Linear Analysis of Nonlinear Constraints for Interactive Geometric Modeling

Martin Habbecke and Leif Kobbelt

Computer Graphics Group, RWTH Aachen University, Germany

Abstract

Thanks to its flexibility and power to handle even complex geometric relations, 3D geometric modeling with nonlinear constraints is an attractive extension of traditional shape editing approaches. However, existing approaches to analyze and solve constraint systems usually fail to meet the two main challenges of an interactive *3D modeling system: For each atomic editing operation, it is crucial to adjust as few auxiliary vertices as possible in order to not destroy the user's earlier editing effort. Furthermore, the whole constraint resolution pipeline is required to run in real-time to enable a fluent, interactive workflow. To address both issues, we propose a novel constraint analysis and solution scheme based on a key observation: While the computation of actual vertex positions requires nonlinear techniques, under few simplifying assumptions the determination of the minimal set of to-be-updated vertices can be performed on a linearization of the constraint functions. Posing the constraint analysis phase as the solution of an under-determined linear system with as few non-zero elements as possible enables us to exploit an efficient strategy for the Cardinality Minimization problem known from the field of Compressed Sensing, resulting in an algorithm capable of handling hundreds of vertices and constraints in real-time. We demonstrate at the example of an image-based modeling system for architectural models that this approach performs very well in practical applications.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Modeling packages

1. Introduction

Modeling with constraints is an important tool for the construction and modification of 3D geometric models. Especially in the case of modeling man-made structure like architecture or machine parts, geometric constraints are able to create and preserve ubiquitous alignment properties like element parallelism, collinearity, fixed angles and distances, or symmetry relations. The automatic satisfaction of these constraints greatly simplifies the modeling process by reducing the degrees of freedom and furthermore strongly improves the quality of the results. Consequently, geometric constraints have a long-standing history in Computational Geometry and CAD/CAM. Thanks to the high computation power of commodity PCs, several interactive constraint-based shape editing [ZFCO*11, GSMCO09, XWY*09] and resizing [KSSCO08, CLDD09] approaches have been proposed recently.

The central problem of interactive constrained editing is, given a user modification in the form of re-positioning a set of vertices, to adjust the positions of the remaining vertices such that all constraints are satisfied. Various solutions to handle this problem have been proposed, ranging from simple (weighted) least squares solutions [XWY*09] over ad-hoc propagate-and-fix approaches [ZFCO*11, GSMCO09] to elaborate strategies from the field of Computational Geometry [JTNM06, HL01, FASR08]. However, the main challenge of any *incremental* editing approach is not handled satisfyingly: In order to not destroy the results of earlier (manual or automatic) editing operations, it is of crucial importance to modify the positions of *as few additional vertices as possible* during the automatic constraint satisfaction phase. Figure 1 illustrates this problem at a simple example.

The main contribution of our work is an interactive constrained modeling approach with a well-defined strategy that, for an atomic editing operation, computes *as small as possible* model updates in terms of the total number of adjusted vertices. Similar to traditional constraint satisfaction approaches [JTNM06], our method consists of two phases,

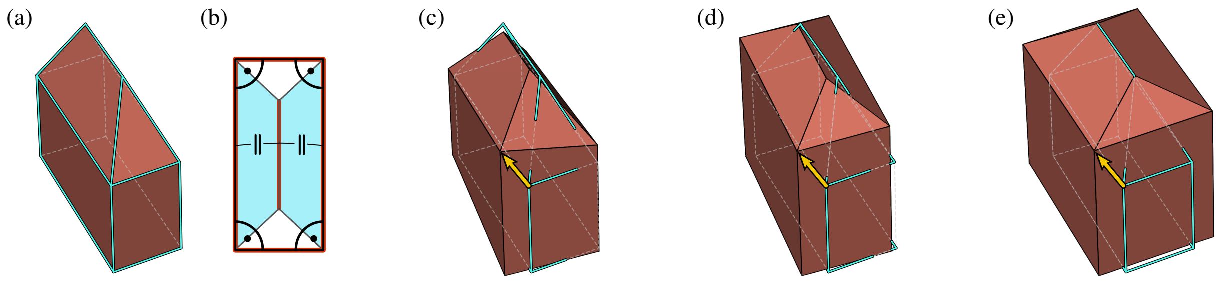

Figure 1: *Example of a simple modeling operation with nonlinear geometric constraints on the building model (a). In addition to the alignments depicted in (b), all red edges are constrained to be horizontal, and the shaded polygons should be symmetric. Furthermore, all vertices should stay as close as possible to their original positions in order to change the model as little as possible. When interactively moving one corner vertex, optimizing the positions of all remaining vertices with uniform constraint weights (c) as well as different constraint weight classes (d) does not yield a consistent result. Only when the optimization is performed on a minimal subset of vertices such that all constraints are guaranteed to be satisfiable (i.e., when leaving the vertices of the roof ridge fixed), we obtain the desired result in (e).*

an *analysis phase* that determines a set of vertices that need to be updated in order to satisfy all constraints, and a *solution phase* that computes actual vertex positions. The central idea of our approach is to perform a linear analysis by considering *infinitesimal* editing operations, and to take the full editing operations into account in the nonlinear solution phase only. Inspired by the Inverse Kinematics approach [BB04], the analysis phase is based on the nullspace of the constraints' Jacobian. For this to be feasible, we make three simplifying assumptions:

1. Editing operations performed by the user are limited to *linear* displacements of one or more vertices. That is, the vertices affected by an atomic editing operation are considered to be simultaneously shifted on straight lines towards their target positions.
2. All constraints are invariant under global translation.
3. Each editing operation is performed on a (input) model instance that satisfies all constraints.

The first assumption is quite natural, especially in the case of modeling man-made structure where it is common to move vertices along existing edges or known scene directions. Section 5 discusses how we integrate this assumption with an interactive modeling system that requires the model to be updated for visual feedback before the actual target position of an element is known. While the third assumption might seem to be rather strong from a traditional constraint satisfaction point of view, we will demonstrate it to be easily satisfiable by an effective, incremental initialization procedure that is able to "bootstrap" a suitable model instance. Thus, all three assumptions do not pose limitations in practice.

In our approach we assume the given constraint system is under-constrained. Well-constrained systems are of little interest for interactive modeling since no degrees of freedom are left. Over-constrained systems or contradicting constraints can be detected but we leave the resolution up to the user. Redundant sets of constraints do not pose a problem to our algorithm. The three above assumptions enable an algorithm that has several advantages over existing systems: By linearizing the constraint functions and examining their nullspace, the analysis phase can be formulated as a *Cardinality Minimization Problem* [RFP10] for which efficient (approximate) solutions are known from the field of *Compressed Sensing* [DDEK12]. Furthermore, the solution phase can be based on a standard nonlinear solver since the initialization required for reliable convergence is always provided by the model instance satisfying all constraints. Consequently, in contrast to existing image-based constrained reconstruction approaches [FF09, TW06], our algorithm is able to cope with hundreds of vertices and constraints in realtime, enabling a truly interactive modeling system.

