EUROGRAPHICS 2019 / P. Alliez and F. Pellacini
(Guest Editors)

Volume 38 (2019), Number 2

Exact Constraint Satisfaction for Truly Seamless Parametrization

M. Mandad and M. Campen

Osnabrück University, Germany

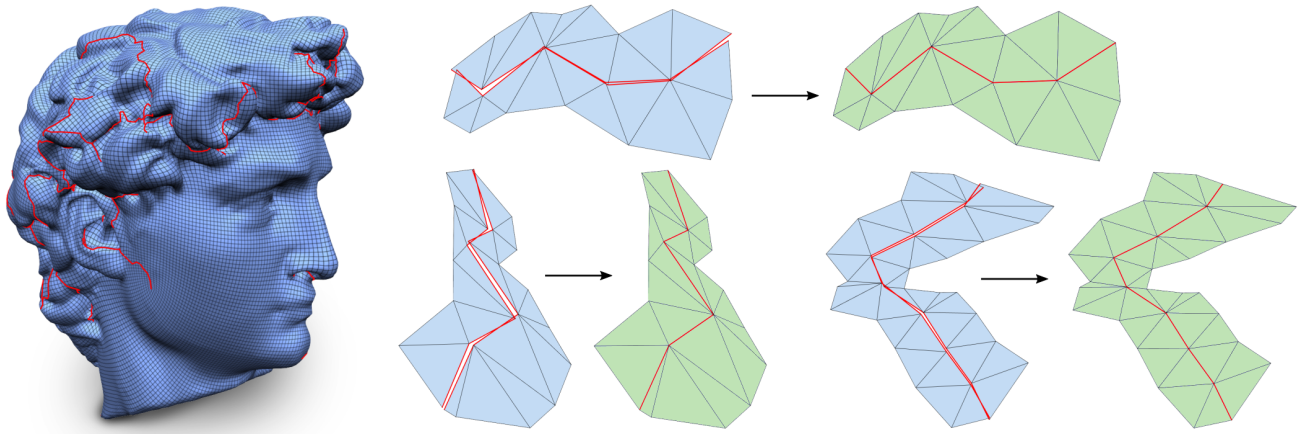


Figure 1: *Left: Parametrization of a mesh, with so-called seamless transitions across cuts (red). Right: Pieces of this mesh in parameter space $\mathbb{R}^2$ near the cut (both sides brought into common coordinate charts via transitions). Notice the gaps/overlaps due to numerical inaccuracies in the parametrization. As these typically are of minuscule magnitude, thus virtually invisible, they were amplified by a large factor here for clarity. The green instances show the result obtained by our method, with no inconsistencies at all – the parametrization is truly seamless.*

Abstract

In the field of global surface parametrization a recent focus has been on so-called seamless parametrization. This term refers to parametrization approaches which, while using an atlas of charts to enable the handling of surfaces of arbitrary topology, relate the parametrization across the cuts between charts via transition functions from special classes of transformations. This effectively makes the cuts invisible to applications which are invariant to these specific transformations in some sense. In actual implementations of these parametrization approaches, however, these restrictions are obeyed only approximately; errors stem from the tolerances of numerical solvers employed and, ultimately, from the limited accuracy of floating point arithmetic. In practice, robustness issues arise from these flaws in the seamlessness of a parametrization, no matter how small. We present a robust global algorithm that turns a given approximately seamless parametrization into an exactly seamless one – that still is representable by standard floating point numbers. It supports common practically relevant additional constraints regarding boundary and feature curve alignment or isocurve connectivity, and ensures that these are likewise fulfilled exactly. This allows subsequent algorithms to operate robustly on the resulting truly seamless parametrization. We believe that the core of our method will furthermore be of benefit in a broader range of applications involving linearly constrained numerical optimization.

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics;*** *Mesh models; Mesh geometry models; Shape modeling;*

1. Introduction

Optimization problems are at the core of many tasks in Computer Graphics and Geometry Processing. More precisely, one often has to deal with some form of constrained optimization. Constraints are imposed on the problem's variables to ensure that the computed solution exhibits certain properties required by a specific application. These constraints can be linear or nonlinear, and they can require equality or, to impose certain bounds, inequality between expressions formulated in terms of the variables.

In practical implementations, the result of a numerical optimization procedure rarely is a true solution to the problem: due to inaccuracies of the typically employed floating point arithmetics and possibly due to tolerances of the solver, neither is the result a true optimum, nor are the constraints satisfied precisely.

While small deviations from being a true optimum are relevant for the success or failure of subsequent processing steps only in very specific, rather uncommon scenarios, the binary question of a constraint being satisfied or violated can be entirely decisive. For this it may not even matter how slightly it is violated (cf. Sec. 6.2). A relevant question thus is:

How can constraint violations be avoided entirely?

For certain types of optimization problems, one might be able to get around this issue through the use of solvers based on adaptive precision arithmetics. However, not only can this cause a huge performance impact (cf. Sec. 6.1), it will in general yield solutions not representable by standard floating point numbers, and conversion will lead to constraint violations again. Thus, all subsequent processing steps would need to be compatible with, and carried out based on the same adaptive precision arithmetics as well.

1.1. Exact Constraint Satisfaction despite Limited Precision

To avoid all these issues, our goal is to obtain a result which

- exactly satisfies all constraints,
- while being expressed in standard floating point numbers.

We present a method that handles the case of optimization problems with linear equality constraints. It operates as a post-process and thus is independent of the type of solver employed and the nature (linear, non-linear, convex, non-convex, ...) of the underlying optimization problem.

More precisely, our method takes as input the result of a solver, i.e. an assignment of variables, which typically does not satisfy the linear equality constraints exactly. It then finds a nearby assignment of variables (*in the standard floating point numbers*) that exactly (*not only in floating point arithmetic*) satisfies the constraints. To illustrate the idea, assume constraints are univariate, i.e. of type $x_i = c_i$, where $\boldsymbol{x}$ are variables and $\boldsymbol{c}$ are constants expressed as floating point numbers. In this simplest possible of all non-trivial cases, one simply needs to override each variable x_i involved in such a constraint by the constant c_i in the solver's output. This, obviously, leads to exact constraint satisfaction. As soon as a constraint is multivariate, involving more than one variable, the situation becomes significantly more complicated – here our method comes into play.

1.2. Seamless Parametrization

A good example (and original motivation for our work) that involves such multivariate constraints is surface parametrization – one of the fundamental mathematical tools made use of in countless geometry and graphics applications. While simple instances involve merely point pinning constraints (of the above univariate type), lately more advanced instances with constraints for feature alignment, boundary alignment, seamlessness across charts, and isocurve connectivity came into focus, see Fig. 1 and Sec. 5.

Algorithms that build on such parametrizations, for instance for mesh generation tasks, may suffer from severe robustness issues when these constraints are not satisfied, as in Fig. 1. The level of robustness may even be independent of how strongly these are violated – in certain scenarios the slightest violation can guarantee failure, unless application-specific workarounds are designed.

We discuss this in more detail and evaluate our method in this use case in Sec. 6. Our approach is generic and can in principle be applied to any use case involving linearly constrained optimization. Its performance depends on the number and structure of the constraints, though, and it may be impractical for certain applications. Analysis in other fields is thus an interesting topic for future work, and at this point we make no claims about practical genericity.

2. Overview

The generic problem we tackle is an optimization problem with linear equality constraints

$$\min_{\boldsymbol{x}} E(\boldsymbol{x}) \quad s.t. \quad C\boldsymbol{x} = \boldsymbol{b}. \tag{1}$$

True minimization in the floating point (or any other discrete) domain is, in general, NP-hard. In practice thus approximative approaches are common. Our goal is to approximate only the minimality, but *not* the constraint satisfaction, because this can be essential for the correctness of subsequent processing steps. This is, of course, only possible if matrix C and vector $\boldsymbol{b}$ are exactly representable in the first place. We thus assume that the entries of C and $\boldsymbol{b}$ are rational or, equivalently, integer.

The central idea of our method: take an approximate minimizer $\bar{\boldsymbol{x}}$ of (1) which approximately satisfies the constraints ($C\bar{\boldsymbol{x}} \approx \boldsymbol{b}$) – obtained by an arbitrary solver – and project it to a point $\boldsymbol{x}$ in the solution space $\Omega = \{\boldsymbol{x} \mid C\boldsymbol{x} = \boldsymbol{b}\}$ (the kernel of C in case $\boldsymbol{b} = \boldsymbol{0}$). The key challenge: we require not only $\boldsymbol{x} \in \Omega$, but $\boldsymbol{x} \in \Omega \cap \mathbb{F}^n$, where $\mathbb{F}$ is the (*discrete*) set of values representable by a given floating point number type, and n the dimension of $\boldsymbol{x}$. The result $\boldsymbol{x}$ is thus, due to $\boldsymbol{x} \in \mathbb{F}^n$, representable by standard floating point numbers – but at the same time, due to $\boldsymbol{x} \in \Omega$, it *exactly* satisfies the constraints under exact (real number) arithmetic. In essence, this paper describes how to find one such point $\boldsymbol{x}$.

We demonstrate the effectiveness of our approach in the context of seamless parametrization, where exact constraint satisfaction is particularly important.

3. Related Work

Seamless Parametrization

Parameterizing a manifold surface over (a subset of) $\mathbb{R}^2$ is one of the fundamental mathematical tools made use of in countless applications [SPR06]. While locally (and on, e.g., disk-topology surfaces) this is always possible (as per the very definition of a manifold), global parametrization of arbitrary-topology surfaces can only be facilitated through the use of an *atlas* of *charts*.

