod{u'}]$ with weight $W_2$.
24: $w = (f - v'(v' + h))/c_4$, weighted.
25: Compute $1/\mathrm{lcf}(w)$, $1/W$, $1/W_2$ and unweighted $u', v'$
26: $u_n = w/u'$   (exact division, made monic).
27: $v_n = V^- - (V^- + v' + h) \pmod{u_n}$.
28: $n_n = 3 - \deg(u_n)$.
29: **return** $[u_n, v_n, n_n]$.
```

Degree 1 and 3 Addition

Let $[u_1, v_1, 0]$ and $[u_2, v_2, n_2]$ be the reduced divisor class representatives with $u_1 = x^3 + u_{12}x^2 + u_{11}x + u_{10}$, $v_1 = V_4^- x^4 + V_3^- x^3 + v_{12}x^2 + v_{11}x + v_{10}$ and $u_2 = x + u_{20}$, $v_2 = V_4^- x^4 + V_3^- x^3 + V_2^- x^2 + V_1^- x + v_{20}$, and $n_2 = 0, 1, 2$. There are three degree 1 and 3 addition algorithms required for complete divisor class arithmetic; degree 1 and 3 addition, degree 1 and 3 addition with up adjust, and degree 1 and 3 addition with two up adjusts. Not considering special output cases, the degree of the divisor after degree 1 and 3 addition is 4, where an adjustment is required to produce a reduced divisor class representative.

The choice of algorithm depends on the input balancing coefficient n_2 where the possibilities for $0 + n_2 = n_2$ are 0,1,2. If $n_2 = 2$ then the balancing coefficient computed in Step 12 of Addition Algorithm 33 is $n'' = n_2 - 2 = 0$ and a degree 1 and 3 addition is required. There is an inherent down adjustment that does not alter the balancing coefficient. If $n_2 = 1$, then similarly $n'' = 1 - 2 = -1$, then degree 1 and 3 addition with up adjust is required. In this case, applying the up adjustment does increase the balancing coefficient, bringing the balancing coefficient into range. Finally, if $n_2 = 0$, then $n'' = 0 - 2 = -2$ and a degree 1 and 3 addition with two adjustment steps is required. The first up adjustment step is applied as a reduction, where a second up adjustment step is required to bring the output balancing coefficient n_n into range if no special output case is computed. The composition and reduction via adjustment are combined through the continued fraction steps of Addition Algorithm 33 as described in T10, and the second up adjustment required for degree 1 and 3 addition with two up adjusts is applied as described in Steps 25–27 of Algorithm 33 using Montgomery's inversion trick (Section 4.3.7).

For degree 1 and 3 addition, Addition Algorithm 33 is adapted to this case similarly to genus 2 degree 1 and 2 addition Algorithm 49. The main differences are that v_1 is normalized to positive reduced basis and the computation of higher degree z and output polynomials in genus 3 (see for example Algorithm 69). Computation of the output balancing coefficient is described in Section 6.1.6. The basic formulation for degree 1 and 3 addition is described in Algorithm 77.

Algorithm 77 Genus 3 Split Model Degree 1 and 3 Addition

Input: $[D_1] = [u_1, v_1, n_1]$, $\deg(u_1) = 3$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_2) = 1$, $n_1 + n_2 = 2$.
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
1:  $d = u_1 \pmod{u_2}$.
2:  if $d = 0$ then
3:      $d_w = (v_2 + v_1 + h) \pmod{u_2}$.
4:      if $d_w = 0$ then
5:          $u_n = u_1/u_2$  (exact division).
6:          $v_n = V^- - (V^- - v_1) \pmod{u_n}$.
7:          return $[u_n, v_n, 1]$.
8:      $v_1 = V^+ - (V^+ - v_1) \pmod{u_1}$.
9:      $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$  (exact division).
10:     $s_0(d_w) = z \pmod{u_2}$.
11:     Let $d = d_w$ and goto line 14.
12: $v_1 = V^+ - (V^+ - v_1) \pmod{u_1}$.
13: $s_0' = (v_2 - v_1) \pmod{u_2}$.
14: $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$  (exact division).
15: if $s_0'$ is 0 then
16:     if $z_3 = 0$ then
17:         if $z_2 = 0$ then return $[1, V^+, 0]$.
18:         else Compute $1/z_2$.
19:     else Compute $1/z_3$.
20:     $u_n = z/u_2$  (exact division).
21:     $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
22:     return $[u_n, v_n, 0]$.
23: Compute $1/s_0$ and $s = s_0$.
24: $l = -u_1s$,  $\tilde{v} = l - v_1 - h$.
25: $u_n = (s(\tilde{v} - v_1)/c_4 + z)/u_2$  (exact division, made monic).
26: $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
27: return $[u_n, v_n, 0]$.
```

For degree 1 and 3 addition with up adjust, the basic approach is similar to Algorithm 77, but v_1 is kept in negative reduced basis and the computation of the balancing coefficient changes accordingly. The basic formulation for degree 1 and 3 addition with up adjust is given in Algorithm 78.