In this work, we assume that constraints are defined on the vertices of a 3D polygonal model. Our main target application is image-based 3D modeling (i.e., modeling in image overlay). At the example of a prototypical image-based modeling system for 3D building models from oblique aerial images we demonstrate that our approach works very well in practice. Notice, however, that the proposed method is not limited to this particular application.

2. Related Work

Constrained Geometric Modeling In Computer Graphics, several methods have been proposed recently to efficiently modify 3D models while preserving important global features. Two particularly innovative systems are iWires [GSMCO09] and its recent generalization [ZFCO*11]. The iWires-system constructs a wire structure for an input model and derives modeling constraints like parallelism or symmetry. Modeling operations are propagated over the wire structure resulting in a deformed 3D model with global features being preserved. [ZFCO*11] generalize this concept by embedding model components in cage-like controllers, thereby lowering the burden of constructing a wire structure which can be difficult in case a model does not exhibit clear sharp features. Both approaches yield very powerful shape editing metaphors. However, the employed propagate-and-fix constraint resolution strategy in each step has a local view on a subset of constraints only. Thus, in certain situations (e.g. cyclic dependencies) it may get stuck in a locally unsolvable configuration although a global solution exists. Since our constraint analysis approach has a global view on all constraints, it is not prone to such failures. Our algorithm could hence be employed as a drop-in replacement for the constraint solver in both approaches.

[XWY*09] present a modeling system capable of handling joints naturally arising in 3D models of man-made, technical objects. After an analysis of joint properties to construct suitable modeling constraints, this approach performs a global nonlinear optimization of *all* joint positions. In addition, a sophisticated constraint weighting scheme is introduced that requires manual adjustments in order to achieve certain modeling effects. [KSSCO08] and [CLDD09] present methods to perform structure-aware resizing of 3D models. The first method applies a sizing field that is adapted to model features and structural detail, the second performs an optimization of vertex positions similar to our approach. However, the available set of constraints is limited to linear functions. While both approaches are able to preserve existing alignments, they do not allow for the creation of new relations between elements of a model. Yang et al. [YYPM11] present a system for the interactive deformation of constrained polygonal models. Similar to our approach, all model instances satisfying the constraints are considered to define a manifold in a high-dimensional space. Editing operations are mapped to finding suitable points on this manifold. In contrast to our goal of finding a model instance with only a few vertices changed, Yang et al. perform global model updates and define additional regularization energies (e.g. surface fairness) to handle degrees of freedom not fixed by the constraints. GlobFit [LWC*11] fits geometric primitives to 3D point clouds and detects their global relations. While this approach is not concerned with the modification of the resulting models, similar ideas could be used to further automate our target editing applications.

Geometric constraints and various strategies to fulfill them are at the core of every CAD/CAM system. The surveys of [JTNM06] and [HL01] give an excellent overview of the field. As outlined by Jermann et al. [JTNM06], traditional constraint satisfaction approaches usually try to identify solvable subproblems and then incrementally construct a complete solution. The more powerful bottom-up strategy (as, for instance, in the image-based 3D reconstruction approach [TW06]) has the disadvantage that it is impossible to control which vertices of the solution are free and which are dependent. Thus it is unclear how to integrate user input. Furthermore, such approaches are known to have problems with redundant constraints. Similarly, the image-based reconstruction system [FF09] stabilizes the reconstruction process by automatically detected constraints, but is not able to incorporate interactive editing. Traditional constraint solution strategies like [TW06, FF09] often require computation times in the order of at least a few seconds even for moderately complex problems and are thus not applicable to real-time interactive editing systems. Also related to our work is the Dynamic Geometry system based on geometric constraints [FASR08]. However, this system is able to handle sets of constraints with exactly one degree of freedom only.

Interactive Image-Based Modeling The *Façade* system [DTM96] is an early image-base reconstruction system. Façade is based on the manual construction of box-like geometric elements and the specification of links to 2D features in the input images. The system then recovers correct model dimensions and camera calibration parameters automatically. Façade does not, however, provide modeling operations in images space. Thus, in case a reconstruction fails, the only way of interacting is to add more 3D-to-2D links. VideoTrace [vdHDT*07] as well as the architectural modeling system [SSS*08] are also based on the user-guided construction of planar polygons. Both require additional scene information like 3D points or vanishing lines recovered by a pre-process and thus work for relatively dense image sequences only. While both approaches allow for modifications of the 3D model in image overlay, the interactions are rather simple thus often requiring tedious adjustments of many elements.

Inverse Kinematics The problem solved by Inverse Kinematics is strongly related to the problem of interactive constrained editing: For a robotic component consisting of joints and limbs, the goal is to find joint positions such that an end effector reaches a desired target. The resulting optimization problem usually is under-constrained. Instead of computing simple least-squares solutions, several ideas have been proposed to exploit the remaining degrees of freedom: For instance, [BB04] tries to reach secondary target positions, [DW97] minimize the maximal joint velocity. To our knowledge, however, the problem of moving as few joints as possible has not been considered.

Compressed Sensing The central insight of Compressed Sensing [DDEK12] in signal processing is that many real-world signals are sparse, i.e., can be represented as sparse vector $\mathbf{x} \in \mathbb{R}^n$ with respect to a suitable basis. Given a measurement process modeled as $\mathbf{y} = A\mathbf{x}$ with $A \in \mathbb{R}^{m \times n}$, $m \ll n$, the goal is to reconstruct the sparse signal $\mathbf{x}$ from the measurement $\mathbf{y}$. We will see in Section 3.2 that the constraint analysis can be formalized in a very similar way, i.e. as the solution of a linear system with as few non-zero elements as possible. The main difference of our setting regards the measurement matrix A: While finding suitable matrices A with favorable properties is a major research topic in Compressed Sensing, in our constraint analysis it is defined by the modeling constraints and does not allow for adjustments.

3. Constraint Resolution Approach

3.1. Method Overview

In our modeling system, constraints are encoded as functions $c_j : \mathbb{R}^{3n} \rightarrow \mathbb{R}$ with $c_j(\mathbf{x}_1, \dots, \mathbf{x}_n) \overset{!}{=} 0$, where n is the total number of vertices and $\mathbf{x}_i \in \mathbb{R}^3$ are the vertex positions. Constraint functions c_j can be nonlinear, with the only requirement that the gradients are well defined and $|c_j|$ is a meaningful measure of constraint deviation. We denote by $\mathbf{X} = (\mathbf{x}_1^T, \dots, \mathbf{x}_n^T)^T$ a vector in $\mathbb{R}^{3n}$ with all vertex coordinates stacked upon each other, and by $\mathbf{c}(\mathbf{X}) = (c_1(\mathbf{X}), \dots, c_m(\mathbf{X}))^T : \mathbb{R}^{3n} \rightarrow \mathbb{R}^m$ a vector-valued function that contains all constraints.