Across chart boundaries (or *cuts*), the local parametrizations are related by transition functions, which can be arbitrary [SCOGL02, BCGB08, SSP08] or of restricted classes of transformations [TACSD06, KNP07, BZK09, NPPZ12, MZ12, BCE*13, MZ13, MPZ14, KCPS15, APL15, AL15, CZ17, BCW17] . Most importantly, *seamless* transitions [BZK09, MZ12, RNLL10] are commonly employed for mesh generation purposes, cf. Fig. 1. In this context also even further restricted classes of transitions, e.g. with quantized/integer translational components [BCE*13, CBK15], are of relevance.

Inaccurate Seams

The issue of (even just slightly) violated constraints in seamless parametrizations was addressed before [EBCK13, LBK16]. Unfortunately, the parametrization *sanitization* described in these works fundamentally relies on the parametrization being a quantized, integer grid parametrization [BCE*13, CBK15]: it assumes that certain key entries in $\boldsymbol{x}$ are supposed to be exact integers and rounds them accordingly. This rounding cannot simply be omitted as it is responsible for resolving most of the potential inaccuracy issues.

Therefore, this kind of sanitization cannot be applied to general seamless parametrizations. While quantized seamless parametrizations are of particularly high relevance in practice, note that recent methods for their creation work with and rely on initially continuous seamless parametrizations [CBK15]. In other fields, methods likewise rely on continuous seamless parametrizations [MPKZ10, RRP15, CZ17] – and suffer from robustness issues if these are not exact, cf. Sec. 6.2.

The problem is much more involved when considering general (non-quantized, non-integer) parametrizations. While [EBCK13, LBK16] only need local operations, we have to deal with entirely global interdependencies, induced by the network formed by the involved constraints. At the heart of our approach we thus operate with a global equation system, solved by a custom-tailored algorithm to not introduce any numerical errors. This global strategy furthermore allows for the consideration of a variety of relevant constraints besides seamless transition constraints, cf. Sec. 5.1. None of these are considered by the local sanitization procedures.

Exact Numerics

A variety of techniques for error-free numerical processing have been described, from exact predicates [She97] over adaptive precision numbers [Yap97, G*15] to exact solvers [GS00, Gär99, GSW16]. These are not easily applicable in our case: the optimization of constrained parametrizations is already hard and slow with fast standard floating point arithmetic. But even if performance impacts (cf. Sec. 6.1) can be tolerated: the results are generally not representable by standard floating point numbers without error. We instead take the (inaccurate) output of fast standard solvers and repair it while making it representable by standard numbers.

Other works, e.g. [MRL01], deal with constraints to be satisfied *relative to* a specific floating point standard, rounding mode, and evaluation procedure. This is of interest, e.g., for program analysis, but not of benefit in our context; the fact that a constraint is considered satisfied in a specific notion does not imply that subsequent calculations (even if carried out exactly) will turn out as expected and be consistent. Already linear combinations of constraints are not generally satisfied in this case. Our results, by contrast, exactly satisfy constraints – relative to exact, infinite precision arithmetic.

4. Exact Constraint Satisfaction

We begin by describing our key algorithms to project an arbitrary vector $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ to a vector $\boldsymbol{x} \in \mathbb{R}^n$ that exactly satisfies

$$C\boldsymbol{x} = \boldsymbol{b}, \quad C \in \mathbb{Z}^{m \times n}, \boldsymbol{b} \in \mathbb{Z}^m,$$

i.e. $\boldsymbol{x} \in \Omega$, where $\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid C\boldsymbol{x} = \boldsymbol{b}\}$. We note that the analogous problem with $C \in \mathbb{Q}^{m \times n}$, $\boldsymbol{b} \in \mathbb{Q}^m$ can always be reduced to this integer setting, simply by multiplying each system row with the least common multiple of its entries' denominators. We will thus assume an integer matrix C and vector $\boldsymbol{b}$ in the following.

Afterwards, in Sec. 4.3, we augment the procedure to yield a result $\boldsymbol{x} \in \Omega \cap \mathbb{F}^n$, where $\mathbb{F}$ corresponds to a finite precision number type, like the ubiquitous IEEE 754 floating point numbers.

Structurally, our approach has ties to the classical master-slave method employed to handle multi-variate constraints in finite element systems [CMPW01, Ch. 13.1]. Variables $\boldsymbol{x}$ are separated into two sets $\boldsymbol{x}|_{\text{free}}$ and $\boldsymbol{x}|_{\text{impl}}$ of *free* and *implied* variables, respectively, such that the free ones form an unconstrained system, and values for $\boldsymbol{x}|_{\text{impl}}$ can be computed (cf. Eq. 2) from $\boldsymbol{x}|_{\text{free}}$ via transformed constraint equations. In theory (assuming exact arithmetic) this allows performing the desired projection into Ω by simply setting $\boldsymbol{x}|_{\text{free}} \leftarrow \bar{\boldsymbol{x}}|_{\text{free}}$ and deducing $\boldsymbol{x}|_{\text{impl}}$. In practice, however, we need to deviate in order to overcome a series of challenges:

- The transformation of constraint equations C to yield expressions for the computation of $\boldsymbol{x}|_{\text{impl}}$ from $\boldsymbol{x}|_{\text{free}}$ needs to be carried out in an exact manner. We achieve this through the use of integer arithmetic (Sec. 4.1, Alg. 1 and 2).
- The values for $\boldsymbol{x}|_{\text{free}}$ cannot actually be chosen freely: we not only want *them* to be representable by floating point numbers, $\boldsymbol{x}|_{\text{free}} \in \mathbb{F}^{\hat{n}}$, but also the values of the implied variables $\boldsymbol{x}|_{\text{impl}}$, that non-trivially depend on them (Sec 4.3, Alg. 3 and 4).
- The computation of $\boldsymbol{x}|_{\text{impl}} \in \mathbb{F}^{\hat{m}}$ from $\boldsymbol{x}|_{\text{free}}$ needs to be carried out in an exact manner, without introducing any rounding errors (Sec. 4.3, Alg 5).

4.1. Fraction-Free System Transformation

We transform the constraint system matrix C into *integer reduced row echelon form* (IRREF) $\hat{C} \in \mathbb{Z}^{m \times n}$, which is defined as follows:

- if a row has a *pivot* (a first non-zero element) in column i, the row above it (if exists) has a pivot in column $j < i$,
- each column that contains a pivot contains no other non-zero element.

The transformation into this form can be performed entirely in $\mathbb{Z}$, i.e. without introducing any numerical inaccuracies. We first transform the matrix C into *integer (non-reduced) row echelon form* (IREF) (i.e. elements above pivots may be non-zero) using a fraction-free integer version of Gaussian elimination for rank-deficient rectangular matrices (adapted from [Dur12, Alg. 1] based on [Tur95, Alg. 6]). Then we follow up with fraction-free Jordan steps to cancel all elements above pivots. These algorithms are given below as Algorithms 1 and 2.

Remarks: the lower left triangle of C is known to become zero through Gaussian elimination; for efficiency, this part of the matrix is not modified by the algorithms, and the following algorithms ignore it accordingly.

The division by the greatest common divisor per matrix row in Algorithms 1 and 2 is to remove common factors and thereby reduce the growth of values as much as possible in $\mathbb{Z}$. This can be beneficial as, as we will see in the following, the magnitude of the matrix entries has some influence on how freely the values for variables $\boldsymbol{x}|_{\text{free}}$ can be chosen.

Algorithm 1 IREF by fraction-free Gaussian Elimination

for $k = 1,\dots,m-1$ **do**
  if $C_{kk} = 0$ **then** ▷ pivoting required
    $l \leftarrow \min\{l > k \mid C_{lk} \neq 0\}$ ▷ find pivot in column
    if l undefined **then continue** ▷ column is all zeroes
    $C_k \leftrightarrow C_l$ ▷ swap rows
  end if
  for $i = k+1,\dots,m$ **do**
    for $j = k+1,\dots,n$ **do** ▷ elimination (row i by k)
      $C_{ij} \leftarrow (C_{kk} C_{ij} - C_{ik} C_{kj})$
    end for
    $b_i \leftarrow (C_{kk} b_i - C_{ik} b_k)$
    $o \leftarrow \gcd(C_i, b_i)$ ▷ gcd of all elements of row i and b_i
    $C_i \leftarrow C_i / o$
    $b_i \leftarrow b_i / o$
  end for
end for

Algorithm 2 IRREF by fraction-free Jordan Elimination

for $k = \hat{m},\dots,2$ **do**
  $l \leftarrow \min\{l \geq k \mid C_{kl} \neq 0\}$ ▷ find row pivot
  for $i = k-1,\dots,1$ **do** ▷ elimination (row i by k)
    $c \leftarrow C_{il}$
    for $j = n,\dots,i$ **do**
      if $j \geq k$ **then**
        $C_{ij} \leftarrow (C_{kl} C_{ij} - c C_{kj})$
      else
        $C_{ij} \leftarrow (C_{kl} C_{ij})$
      end if
    end for
    $b_i \leftarrow (C_{kl} b_i - c b_k)$
    $o \leftarrow \gcd(C_i, b_i)$ ▷ gcd of all elements of row i and b_i
    $C_i \leftarrow C_i / o$
    $b_i \leftarrow b_i / o$
  end for
end for

4.2. Evaluation of Implied Values

Let the result of applying Algorithms 1 and 2 to $C, \boldsymbol{b}$ be denoted $\hat{C}, \hat{\boldsymbol{b}}$. The matrix $\hat{C}$ contains $\hat{m} \leq m$ rows with pivots, and $m - \hat{m}$ rows of all zeroes at the bottom, due to potential linear dependencies in the constraints in C. Based on this matrix $\hat{C}$, it is now conceptually easy to construct a solution, i.e. a vector $\boldsymbol{x} \in \mathbb{R}$ of variable assignments, such that $\hat{C}\boldsymbol{x} = \hat{\boldsymbol{b}}$ (and thus $C\boldsymbol{x} = \boldsymbol{b}$) – assuming theoretical real-valued arithmetics for now.