Algorithm 78 Genus 3 Split Model Degree 1 and 3 Addition with Up Adjust

Input: $[D_1] = [u_1, v_1, n_1]$, $\deg(u_1) = 3$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_2) = 1$, $n_1 + n_2 = 1$.
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
1: $d = u_1 \pmod{u_2}$.
2: if $d = 0$ then
3:     $d_w = (v_2 + v_1 + h) \pmod{u_2}$.
4:     if $d_w = 0$ then
5:         $u_n = u_1/u_2$ (exact division).
6:         $v_n = V^- - (V^- - v_1) \pmod{u_n}$.
7:         return $[u_n, v_n, 0]$.
8:     $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$ (exact division).
9:     $s_0' = d_ws_0 = z \pmod{u_2}$.
10:    Go to Step 12 and let $d = d_w$.
11: $s_0' = ds_0 = (v_2 - v_1) \pmod{u_2}$ with weight $d$.
12: $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$ (exact division).
13: if $s_0'$ is 0 then
14:     if $z_3 = 0$ then
15:         if $z_2 = 0$ then return $[1, V^+, 3]$.
16:         else Compute $1/z_2$.
17:     else Compute $1/z_3$.
18:     $u_n = z/u_2$ (exact division).
19:     $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
20:     $n_n = 3 - \deg(u_n)$.
21:     return $[u_n, v_n, n_n]$.
22: Compute $1/s_0$ and $s_0$.
23: $l = -u_1s_0$, $\tilde{v} = l - v_1 - h$.
24: $u_n = (s_0(\tilde{v} - v_1)/c_4 + z)/u_2$ (exact division, made monic).
25: $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
26: return $[u_n, v_n, 0]$.
```

For degree 1 and 3 addition with two up adjusts, the basic approach is similar to Algorithm 78, with the addition of computing a second up adjustment after reduction in some cases as described in Steps 24–27 of Algorithm 33 using Montgomery's inversion trick (Section 4.3.7). Technique T11 is not applied because there is no benefit when combined with Montgomery's inversion trick. The basic formulation for degree 1 and 3 addition with two up adjusts is described in Algorithm 79.

Algorithm 79 Genus 3 Split Model Degree 1 and 3 Addition with Two Up Adjusts

Input: $[D_1] = [u_1, v_1, n_1]$, $\deg(u_1) = 3$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_2) = 1$, $n_1 + n_2 = 0$.

Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
1: $d = u_1 \pmod{u_2}$.
2: **if** $d = 0$ **then**
3:     $d_w = (v_2 + v_1 + h) \pmod{u_2}$.
4:     **if** $d_w = 0$ **then**
5:         $u' = u_1/u_2$   (exact division).
6:         $v' = V^- - (V^- - v_1) \pmod{u'}$,
7:         $w = w_4x^4 + w_3x^3 + w_2x^2 + w_1x + w_0 = (f - v'(v' + h))/c_4$.
8:         Compute $1/\mathrm{lcf}(w)$.
9:         $u_n = w/u'$   (exact division, made monic).
10:        $v_n = V^- - (V^- + v' + h) \pmod{u_n}$.
11:        $n_n = 3 - \deg(u_n)$.
12:        **return** $[u_n, v_n, n_n]$.
13:    $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$   (exact division).
14:    $s'_0 = d_w s_0 = z \pmod{u_2}$.
15:    Go to Step 18 and let $d = d_w$.
16: $s'_0 = ds_0 = (v_2 - v_1) \pmod{u_2}$ with weight $d$.
17: $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$   (exact division).
18: **if** $s'_0 = 0$ **then**
19:    **if** $z_3 = 0$ **then**
20:        **if** $z_2 = 0$ **then return** $[1, V^+, 2]$.
21:        **else** Compute $1/z_2$.
22:    **else** Compute $1/z_3$.
23:    $u_n = z/u_2$   (exact division).
24:    $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
25:    $n_n = 2 - \deg(u_n)$.
26:    **return** $[u_n, v_n, n_n]$.
27: $W = s'_1 d$,   $W_2 = W^2$.
28: $l = -u_1 s'_0$.
29: $\tilde{v} = l - v_1 - h$.
30: $u' = (s'(\tilde{v} - v_1) + k)/u_2$   (exact division) with weight $W$.
31: $v' = V^+ - [(V^+ - \tilde{v}) \pmod{u'}]$ with weight $W_2$.
32: $w = (f - v'(v' + h))/c_4$, weighted.
33: Compute $1/\mathrm{lcf}(w)$, $1/W$, $1/W_2$ and unweighted $u'$, $v'$
34: $u_n = w/u'$   (exact division, made monic).
35: $v_n = V^- - (V^- + v' + h) \pmod{u_n}$.
36: $n_n = 3 - \deg(u_n)$.
37: **return** $[u_n, v_n, n_n]$.
```

Degree 1 and 2 Addition

Let $[u_1, v_1, n_1]$ and $[u_2, v_2, n_2]$ be the reduced divisor class representatives with $u_1 = x^2 + u_{11}x + u_{10}$, $v_1 = V_4^- x^4 + V_3^- x^3 + V_2^- x^2 + v_{11}x + v_{10}$, $u_2 = x + u_{20}$, $v_2 = V_4^- x^4 + V_3^- x^3 + V_2^- x^2 + V_1^- x + v_{20}$, $n_2 = 0, 1, 2$ and $n_1 = 0, 1$. There are four degree 1 and 2 addition algorithms required for complete divisor class arithmetic; degree 1 and 2 addition, degree 1 and 2 addition with down adjust, degree 1 and 2 addition with up adjust, and degree 1 and 2 addition with two up adjusts. Not considering special output cases, the degree of the divisor after degree 1 and 2 addition is 3 so no reduction steps are required. In this case, continued fraction steps can be applied to combine the balancing adjustment with addition as described in Technique T10.

The choice of algorithm depends on the input balancing coefficients n_1 and n_2 where the possibilities for $n' = n_1 + n_2$ are 0,1,2 and 3. If $n' = 3$ then the output $n = n' - 2 = 1$ and a degree 1 and 2 addition with down adjust is applied. If $n' = 2$ then the output $n = n' - 2 = 0$ and a degree 1 and 2 addition is applied with no adjustments. If $n' = 1$, then $n = n' - 2 = -1$, then degree 1 and 2 addition with up adjust is applied. Finally, if $n' = 0$, then $n = n' - 2 = -2$ so degree 1 and 2 addition with two adjustment steps is applied. The first up adjustment is computed through continued fraction steps, and a second up adjustment is required to bring n into range if no special output case is computed. In all cases that require adjustment steps, the addition and first adjustment are combined through the continued fraction steps of Addition Algorithm 33 as described in T10, and the second up adjustment required for degree 1 and 2 addition with two up adjusts is applied as described in Steps 25–27 of Algorithm 33 using Montgomery's inversion trick (Section 4.3.7).

For degree 1 and 2 addition, Addition Algorithm 33 is adapted to this case similarly to genus 2 degree 1 and 2 addition Algorithm 49. The main differences are that v_1 is normalized to positive reduced basis and the computation of higher degree z and output polynomials in genus 3 (see for example Algorithm 69). Computation of the output balancing coefficient is described in Section 6.1.6. The basic formulation for degree 1 and 2 addition is described in Algorithm 80.