As stated earlier, we assume that each editing operation is performed on a model instance $\mathbf{X}_0$ with $\mathbf{c}(\mathbf{X}_0) = \mathbf{0}$. An editing operation is given in the form of a displacement $\mathbf{d} \in \mathbb{R}^{3n}$ where $\mathbf{d}$ has non-zero elements for the explicitly modified vertices only. The central goal of our constraint resolution approach is, for a given editing displacement $\mathbf{d}$, to find a *correction displacement* $\mathbf{d}'$ such that $\mathbf{c}(\mathbf{X}_0 + \mathbf{d} + \mathbf{d}') = \mathbf{0}$. Clearly, $\mathbf{d}'$ has to be chosen in a way such that the non-zero elements of $\mathbf{d}$ and $\mathbf{d}'$ are disjoint. More formally, let $I(\mathbf{d}) \subseteq \{1, \dots, n\}$ be the indices of the vertices affected by the displacement $\mathbf{d}$, then $I(\mathbf{d}) \cap I(\mathbf{d}') = \emptyset$. Otherwise the explicitly placed vertices would not remain at their intended position. Furthermore, $\mathbf{d}'$ should be as sparse as possible in order to modify as few auxiliary vertices as possible.

We represent the space of possible movements of each vertex $\mathbf{x}_i$ with a basis $\{\mathbf{b}_{i,1}, \mathbf{b}_{i,2}, \mathbf{b}_{i,3}\}$, $\mathbf{b}_{i,k} \in \mathbb{R}^3$. The canonical basis $\mathbf{b}_{i,1} = (1,0,0)^T$, $\mathbf{b}_{i,2} = (0,1,0)^T$, $\mathbf{b}_{i,3} = (0,0,1)^T$ is a viable choice. However, the basis vectors allow for the integration of application-specific knowledge such as local geometric alignment into the constraint analysis phase. In Section 5.2 we will discuss the construction of a basis specifically targeted at image-based modeling. For the sake of notational correctness, we extend the 3-dimensional basis vectors to vectors $\mathbf{B}_{i,k} := (0, \dots, 0, \mathbf{b}_{i,k}^T, 0, \dots, 0)^T \in \mathbb{R}^{3n}$ with $3(i-1)$ leading zeros. The correction displacement $\mathbf{d}'$ can then be represented as linear combination

$$\mathbf{d}' := \sum_{i \notin I(\mathbf{d})} \sum_{k=1}^{3} \alpha_{i,k} \mathbf{B}_{i,k}. \tag{1}$$

The computation of the correction displacement $\mathbf{d}'$ is split into two parts. In the analysis phase, only an as small as possible set of basis vectors, i.e., set of non-zero $\alpha_{i,k}$, is determined. The actual displacement, i.e., the values of the previously selected $\alpha_{i,k}$, are computed in the solution phase.

3.2. Analysis Phase

The determination of the basis vectors required to compute a correction displacement is formulated as a *relaxation* process. Initially, all vertices $\mathbf{x}_i, i \notin I(\mathbf{d})$ are fixed at their positions defined by the initial configuration $\mathbf{X}_0$ by setting all $\alpha_{i,k} = 0$. The algorithm then allows for specific values $\alpha_{i,k}$ to take on non-zero values.

For the analysis, we examine the nullspace of the constraints' Jacobian $J_\mathbf{c} \in \mathbb{R}^{m \times 3n}$, where the jth row of $J_\mathbf{c}$ contains the gradient of c_j. The Jacobian can be considered as a map from vertex movement directions in $\mathbb{R}^{3n}$ to variations of the constraint functions $\mathbf{c}$. Thus, given that all constraints are satisfied at $\mathbf{X}_0$, a vertex movement in the nullspace of $J_\mathbf{c}(\mathbf{X}_0)$ does not violate any constraint. In most cases, the vertex displacement $\mathbf{d}$ is not in the nullspace of $J_\mathbf{c}(\mathbf{X}_0)$. Consequently, our goal is to construct a correction displacement $\mathbf{d}'$ such that the total displacement $\mathbf{d} + \mathbf{d}'$ is in the nullspace again. Clearly, in general the above argument is valid for infinitesimal displacements $\mathbf{d}$ and $\mathbf{d}'$ only, while actual displacements have finite extend. However, the nullspace is employed to solve the combinatorial problem of relaxing vertices (respectively basis vectors) only. The actual correction displacement is determined by a nonlinear solver in the solution phase. As discussed in Section 3.4, the set of relaxed vertices obtained by this approach is correct except for a few singular cases which are easy to handle.

An alternative interpretation in analogy to [YYPM11] is derived from the observation that the vertex positions $\mathbf{X}$ of all model instances satisfying $\mathbf{c}(\mathbf{X}) = \mathbf{0}$ define a manifold $\mathcal{M}$ in $\mathbb{R}^{3n}$. The nullspace of the constraint's Jacobian is the tangent space of $\mathcal{M}$ at $\mathbf{X}$, each non-zero coordinate in $\mathbf{d}$ can be considered as an intersection of the tangent space with a $(3n-1)$-dim. hyperplane. Hence, finding the correction displacement $\mathbf{d}'$ is equivalent to finding a point in the intersection space with the least number of non-zero coordinates.

A straight forward formalization of the search for a total displacement $\mathbf{d} + \mathbf{d}'$ in the nullspace of $J_\mathbf{c}$ is to solve the linear system $P(\mathbf{d} + \mathbf{d}') = \mathbf{0}$ with $P := J_\mathbf{c}$. However, as discussed in Section 6, superior results can often be achieved by exploiting the projection onto the nullspace of $J_\mathbf{c}$ instead of using the Jacobian directly. Let $N_J = (\mathbf{n}_1 \dots \mathbf{n}_l) \in \mathbb{R}^{3n \times l}$ be the matrix of nullspace basis vectors of $J_\mathbf{c}(\mathbf{X}_0)$. Then $N_J N_J^T$ is an orthogonal projection from $\mathbb{R}^{3n}$ onto the nullspace, $I - N_J N_J^T$ yields the residual of the projection. To find the correction displacement $\mathbf{d}'$, we set $P := I - N_J N_J^T$ and again solve $P(\mathbf{d} + \mathbf{d}') = \mathbf{0}$. With (1) this expand to

$$\sum_{i \notin I(\mathbf{d})} \sum_{k=1}^{3} \alpha_{i,k} P \mathbf{B}_{i,k} = -P\mathbf{d}, \tag{2}$$

which is a linear system in the unknowns $\alpha_{i,k}$.