To this end we can treat the $\hat{n} = n - \hat{m}$ variables corresponding to columns *without* pivots as free variables $\boldsymbol{x}|_{\text{free}}$, and the $\hat{m}$ variables corresponding to columns *with* pivots as implied variables $\boldsymbol{x}|_{\text{impl}}$. Values for free variables can be chosen arbitrarily, while the implied variables are to be computed from these such that $\hat{C}\boldsymbol{x} = \hat{\boldsymbol{b}}$:

$$x_{\pi(j)} = \hat{b}_j - \frac{1}{\hat{C}_{j\pi(j)}} \sum_{i=\pi(j)+1}^{n} \hat{C}_{ji} x_i = \hat{b}_j - \sum_{i=\pi(j)+1}^{n} \hat{C}_{ji} \frac{x_i}{\hat{C}_{j\pi(j)}} \quad (2)$$

Here $\pi(j)$ is the index of the pivot, the first non-zero element, in row j. Note that the sum only depends on free variables, not on any implied variables: whenever x_i is an implied variable, $\hat{C}_{ji}$ is 0 in this sum because $\hat{C}$ is in IRREF.

A solution $\boldsymbol{x} \in \mathbb{R}^n$ that satisfies all constraints exactly while being close to $\bar{\boldsymbol{x}}$ can thus be obtained by choosing $x_i = \bar{x}_i$ for each free variable, and computing values for the implied variables via (2). Note that $\boldsymbol{x}$ determined in this (asymmetric) way is, in general, not optimally close to $\bar{\boldsymbol{x}}$; alternative constraint handling techniques, like Lagrange multipliers or the penalty method, however, are not amenable to the strategy we present in the following to achieve numerical exactness in tandem with floating-point-representable results. Under the practically realistic assumption that $\bar{\boldsymbol{x}}$ violates constraints only slightly (in the targeted parametrization use case we typically encounter relative violations around 10^{-8}), the difference is likely to be small, anyway.

4.3. Numerical Exactness

The fraction-free matrix transformations by Algorithms 1 and 2 happen entirely in $\mathbb{Z}$, thus can be carried out without any error using integer number types – given that no overflow occurs (cf. Sec. 7).

The challenge of achieving exact constraint satisfaction, in particular achieving it using values $x_i \in \mathbb{F}$, thus is concentrated to the step of evaluation (2).

For a free variable x_i, we could simply choose the value $\mathbb{F}(\bar{x}_i)$, where $\mathbb{F}(x) = \arg\min_{y \in \mathbb{F}} \|x - y\|$ "rounds" to $\mathbb{F}$ – if $\bar{x}_i$ is not given in this form already anyway.

For a dependent variable x_i, however, we need to ensure that formula (2) can be evaluated without error and that it yields a result $x_i \in \mathbb{F}$. We employ three measures to achieve this:

- Values for free variables are chosen from a (*fixed point*) subset $\mathbb{F}_\delta \subset \mathbb{F}$, so as to guarantee that the sum in (2) can be evaluated without error in $\mathbb{F}$ and yields a result that is from $\mathbb{F}_\delta$ as well.
- Values for free variables are chosen such that the *division* in (2) is without error in $\mathbb{F}$ and yields a result that is from $\mathbb{F}_\delta$.
- The computation of the sum in (2) (effectively a *dot product*) is performed by a special algorithm that guarantees that no intermediate value exceeds the range representable exactly by $\mathbb{F}$.

Fixed Point Representation. When performing the additions, subtractions, and multiplications in the evaluation (2) using a floating point representation, there is a potential for numerical error due to the *floating* point, i.e. the varying exponent. For instance, adding a number a with floating point exponent $k + l$ to a number b with exponent k and subtracting a again, does not yield the original number b because its l least significant bits get lost – unless they were zero in the first place. As pointed out by Ebke et al. [EBCK13], numerical error can thus be avoided by making sure that the l least significant bits of any number with floating point exponent $K - l$ involved in an operation are zero to begin with, where K is the maximum exponent over all involved values (initial, intermediate results, final result). We determine the largest floating point exponent K over all values of $\bar{\boldsymbol{x}}$ as $K = \max_i \lceil \log_2 |\bar{x}_i| \rceil + 1$ (where the additional 1 adds a margin), and define $\delta = 2^K$.

Let $\mathbb{F}_{\delta=2^\kappa} \subset \mathbb{F}$ be the set of floating point numbers with floating point exponent $k \leq K$ and their $K - k$ least significant bits being

zero. A value $v \in \mathbb{F}$ can be truncated to a value $v' \in \mathbb{F}_\delta$ by the floating point operation $v' \leftarrow (v + s_v\delta) - s_v\delta$, where s_v is the sign of v [EBCK13]. We define $\mathbb{F}_\delta(v) = v'$ through this operation.

Note that every addition, subtraction, or multiplication of values from $\mathbb{F}_\delta$ is exact in $\mathbb{F}_\delta$ as long as the result does not exceed the range $(-\delta, +\delta)$. The same holds for a multiplication of a value v from $\mathbb{F}_\delta$ with a value $n \in \mathbb{Z}$ (as it is equivalent to an n-fold addition of $v \in \mathbb{F}_\delta$).

Divisibility. In order to guarantee that the division of the sum by $\hat{C}_{j\pi(j)} \in \mathbb{Z}$ is exact in $\mathbb{F}_\delta$, we choose the free variables such that each summand $\hat{C}_{ji}x_i$ is exactly $\mathbb{F}_\delta$-divisible by $\hat{C}_{j\pi(j)}$. Each row j poses such a condition on each free variable x_i (unless $\hat{C}_{ji} = 0$).

We can conservatively meet all these conditions on a free variable x_i by choosing it such that it is $\mathbb{F}_\delta$-divisible by the least common multiple of the pivots of all rows j where $\hat{C}_{ji} \neq 0$, i.e.

$$x_i \mid_{\mathbb{F}_\delta} \operatorname{lcm}_{\{j|\hat{C}_{ji}\neq 0\}} \hat{C}_{j\pi(j)}.$$

We achieve this by setting the value of free variables as follows:

$$x_i \leftarrow \mathbb{F}_\delta(\bar{x}_i/\ell_i)\ell_i, \text{ where } \ell_i = \operatorname{lcm}_{\{j|\hat{C}_{ji}\neq 0\}} \hat{C}_{j\pi(j)} \in \mathbb{Z}$$

This is spelled out in Algorithm 4.

Dot Product Evaluation. The dot product in (2) is evaluated by a special procedure that guarantees that no intermediate value exceeds the final value or the range of values chosen for the free variables. This guarantees that the result is exact and from $\mathbb{F}_\delta$ as detailed above, if only this final result does not exceed the range $(-\delta, +\delta)$ (cf. Sec. 7).

As $\hat{C}_{ji} \in \mathbb{Z}$, the dot product can be written as a plain sum where each $x_i/\cdot$ appears $\hat{C}_{ji}$ times as summand. We sum these summands in a specific order: if an intermediate sum is positive, a negative summand is added next (and vice versa), unless there is none left.

The algorithm for consistent evaluation, including the safe dot product Algorithm 5, is given as Algorithm 3.

Algorithm 3 Evaluation

```
for k = n, ..., 1 do
    if column k contains no pivot then
        D ← ∅
        for i = 1, ..., min(k, m̂) do                ▷ collect divisors
            if Ĉ_ik ≠ 0 then
                D ← D ∪ {Ĉ_iπ(i)}
            end if
        end for
        x_k ← makeDiv(x̄_k, D)                       ▷ fix free variable
    else                                            ▷ compute implied variable
        j ← index of row containing the pivot
        x_k ← b̂_k − safeDot({(Ĉ_ji, x_i/Ĉ_jk) | k+1 ≤ i ≤ n})
    end if
end for
```

Algorithm 4 *makeDiv*(x, D)

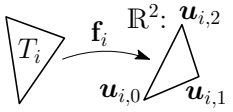
```
if D = ∅ then return F_δ(x)
d ← lcm(D)
return F_δ(x/d)d    ▷ multiple of d, thus divisible by each e ∈ D
```

Algorithm 5 *safeDot*(S)

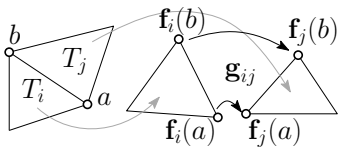
```
P ← {(|c|, +|x|) | (c,x) ∈ S ∧ cx > 0}      ▷ postive summands
N ← {(|c|, −|x|) | (c,x) ∈ S ∧ cx < 0}      ▷ negative summands
r ← 0                                        ▷ accumulator
while P ≠ ∅ ∨ N ≠ ∅ do
    if P ≠ ∅ ∧ (r < 0 ∨ N = ∅) then
        P ← P\(c,x)                          ▷ take arbitrary element
        k ← min(c, ⌊(δ − r)/x⌋)
        r ← r + kx
        if k < c then P ← P ∪ {(c − k, x)}   ▷ re-add remainder
    else
        N ← N\(c,x)                          ▷ take arbitrary element
        k ← min(c, ⌊(−δ − r)/x⌋)
        r ← r + kx
        if k < c then N ← N ∪ {(c − k, x)}   ▷ re-add remainder
    end if
end while
return r
```

5. Seamless Parametrizations

A piecewise linear parametrization F of a triangle mesh M consists of a linear map per triangle T_i to the plane, $\mathbf{f}_i : T_i \to \mathbb{R}^2$. Such a map $\mathbf{f}_i$ is represented by coordinates $\boldsymbol{u}_{i,0}, \boldsymbol{u}_{i,1}, \boldsymbol{u}_{i,2} \in \mathbb{R}^2$ at each of T_i's three vertices. To enable individual coordinates per triangle, these are not represented per (shared) vertex, but per wedge (a corner of a triangle).