Algorithm 80 Genus 3 Split Model Degree 1 and 2 Addition

Input: $[D_1] = [u_1, v_1, n_1]$, $\deg(u_1) = 2$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_2) = 1$, $n_1 + n_2 = 2$.
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
1: $d = u_1 \pmod{u_2}$.
2: if $d = 0$ then
3:     $d_w = (v_2 + v_1 + h) \pmod{u_2}$.
4:     if $d_w = 0$ then
5:         $u_n = u_1/u_2$   (exact division).
6:         $v_n = V^- - (V^- - v_1) \pmod{u_n}$.
7:         return $[u_n, v_n, 1]$.
8:     $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$   (exact division).
9:     $s_0 = z/d_w \pmod{u_2}$.
10:    $u_n = u_1u_2$.
11:    $v = V^- - [(V^- - v_1 - s_0u_1) \pmod{u_n}]$.
12:    return $[u_n, v_n, 0]$.
13: $s_0 = (v_2 - v_1)/d \pmod{u_2}$.
14: $u_n = u_1u_2$.
15: $v_n = V^- - [(V^- - v_1 - s_0u_1) \pmod{u_n}]$.
16: return $[u_n, v_n, 0]$.
```

The basic formulations for degree 1 and 2 addition with down adjustment, degree 1 and 2 addition with up adjustment and degree 1 and 2 addition with two up adjustments are identical to degree 1 and 3 addition Algorithm 77, degree 1 and 3 addition with up adjustment Algorithm 78, and degree 1 and 3 addition with two up adjustments Algorithm 79.

Degree 1 Addition

Let $[u_1, v_1, n_1]$ and $[u_2, v_2, n_2]$ be the reduced divisor class representatives with $u_1 = x + u_{10}$, $v_1 = V_4^-x^4 + V_3^-x^3 + V_2^-x^2 + V_1^-x + v_{10}$ and $u_2 = x + u_{20}$, $v_2 = V_4^-x^4 + V_3^-x^3 + V_2^-x^2 + V_1^-x + v_{20}$. The balancing coefficients $n_2 = 0, 1, 2$ and $n_1 = 0, 1, 2$. There are four degree 1 addition algorithms required for complete divisor class arithmetic; degree 1 addition, degree 1 addition with down adjust, degree 1 addition with up adjust, and degree 1 addition with two up adjusts. Not considering special output cases, the degree of the divisor after degree 1 addition is 2 so no reduction steps are required. In this case, continued fraction steps can be applied to combine the balancing adjustments with addition as described in Technique T10.

The choice of algorithm depends on the input balancing coefficients n_1 and n_2 where the possibilities for $n' = n_1 + n_2$ are 0,1,2,3 and 4. If $n' = 4$ then the output $n_n = n' - 2 = 2$, which is out of range for a degree 2 divisor, so a degree 1 addition with down adjust is required. If $n' = 2$ or $n' = 3$ then the output $n_n = n' - 2 = 0$ or $n = n' - 2 = 1$ and a degree 1 addition is required with no adjustments. If $n' = 1$, then $n = n' - 2 = -1$, and a degree 1 addition with up adjust is required. Finally, if $n' = 0$, then $n = n' - 2 = -2$ and a degree 1 addition with two adjustment steps is required. In all cases that require adjustments, the addition and first adjustment are combined through the continued fraction steps of Addition Algorithm 33 as described in T10, and the second up adjustment required for degree 1 addition with two up adjusts is applied as described in Steps 25–27 of Algorithm 33 using Montgomery's inversion trick (Section 4.3.7).

For degree 1 addition, Addition Algorithm 33 is adapted similarly to genus 2 degree 1 addition Algorithm 52. Computation of the output balancing coefficient is described in Section 6.1.6. The basic formulation for degree 1 addition is given in Algorithm 81.

Algorithm 81 Genus 3 Split Model Degree 1 Addition

Input: $[D_1] = [u_1, v_1, n_1]$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_2) = \deg(u_1) = 1$, $n_1 + n_2 = 2, 3$.
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
1:  $d = u_1 \pmod{u_2}$.
2:  if $d = 0$ then
3:      $d_w = (v_2 + v_1 + h) \pmod{u_2}$.
4:      if $d_w = 0$ then return $[1, V^-, n_1 + n_2 - 1]$.
5:      $k = (f - v_1(v_1 + h))/u_1$   (exact division).
6:      $s_0 = k/d_w \pmod{u_2}$.
7:      $u_n = u_1 u_2$.
8:      $v_n = V^- - [(V^- - v_1 - s_0 u_1) \pmod{u_n}]$.
9:      return $[u_n, v_n, 0]$.
10: $s_0 = (v_2 - v_1)/d \pmod{u_2}$.
11: $u_n = u_1 u_2$.
12: $v_n = V^- - [(V^- - v_1 - s_0 u_1) \pmod{u_n}]$.
13: return $[u_n, v_n, 0]$.
```

For the degree 1 additions with adjustments, the resulting $n = n_1 + n_2 - 2$ value is -1 or 3, depending on n_1 and n_2. Addition Algorithm 29 is followed closely combining the composition and adjustment together via the continued fraction step. Similar to discussions in degree 1 with 3 addition algorithms, the reduced basis of v_1 dictates which direction is applied, and similarly v_1 is already in the form required to facilitate an up adjustment.

Degree 1 addition with down adjustment is almost identical to degree 1 and 3 addition Algorithm 77 other than when $d = 0$. If $d = 0$, then either $d_w = v_2 + v_1 + h \pmod{u_2} = 0$ or $d_w \neq 0$, but by Step 6 of Algorithm 33 $b_2 = 0$ by cancellation in the opposite normalizations of v_1 and v_2, resulting in $s = u_2(v_2 - v_1) \pmod{u_2} = 0$. So the degree zero divisor class representative $[1, V^-, 3]$ is returned. Computation of the output balancing coefficient n_n is described in Section 6.1.6. The basic formulations for degree 1 addition with down adjustment is given in Algorithm 82.

Algorithm 82 Genus 3 Split Model Degree 1 Addition with Down Adjust

Input: $[D_1] = [u_1, v_1, n_1]$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_2) = \deg(u_1) = 1$, $n_1 + n_2 = 4$.
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
 1: $d = u_1 \pmod{u_2}$.
 2: **if** $d = 0$ **then** **return** $[1, V^-, 3]$.
 3: $v_1 = V^+ - (V^+ - v_1) \pmod{u_2}$.
 4: $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$   (exact division).
 5: $s_0 = (v_2 - v_1)/d \pmod{u_2}$.
 6: **if** $s_0'$ is 0 **then**
 7:     **if** $z_3 = 0$ **then**
 8:         **if** $z_2 = 0$ **then return** $[1, V^+, 0]$.
 9:         **else** Compute $1/z_2$.
10:     **else** Compute $1/z_3$.
11:     $u_n = z/u_2$   (exact division).
12:     $v_n = V^- - (V^- + v_n + h) \pmod{u_n}$.
13:     $n_n = 3 - \deg(u_n)$.
14:     **return** $[u_n, v_n, 0]$.
15: Compute $1/s_0$ and $s_0$.
16: $l := -u_1s_0$,   $\tilde{v} = l - v_1 - h$.
17: $u_n = (s_0(\tilde{v} - v_1)/c_4 + z)/u_2$   (exact division, made monic).
18: $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
19: **return** $[u_n, v_n, 0]$.
```

Degree 1 addition with up adjustment is almost identical to degree 1 and 3 addition with up adjust Algorithm 78 other than when $d = 0$. If d is zero, and $d_w = v_2 + v_1 + h \pmod{u_2} = 0$, then the degree zero divisor class representative $[1, V^-, 3]$ is returned. Otherwise if $d_w \neq 0$, s is computed using $z = k/c_4$ as described in Technique T11. Computation of the output balancing coefficient n_n is described in Section 6.1.6. The basic formulations for degree 1 addition with up adjustment is given in Algorithm 83.

Algorithm 83 Genus 3 Split Model Degree 1 Addition with Up Adjust

Input: $[D_1] = [u_1, v_1, n_1]$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_2) = \deg(u_1) = 1$, $n_1 + n_2 = 1$.
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
 1: $d = u_1 \pmod{u_2}$.
 2: if $d = 0$ then
 3:     $d_w = (v_2 + v_1 + h) \pmod{u_2}$.
 4:     if $d_w = 0$ then return $[1, V^-, 0]$.
 5:     $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$ (exact division).
 6:     $s_0(d_w) = z \pmod{u_2}$.
 7:     Go to Step 10 with $d = d_w$.
 8: $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$ (exact division).
 9: $s_0 = (v_2 - v_1)/d \pmod{u_2}$.
10: if $s'_0$ is 0 then
11:     if $z_3 = 0$ then
12:         if $z_2 = 0$ then return $[1, V^+, 3]$.
13:         else Compute $1/z_2$.
14:     else Compute $1/z_3$.
15:     $u_n = z/u_2$ (exact division).
16:     $v_n = V^- - (V^- + v_n + h) \pmod{u_n}$.
17:     $n_n = 3 - \deg(u_n)$.
18:     return $[u_n, v_n, n_n]$.
19: Compute $1/s_0$ and $s_0$.
20: $l := -u_1s_0$,   $\tilde{v} = l - v_1 - h$.
21: $u_n = (s_0(\tilde{v} - v_1)/c_4 + z)/u_2$ (exact division, made monic).
22: $v_n = V^- - [(V^- - \tilde{v}) \pmod{u_n}]$.
23: return $[u_n, v_n, 0]$.
```

For degree 1 addition with two up adjusts, the basic approach is similar to Algorithm 83, with the addition of computing a second up adjustment after reduction in some cases as described in Steps 24–27 of Algorithm 33 using Montgomery's inversion trick (Section 4.3.7). Technique T11 is not applied because there is no benefit when combined with Montgomery's inversion trick. Computation of the output balancing coefficient n_n is described in Section 6.1.6. The basic formulation for degree 1 addition with two up adjusts is described in Algorithm 84.

