Computer-Aided Design 61 (2015) 42–57

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

A fully geometric approach for interactive constraint-based structural equilibrium design

Corentin Fivet *, Denis Zastavni

LOCI Faculty of Architecture, Architectural Engineering, Urbanism - UCLouvain, Belgium

HIGHLIGHTS

- The initial definition of structural behaviors is of primary importance when seeking material efficiency.
- A CAD approach is introduced allowing the user to build any plane static equilibrium interactively and graphically.
- All the design freedoms of the structural problem are permanently contained within dynamic graphical regions of positions.
- Techniques benefiting from fully geometric abstraction offer some computational simplifications and new capabilities.
- As a result, this approach frees the designer from the usual hierarchical and chronological structural design processes.

ARTICLE INFO

Keywords:
Structural design
Static equilibrium
Coordinate-free geometric solving
Inequality constraints
Strut-and-tie model
Graphic statics

ABSTRACT

This paper introduces computational techniques to support architects and structural designers in the shaping of strut-and-tie networks in static equilibrium. Taking full advantage of geometry, these techniques build on the reciprocal diagrams of graphic statics and enhance the interactive handling of them with two devices: (1) nodes – considered as the only variables – are constrained within Boolean combinations of graphic regions, and (2) the user modifies the diagrams by means of successive operations whose geometric properties do not at any time jeopardize the static equilibrium. This constructive approach enables useful design-oriented capabilities: a graphical control of multiple solutions, the direct switching of the dependencies hierarchy, the execution of dynamic conditional statements using static constraints, the computation of interdependencies, and coordinate-free methods for ensuring consistency between certain continuums of solutions. The paper describes a computer implementation of these capabilities.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

1.1. Context

Today's structural design practices are usually isolated in a process that sizes the structural parts after an analysis of a predetermined shape. Although satisfactory for most common routines, this process does not encourage a direct return to initial choices and hence does not favor adequate control of the structural behavior being shaped. If the structural behavior is not mastered throughout the process, the result consists of a collection of uncoordinated local arrangements without any kind of global assessment. The structure is therefore likely to have a lack of efficiency and durability or more generally a lack of internal consistency: structural parts might be over-dimensioned or superfluous, the structure might not fit with initial design intentions and, in the worst case scenario, the process might lead to a mechanically unsafe structure.

1.2. Objectives

Approaches exist to master the initial definition of the structural behavior. They generally consist of design-oriented uses of simplifying assumptions, problem reductions guaranteeing permanent control and extensive use of graphical methods and geometry. If executed properly, these approaches may render common computational analysis superfluous and hence reverse the classical process of structural design since the features of the structural behavior are a preliminary and not a result of the geometry and its structural analysis.

The computational techniques described in this paper aim to support these approaches in a more interactive way than existing

* Corresponding author. Tel.: +32 10 47 23 45.
E-mail addresses: corentin.fivet@uclouvain.be (C. Fivet), denis.zastavni@uclouvain.be (D. Zastavni).

http://dx.doi.org/10.1016/j.cad.2014.04.001
0010-4485/© 2014 Elsevier Ltd. All rights reserved.

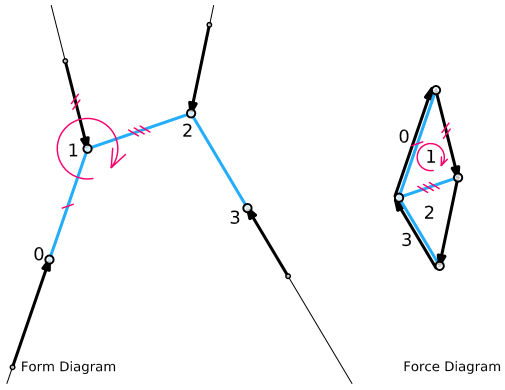

Fig. 1. Forces applied on point 1 in the form diagram are read in the same order as in the force diagram.

structural design tools. They are not intended to replace structural analysis tools, but rather to precede them. In order to meet the need for abstraction, simplicity, speed and interactivity that are inherent to the initial stages of creative design, the proposed environment only deals with static equilibrium. Its main role is to allow the designer to shape strut-and-tie networks and their inner forces graphically and simultaneously – *i.e.* using interactive graphic statics diagrams – by means of geometric operations (1) that guarantee static equilibrium at each step of the process and (2) from which the display of the available design freedoms is inferred.