Our main goal is to relax as few vertices as possible, i.e., to compute a solution vector with minimal cardinality. While this problem is known to be NP-hard in general [RFP10], research in *Compressed Sensing* has developed several approximate solution strategies with favorable properties. One that suits our needs particularly well is the *Orthogonal Matching Pursuit* (OMP) [TG07]. Its main advantage over more elaborate techniques is that the actual cardinality does not have

Algorithm 1 Orthogonal Matching Pursuit (OMP)

```
procedure OMP(Matrix $A$, right-hand side $\mathbf{y}$)
    $\mathbf{r} \leftarrow \mathbf{y}$                                  // Residual vector
    $\Lambda = \emptyset$                                  // Set of column indices of $A$
    while $||\mathbf{r}||_2 > \varepsilon_1$ do
        $\mathbf{g} \leftarrow A^T \mathbf{r}$                     // Projection onto columns of $A$
        $\Lambda \leftarrow \Lambda \cup \{\arg\max_l(|\mathbf{g}_l|)\}$     // Idx of largest col.
        solve $A|_\Lambda \mathbf{x} = \mathbf{y}$           // $A$ restricted to columns in $\Lambda$
        $\mathbf{r} \leftarrow \mathbf{y} - A\mathbf{x}$                       // Updated residual
    end while
end procedure
```

to be known a-priori. OMP is outlined in Algorithm 1. It is called with the matrix $A = P(\cdots \mathbf{B}_{i,k} \cdots)$ and the right-hand side vector $\mathbf{y} = -P\mathbf{d}$. OMP then iteratively increases the cardinality of the solution by selecting columns of A (that is, by relaxing basis vectors) which best reduce the residual error. Λ is the set of selected columns of A, $A|_\Lambda$ denotes a matrix that contains these columns only. The procedure terminates when enough basis vectors have been relaxed to solve (2) with sufficient precision. In our implementation, we set $\varepsilon_1 = 10^{-6}$. For an under- or well-constrained set of constraints, the procedure is guaranteed to find a suitable set Λ since we made the assumption that a uniform displacement of all vertices satisfies all constraints. A solution with all columns of A selected and a non-vanishing residual indicates that the constraints are contradicting. Please notice that, while the relaxation of individual basis vectors is possible, it is usually (geometrically) more meaningful to relax all three basis vectors of a vertex at once. In Section 5.2 we will discuss a specific strategy for image-based modeling that either relaxes a single or all three basis vectors of a vertex.

The intuition behind the above procedure is to relax the movement direction that best reduces the residual error, i.e., that brings the total displacement $\mathbf{d} + \mathbf{d}'$ closest to the nullspace. Due to the greedy nature of the algorithm (and the NP-hardness of the problem in general), a minimal solution cannot be guaranteed. A simple means to find a smaller set of relaxed vectors is to loop over all (but the last) basis vectors and to try to individually remove them from Λ again.

3.3. Solution Phase

Due to the linearization, the total displacement $\mathbf{d}+\mathbf{d}'$ emerging from the analysis phase often is not a solution of the nonlinear constraint functions, i.e., $\mathbf{c}(\mathbf{X}_0 + \mathbf{d} + \mathbf{d}') \neq \mathbf{0}$. However, the degrees of freedom provided by the relaxed basis vectors allows for the computation of a correct solution (except for rare singular cases discussed in Section 3.4). To find a suitable correction $\mathbf{d}'$, we minimize the objective function

$$E\left(\{\alpha_{i,k} | (i,k) \in \Lambda\}\right) = \sum_{j \in C} c_j^2 \left(\mathbf{X}_0 + \mathbf{d} + \sum \alpha_{i,k} \mathbf{B}_{i,k}\right), \quad (3)$$

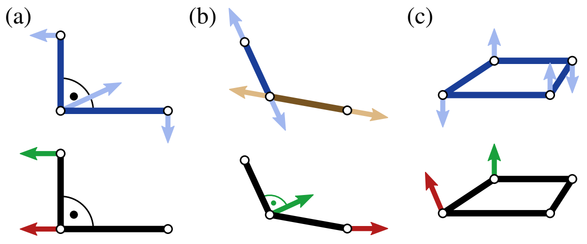

Figure 2: *Analysis of (a) an orthogonality constraint, (b) two edges with fixed lengths, and (c) a planarity constraint. (a) and (b) are 2D, (c) is a 3D example. Top: the arrows depict the gradient directions of the respective constraints, i.e., the directions of maximal constraint violation (orthogonal to the nullspace). Bottom: red arrows depict user-specified displacements* $\mathbf{d}$*, green arrows the computed corrections* $\mathbf{d}'$.

with C being the set of all constraint indices that involve the vertices affected by the displacements $\mathbf{d}$ or $\mathbf{d}'$, and Λ being the set of relaxed basis vectors. We employ the well-established, iterative Levenberg-Marquardt algorithm (cf. [NW06]) to solve (3). Since the number of relaxed vertices (respectively basis vectors) usually is much smaller than the total number of vertices, the Levenberg-Marquardt iteration can be computed in real-time to provide visual feedback during interactive editing operations.

The set of relaxed vertices, combined with all affected constraint functions, often yields a slightly under-constrained system, in particular when the basis vector selection strategy of Section 5.2 is employed. That is, the number of relaxed basis vectors is slightly larger than the degrees of freedom of the involved constraint functions. To fix these redundant degrees of freedom, we add a penalty term for each relaxed basis vector that drags the respective vertex back to its original position. More formally, let $\mathbf{x}_{i,0} \in \mathbb{R}^3$ be the position of vertex i in the initial configuration $\mathbf{X}_0$. We add a constraint function $\omega(\mathbf{x}_i - \mathbf{x}_{i,0})^T \mathbf{b}_{i,k}$, with $\mathbf{b}_{i,k}$ being the relaxed basis vector and ω a small weight. In our implementation, we set $\omega = 10^{-3}$. This approach has the advantage that each relaxed vertex, independent of the actual degrees of freedom, stays as close to its original position as possible. The weight ω is chosen small enough such that all constraints in $\mathbf{c}$ can be satisfied with sufficient numerical precision.

3.4. Discussion and Solution of Singular Cases

As illustrated in Figure 2, posing the constraint analysis as a linear problem by considering infinitesimal displacements works well in practice. Although the initial correction displacements usually are not a solution of the nonlinear constraints $\mathbf{c}$, in most situations they correctly determine which vertices need to be relaxed.

In two particular cases, however, the consideration of infinitesimal displacements may not yield enough relaxed vertices. The first case is caused by a user-specified displace-

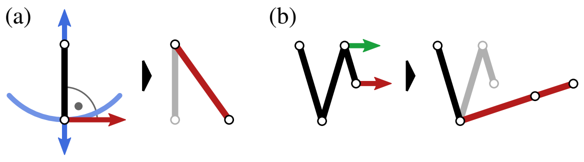

Figure 3: *Problematic cases caused by the consideration of infinitesimal displacements. Constraints of red edges are not satisfied. (a) For a length-constrained edge, displacements that lie exactly in the linearization of the curved nullspace yield too few relaxed vertices. (b) Combination of three length-constrained edges. While the relaxation of the indicated vertex is correct for an infinitesimal displacement, the length constraints cannot be satisfied for a very far actual displacement. Both cases are solved by a simple extension of the 2-phase process.*

ment (or a correction displacement) that happens to *exactly* lie in the linearization of an actually curved (e.g., quadratic) nullspace (cf. Figure 3a). Notice, however, that this happens almost never in practice: in the example in Figure 3a, the displacement has to be exactly horizontal while the edge is exactly vertical *without being constrained as such*. If the edge was constrained to be vertical, the additional constraint would cause more vertices to be relaxed. In fact, to provoke such cases in our experiments, we had to artificially construct them, e.g. by first adding and subsequently removing an orthogonality constraint from a length-constrained edge.