Algorithm 84 Genus 3 Split Model Degree 1 Addition with Two Up Adjusts

Input: $[D_1] = [u_1, v_1, n_1]$, $[D_2] = [u_2, v_2, n_2]$, $\deg(u_2) = \deg(u_1) = 1$, $n_1 = n_2 = 0$.
Output: $[D] = [u_n, v_n, n_n] = [D_1] + [D_2]$.

```
1: $d = u_1 \pmod{u_2}$.
2: if $d = 0$ then
3:     $d_w = (v_2 + v_1 + h) \pmod{u_2}$.
4:     if $d = 0$ then return Precomputed up adjustment of $[1, V^-, -1]$.
5:     $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$    (exact division).
6:     $s_0' = d_ws_0 = z \pmod{u_2}$.
7:     Go to Step 10 with $d = d_w$.
8: $s_0' = ds_0 = (v_2 - v_1) \pmod{u_2}$ with weight $d$.
9: $z = z_3x^3 + z_2x^2 + z_1x + z_0 = (f - v_1(v_1 + h))/(c_4u_1)$    (exact division).
10: if $s_0'$ is 0 then
11:     if $z_3 = 0$ then
12:         if $z_2 = 0$ then return $[1, V^+, 2]$.
13:         else Compute $1/z_2$.
14:     else Compute $1/z_3$.
15:     $u_n = z/u_2$    (exact division).
16:     $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
17:     $n_n = 2 - \deg(u_n)$.
18:     return $[u_n, v_n, n_n]$.
19: $W = s_0'd$,    $W_2 = W^2$.
20: $l = -u_1s_0$,  $\tilde{v} = l - v_1 - h$.
21: $u' = (s(\tilde{v} - v_1) + k)/u_2$    (exact division) with weight $W$.
22: $v' = V^+ - [(V^+ - \tilde{v}) \pmod{u}]$ with weight $W_2$.
23: $w = (f - v'(v' + h))/c_4$, weighted.
24: Compute $1/\mathrm{lcf}(w)$, $1/W$, $1/W_2$ and unweighted $u'$, $v'$
25: $u_n = w/u'$    (exact division, made monic).
26: $v_n = V^- - (V^- + v' + h) \pmod{u_n}$.
27: $n_n = 3 - \deg(u_n)$.
28: return $[u_n, v_n, n_n]$.
```

Degree 0 Addition

Let $[u_1, v_1, n_1]$ and $[u_2, v_2, n_2]$ be the reduced divisor class representatives with $u_1 = x^3 + u_{12}x^2 + u_{11}x + u_{10}$, $v_1 = V_4^-x^4 + V_3^-x^3 + v_{12}x^2 + v_{11}x + v_{10}$ and $u_2 = 1$, $v_2 = V^-$. There are three main degree 0 addition algorithms required for complete divisor class arithmetic; degree 0 with down adjust, degree 0 with up adjust, and degree 0 with two up adjusts, all of which have variants based on the input degree of the divisor class representatives. The basic formulations of the degree 1, 2 and 3 variants are identical.

By Technique T10 degree 0 add with up adjust is formulated from Addition Algorithm 33 directly, similar to genus 2 split model degree 0 addition (Section 5.2.7). Degree 0 with down adjust, first normalizes v_1 into negative reduced basis, before performing identical computations to the up adjustment case. Applying Technique T11 changes the computation of u_n to $u_n = (f - v_1(v_1 + h))/u_1 c_3$ made monic. The polynomial $z' = zu_1 = ku_1/c_4$ is computed first without the exact division by u_1 to check for special output cases. Then the output u_n polynomial is exactly $z = z'/u_1$ made monic in all cases.

For degree 0 addition with two up adjustments , the basic approach is similar to degree 0 addition with up adjustment, but with the addition of computing a second up adjustment in some cases as described in Steps 24–27 of Algorithm 33 using Montgomery's inversion trick (Section 4.3.7). The basic formulations for degree 0 additions are the same whether degree 1, 2 or 3 input divisor class representatives are added with the degree 0 divisor class representative. The formulations are given in Algorithms 85, 86 and 87.

Algorithm 85 Genus 3 Split Model Degree 0 Add with Down Adjust

Input: $[D_1] = [u_1, v_1, n_1]$.

Output: $[u_n, v_n, n_n] = [D_1] + [1, V^+, 0]$.

```
1: $v_1 = V^+ - [(V^+ - v_1) \pmod{u_1}.]$
2: $z' = (f - v_1(v_1 + h))/c_4$
3: Compute $1/\mathrm{lcf}(z')$.
4: $u_n = z'/u_1$    (exact division, made monic).
5: $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
6: **return** $[u_n, v_n, 0]$.
```

Algorithm 86 Genus 3 Split Model Degree 0 Add with Up Adjust

Input: $[D_1] = [u_1, v_1, n_1]$.

Output: $[u_n, v_n, n_n] = [D_1] + [1, V^+, 0]$.

```
1: $z' = (f - v_1(v_1 + h))/c_4$
2: Compute $1/\mathrm{lcf}(z')$.
3: $u_n = z'/u_1$    (exact division, made monic).
4: $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
5: $n_n = 3 - \deg(u_n)$.
6: **return** $[u_n, v_n, n_n]$.
```

Algorithm 87 Genus 3 Split Model Degree 0 Add with Two Up Adjusts

Input: $[D_1] = [u_1, v_1, n_1]$.

Output: $[u_n, v_n, n_n] = [D_1] + [1, V^+, 0]$.

```
1: $z' = (f - v_1(v_1 + h))/c_4$
2: **if** $\deg(z') - \deg(u_1 < 3)$ **then**
3:     Compute $1/\mathrm{lcf}(z')$.
4:     $u_n = z'/u_1$   (exact division, made monic unless $\deg(u_n) = 3$).
5:     $v_n = V^- - (V^- + v_1 + h) \pmod{u_n}$.
6:     $n_n = 3 - \deg(u_n)$. **return** $[u_n, v_n, n_n]$
7: $W = \mathrm{lcf}(z')$,   $W_2 = W^2$.
8: $u' = z'/u_1$   (exact division) with weight $W$.
9: $v' = V^- - (V^- + v_1 + h) \pmod{u'}$ with weight $W_2$.
10: $w = (f - v'(v' + h))/c_4$, weighted.
11: Compute $1/\mathrm{lcf}(z')$, $1/W$, $1/W_2$ and unweighted $u'$, $v'$
12: $u_n = w/u'$   (exact division, made monic).
13: $v_n = V^- - (V^- + v' + h) \pmod{u_n}$.
14: $n_n = 3 - \deg(u)$.
15: **return** $[u_n, v_n, n_n]$.
```

6.1.7 Field Operation Costs and Comparisons

Field operation costs of the doubling and addition algorithms presented above are given in Tables 6.1 and 6.2 respectively, and compared to previous methods in Table 6.3. Standard protocol is followed using M to denote a multiplication, S a squaring, C a multiplication by a constant curve coefficient and A an addition. Unlike many previous works additive field operation counts are included, as these have been shown to have non-trivial cost working with large fields [16], and significant cost relative to multiplications working with word sized fields [36]. Recall that based on the findings of Sutherland [36, 37], techniques applied to any explicit formula never trade a multiplication for more than 3 additions based on [36]. The assumption that division by two over fields with characteristic not equal to 2 is equivalent in cost to an addition [37] is applied to the analysis.