1.3. Ranges of applications

The range of structures that can be explored with graphic statics diagrams is quite large but is limited to those that can be modeled with a finite number of forces, struts and ties. They include the design and analysis of any reticular systems regardless of whether they are isostatic, indeterminate, articulated, pre-stressed or self-constrained; the design and analysis of any regular or irregular beam subjected to bending; and the sketch of discontinuous stress fields (*e.g.* within concrete shear walls [1]) or lines of thrusts (*e.g.* within masonry arches [2]) — the use of strut-and-tie models is actually ideally suited to working with the lower-bound theorem of plastic theory [3,4].

This paper only considers planar systems although the techniques presented can also be used to control spatial systems by means of fixed projected planes.

1.4. Organization of the paper

The remainder of this paper is organized into six sections. Section 2 highlights the overall functioning and reviews past contributions. Section 3 then specifies the data to be processed. Section 4 describes the various native operations that the user can invoke in order to constrain, shape and modify strut-and-tie networks in static equilibrium. It also explains how the update of position is performed and to what extent the environment may ease coordinate-free techniques to ensure the existence of consistent solutions. Section 5 introduces the role of higher procedures and specifies some of them. Lastly Section 6 screens one example of application, before Section 7 discusses the paper.

2. General overview and state of the art

2.1. Graphic statics

Graphic statics is a theory stretching back 150 years [5,6] that is used to represent and compute the geometry of a structure and its inner forces in a fully graphic-based approach. The geometry of the structure is drawn in a diagram called the form diagram, while its inner forces are drawn in another diagram called the force diagram in which a distance between two points measures a magnitude of force — *i.e.* in newtons or kilograms. These two diagrams must satisfy the following rules:

(1) In order for any sub-structure – including any isolated node and the overall structure itself – in the form diagram to be in static equilibrium, the sum of the forces applied to it – including those applied by a strut or a tie – must be zero. This means that the vector description of these forces in the force diagram has to be a closed polygon. If this condition is satisfied for all the nodes of the strut-and-tie structure, the sum of moments is always zero and the rotational equilibrium does not have to be checked *per se*.
(2) In order for each force, strut and tie to be drawn only once in each diagram, the forces applied to any sub-structure must be read in the form diagram on a cycle that (a) is always read either clockwise or anti-clockwise and (b) is identical to the order described by the corresponding closed polygon in the force diagram (Fig. 1).

Graphic statics presents considerable benefits for computer-aided structural design. From the point of view of implementation, strut-and-tie networks and their static equilibrium can be fully described in a geometric and homogeneous environment the data of which are only positions of nodes on two planes. From the point of view of the user, it allows a visual description of the force distribution, its geometries and magnitudes.

Some software implementations of interactive graphic statics diagrams have already been developed for education purposes [7,8]. They are however only capable of dynamic displacements of nodes on preassembled diagrams. For this reason, the following two paragraphs introduce two new concepts that enhance the interactive use of graphic statics.

2.2. Graphical domains of solutions

The first device consists of assigning to each node of both diagrams a graphical region representing all its positions that solve the desired structural problem. If a node stays within its graphical domain, then the user is assured that the structure satisfies all the geometric and physical conditions applied. As an illustration, the strut-and-tie network presented by the form diagram in Fig. 2 stays within the shaded area as long as the highlighted point in the force diagram remains within the shaded triangle, limiting its movement and hence the maximal magnitude of the three convergent rods. These domains are the result of the application of constraints by the user or are computed by the software in order to ensure consistency with all the constraints applied on the other points.