The second case is caused by the fact that considering infinitesimal displacements does not take possible length restrictions into account. This happens, for instance, when two or more length-constrained edges are combined as illustrated in Figure 3b. The linear analysis correctly relaxes the vertex next to the vertex moved by the user. However, once the displacement becomes too large during the interaction, the length constraints cannot be satisfied anymore.

Both above cases are easily detectable by a non-zero residual of the nonlinear solution algorithm and allow for a simple solution that seamlessly integrates with the linear constraint analysis approach. In both cases the (linearized) nullspace is too "permissive", i.e., allows displacements which are actually not feasible. Hence, this problem can be solved by reducing the degrees of freedom of the nullspace by adding more constraints and then re-run the linear relaxation process. In our implementation, we add *stiffening* constraints to affected vertices. More specifically, for two vertices $\mathbf{x}_{i_1}$, $\mathbf{x}_{i_2}$ we add a constraint

$$c_{\text{stiff}}(\mathbf{x}_{i_1},\mathbf{x}_{i_2}) \;=\; (\mathbf{x}_{i_1}-\mathbf{x}_{i_2})-(\mathbf{x}_{0,i_1}-\mathbf{x}_{0,i_2}) \;\overset{!}{=}\; \mathbf{0},$$

that enforces the vertices to remain in the same relative configuration as in $\mathbf{X}_0$. Stiffening is applied to all vertices of the constraint with the largest residual error of the linearly determined displacement, i.e., to all vertices affected by $c_{\max} = \arg\max_{c_j}(|c_j(\mathbf{X}_0+\mathbf{d}+\mathbf{d}')|)$. Notice that the stiffening constraints are only considered in the analysis phase to relax more vertices, but are not used in the solution phase. The resulting algorithm that interleaves the linear analysis and the nonlinear solution phases is outlined in Algorithm 2. The threshold ε_2 to detect unsatisfied constraints clearly depends on the actual implementation of the constraints. In our modeling system, we formulate all constraints in terms of Euclidean distances and set $\varepsilon_2 = 10^{-3}$, i.e., all constraints have to be satisfied with a precision of at least 1mm.

Algorithm 2 Combination of linear constraint analysis with nonlinear solution.

```
input: editing displacement $\mathbf{d}$
$\Lambda = \emptyset$
repeat
    run nonlinear solver with $\mathbf{d}$, $\Lambda$; compute residual $\mathbf{r}$
    if $||\mathbf{r}||_\infty > \varepsilon_2$ then
        if $\Lambda \neq \emptyset$ then
            add stiffening constraint
        end if
        run linear analysis on $\mathbf{d}$, extend $\Lambda$
    end if
until nonlinear residual $||\mathbf{r}||_\infty \leq \varepsilon_2$
```

4. Constraint Initialization

To initialize new constraints we basically perform the same procedure as for the regular editing (linear analysis and nonlinear solution), with a slight modification of the linear system used in the OMP algorithm. We separate all constraints into a set of satisfied constraints $\mathbf{c}_{\text{sat}}(\mathbf{X}_0) = \mathbf{0}$ and a set of new, unsatisfied constraints $\mathbf{c}_{\text{new}}(\mathbf{X}_0) \neq \mathbf{0}$. As before, the nullspace and the projection $P = I - N_J N_J^T$ is computed from the satisfied constraints $\mathbf{c}_{\text{sat}}$ only. Since no explicit displacement $\mathbf{d}$ is given, we employ the Taylor expansion

$$\mathbf{c}_{\text{new}}(\mathbf{X}_0+\mathbf{d}') \;\approx\; \mathbf{c}_{\text{new}}(\mathbf{X}_0) + J_{\text{new}}(\mathbf{X}_0)\mathbf{d}' \;\overset{!}{=}\; \mathbf{0}$$

to construct a suitable right-hand side of the linear system. That is, to compute a correction displacement $\mathbf{d}'$ that lies in the nullspace of $\mathbf{c}_{\text{sat}}$ and in addition fulfills the above Taylor approximation, the input to the OMP algorithm is

$$A = \begin{pmatrix} P \\ J_{\text{new}} \end{pmatrix} (\cdots \mathbf{B}_{i,k} \cdots), \qquad \mathbf{y} = \begin{pmatrix} \mathbf{0} \\ -\mathbf{c}_{\text{new}}(\mathbf{X}_0) \end{pmatrix}.$$

Notice that, similar to the regular analysis phase, in rare cases the relaxed basis vectors do not allow for the nonlinear computation of a correct solution. We hence employ Algorithm 2 for the initialization of new constraints as well.

5. Image-Based Modeling System

In this section we discuss the integration of our constraint resolution approach into a prototypical image-based 3D modeling system. For more details on the system itself, please refer to [Hab12] and the supplemental video.

5.1. System Overview

As for most image-based modeling systems, the main source of information is a set of calibrated input images. The interface provides one or more views of the current model, rendered over the images that show the object to be reconstructed. The actual editing is performed by dragging vertices, edges, and faces or by adding constraints. Similar to other image-based modeling systems, we exploit epipolar geometry (cf. [HZ03]) to simplify the editing process. During editing, the user declares one view as *reference*. In this view, elements of the 3D model are allowed to be moved according to simple rules (vertices move on adjacent edges, edges move on adjacent facet planes, etc.) In contrast, in all other (non-reference) views, vertices are restricted to move on *viewing rays* through the reference camera center. This enables the precise 3D positioning of a vertex by (at most) a 1D / 2D operation in the reference image and a subsequent 1D adjustment in any other view.

Due to ubiquitous alignments, in the specific application of architectural modeling from aerial images large parts of a building can be implicitly generated by extrusion operations. Consequently, in many cases it is sufficient to construct a sparse set of polygons to define the roof shape and possibly geometric detail on the facades, cf. Figure 5. For the generation of building surfaces from a sparse set of polygons we employ a volumetric CSG-like approach. The creation of new facets is based on a sketch-based interface enabling the user to draw the shape of a planar polygon in one of the images. Automatic image fitting procedures are employed to initialize the supporting planes of new polygons and to precisely align existing model components with the underlying images.

5.2. Integration with Constrained Modeling

Basis Construction While the canonical basis works well in many modeling scenarios, Figure 4 illustrates that image-based modeling with the concept of epipolar geometry requires adjusted basis vectors. For a vertex $\mathbf{x}_i$ and a reference camera center $\mathbf{p} \in \mathbb{R}^3$ we therefore construct the first basis vector as $\mathbf{b}_{i,1} = \mathbf{x}_i - \mathbf{p}$, and the two remaining vectors as $\mathbf{b}_{i,2} = \mathbf{b}_{i,1} \times \mathbf{o}$, $\mathbf{b}_{i,3} = \mathbf{b}_{i,1} \times \mathbf{b}_{i,2}$, where $\mathbf{o}$ is an arbitrary direction not parallel to $\mathbf{b}_{i,1}$. Clearly, relaxing only $\mathbf{b}_{i,1}$ and keeping $\mathbf{b}_{i,2}$ and $\mathbf{b}_{i,3}$ constrained in the analysis phase enables the vertex $\mathbf{x}_i$ to move on its respective viewing ray.