All divisor class operations require one inversion, and field operations counts are presented for the frequent cases. Field operation counts can be reduced if assumptions about the characteristic of the base field are made. Three base field and curve equation cases are presented; the generic split model curve equation over arbitrary fields, and the curve equation assumptions described in

Section 6.1.1 where $h_4 = h_3 = h_2 = h_1 = h_0 = 0$ for fields with characteristic not equal to 2 and $h_4 = 1$, $h_3 = f_4 = f_3 = f_2 = 0$ for fields with characteristic 2. Several new formulas that require exactly one inversion in all cases and fully encapsulate divisor class arithmetic over the divisor class group of a genus 3 split model hyperelliptic curve are introduced. Operation counts for all doubling explicit formulas are given in Table 6.1, and addition explicit formulas in Table 6.2.

Table 6.1: Frequent case field operation counts of all one inversion doubling operations required for complete divisor class group arithmetic over genus 3 hyperelliptic curve split models.

	Arbitrary				**char $\neq$ 2**				**char = 2**			
Operations	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**
Doubling												
Degree 1	7	1	19	6	7	1	15	3	7	1	14	3
Degree 1 with Down Adjust	14	3	25	8	14	3	24	3	12	3	19	4
Degree 1 with two Up Adjusts	42	5	67	27	40	6	57	7	38	7	44	10
Degree 2	37	0	56	13	35	2	54	3	36	1	48	3
Degree 2 with two Up Adjusts	62	5	95	31	60	7	89	4	60	7	80	7
Degree 3	73	3	101	19	72	4	97	0	71	4	86	1

Comparison Summary

Comparisons of explicit formulas from this work to the previous best are presented in Table 6.3. Only frequent case operations with the simplifying assumptions resulting from fields of characteristic not equal to 2 and equal to 2 have direct comparisons to previous work, where comparisons for doubling (3DBL) and addition (3ADD) of degree 3 divisors are presented. Note that Baby Step and Inverse Baby Step operations in [32] are equivalent to the Degree 0 and 3 ADD with Up adjust, and Degree 0 and 3 Add with Down Adjust in this work and have the same multiplicative field operation counts, but this work saves five additions in both operations over both types of fields. All other divisor class operations presented above do not have a direct comparison to previous works.

Table 6.2: Frequent case field operation counts of all one inversion addition operations required for complete divisor class group arithmetic over genus 3 split hyperelliptic curve split models.

Operations	Arbitrary				char ≠ 2				char = 2			
	M	**S**	**A**	**C**	**M**	**S**	**A**	**C**	**M**	**S**	**A**	**C**
Addition												
Degree 0 and 1 with Down Adjust	8	0	12	4	6	1	10	3	6	1	10	3
Degree 0 and 1 with Up Adjust	8	0	17	5	6	1	14	3	6	1	12	3
Degree 0 and 1 with two Up Adjusts	32	2	48	22	30	4	37	7	31	3	37	11
Degree 0 and 2 with Down Adjust	10	0	23	6	9	1	21	2	9	1	20	2
Degree 0 and 2 with Up Adjust	10	0	23	6	8	2	19	2	9	1	17	2
Degree 0 and 2 with two Up Adjusts	36	2	54	22	33	5	42	6	34	4	39	10
Degree 0 and 3 with Down Adjust	11	0	29	7	9	1	21	0	9	1	21	2
Degree 0 and 3 with Up Adjust	11	0	29	7	9	1	21	0	9	1	21	2
Degree 0 and 3 with two Up Adjusts	39	2	60	23	36	5	44	3	37	3	46	9
Degree 1	3	0	5	0	3	0	5	0	3	0	5	0
Degree 1 with Down Adjust	17	2	25	5	15	4	23	1	15	4	21	1
Degree 1 with Up Adjust	14	2	26	5	13	3	23	1	13	3	20	1
Degree 1 with two Up Adjusts	40	4	57	24	38	6	46	8	38	6	45	10
Degree 1 and 2	6	1	10	0	6	1	10	0	6	1	10	0
Degree 1 and 2 with Down Adjust	20	3	39	7	19	4	36	1	18	5	33	1
Degree 1 and 2 with Up Adjust	18	3	36	6	17	4	30	1	17	4	28	1
Degree 1 and 2 with two Up Adjusts	46	5	68	23	44	7	54	6	44	7	50	9
Degree 1 and 3	25	2	49	8	23	4	42	0	23	4	40	0
Degree 1 and 3 with Up Adjust	22	2	43	7	21	3	36	0	22	2	33	0
Degree 1 and 3 with two Up Adjusts	52	4	75	23	49	7	62	3	51	5	56	7
Degree 2	37	1	56	7	36	2	57	0	33	4	53	0
Degree 2 with Up Adjust	30	1	47	8	29	2	48	0	29	2	41	0
Degree 2 with two Up Adjusts	59	3	79	27	56	6	73	5	57	5	66	8
Degree 2 and 3	41	1	62	3	41	1	60	0	41	1	55	0
Degree 2 and 3 with Up Adjust	74	3	90	19	72	5	78	3	72	4	76	7
Degree 3	65	3	87	12	65	3	85	0	65	3	80	0

Table 6.3: Field operation comparisons for frequent case divisor class group operations on genus 3 hyperelliptic curve split models.

	char(k) ≠ 2		**char(k) = 2**	
M S A C	3DBL	3ADD	3DBL	3ADD
Rezai Rad *et al.* [32]	85 2 163 0	75 2 138 0	89 1 116 0	81 0 118 0
Sutherland [37]	74 8 127 0	73 6 127 0		
This	72 4 97 0	65 3 85 0	71 4 86 1	65 3 80 0

Frequent Case Comparisons

For frequent case degree 3 addition and doubling, errors were found in the explicit formula tables provided in [32] over characteristic not equal to 2 fields. In Step 3 of degree 3 addition, the first line

$$\hat{u}_3 = s'_2(s'_2u_{11} - s'_0) - s'_1(s'_2u_{12} - s'_1)$$

should be

$$\hat{u}_3 = s'_2(s'_0 - u_{11}s'_2 + 2r(u_{12} - u_{22})) - s'_1(s'_1 - s'_2u_{12} + 4r) - 4r^2,$$

adding **1M + 1S + 8A** to the operation counts reported in [32]. In Step 4 of degree 3 doubling, the first line

$$\hat{u}_3 = s'_2(s'_2u_2 - s'_0) - s'_1(s'_1 - s'_2u_2)$$

should be

$$\hat{u}_3 = s'_2(s'_0 - s'_2u_1) - s'_1(s'_1 - u_2s'_2) - 4r(r + s'_1),$$

adding **1M + 4A** to the operation counts reported. The corresponding formulas were correct in the characteristic 2 setting tables presented in [32].