Parametrization F is *seamless* if for each edge e_{ij}, between triangles T_i, T_j, with incident vertices a, b, there is a rigid transformation $\mathbf{g}_{ij} : \boldsymbol{u} \mapsto \boldsymbol{r}_{ij}(\boldsymbol{u}) + \boldsymbol{t}_{ij}$ with rotation $\boldsymbol{r}_{ij}$ by angle $k_{ij}\pi/2$, $k_{ij} \in \mathbb{Z}$, and translation $\boldsymbol{t}_{ij} \in \mathbb{R}^2$ such that



$$\mathbf{g}_{ij}(\mathbf{f}_i(a)) = \mathbf{f}_j(a), \quad \mathbf{g}_{ij}(\mathbf{f}_i(b)) = \mathbf{f}_j(b). \tag{3}$$

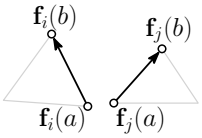
The sum of parametric angles around each vertex thus is a multiple of $\pi/2$. Vertices where it is 2π are called *regular*, others *singular*.

An edge e_{ij} for which $\mathbf{g}_{ij}$ is not identity is called a *cut edge*. Commonly, methods for the generation of seamless parametrizations aim to minimize the number (or total length) of cut edges. For topological reasons, at least a *cutgraph* which includes all singular vertices has to be included [BZK09], as in Fig. 1 left.

5.1. Constraints

We identify four kinds of commonly relevant constraints. Besides the above per-edge *transition constraints* (3), which ensure seamlessness, there are *alignment constraints*, *connection constraints* and *cycle constraints*, depending on the specific use case.

Transition Constraints. For each edge e_{ij} between triangles T_i, T_j and vertices a, b, equations (3) impose constraints relating the parametrizations of two adjacent faces to each other. As $\boldsymbol{t}_{ij}$ is arbitrary, the constraint can equivalently be simplified to

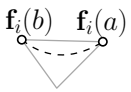


$$\boldsymbol{r}_{ij}(\mathbf{f}_i(a) - \mathbf{f}_i(b)) = \mathbf{f}_j(a) - \mathbf{f}_j(b). \tag{4}$$

Note that this actually represents two constraint equations, as we are working with two-dimensional coordinates $\boldsymbol{u}$, i.e. $\mathbf{f}(\cdot) \in \mathbb{R}^2$.

The integer rotation coefficient k_{ij} per edge is fixed a priori – it can be given as additional input to our method or extracted from the parametrization as detailed in [EBCK13].

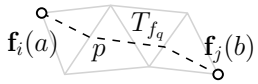
Alignment Constraints. For sharp features or boundaries, it is often required that isocurves of the parametrization align with them [BZK09]. To ensure this, one has to constrain one component of the parameters $\boldsymbol{u}$ of the two vertices a, b incident to the feature or boundary edge in triangle T_i to be equal:



$$\mathbf{f}_i(a)|_k = \mathbf{f}_i(b)|_k\,, \tag{5}$$

where k is either 0 or 1 and $|_k$ extracts the k^{th} component of a vector.

Connection Constraints. For purposes such as mesh or layout structure optimization, it can be desirable to constrain the connectivity of certain surface points a, b (feature vertices, singularities, etc.) in terms of parametrization isocurves. If one intends to prescribe the precise path of the isocurve between a and b, this can be achieved using the above feature alignment constraints. If one, however, wants to enforce that they are connected by *some* isocurve, while leaving the concrete path to optimization, so-called connection constraints can be used [MPKZ10]:



Let p be a path on the input mesh starting at vertex a and ending at vertex b while passing through the faces $T_{f_0}, \ldots, T_{f_m}$. Further, let $\mathbf{g}_p = \mathbf{g}_{f_{m-1}f_m} \circ \cdots \circ \mathbf{g}_{f_0 f_1}$ be the accumulated transition function of that path. The constraint to be imposed then is

$$\mathbf{f}_i(a)|_k = \mathbf{g}_p(\mathbf{f}_j(b))|_l\,, \tag{6}$$

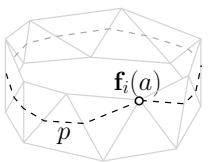
where l is the component corresponding to component k when taking the transition functions into account, i. e. $l = k$ if the rotational part of $\mathbf{g}_p$ is by an angle that is some multiple of π, and $l = 1 - k$ otherwise.

Note that the translation $\boldsymbol{t}_{f_{q-1}f_q}$ of a transition $\mathbf{g}_{f_{q-1}f_q}$ can be expressed in terms of $\mathbf{f}_{f_{q-1}}$ and $\mathbf{f}_{f_q}$ using Eq. (3) as

$$\boldsymbol{t}_{f_{q-1}f_q} = \mathbf{f}_{f_q}(v) - \boldsymbol{r}_{f_{q-1}f_q}\mathbf{f}_{f_{q-1}}(v), \tag{7}$$

where v is one (arbitrary) of the two vertices of the common edge.

Cycle Constraints. Another way to control the connectivity structures implied by global parametrizations is through isocurve cyclicity. Instead of enforcing specific points to be connected by isocurves, one can enforce isocurves to form cyclic loops rather than spirals or helices around parts of an object [BLK11]. This can be achieved using a variation, or special case, of connection constraints, with $a = b$ and the path p forming a closed loop [CIE*16, 4.3]:



$$\mathbf{f}_i(a)|_k = \mathbf{g}_p(\mathbf{f}_i(a))|_l\,. \tag{8}$$

5.2. Constraint System

For a particular parametrization task, with prescribed transition rotations, tagged feature and boundary edges to be aligned to, given connection and cycle paths, the constraints of the above types form a constraint equation system $C\boldsymbol{u} = \boldsymbol{b}$ as follows:

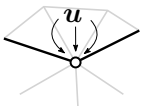
- one constraint (4) per mesh edge (consisting of two equations, one per component of $\boldsymbol{u}$),
- one constraint (5) per to-be-aligned feature or boundary edge,
- one constraint (6) per prescribed connection path,
- one constraint (8) per prescribed cycle path.

Note that here actually $C \in \{-1, 0, 1\}^{m \times n}$, except for unusual connection/cycle constraints whose associated paths cross the same or adjacent cut edges multiple times. Also note that all these constraints are homogeneous, i.e. $\boldsymbol{b} = \mathbf{0}$. However, our algorithm handles general non-zero righthand sides $\boldsymbol{b}$, thus more general less common parametrization constraints, like node spacing constraints [CK14], could be incorporated as well.

This system is sparse and usually highly underdetermined. We are interested in one of its solutions $\boldsymbol{u}$ close to $\bar{\boldsymbol{u}}$, the values given by an imperfect input parametrization computed with these constraints under consideration using any state-of-the-art parametrization technique. To this end, we could now directly apply our algorithms presented in Sec. 4. However, the process can significantly by simplified (thus sped up) by not treating the problem instance as a generic constraint system, but by exploiting the specific nature of the constraints relevant in the parametrization context, as detailed in the following.

5.3. Reduced Constraint System

At each vertex, cut edges partition the cycle of incident triangles into *sectors*; at a vertex not incident to any cut edges, the incident triangles form a single sector. As noted in [BZK09], and as is common practice, the parametrization can be represented by $\boldsymbol{u}$-coordinates per sector (called *sector variables* in the following), rather than per individual wedge, because wedges within the same sector are related by identity transitions (across non-cut edges).



We make use of the following additional terminology: An edge is called an *align edge*, if it is the edge of an alignment constraint. A vertex is called a *node* if one of the following holds:

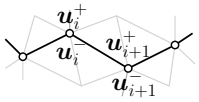
- it is a singular vertex,
- it is an endpoint of a connection or cycle constraint,
- it has more than two incident cut edges,
- it has an incident cut edge and an incident align edge,
- it has incident align edges aligning different coord. components.

We are going to derive a formulation such that only the $\boldsymbol{u}$-coordinates of node sectors, i.e. typically a small fraction of all

$\boldsymbol{u}$-coordinates, need to be considered in the constraint system matrix C. It is based on the concept of branches:

Nodes partition the union of all cut edges and align edges into *branches*. Each branch is a sequence of either cut or align edges forming a connected chain. Note that the rotation $\boldsymbol{r}$ of the transition $\mathbf{g}$ (cf. Eq. (3)) is constant per cut branch (a change in $\boldsymbol{r}$ would imply a singular vertex, thus a node, which splits branches). Likewise, the aligned coordinate component is constant per align branch.