Basis Vector Relaxation Strategy The above construction implies a simple basis vector relaxation strategy for image-based modeling. In each relaxation step, we only consider two possible cases, either $\{\mathbf{b}_{i,1}\}$ is relaxed alone, or all three basis vectors $\{\mathbf{b}_{i,1}, \mathbf{b}_{i,2}, \mathbf{b}_{i,3}\}$ of a vertex are relaxed at once. In the actual implementation (cf. Algorithm 1), after finding the basis vector with the largest dot product with the residual vector, we potentially add two more indices to the set Λ, depending on which basis vector has been selected in the first place.

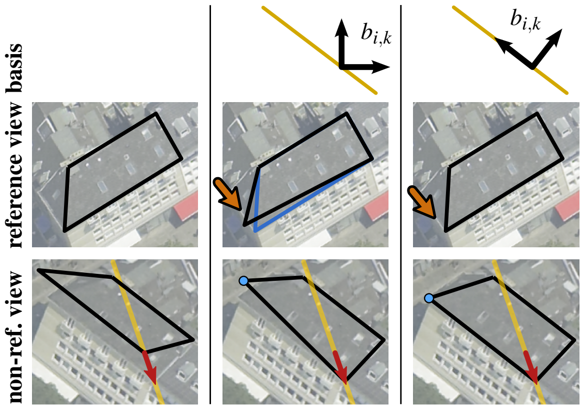

Figure 4: *Epipolar modeling with and without basis vectors aligned to vertex viewing rays. Left: a polygon correctly aligned with the reference view. Center: moving the indicated vertex on its viewing ray triggers the relaxation of all three canonical basis vectors of the blue vertex, resulting in a destroyed alignment. Right: in case of basis vectors aligned with their respective viewing rays, it is sufficient to relax only a single basis vector of the blue vertex, resulting in the preservation of the alignment in the reference view.*

Interactive Editing Thanks to the interleaved application of the linear analysis and the nonlinear solution discussed in Section 3.4, it is an easy task to extend the algorithm to a fully interactive editing system. The basic idea is to initialize $\Lambda = \emptyset$ in Algorithm 2 only when an interactive editing operation is started. When the displacement $\mathbf{d}$ is updated by the user dragging a model element, we call Algorithm 2 again but reuse the set of previously relaxed basis vectors. Consequently, in most cases the nonlinear solver quickly converges to an updated solution. If the residual of any constraint is larger than ε_2, we distinguish two cases. In case the *direction* of $\mathbf{d}$ has not changed (w.r.t. a small tolerance) since the last run of the linear analysis, we add stiffening constraints in order to trigger the relaxation of more vertices. If the direction has changed, we simply run the analysis again on the new direction.

Implemented Constraint Types In our modeling system, all facets are constrained to be planar by default. In addition, we have implemented the following set of constraints:

- Plane / edge horizontal,
- plane / edge vertical,
- pair of planes / edges parallel,
- pair of planes / edges orthogonal,
- vertices / planes coplanar,
- vertices / edges collinear,
- fixed vertex distance,
- two vertices symmetric with a vertical symmetry plane,
- “cloned” groups of vertices with identical shape.

Notice that in this list “planes” can either be the supporting planes of facets (e.g., when snapping vertices to a facet),

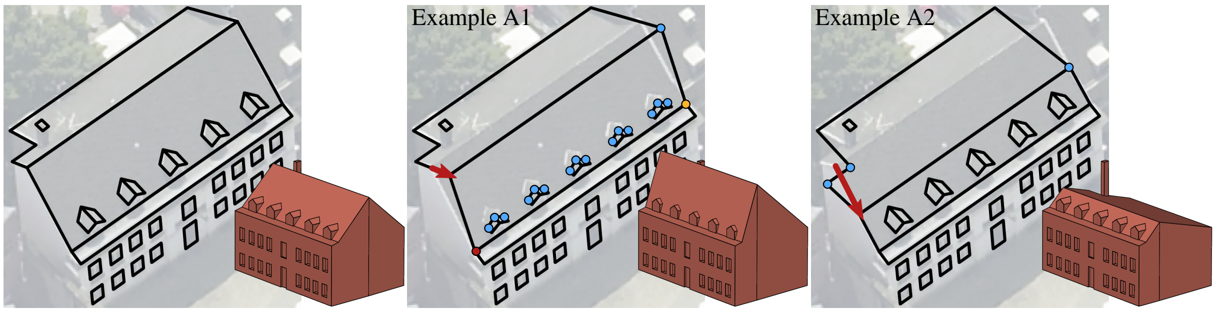

Figure 5: *Modeling of a roof structure with dormers. Left: original configuration. Center: editing operation such that the base-plane of the dormers changes its orientation (example A1). Right: the dormers' base plane does not change (example A2). Blue vertices are relaxed in the analysis phase and automatically updated by the editing system. Please see text for more details.*

but also more general planes (e.g., a vertical plane spanned by a (non-vertical) edge). Clearly, this list is by no means exhaustive and other modeling tasks might require different constraints. A major advantage of our general constraint analysis and solution scheme is that it can easily be extended by additional constraints (for instance, rotational symmetry). For the task of architectural modeling in aerial images, however, we have found that this set of constraints is sufficient.

6. Results and Discussion

We have performed several experiments to demonstrate the practical applicability of our approach. Table 1 lists the details. Figure 5 (examples A1 and A2) demonstrates two editing operations on a building roof with several dormers. The dormers' base-vertices are constrained to be coplanar with the roof. Thus, in case A1 these vertices are required to be updated in order to satisfy all constraints. In example A2, the plane to which the dormers are attached does not change. This situation is correctly recognized and only the required vertices are updated. Examples B1 and B2 (cf. Figure 6) show two editing operations on a snake-like roof structure. While B1 is performed in the reference view, the operation of B2 is performed in a non-reference view. Again, in both cases the correct minimum cardinality solution is found.

In the spring example S depicted in Figure 7 all edges are constrained to have fixed lengths. This case requires the incremental relaxation of additional vertices by Algorithm 2 during the interactive dragging. Our algorithm correctly relaxes one vertex after the other, enabling the structure to unfold. In Table 1 the respective row contains the values of the last analysis step only. As all previous steps work on smaller input data sets, their running times are even shorter. Notice that such combinations of length-constrained edges are difficult for propagate-and-fix solution strategies such as employed in iWires [GSMCO09]: When propagating the modification from vertex to vertex, each vertex has (in the 2D case) one degree of freedom. However, by fixing these degrees of freedom with only local knowledge, it cannot be guaranteed that e.g. a desired target position is reached.