With these corrections, for the frequent case formulas developed in this work, degree 3 doubling requires an additional **2S** but saves **13M + 66A** and degree 3 addition requires an additional **1S**, but saves **10M + 51A** over characteristic not equal to 2 fields. Over characteristic 2 fields, degree 3 doubling requires an additional **3S + 1C** but saves **18M + 30A** and degree 3 addition requires an additional **3S**, but saves **16M + 38A**. Similarly, compared to formulas developed using an adapted basis over characteristic not equal to 2 fields in [37], degree 3 doubling and addition in this work requires **2M + 4S + 30A** and **8M + 3S + 42A** fewer operations respectively.

Special Case Comparisons

All implementations of frequent case arithmetic include explicit handling of special cases where the input divisor class representatives have points in common (or are opposite) as they come up naturally in the computations. Rather than making calls to Cantor's Algorithm, the special cases are computed explicitly and with only one inversion. These cases are made as fast as possible given the techniques that can be applied and are not analyzed here because there is no previous best to compare to.

6.2 Empirical Analysis

In this section, empirical data is presented that illustrates the relative performance of the genus 3 split model addition and doubling algorithms presented in this chapter and previous best works. All algorithms for addition and doubling are implemented in Magma as a proof of concept. Therefore, the absolute timings are not of great interest. The reader should rather focus on the relative cost between the various algorithms and models. The experiments were performed on a workstation with an Intel Xeon 7550 processor that has 64 cores, each of which is 64-bit and runs at 2.00 GHz.

Explicit formulas are very complex and prone to errors. Both black-box and white-box testing programs are utilized to provide thorough evidence that the formulas as they appear in the Magma code are all correct. Black-box testing computes numerous additions of randomly chosen divisor classes over randomly chosen curve models with output of each addition compared to Cantor's Algorithm. White-box testing computes a divisor class addition for every possible computation path within each formula, and similarly compares the output to Cantor's Algorithm.

As a preliminary benchmark, doubling and addition algorithms are compared separately. For addition, which includes previous best explicit algorithms, Cantor and Magma's built-in arithmetic over split models, a Fibonacci-like sequence of divisors using $D_{i+1} = D_i + D_{i-1}$ was computed, starting from two random divisors. Similar experiments were performed for the doubling algorithms over ramified and split models, by computing series of thousands of additions of a divisor class with itself. Timings range over prime fields of sizes 2, 4, 8, 16, 32, 64, 128, 256, 512 and 1024 bits were collected. All timings were run over random hyperelliptic curves with $h = 0$ defined over prime fields, using algorithms specialized for this setting.

All implementations of previous best formulas fall back to the appropriate Cantor's algorithm in any special cases not considered by the authors. Adaptations of the complete addition and doubling algorithms of this work were implemented similarly for direct comparison. Only degree 2 addition and doubling frequent cases were explicitly developed in the work of Sutherland [37]. Additionally, special output cases were considered in the work of Rezai Rad [32], but do not apply to the balanced setting. Therefore, only the frequent cases are considered in the work of Rezai Rad [32] as well.

The algorithms used for addition in the balanced setting are:

- Complete explicit addition using formulas of this work (Section 6.1.6),
- Sutherland explicit addition formulas [37],
- Rezai Rad explicit addition formulas [32],
- Explicit addition but calling Cantor's algorithm in the same cases as Rezai Rad and Sutherland,
- Improved Balanced Addition (Negative Reduced Basis) (Algorithm 15).

The algorithms used for doubling in the balanced setting are:

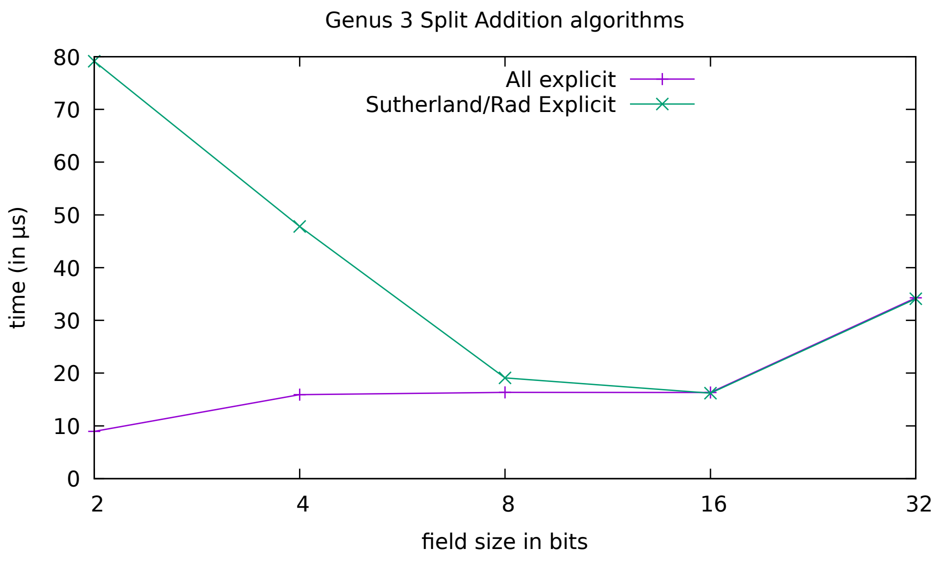
- Complete explicit doubling using formulas of this work (Section 6.1.5),
- Sutherland explicit doubling formulas [37],
- Rezai Rad explicit doubling formulas [32],
- Explicit doubling but calling Cantor's algorithm in the same cases as Rezai Rad and Sutherland,
- Improved Balanced Doubling (Negative Reduced Basis) (Algorithm 16).

Magma's built-in arithmetic was also included for comparison. In the following, several observations are discussed organized into sub-sections.

6.2.1 Explicit Special Cases

In the next figure, complete explicit formulas of this work are compared with adaptations of this work for which some special cases fall back on the appropriate Cantor's algorithm. Only addition comparisons are presented, as doubling shows similar results. The following conclusions are drawn from these plots:

- Our complete explicit formulas greatly outperform any variation of formulas that rely on Cantor's algorithm for special cases over fields with small cardinality.
- Convergence in the timings occur at 16-bit fields, suggesting that frequent case only implementations are effectively as efficient as completely explicit algorithms when working over 16-bit fields or larger when implemented in Magma.