Cut Branches. Consider a chain of m cut edges forming a branch with rotation $\boldsymbol{r}$. Let the $\boldsymbol{u}$-coordinates of the sectors along this branch be denoted $\boldsymbol{u}_0^{\pm},\dots,\boldsymbol{u}_m^{\pm}$. Then the transition constraint equations τ_i, $i=0,\dots,m-1$, following (4) for the edges of this branch are



$$\tau_i:\ \boldsymbol{r}(\boldsymbol{u}_i^+ - \boldsymbol{u}_{i+1}^+) - (\boldsymbol{u}_i^- - \boldsymbol{u}_{i+1}^-) = 0,$$

Note that the cumulative sum of these equations has a simple form:

$$\sum_{i=0}^{k}\tau_i:\ \boldsymbol{r}(\boldsymbol{u}_0^+ - \boldsymbol{u}_{k+1}^+) - (\boldsymbol{u}_0^- - \boldsymbol{u}_{k+1}^-) = 0, \tag{9}$$

Instead of equations τ_i, $i=0,\dots,m-1$, we can equivalently use equations $\sum_{i=0}^{k}\tau_i$, $k=0,\dots,m-1$. Ordering variables in the system such that those corresponding to node sectors come last, this yields the following structure per branch (exemplarily for $m=5$):

$$\left[\begin{array}{ccccc|cc}
\mathbf{1} & & & -\boldsymbol{r} & & -\mathbf{1} & \boldsymbol{r}\\
 & \mathbf{1} & & & -\boldsymbol{r} & -\mathbf{1} & \boldsymbol{r}\\
 & & \mathbf{1} & & & -\mathbf{1} & \boldsymbol{r}\\
 & & & \mathbf{1} & & -\mathbf{1} & \boldsymbol{r}\\
 & & & & -\boldsymbol{r} & -\mathbf{1} & \boldsymbol{r}\\
\hline
 & & & & & \mathbf{1}\ -\boldsymbol{r} & -\mathbf{1}\ \boldsymbol{r}
\end{array}\right]
\begin{array}{c}\\\\\\\\\\ \boldsymbol{u}_m^-\ \boldsymbol{u}_m^+\ \boldsymbol{u}_0^-\ \boldsymbol{u}_0^+\end{array}$$

where each entry is a 2×2 block ($\mathbf{1}$ denoting identity, $\boldsymbol{r}$ a rotation matrix by $0°/90°/180°/270°$), because each $\boldsymbol{u}$ has two components.

Align Branches. Analogously, cumulative sums of alignment constraints can be built, forming this structure per align branch:

$$\left[\begin{array}{ccccc|c}
\mathbf{1}^k & & & & & -\mathbf{1}^k\\
 & \mathbf{1}^k & & & & -\mathbf{1}^k\\
 & & \mathbf{1}^k & & & -\mathbf{1}^k\\
 & & & \mathbf{1}^k & & -\mathbf{1}^k\\
\hline
 & & & & & \mathbf{1}^k\ -\mathbf{1}^k
\end{array}\right]
\begin{array}{c}\\\\\\\\ \boldsymbol{u}_m^-\ \boldsymbol{u}_0^-\end{array}$$

where $\mathbf{1}^k$ is a 1×2 block, $[1,0]$ for $k=0$, $[0,1]$ for $k=1$, where k is the aligned coordinate component from (5).

Connections and Cycles. Connection and cycle constraints involve node sector variables corresponding to their paths' endpoints and additional cut vertex sector variables (due to the translation (7), which in the notation used here reads $\boldsymbol{t}=\boldsymbol{u}_i^- - \boldsymbol{r}\boldsymbol{u}_i^+$), which may be node or non-node sector variables. If non-node variables are involved, we can, as they are on a cut branch, effectively replace them by node variables by adding cumulative transition constraint equations (9) to the connection or cycle constraint equation. Concretely, $\boldsymbol{t}=\boldsymbol{u}_i^- - \boldsymbol{r}\boldsymbol{u}_i^+$ turns into $\boldsymbol{t}=\boldsymbol{u}_0^- - \boldsymbol{r}\boldsymbol{u}_0^+$ by adding $-\tau_{i-1},\dots,-\tau_0$.

Global System Globally ordering all variables in the system such that those corresponding to node sectors come last, the cumulative transition equations per cut branch, the cumulative alignment equations per align branch, and the modified connection and cycle equations together form the following constraint system (when the number of cut and align branches is q and p, respectively):

$$\left[\begin{array}{ccccccc|c}
\mathbf{1} & & -\boldsymbol{r}_1 & & & & & \\
 & \ddots & & \ddots & & & & \\
 & & \mathbf{1} & & -\boldsymbol{r}_q & & & B\\
 & & & & & \mathbf{1}^{k_1} & & \\
 & & & & & & \ddots & \\
 & & & & & & & \mathbf{1}^{k_p}\\
\hline
 & & & \vdots & & & & \\
 & & \cdots & 0 & \cdots & & & C\\
 & & & \vdots & & & &
\end{array}\right]$$

transition
alignment
transition
alignment
conn./cycle

Here we moved all rows involving only node sector variables (i.e. the last row of each of the above two local system types) to the bottom. We assumed here that no edge is both, cut and aligned. While this can easily be ensured in practice, it is not hard to also generalize to this case (the only difference is that the $-\boldsymbol{r}$ block and the $\mathbf{1}^k$ block may share some columns).

As can be seen, the upper left block of the constraint system matrix has a very particular structure: it is an upper triangular sign matrix, with entries from $\{-1,0,1\}$ only. This together with the lower left all-zero block enables the following efficient algorithm.

5.3.1. Algorithm

Given an input mesh with an imperfect parameterization, given by per-sector coordinates $\bar{\boldsymbol{u}}$, along with the set of constraints that was used to generate this parametrization and that the parametrization is supposed to satisfy, we first determine the nodes and the branches.

Let n be the total number of sectors involved in constraints, n_s the number of node sectors, and m_b, m_a, m_c the numbers of branches, align branches, and connection/cycle constraints, respectively.

We build the system matrix in the specific way just described and extract the submatrix C (lower right block). Note that C has only $2m_b+m_a+m_c$ rows and n_s columns. We then proceed as follows:

1. Our method from Sec. 4 is applied to only the small lower right block C and corresponding input values $\bar{\boldsymbol{u}}$; this yields values $\boldsymbol{u}$ for all node sector variables.
2. Then values for (non-node) variables involved in the $-\boldsymbol{r}$ block are chosen freely as $x_i \leftarrow \mathbb{F}_\delta(\bar{x}_i)$; this leaves one non-determined variable per row in the upper part (involved in the $\mathbf{1}$ or $\mathbf{1}^k$ block).
3. Values for these remaining variables are computed directly as an integer linear combination (2) of the already determined variables using Algorithm 5, so as to satisfy each row's equation.
4. Values of sectors not involved in any constraint obviously do not need to be touched.

Note that for the exactness of step 3 it is essential that the remaining variables (in the upper left block of the system matrix) have unit coefficients, as is the case here by construction – otherwise we would have to deal with exact divisibility issues.

This approach, based on applying the method from Sec. 4 to the reduced system matrix C rather than the entire system, leads to significant run time benefits – due to the circumstance that C is commonly smaller by several orders of magnitude. Compared to the direct application of the generic method, this specialized algorithm leads to run time improvements by factors of 10^3 or even 10^4 in our experiments. For instance, on an average model like focal-octa (cf. Table. 1), 0.013s rather than 52s are taken by the algorithms.

5.4. Inequality Constraints

Our approach focuses on the satisfaction of linear equality constraints. Parametrization problems often involve (non-linear, or linearized [Lip12, BCE*13]) inequality constraints as well, in order to achieve local injectivity (i.e. absence of fold-overs) – though these may also be intentionally omitted in some scenarios [ECBK14]. If these are not satisfied in the input, our method obviously does not repair the parametrization accordingly. This would require entirely different techniques – especially considering the non-linear nature of these injectivity constraints.

If the inequalities are satisfied in the input, there is a small chance the small adjustments our method makes to the parametrization lead to a local violation of these inequality constraints, i.e. a fold-over may get introduced – though this is rare, cf. Sec. 6. As such a violation is local, it can typically be fixed, as demonstrated in Sec. 6, by parametrically moving one of the vertices of the inverted triangle into the kernel of its 1-ring (if non-empty), i.e. the intersection of the half-planes spanned by the vertex's link edges. By making sure the inequalities are satisfied *with a margin* in the first place, one could in principle prevent such inequality violations from happening entirely, cf. Sec. 7.

6. Results

We applied our method to numerous parametrizations, generated by state of the art seamless parametrization methods, namely 113 models parametrized using the method of [MPZ14] and 92 using the method of [BCE*13]. These two datasets were provided as supplemental material by the authors of [MPZ14].

We processed these using our method, with seamless transition constraints across all cut edges as well as alignment constraints for boundary edges. Tables 1 and 2 show statistics of a sample of the results, as well as aggregate information over the entire datasets. While the input parametrization is inexact, i.e. constraints are violated, in every single case, all constraints are exactly satisfied by the output parametrization computed by our method, represented in standard floating point numbers, for each case from the datasets. The exact output files are available from the authors' websites.

As a consequence of the way parametrizations are generated by the method of [MPZ14], there often are badly conditioned triangles, whose relative height in parameter space is $< 10^{-11}$, particularly near cuts. At the same time, constraints are typically violated by $> 10^{-10}$ (cf. Table 1). Hence the adjustment that necessarily needs to be made for constraint satisfaction has potential to introduce flips. This occurred in 13 of the 113 cases, most often (in 9 cases) just a single flip. The simple local 1-ring-kernel based post-process described in Sec. 5.4 removed these flips again (even one flip that was already present in the dataset), except for one single model (FOCAL-OCTA, the one with the worst parametric distortion of the entire dataset), where the extremely small triangles surrounding two adjacent singularities cannot locally be un-flipped while preserving exact constraint satisfaction. The [BCE*13] dataset has better conditioned triangles in parameter space (as inequalities are imposed with an ε-margin in that method, cf. Sec. 7). In these cases not a single flip was introduced in the 92 models (and most of those already present in the input in that dataset were actually removed).

The tables also contain data about the average and maximum adjustment $\Delta\boldsymbol{u}$ made to the input parametrization's coordinates by our method. These quantities were normalized and are specified relative to the total extent of the parametrization for comparability.