Ex.	#V	#C	#N	#rx	#fx	Alg2	T_{Ba}	T_{OMP}	T_{Up}
A1	116	812	73	51	3	1	227ms	16ms	5ms
A2	116	812	73	9	0	1	227ms	1.2ms	1.1ms
B1	45	117	62	90	6	1	16ms	34ms	6ms
B2	45	117	62	22	0	1	16ms	1.9ms	5ms
G5	100	395	41	24	0	1	134ms	3.1ms	1.2ms
G7	196	819	57	36	0	1	1.41s	14.7ms	3.2ms
G10	400	1740	81	54	0	1	13s	85ms	5.3ms
S	11	32	11	3	0	7	1.5ms	1.2ms	4ms

Table 1: *Details of the experiments. "Ex" denotes a particular example, #V the number of vertices, #C number of constraint functions, #N the dimension of the nullspace. The columns #rx and #fx contain the numbers of relaxed and subsequently re-fixed basis vectors. "Alg2" denotes iterations in Algorithm 2, the last three columns contain the times for the basis construction, the linear analysis, and the nonlinear solution. All experiments were run on an Intel Core i7 920.*

All experiments in Table 1 employ the nullspace projection in (2) rather than directly using the constraints' Jacobian. This choice influences the two main computation steps of the analysis phase, the construction of the transformed basis $P\mathbf{B}_{i,k}$ (T_{Ba} in Table 1) and the OMP algorithm (T_{OMP}). While the basis construction usually is faster for the Jacobian-only approach (e.g. A1: T_{Ba}=110ms, B1: T_{Ba}=4.6ms), for two reasons the OMP algorithm performs better with the nullspace projection. First, the size of the matrix P usually is smaller, leading to faster solutions of the linear system in Algorithm 1. Second, the greedy relaxation often is more effective. The orange vertices in examples A1 and B1 depict cases in which the respective vertices have been relaxed during the OMP iteration and then have been fixed again in the final pass over all relaxed basis vectors. Notice that these are the only such cases for the nullspace projection. When using the Jacobian in (2) directly, in example A1 the algorithm relaxes #rx=75 and later fixes #fx=27 basis vectors, and thus takes T_{OMP}=72ms. In example B1, #rx=117, #fx=33, T_{OMP}=58ms. Hence, due to the better performance of the greedy OMP algorithm, the nullspace pro-

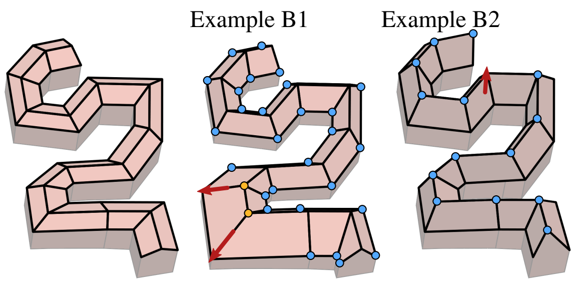

Figure 6: *Editing of a snake-like roof structure. In each quad, the upper and lower edges are constrained to be parallel. Furthermore, all ridge vertices as well as all lower vertices are aligned horizontally. Center: moving a lower quad-edge downwards in its supporting plane results in the relaxation of* all *lower vertices (blue). Right: moving one of the ridge vertices along its viewing ray in a non-reference view yields the relaxation of only the first basis vectors (which are aligned with the respective viewing rays) of all other ridge vertices (blue).*

jection is the method of choice for interactive systems. The main drawback of the nullspace projection is the requirement of computing the nullspace basis. In our current implementation, the basis is computed by a standard singular value decomposition (SVD) and thus constitutes the main bottleneck especially for the larger examples. However, notice that the transformed basis does not depend on the actual editing operation. The basis for editing operation k can therefore be constructed immediately after operation $k-1$, thereby effectively hiding its computation from the user. Furthermore, the Jacobian $J_\mathbf{c}$ has a sparse structure which can be exploited during the nullspace computation [GT08].

Examples G5, G7, and G10 are based on the grid structure depicted in Figure 8(a). G5 has been performed on a grid of 5×5 quads, G7 on 7×7 quads, and G10 on 10×10 quads. Notice that, as in all examples, all vertices of the quads are defined in $\mathbb{R}^3$. All quads are constrained to be coplanar, all neighboring edges (depicted by arrows in the figure) are constrained to be collinear. The results in Table 1 correspond to the operation of moving a vertex along an adjacent edge. These artificial examples clearly are extremal cases. However, they demonstrate that the OMP algorithm and the nonlinear solution phase ($\mathrm{T}_{\mathrm{OMP}}$ and T_{Up} in Table 1) can be performed in real-time even for very large constraint systems.

Figure 8(b) compares the behavior of our approach to propagate-and-fix strategies (like iWires) on the same grid structure with collinear edges. The red vertex is assumed to be fixed. Our approach detects that the blue vertices have to be relaxed, and then computes suitable positions in the nonlinear solution phase. In contrast, propagate-and-fix strategies determine *final* vertex positions for each individual quad. When fixing the first quad (affected by the editing operation), the algorithm is not aware of the fixed vertex at the end of the propagation chain. It is thus not able to determine

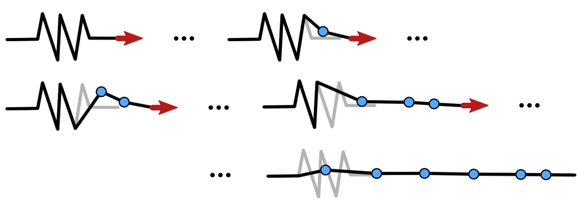

Figure 7: *Unfolding a spring-like structure. All edges are length-constrained. Due to the incremental relaxation of vertices in Algorithm 2, the structure unfolds as expected.*

the correct position of the lower left corner in the first quad and the propagation strategy gets stuck in an unsolvable situation eventually.

Column “Alg2” in Table 1 demonstrates that in all experiments except the spring configuration in example S a single iteration of Algorithm 2 was sufficient. Thus, in practice the linear analysis finds suitable sets of vertices, stiffening constraints are required to handle rare cases only.

Notice that the minimal set of vertices which are required to be updated in general is not unique. Furthermore, in certain situations editing operations with more relaxed vertices “feel” more natural than the minimal solution. This is the case in example A1: Each dormer has three vertices in the roof plane. The minimal solution (cf. the supplemental video) rotates the roof plane about the axis through the fixed lower pairs of vertices, only the five upper vertices are relaxed. As a simple means to guide the analysis algorithm, our interface allows for the manual exclusion of vertices from the relaxation process. In particular, the red vertex in example A1 has been marked for exclusion to achieve a more natural editing operation with more relaxed vertices.

In its current formulation, the choice of vertices affected by the correction displacement $\mathbf{d}'$ is dominated by a combinatorial process rather than by the actual geometry of the model. Due to our goal of altering as few auxiliary vertices as possible, geometric primitives with low numbers of vertices may be preferred over primitives with many vertices in the relaxation process. In our interactive modeling system we consider this behavior a feature: primitives with many vertices usually require more editing effort and thus should be changed with lower probability. Cases in which this is not desired could, for instance, be handled with an extended relaxation strategy. Simultaneously relaxing groups of vertices belonging to the same primitive would reduce the strong combinatorial influence of the analysis procedure.