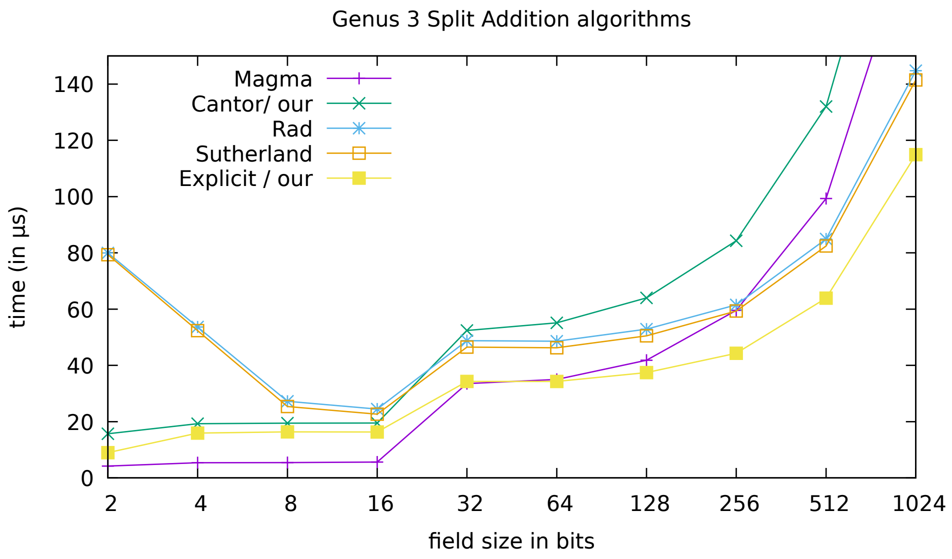


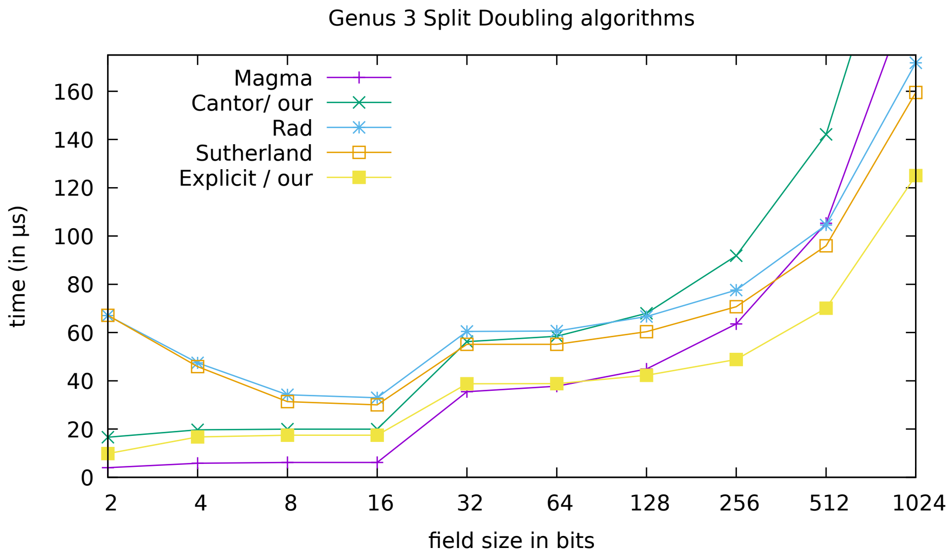
6.2.2 Split Model Comparisons

In the next figures, explicit formulas introduced in this work are compared to all relevant previous best algorithms over split models. The following conclusions are drawn from these plots:

- On the same platform, our explicit formulas outperform all relevant previous algorithms, including state-of-the-art and Cantor based algorithms for every field size.

- Our explicit formulas outperform Magma's built-in arithmetic over 64-bit fields and up. Based on the relative performance of prior state-of-the-art, Magma is likely accessing internal *C* functions that are not accessible via the interface provided.





The empirical results indicate that complete explicit formulas provide an improvement for computing split model arithmetic over small cardinality fields with a cross-over at 16-bits. As expected, the relatively fewer field operations (see Section 6.1.7) in the formulas of this work translate to faster implementations when using the same platform. The relative performance of Magma's built-in arithmetic and the formulas of this chapter likely due to Magma's high-level interface. Integrating algorithms from this chapter directly into Magma's built-in arithmetic, or implementing the algorithms directly in C/C++ so that they do not suffer from the overhead associated to working in a high-level system like Magma, might reduce the relative performance, and increase the field size for performance convergence between complete explicit formulas and frequent case only implementations.

Chapter 7

Conclusion

In this thesis, efficient computational approaches for computing divisor class arithmetic on ramified and split model curves were developed, introducing several novel algorithms that outperform previous best. Balanced NUCOMP for generic genus split model curves, explicit formulas based on NUCOMP and Balanced NUCOMP for genus 2 ramified and split model curves, and genus 3 split model curves were developed, and thoroughly tested. Balanced NUCOMP introduces a new "negative reduced" basis for the normalization of the v Mumford polynomial, eliminating the inefficiencies for odd genus split model curves found in previous work on arithmetic in the infrastructure of a split model curve. Moreover, in previous work it was suggested that NUCOMP does not benefit divisor class arithmetic over hyperelliptic curves of small genera, but explicit formulas based on NUCOMP over genus 2 ramified model curves and Balance NUCOMP over genus 2 and 3 split model curves prove to be computationally more efficient than previous best in the above cases mentioned. Most notably over split model genus 3 curves, there is a considerable improvement overall in the operation counts of divisor class addition and doubling supported by empirical evidence using Magma. Ramified model genus 3 arithmetic would also benefit from NUCOMP techniques but was not considered in this thesis due to ongoing work elsewhere; this will be presented along with our split model results in a forthcoming paper. All algorithms are implemented, tested for correctness, and empirically tested for comparisons to previous best. All software developed in support of this thesis is available at https://github.com/salindne/divisorArithmetic.

7.1 Future Work

The work of this thesis has to potential to be extended in the following ways:

Adapt NUCOMP to divisor arithmetic over non-hyperelliptic $C_{a,b}$ curves as this setting also plays a role in computational number theoretic applications [36]. Algorithms for divisor arithmetic on generic $C_{a,b}$ curves [14], specialized subsets of cases such as superelliptic curves [35], or specific cases for example $a = 3, b = 4$ have been worked on. Explicit formulas for the case $a = 3, b = 4$ have recently been developed in [26]. The general pattern of composing and reducing divisor class representatives to a reduced form is common among all algorithms mentioned. An adaptation of NUCOMP to these settings may result in similar improvements seen in the hyperelliptic curve setting.

Automating the generation of near-optimal explicit formulas for any genus based on this work would also be of interest. This would provide further improvements for $g > 3$ where developing formulas by hand is far too tedious, yielding algorithms and implementations that are greatly improved over polynomial arithmetic based algorithms. Choosing the right explicit techniques to apply generically may prove to be challenging, because finding optimal explicit formulas most certainly would require cycles of testing and optimization. The program could apply as many combinations of techniques as possible and output explicit formulas based on the intended application through specification inputs such as the relative cost of field additions compared to field multiplications. This type of work would also benefit from automatically generating testing scripts and example inputs, that could be based on the testing scripts of this work.

At this point, divisor class arithmetic in genus 2 and 3 is quite thoroughly optimized. Using ideas of the thesis (NUCOMP), there is still lots of room for improvement in genus 4 and above, and perhaps for non-hyperelliptic curves. Automation is possibly within reach as well for those cases. As discussed in Section 2.1.4, it is unlikely that arithmetic on inert models will be needed in the future for numerical investigations, but if that changes it should be possible to extend the ideas in this thesis to develop efficient formulas for inert models as well. Perhaps the ideas of this work can be applied to special families of curves, such as those with Kummer surfaces [13] and higher-genus analogues of DIK curves [7] or Edwards curves [8]. Isogeny based cryptographic protocols (if divisor class arithmetic plays a role) could be an application, or even direct improvements to isogeny computations themselves. The impactful results of this thesis leave us with many interesting avenues to explore.