Table 3 contains information about the range of values in matrix $\hat{C}$ after processing by Algorithms 1 and 2. While with transition constraints and alignment constraints we never encountered a value exceeding 2, connection and cycle constraints prove to cause some increase. To test this, we added 10, 20, 50 and 100 randomly generated connection constraints between pairs of parametrization singularities. Fig. 2 plots the total time taken by our matrix conversion, demonstrating that also these constraints, in addition to transition and alignment constraints, can easily be handled.

6.1. Comparison

Exact Arithmetic Solvers. An alternative strategy to yield a result exactly satisfying all constraints is the use of a solver based on adaptive precision numbers and exact arithmetic. We performed experiments using CGAL's QP solver with rational arithmetic on the models from Tables 1 and 2. Compared to our method, run time was more than 4 orders of magnitude longer in each case. Furthermore, the resulting values are rational numbers, with up to several hundred bits in numerator and denominator, not representable using standard floating point numbers without introducing error again.

Constraint Elimination. As detailed in Sec. 4 part of our strategy is related to the master-slave method, also referred to as constraint or variable elimination [CMPW01, Ch. 13.1]. To illustrate the importance of our specific algorithmic strategy, i.e. our deviation from this classical procedure, we also tried to perform the implied variable value computation not using our Alg. 3, 4, and 5 but by a straightforward evaluation of Eq. (2) in floating point arithmetic. The resulting values violate the constraints by up to 10^{-11}; Eq (2) consists of up to 51 terms in the test cases, giving ample chance to the accumulation of rounding errors in the involved additions, multiplications and divisions. Constraints are violated in every single case, with errors generally larger than 10^{-16} in all cases.

6.2. Demonstration of Benefits

Small violations of constraints may be negligible in some applications; in other cases they are fatal. A common operation on seamless parametrizations is the tracing of isocurves, e.g. for the construction of motorcycle graphs [CBK15, CZ17], quadrilateral base complexes [RRP15], spline domains [MPKZ10], quad meshes [EBCK13], or other graph structures. Already when considering this specific operation of isocurve tracing, one can observe a wide range of critical issues arising out of non-exactly satisfied constraints – regardless of the magnitude of constraint violations.

model	#F	#N	#B	time (ms) IREF	IRREF	Eval	sparsity (%) C	$\hat{C}$	($\times 10^{-8}$) max $C\bar{\boldsymbol{x}}$	change ($\times 10^{-8}$) avg $\Delta\boldsymbol{u}$	max $\Delta\boldsymbol{u}$	# flips $\bar{\boldsymbol{u}}$	$\boldsymbol{u}$	final
BRAIN	317K	4,500	4,613	1,212	1,054	548	0.02	0.03	0.021	0.041	0.873	0	10	0
VH_SKIN	195K	1,850	2,007	218	215	97	0.04	0.08	175.0	0.109	234.2	1	1	0
DAVID	92K	560	559	15	14	7	0.13	0.25	0.011	0.025	0.438	0	0	0
VASE-LION	152K	477	476	11	10	5	0.15	0.30	1.049	0.075	1.727	0	0	0
JULIUS	130K	406	317	5	2	2	0.16	0.25	0.034	0.042	0.541	0	0	0
FOCAL-OCTA	48K	339	338	6	5	2	0.22	0.40	0.090	0.041	0.512	0	16	15
EROS	136K	294	293	5	4	2	0.26	0.48	0.014	0.028	0.374	0	0	0
BEETLE	62K	244	206	3	2	1	0.29	0.51	0.663	0.037	0.741	0	0	0
BLADE	94K	216	215	3	2	1	0.35	0.60	0.085	0.029	0.383	0	0	0
AIRCRAFT	11K	134	133	1	1	0	0.56	0.97	0.002	0.026	0.307	0	0	0
PULLEY	132K	111	112	1	1	0	0.68	1.31	0.004	0.059	0.690	0	1	0
WRENCH	68K	54	55	0	0	0	1.39	2.39	0.003	0.060	0.178	0	0	0
entire dataset (avg):				21	19	9	0.63	1.18	7.779	0.401	11.91			

Table 1: *Statistics of a sample of models (with #F triangles) parametrized using [MPZ14] and processed by our method. The processing time mostly depends on the number of nodes (#N) and branches (#B). The maximum input constraint violation* max $C\bar{\boldsymbol{x}}$*, as well as our adjustment magnitude* $\Delta\boldsymbol{u}$ *is given, all relative to total extent for comparability. The average of* $\Delta\boldsymbol{u}$ *was computed only over variables involved in constraints. The number of flipped triangles in input* $\bar{\boldsymbol{u}}$*, our initial output* $\boldsymbol{u}$*, and our final output (after the 1-ring-kernel based fix) is given.*

model	#F	#N	#B	time (ms) IREF	IRREF	Eval	sparsity (%) C	$\hat{C}$	($\times 10^{-7}$) max $C\bar{\boldsymbol{x}}$	change ($\times 10^{-7}$) avg $\Delta\boldsymbol{u}$	max $\Delta\boldsymbol{u}$	# flips $\bar{\boldsymbol{u}}$	$\boldsymbol{u}$	final
GARGOYLE	100K	383	382	9	17	6	0.3	0.5	1.19	0.02	1.87	7	7	0
DAVID	50K	194	193	3	5	2	0.5	1.1	0.35	0.01	0.34	0	0	0
BEETLE	39K	140	84	1	1	1	0.9	1.1	0.22	0.01	0.24	0	0	0
ROBOCAT	7K	105	106	1	2	1	0.9	1.8	342.5	20.6	391.1	97	97	12
SHARK	20K	83	82	1	1	1	1.2	2.5	15.4	0.59	17.8	0	0	0
KNOT	100K	66	67	1	1	0	1.5	2.8	0.77	0.08	1.35	0	0	0
PULLEY	100K	48	49	0	0	0	2.0	3.9	74.5	13.1	123.6	1	1	0
FEMUR	8K	44	47	0	0	0	2.1	3.4	1971	157.6	1971	30	30	5
VASE	100K	33	32	0	0	0	3.1	5.9	1.00	0.12	1.10	0	0	0
CUP	11K	17	20	0	0	0	4.9	7.0	0.06	0.01	0.048	0	0	0
entire dataset (avg):				1	2	1	2.2	4.2	47.48	2.38	49.31			

Table 2: *Statistics for our processing of the dataset generated using the seamless parametrization method of [BCE*13], analogous to Table 1.*

Tracing is commonly performed by starting from a particular seed point on the surface (such as a singularity or a feature point) and following the parametrization's u- or v-isocurve (taking transitions into account whenever a cut is crossed) until a specific target point is reached or some other condition is met. If there is even just the slightest violation of a constraint (whether concerning transition seamlessness, alignment, connections, cycles) involved in the path taken by the trace, the target *will* be missed, or the stopping condition may never (or wrongly) be met – or an isocurve simply falls into the numerical gap of the parametrization at a cut edge. The key issue in this context is the circumstance that (even minuscule) geometrical errors translate into structural, combinatorial errors.

To concretely demonstrate the issues due to constraint violations, we performed tracing operations on the inexact input parametrizations from Sec. 6 – as well as, for comparison, on the exact output of our method. The following is a list of some of the encountered issues; one instance of each type is shown in Fig. 3 and 4.

- When tracing an isocurve into a cut edge where seamlessness is not exact, the corresponding trace direction on the other side of the cut may point directly back onto the edge (instead of away, as would be guaranteed if exact). The tracing gets stuck, oscillating between two sides, remaining at that point indefinitely (Fig. 3a).
- When tracing onto a regular cut vertex, there may be multiple or none (instead of a single, as would be guaranteed if exact) outgoing trace directions (Fig. 3b).
- When tracing along a sequence of aligned boundary edges, the isocurve may "fall off" the mesh due to inexact transitions when crossing a cut (Fig. 3c) – or at any point, if alignment is inexact.
- When two surface points are connected by a connection constraint, tracing an isocurve starting from one is expected to reach the other. Even the tiniest constraint violation (on the connection constraint or some intermediate cut transition), will cause it to be missed, however. Then, e.g., the two points are not connected in

model \ #connections	max $\lvert \hat{C}_{ij} \rvert$ 0	10	20	50	100
BRAIN	2	6	4	8	24
FILIGREE	2	6	8	6	16
RED CIRCULAR BOX	2	2	8	10	8
THAI STATUE	2	5	4	8	6
GARGOYLE	2	3	6	6	6
LUCY	2	2	4	4	8
CHINESE LION	2	4	4	8	5
DANCING CHILDREN	2	4	4	4	16
POLYGIRL	2	4	4	6	6
BEETLE	2	2	4	4	70
HEPTOROID	2	4	4	8	12
AIRCRAFT	2	4	8	12	8
HELMET	2	5	2	2	3

Table 3: *Maximum absolute entry in final matrix $\hat{C}$, for different numbers (0, 10, 20, 50, 100) of connection constraints imposed.*

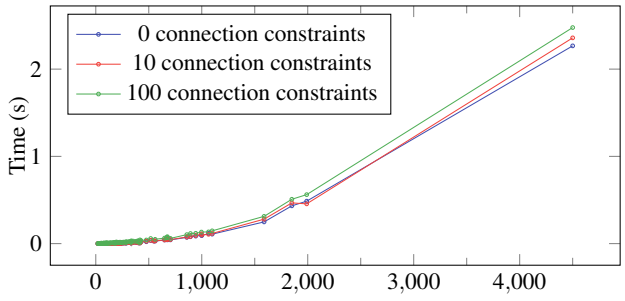


Figure 2: *Total time taken by the matrix conversion into IRREF relative to the number of nodes, over the entire dataset of [MPZ14]. Various numbers of random connection constraints were added; notice that these have little impact on the run time.*

the graph or mesh structure arising from the traces as expected – contrary to the intention of the connection constraint (Fig. 4 top).