Computing a correction displacement $\mathbf{d}'$ may be achievable by alternative means, e.g. by optimizing a displacement of all vertices while imposing a l_1-norm regularizer. However, we chose the 2-phase procedure presented in Section 3 over such approaches for two main reasons: Regularizing energies are usually unable to completely prevent undesired vertex movements. That is, while the majority of $\mathbf{d}'$s “en-

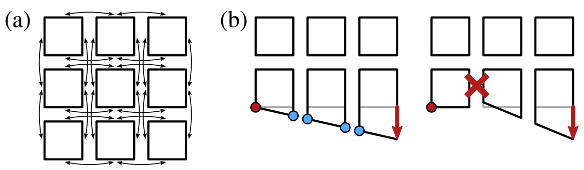

Figure 8: *(a) 3×3 example of grid-structure used in examples G5, G7, and G10. (b) Behavior of our approach (left) and a propagate-and-fix strategy (right). See text for details.*

ergy" is distributed to a few vertices only, many other vertices move slightly and thereby compromise previously generated alignments. Furthermore, such a formulation would require the optimization of all vertices in parallel. Even for only moderately complex models this would contradict our goal of real-time interactive modeling operations.

In contrast to traditional constraint analysis approaches, our algorithm is based on several (well-established) *numerical* techniques with two simple thresholds. Due to inevitable numerical inaccuracies, these thresholds cannot be set arbitrarily low. Hence, all constraints are satisfied up to a threshold only (a maximal deviation of 1mm in our implementation) which might not be acceptable for certain applications.

7. Conclusion

We presented a novel approach to analyze and solve nonlinear constraints for geometric modeling. Given an explicit editing operation, the main problem we are considering is to find an as small as possible set of auxiliary vertices such that, after these vertices have been adjusted, all constraints are satisfied again. Our analysis strategy is based on linearized constraint functions which enables a very efficient algorithm and allows for the application of well-established nonlinear solution techniques to compute actual vertex positions. At the example of several real-world modeling problems, we have demonstrated that this approach works well in practice and can easily be integrated into interactive 3D modeling systems with real-time visual user feedback.

Acknowledgment: This project was funded by the DFG Cluster of Excellence *UMIC* (DFG EXC 89) and the DFG Graduate School *AICES* (DFG GSC 111).

References

[BB04] BAERLOCHER P., BOULIC R.: An inverse kinematic architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer 20* (2004), 402–417. 2, 3

[CLDD09] CABRAL M., LEFBVRE S., DACHSBACHER C., DRETTAKIS G.: Structure-preserving reshape for textured architectural scenes. In *Eurographics* (2009). 1, 3

[DDEK12] DAVENPORT M., DUARTE M., ELDAR Y., KUTYNIOK G.: Introduction to compressed sensing. In *Compressed Sensing: Theory and Applications*, Eldar Y. C., Kutyniok G., (Eds.). Cambridge University Press, 2012. 2, 3

[DTM96] DEBEVEC P. E., TAYLOR C. J., MALIK J.: Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH* (1996). 3

[DW97] DEO A., WALKER I.: Minimum effort inverse kinematics for redundant manipulators. *IEEE Trans. on Robotics and Automation 13*, 5 (1997), 767–775. 3

[FASR08] FREIXAS M., ARINYO R. J., SOTO-RIERA A.: A constraint-based dynamic geometry system. In *Proc. of ACM SPM* (2008), pp. 37–46. 1, 3

[FF09] FARENZENA M., FUSIELLO A.: Stabilizing 3d modeling with geometric constraints propagation. *Computer Vision and Image Understanding 113*, 11 (2009), 1147–1157. 2, 3

[GSMCO09] GAL R., SORKINE O., MITRA N., COHEN-OR D.: iWIRES: An analyze-and-edit approach to shape manipulation. *SIGGRAPH* (2009). 1, 2, 8

[GT08] GOTSMAN C., TOLEDO S.: On the computation of null spaces of sparse rectangular matrices. *SIAM J. Matrix Anal. Appl. 30* (2008), 445–463. 9

[Hab12] HABBECKE M.: *Interactive Image-Based 3D Reconstruction Techniques for Application Scenarios at Different Scales*. PhD thesis, RWTH Aachen Univeristy, 2012. 6

[HL01] HOFFMAN C. M., LOMONOSOV A.: Decomposition plans for geometric constraint systems, part i. *J. Symbolic Computation 31* (2001), 367–408. 1, 3

[HZ03] HARTLEY R., ZISSERMAN A.: *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press, 2003. 7

[JTNM06] JERMANN C., TROMBETTONI G., NEVEU B., MATHIS P.: Decomposition of geometric constraint systems: a survey. *IJCGA 16*, 5-6 (2006), 379–414. 1, 3

[KSSCO08] KRAEVOY V., SHEFFER A., SHAMIR A., COHEN-OR D.: Non-homogeneous resizing of complex models. In *Proc. of SIGGRAPH Asia* (2008). 1, 3

[LWC*11] LI Y., WU X., CHRYSATHOU Y., SHARF A., COHEN-OR D., MITRA N. J.: Globfit: Consistently fitting primitives by discovering global relations. In *SIGGRAPH* (2011). 3

[NW06] NOCEDAL J., WRIGHT S.: *Numerical Optimization*, 2nd ed. Springer, 2006. 5

[RFP10] RECHT B., FAZEL M., PARRILO P. A.: Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. 471–501. 2, 4

[SSS*08] SINHA S. N., STEEDLY D., SZELISKI R., AGRAWALA M., POLLEFEYS M.: Interactive 3d architectural modeling from unordered photo collections. In *SIGGRAPH Asia* (2008). 3

[TG07] TROPP J. A., GILBERT A. C.: Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Trans. Inform. Theory 53*, 12 (2007), 4655–4666. 4

[TW06] TROMBETTONI G., WILCZKOWIAK M.: Gpdof - a fast algorithm to decompose under-constrained geometric constraint systems: Application to 3d modeling. *Int. J. Comput. Geometry Appl. 16*, 5–6 (2006), 479–512. 2, 3

[vdHDT*07] V. D. HENGEL A., DICK A. R., THORMÄHLEN T., WARD B., TORR P. H. S.: Videotrace: rapid interactive scene modelling from video. *ACM Trans. Graph. 26*, 3 (2007). 3

[XWY*09] XU W., WANG J., YIN K., ZHOU K., VAN DE PANNE M., CHEN F., GUO B.: Joint-aware manipulation of deformable models. *ACM TOG 28* (July 2009). 1, 3

[YYPM11] YANG Y.-L., YANG Y.-J., POTTMANN H., MITRA N. J.: Shape space exploration of constrained meshes. In *SIGGRAPH Asia* (2011). 3, 4

[ZFCO*11] ZHENG Y., FU H., COHEN-OR D., AU O. K.-C., TAI C.-L.: Component-wise controllers for structure-preserving shape manipulation. In *Eurographics* (2011). 1, 2