Bibliography

[1] B. Balamohan. Accelerating the scalar multiplication on genus 2 hyperelliptic curve cryptosystems. Master's thesis, University of Ottawa, 2009.

[2] J.W. Bos, C. Costello, H. Hisil, and K. Lauter. Fast cryptography in genus 2. In T. Johansson and P.Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of Lecture Notes in Computer Science, pages 194–210. Springer, 2013.

[3] D. Cantor. Computing in the jacobian of a hyperelliptic curve. *Mathematics of Computation*, 48(177):95–101, 1987.

[4] J.W.S. Cassels and E.V. Flynn. *Prolegomena to a middle brow arithmetic of curves of genus 2*, volume 230 of *London Mathematical Society Lecture Notes*. Cambridge University Press, Cambridge, 1996.

[5] H. Cohen and G. Frey. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman and Hall/CRC, 6000 Broken Sound Parkway NW, Boca Raton, FL, 2nd edition, 2006.

[6] C. Costello and K. Lauter. Group law computations on jacobians of hyperelliptic curves. In Miri and S. Vaudenay, editors, *Selected areas in Cryptography*, volume 7118 of Lecture Notes in Computer Science, pages 92–117. Springer, 2011.

[7] C. Doche, T. Icart, and D.R. Kohel. Efficient scalar multiplication by isogeny decompositions. In *Public Key Cryptography-PKC 2006*, pages 191–206. Springer, 2006.

[8] Harold Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, 2007.

[9] S. Erickson, M.J. Jacobson, and A. Stein. Explicit formulas for real hyperelliptic curves of genus 2 in affine representation. *Advances in Mathematics of Communication*, 5(4):623–666, 2011.

[10] X. Fan, T. Wollinger, and G. Gong. Efficient explicit formulae for genus 3 hyperelliptic curve cryptosystems over binary fields. *IET Information Security*, 1(2):65–81, 2007.

[11] S. D. Galbraith, M. Harrison, and D. J. Mireles-Morales. Efficient hyperelliptic arithmetic using balanced representation for divisors. In *Algorithmic Number Theory - ANTS-VIII*, volume 5011 of *Lecture Notes in Computer Science*, pages 342–356, Banff, Canada, 2008. Springer-Verlag, Berlin.

[12] S.D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[13] P. Gaudry. Fast genus 2 arithmetic based on theta functions. *Journal of Mathematical Cryptology*, 1:243–266, 2007.

[14] Ryuichi Harasawa and Joe Suzuki. Fast jacobian group arithmetic on c_{ab} curves. In *Algorithmic Number Theory - ANTS-IV*, volume 1838 of *Lecture Notes in Computer Science*, pages 359–376, Leiden, Netherlands, 2000. Springer-Verlag, Berlin.

[15] R. Harley. Fast arithmetic on genus two curves. Available at http://cristal.inria.fr/~harley/hyper, 2000.

[16] H. Hisil and C. Costello. Jacobian coordinates on genus 2 curves. In *Advances in Cryptology–ASIACRYPT 2014*, pages 338–357. Springer, 2014.

[17] L. Imbert, M. J. Jacobson, and A. Schmidt. Fast ideal cubing in imaginary quadratic number and function fields. In *Advances in Mathematics of Communications*, number 2 in 4, pages 237–260, 2010.

[18] L. Imbert and M. J. Jacobson Jr. Empirical optimization of divisor arithmetic on hyperelliptic curves over $\mathbb{F}_{2^m}$. *Advances in Mathematics of Communication*, 7(4):485–502, 2013.

[19] M. J. Jacobson, Jr. and A. J. van der Poorten. Computational aspects of NUCOMP. In *Algorithmic Number Theory - ANTS-V*, volume 2369 of *Lecture Notes in Computer Science*, pages 120–133, Sydney, Australia, 2002. Springer-Verlag, Berlin.

[20] M.J. Jacobson, R. Scheidler, and A. Stein. Fast arithmetic on hyperelliptic curves via continued fraction expansions. In *Advances in coding theory and cryptography*, pages 200–243. World Scientific, 2007.

[21] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Sov. Phys. Dokl. (Enlgish Translation)*, 7(7):595–596, 1963.

[22] K.S. Kedlaya and A.V. Sutherland. Computing l-series of hyperelliptic curves. In *International Algorithmic Number Theory Symposium*, pages 312–326. Springer, 2008.

[23] D.E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, third edition, 1997.

[24] T. Lange. Formulae for arithmetic on genus 2 hyperelliptic cruves. *Applicable Algebra in Engineering, Communications and Computing*, 15:295–328, 2005.

[25] S.A. Lindner. Fast divisor tripling. Master's thesis, University of Calgary, 2014. https://prism.ucalgary.ca/handle/11023/1723.

[26] E. MacNeil, M.J. Jacobson Jr., and R. Scheidler. Divisor class group arithmetic on non-hyperelliptic genus 3 curves. *The Open Book Series*, 2, 2020.

[27] A. Menezes, Y. Wu, and R. Zuccherato. An elementary introduction to hyperelliptic curves. Technical Report CORR 1996-19, Center for Applied Cryptorgraphy Research (CACR), 1996. Available at http://www.cacr.math.uwaterloo.ca/.

[28] D.J.M. Mireles-Morales. *Efficient arithmetic on hyperelliptic curves with real representation*. PhD thesis, University of London, 2009.

[29] G.L. Mullen and D. Panario. *Handbook of finite fields*. Chapman and Hall/CRC, 2013.

[30] D. Mumford. Tata lectures on theta ii. *Progress in Mathematics*, 43:243–265, 1984.

[31] S. Paulus and H. Rück. Real and imaginary quadratic representations of hyperelliptic function fields. *Mathematics of Computation*, 68(227):1233–1241, 1999.

[32] M. Rezai Rad, M. J. Jacobson, and R. Scheidler. Jacobian versus infrastructure in split hyperelliptic curves. In *Algebra, Codes and Cryptology*, volume 1133 of *Communications in Computer and Information Science*, pages 183–203. Springer, 2019.

[33] P. Rozenhart, M. J. Jacobson, and R. Scheidler. Tabulation of cubic function fields via polynomial binary cubic forms. In *Mathematics of Computation*, number 280 in 81, pages 2335–2359, 2012.

[34] D. Shanks. On Gauss and composition I, II. In R.A. Mollin, editor, *Proc. NATO ASI on Number Theory and Applications*, pages 163–179. Kluwer Academic Press, 1989.

[35] T. Shashka and Y. Kopeliovich. The addition on Jacobian varieties from a geometric viewpoint. *arXiv e-prints*, page arXiv:1907.11070v2, Oct 2019.

[36] A.V. Sutherland. Sato-Tate Distributions. *arXiv e-prints*, page arXiv:1604.01256, Apr 2016.

[37] A.V. Sutherland. Fast Jacobian arithmetic for hyperelliptic curves of genus 3. *The Open Book Series*, 2(1):425–442, 2018.

[38] M. Takahashi. Improving Harley algorithms for jacobians of genus 2 hyperelliptic curves. In *Proceedings of SCIS 2002*, pages 497–502. IEICE Japan, 2002. In Japanese.

[39] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, The Edingburgh Building, Cambridge CB2 2RU, UK, 2nd edition, 2003.