- Analogously, isocurves that are supposed to form closed loops (due to imposed cycle constraints) will instead form long spirals or helices (with extremely narrow pitch) (Fig. 4 bottom).

While one may think of ad hoc means to locally resolve some individual issues, there is no obvious way of doing this in a principled, consistent manner. For instance, artificially skipping *along* an edge where a trace got stuck implies the merging of distinct traces reaching that edge, likely violating the expected and required properties (e.g. quadrilateral faces only) of the structures built from these traces. By employing some kind of ε-based snapping technique one could possibly take care of some near-miss situations but, of course, such heuristics at the same time inevitably enable the occurrence of false positives (merging things that are supposed to be separate), with similarly fatal effects on the resulting graph structures' properties. Such kinds of issues do not have to be dealt with when tracing in a truly seamless parametrization, as provided by our method. In this ideal setting furthermore symmetry is ensured trivially: tracing from a to b yields the same result as tracing from b to a. This likewise would be hard to ensure by means of such ad hoc remedies, ε-snapping, or other application-specific heuristics.

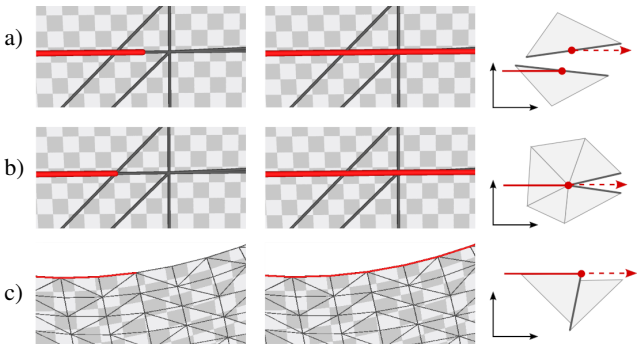


Figure 3: *Left: error cases when tracing in inexact parametrizations. Center: tracing in our exact parametrization. Right: exaggerated illustration of the respective situation in uv-space; dashed arrows indicate where the trace is supposed to but cannot continue.*

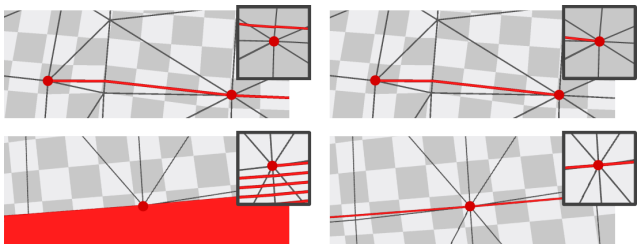


Figure 4: *Left: connectivity errors due to tracing in inexact parametrizations with connection (top) or cycle constraint (bottom). A point (vertex) that is supposed to be reached is missed instead. Such errors arise no matter how small the error (top: around 10^{-16}, bottom: around 10^{-14}). The insets show extreme magnifications of the critical spots. Right: correct behavior using our result.*

7. Limitations and Future Work

Our experiments reveal that in the parametrization context, even for very large problem instances, matrix values remain small (cf. Table 3), i.e. integer overflow issues are far from relevant here. More generally, one could of course employ multi precision integer types (e.g. `mpz` [G*15]) to avoid overflow under any circumstances. Note, however, that if not only intermediate values but also values of the final matrix $\hat{C}$ are very large, this may lead to strongly quantized result values $\boldsymbol{x}$, due to the divisibility requirement described in Sec. 4.3. In the absolute worst case, for a homogeneous system the (valid but probably useless) zero solution would be produced.

The only aspect in which the method thus could actually fail in the sense of producing a result $\boldsymbol{x}$ that does not satisfy $C\boldsymbol{x} = \boldsymbol{b}$, is an overflow of the fixed point range $(-\delta, \delta)$ in the dot product evaluation and implied variable value computation of Sec. 4.3. Note that, by our choice of δ, this may only occur if the computed value for an implied variable x_i would exceed its input value $\bar{x}_i$ by more than the absolute largest value among all input variables, i.e. $\|x_i - \bar{x}_i\| / \max_i |\bar{x}_i| > 1$. Hence, either does the input have to violate constraints extremely rather than slightly (in this case a more conservative choice of δ could help), or the integer values $\hat{\boldsymbol{b}}$ have to have grown beyond being representable by $\mathbb{F}_\delta$ (which cannot occur for homogeneous systems). In our parametrization experiments, the

relative change $\|x_i - \bar{x}_i\| / \max_i |\bar{x}_i|$ never even exceeded 10^{-5}, but for other applications this aspect may be a relevant limitation.

Our approach focuses on equality constraints. An interesting direction is the systematic treatment of inequality constraints. One could aim to adjust values in such a way that satisfaction of inequalities is *preserved* or, more generally, consider the harder problem of ensuring that it is *established* even if inequalities are (slightly) violated initially. Alternatively, if one could determine an a priori bound on the modification performed by the algorithm, one could require the solver used to generate the input to satisfy the inequalities with some appropriate margin (as done in some cases anyway, cf. [BCE*13, Eq. 4]) to ensure preservation despite modification.

We demonstrated the use of our approach in the context of constrained seamless surface parametrization, but it has the potential to be useful in other scenarios. A closely related, thus obvious application is seamless volume parametrization [NRP11]. While our basic algorithm could be applied as is, the specialized optimization we described in Sec. 5.3 is surface specific. Investigation of analogous simplifications for the volume case would thus be worthwhile.

Acknowledgments. The authors thank Jiaran Zhou for her help with early experiments and Max Lyon for helpful discussions.

References

[AL15] AIGERMAN N., LIPMAN Y.: Orbifold Tutte embeddings. *ACM Trans. Graph. 34*, 6 (2015). 2

[APL15] AIGERMAN N., PORANNE R., LIPMAN Y.: Seamless surface mappings. *ACM Trans. Graph. 34*, 4 (2015). 2

[BCE*13] BOMMES D., CAMPEN M., EBKE H.-C., ALLIEZ P., KOBBELT L.: Integer-grid maps for reliable quad meshing. *ACM Trans. Graph. 32*, 4 (2013). 2, 3, 8, 9, 11

[BCGB08] BEN-CHEN M., GOTSMAN C., BUNIN G.: Conformal Flattening by Curvature Prescription and Metric Scaling. *Comp. Graph. Forum 27*, 2 (2008). 2

[BCW17] BRIGHT A., CHIEN E., WEBER O.: Harmonic global parametrization with rational holonomy. *ACM Trans. Graph. 36*, 4 (2017). 2

[BLK11] BOMMES D., LEMPFER T., KOBBELT L.: Global structure optimization of quadrilateral meshes. *Comp. Graph. Forum 30*, 2 (2011), 375–384. 6

[BZK09] BOMMES D., ZIMMER H., KOBBELT L.: Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3 (2009). 2, 5, 6

[CBK15] CAMPEN M., BOMMES D., KOBBELT L.: Quantized global parametrization. *ACM Trans. Graph. 34*, 6 (2015). 2, 3, 8

[CIE*16] CAMPEN M., IBING M., EBKE H.-C., ZORIN D., KOBBELT L.: Scale-invariant directional alignment of surface parametrizations. *Comp. Graph. Forum 35*, 5 (2016). 6

[CK14] CAMPEN M., KOBBELT L.: Quad layout embedding via aligned parameterization. *Comp. Graph. Forum 33*, 8 (2014). 6

[CMPW01] COOK R. D., MALKUS D. S., PLESHA M. E., WITT R. J.: *Concepts and Applications of Finite Element Analysis*, 4 ed. Wiley, 2001. 3, 8

[CZ17] CAMPEN M., ZORIN D.: Similarity maps and field-guided T-splines: a perfect couple. *ACM Trans. Graph. 36*, 4 (2017). 2, 3, 8

[Dur12] DUREISSEIX D.: Generalized fraction-free LU factorization for singular systems with kernel extraction. *Linear Algebra and its Applications 436*, 1 (2012), 27–40. 3

[EBCK13] EBKE H.-C., BOMMES D., CAMPEN M., KOBBELT L.: QEx: robust quad mesh extraction. *ACM Trans. Graph. 32*, 6 (2013). 3, 4, 5, 6, 8

[ECBK14] EBKE H.-C., CAMPEN M., BOMMES D., KOBBELT L.: Level-of-detail quad meshing. *ACM Trans. Graph. 33*, 6 (2014). 8

[G*15] GRANLUND T., ET AL.: *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.1.0 ed., 2015. http://gmplib.org/. 3, 10

[Gär99] GÄRTNER B.: Exact arithmetic at low cost – a case study in linear programming. *Comp. Geom. 13*, 2 (1999), 121 – 139. 3

[GS00] GÄRTNER B., SCHÖNHERR S.: An efficient, exact, and generic quadratic programming solver for geometric optimization. In *Proc. 16th Symp. Comp. Geom.* (2000), SCG '00, pp. 110–118. 3

[GSW16] GLEIXNER A. M., STEFFY D. E., WOLTER K.: Iterative refinement for linear programming. *INFORMS Journal on Computing 28*, 3 (2016), 449–464. 3

[KCPS15] KNÖPPEL F., CRANE K., PINKALL U., SCHRÖDER P.: Stripe patterns on surfaces. *ACM Trans. Graph. 34*, 4 (2015). 2

[KNP07] KÄLBERER F., NIESER M., POLTHIER K.: Quadcover - surface parameterization using branched coverings. *Comp. Graph. Forum 26*, 3 (2007), 375–384. 2

[LBK16] LYON M., BOMMES D., KOBBELT L.: HexEx: Robust hexahedral mesh extraction. *ACM Trans. Graph. 35*, 4 (2016). 3

[Lip12] LIPMAN Y.: Bounded distortion mapping spaces for triangular meshes. *ACM Trans. Graph. 31*, 4 (2012). 8

[MPKZ10] MYLES A., PIETRONI N., KOVACS D., ZORIN D.: Feature-aligned T-meshes. *ACM Trans. Graph. 29*, 4 (2010). 3, 6, 8

[MPZ14] MYLES A., PIETRONI N., ZORIN D.: Robust field-aligned global parametrization. *ACM Trans. Graph. 33*, 4 (2014). 2, 8, 9, 10

[MRL01] MICHEL C., RUEHER M., LEBBAH Y.: Solving constraints over floating-point numbers. In *Int. Conf. Principles and Practice of Constraint Programming* (2001), Springer, pp. 524–538. 3

[MZ12] MYLES A., ZORIN D.: Global parametrization by incremental flattening. *ACM Transactions on Graphics (TOG) 31*, 4 (2012). 2

[MZ13] MYLES A., ZORIN D.: Controlled-distortion constrained global parametrization. *ACM Trans. Graph. 32*, 4 (2013). 2

[NPPZ12] NIESER M., PALACIOS J., POLTHIER K., ZHANG E.: Hexagonal global parameterization of arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics 18*, 6 (2012), 865–878. 2

[NRP11] NIESER M., REITEBUCH U., POLTHIER K.: CubeCover - parameterization of 3D volumes. *Comp. Graph. Forum 30*, 5 (2011). 11

[RNLL10] RAY N., NIVOLIERS V., LEFEBVRE S., LÉVY B.: Invisible seams. *Comp. Graph. Forum 29*, 4 (2010). 2

[RRP15] RAZAFINDRAZAKA F. H., REITEBUCH U., POLTHIER K.: Perfect matching quad layouts for manifold meshes. *Comp. Graph. Forum 34*, 5 (2015). 3, 8

[SCOGL02] SORKINE O., COHEN-OR D., GOLDENTHAL R., LISCHINSKI D.: Bounded-distortion piecewise mesh parameterization. In *Proc. Conf. Vis. '02* (2002), VIS '02, pp. 355–362. 2

[She97] SHEWCHUK J. R.: Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry 18*, 3 (1997), 305–368. 3

[SPR06] SHEFFER A., PRAUN E., ROSE K.: Mesh parameterization methods and their applications. *Found. Trends. Comput. Graph. Vis. 2*, 2 (2006), 105–171. 2

[SSP08] SPRINGBORN B., SCHRÖDER P., PINKALL U.: Conformal equivalence of triangle meshes. *ACM Trans. Graph. 27*, 3 (2008). 2

[TACSD06] TONG Y., ALLIEZ P., COHEN-STEINER D., DESBRUN M.: Designing quadrangulations with discrete harmonic forms. In *Proc. SGP '06* (2006), pp. 201–210. 2

[Tur95] TURNER P. R.: *A Simplified Fraction-free Integer Gauss Elimination Algorithm*. Naval Air Warfare Center, Aircraft Div., 1995. 3

[Yap97] YAP C. K.: Robust geometric computation. In *Handbook of Discrete and Computational Geometry, chapter 35*. CRC Press LLC, Boca Raton, FL, 1997, pp. 653–668. 3

Addendum

Exact Constraint Satisfaction for Truly Seamless Parametrization

In this addendum to the article "Exact Constraint Satisfaction for Truly Seamless Parametrization" we

- make additional remarks on the structure and sparsity of the solution space, which indirectly determines how close to the input $\bar{\boldsymbol{x}}$ one can expect an exactly constraint satisfying solution $\boldsymbol{x}$ in the floating point numbers – and whether there exists one at all;
- point out and correct a mistake in Eq. (2) / Algorithm 4 – which is relevant only to the inhomogeneous case ($\boldsymbol{b} \neq 0$), thus not to the targeted seamless parametrization problem, where all the constraints are homogeneous.

A. Homogeneous Constraint Systems

The homogeneous constraint system $C\boldsymbol{x} = \boldsymbol{0}$ has at least one solution, namely the zero solution $\boldsymbol{x} = \boldsymbol{0}$. If furthermore C is rank-deficient, it has infinitely many solutions, even when restricting to a discrete set like $\mathbb{Z}$ – or a scaled version $s\mathbb{Z}$ thereof. Note that $\mathbb{F}_\delta$ is nothing but a bounded subset of $s\mathbb{Z}$ for a particular choice of $s = 2^r$, $r \in \mathbb{Z}$ (depending on δ and the number of mantissa bits).

Considering, for instance, a single equation $ax_1 - bx_2 = 0$, with $a, b \in \mathbb{Z}$, we have as solution space in the fixed point numbers $s\mathbb{Z}$:

$$\Omega \cap s\mathbb{Z}^2 = \{k\,(b/g, a/g) \mid k \in s\mathbb{Z},\, g = \gcd(a,b)\}.$$

The worst case in terms of sparsity of this space is a and b being large coprime numbers, such that $\gcd(a,b) = 1$. In particular, if in this case furthermore a or b is larger than the largest number from $\mathbb{F}_\delta$, the zero solution $\boldsymbol{x} = \boldsymbol{0}$ is the only representable solution. For systems of multiple equations the situation is more complicated; only after transformation of C into Hermite or Smith normal form [Laz96] do the coefficients become apparent whose greatest common divisors determine the spacing between neighboring solutions in the discrete space.

Fortunately, homogeneous constraints encountered in geometry processing applications usually do not have large prime coefficients. In the targeted seamless parametrization case, almost all coefficients are ± 1, and even when transforming C they do not grow in any significant manner, cf. Table 3.

Note that $\mathbb{F}$ may contain additional solutions that are not in $\mathbb{F}_\delta \subset \mathbb{F}$. The proposed algorithm constructs solutions restricted to $\mathbb{F}_\delta$. Ways to generalize to all of $\mathbb{F}$ are not obvious and the significance of the potential benefit is unclear.

B. Inhomogeneous Constraint Systems

The bracketing in Eq. (2) is incorrect for the inhomogeneous case, where $\boldsymbol{b} \neq \boldsymbol{0}$. The correct expression is

$$x_{\pi(j)} = \left(\hat{b}_j - \sum_{i=\pi(j)+1}^{n} \hat{C}_{ji} x_i \right) / \hat{C}_{j\pi(j)}. \qquad (2')$$

Note that for $\hat{b}_j = 0$, i.e. the homogeneous case – as relevant in the seamless parametrization context – this does not differ from Eq. (2).

This furthermore increases the difficulty of finding a solution: instead of being able to choose the free variables such that they – individually – are divisible by certain values (as possible in the homogeneous case, cf. Algorithm 4), one needs to choose the free variables such that these entire sums (including $\hat{b}_j$) are divisible by the coefficients $\hat{C}_{j\pi(j)}$ of the dependent variables. The choice made by Algorithm 4 is thus actually insufficient for the inhomogeneous case; instead, an algorithm for generic linear Diophantine equation systems can be employed [CD94, KMA08].

While in the homogeneous case the constraint system always has a solution in $\mathbb{F}_\delta$, in the inhomogeneous case a system may have no solution at all in $\mathbb{F}_\delta$ or in $\mathbb{Z}$ (even if it is feasible over $\mathbb{R}$). A minimal example is the single equation

$$3x_1 = 1.$$

For a multivariate equation $\sum a_i x_i = c$ a solution in $\mathbb{Z}$ exists iff $\gcd(a_1, a_2, \dots) \mid c$, i.e. iff the greatest common divisor of all coefficients divides c. Thus, e.g.

$$9x_1 + 14x_2 = 1$$

has a solution in $\mathbb{Z}^2$, while

$$9x_1 + 15x_2 = 1$$

does not. Various algorithms to determine the feasibility of and to solve such linear Diophantine equations (and equation systems) have been described, e.g. [Laz96, CD94, KMA08]. Furthermore, some algorithms supporting range bounds, e.g. [AHL00], can be used to find a solution within the bounded set $\mathbb{F}_\delta$ – or to prove non-existence of such a solution. In general, answering this question is NP-hard.

In conclusion it can be stated that – unless additional a priori knowledge about the characteristics of coefficients and right hand side values is available in a specific application – one cannot expect inhomogeneous constraint systems to be exactly satisfiable in the fixed or floating point numbers in general. The use of rational number types may be the more practical approach in such inhomogeneous cases.

Acknowledgments: The authors thank Kai Hormann and Marco Tarini for helpful input on the issue of feasibility and sparsity.

References

[AHL00] AARDAL K., HURKENS C. A., LENSTRA A. K.: Solving a system of linear Diophantine equations with lower and upper bounds on the variables. *Mathematics of Operations Research 25*, 3 (2000), 427–442. 1

[CD94] CONTEJEAN E., DEVIE H.: An efficient incremental algorithm for solving systems of linear Diophantine equations. *Information and computation 113*, 1 (1994), 143–172. 1

[KMA08] KHORRAMIZADEH M., MAHDAVI-AMIRI N.: On solving linear diophantine systems using generalized Rosser's algorithm. *Bulletin of the Iranian Mathematical Society 34*, 2 (2008), 1–25. 1

[Laz96] LAZEBNIK F.: On systems of linear Diophantine equations. *Mathematics Magazine 69*, 4 (1996), 261–266. 1