The constraint-based geometric solving techniques presented in this paper are logic-based and constructive (see [9–11] for related reviews). Domains of solutions are solved incrementally on graphical regions. In this regard, the approach can be related to the work of [12]. However, it stands out for two main reasons.

The first is that it extends the classical scope of compass-and-straightedge constructions by considering graphical inequalities and relative directions as fundamental constraints, leading to the possible formulation of infinite domains of solutions. Current geometric software usually offers very little support for geometric constructions allowing multiple solutions [13,14], *i.e.* allowing points to be constrained on graphical inequalities (including half-planes) and on unions and negations of regions defined by other points. Software that enables these devices [15] does not ensure consistency between constraints (*i.e.* regions may become empty).

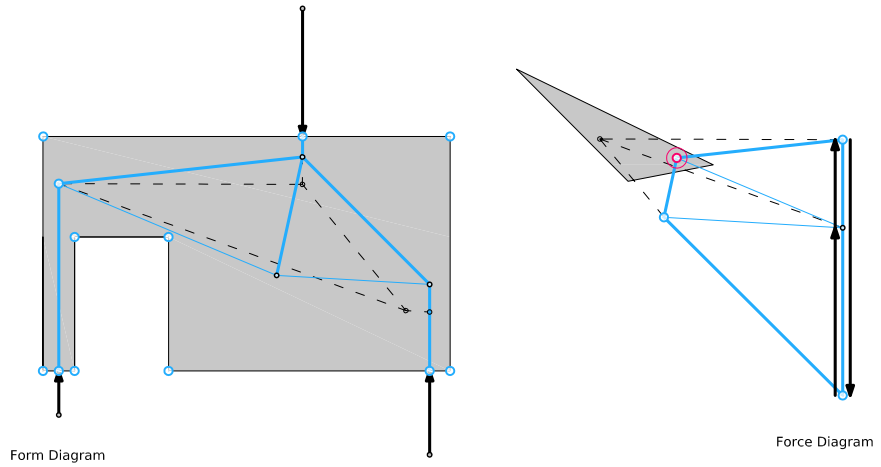

Fig. 2. The strut-and-tie network remains within the shaded area on the form diagram (*e.g.* the boundary of a reinforced concrete shear-wall) as long as the highlighted point in the force diagram remains within the shaded triangle limiting its dragging. The dashed lines represent another solution.

The second is that all the constraints here only take positions as parameters, whereas other solvers deal with constraints and parameters of various kinds (distances, angles, parallelism, tangency, proportionality etc.). Although this second feature does not imply any restriction of application – *e.g.* an angle can still be defined with three positions – it does allow some helpful simplifications in the problem terms that enhance solver capabilities, such as inverting the dependencies hierarchy between two relative points without requiring the construction to be rebuilt, guaranteeing some consistency by means of coordinate-free propagation and processing constraints that are interdependent. These features are developed further in Sections 4 and 5.

2.3. Operations of equilibrium

The second device consists of simultaneously constructing and modifying both diagrams with geometric operations that transform a given strut-and-tie network in static equilibrium into another one. This single feature brings new interactivity to graphic statics diagrams. Indeed, current and classical graphic statics methods [16] require the force diagram to be built with successive parallels copied from the form diagram, meaning that global equilibrium is only reached at the end of the construction process and that this construction must be re-started everytime the typology of the form diagram changes. Here, equilibrium exists from the start, the modifications affect both diagrams simultaneously and it does not require their complete reconstruction.

As an illustration, Fig. 3 describes the states before the combination of two sub-equilibriums into a new one (Fig. 5). Fig. 4 is an intermediate step where points are dragged or geometrically constrained by the user in order to make paired forces opposite, equal and parallel. Since the change from one state of equilibrium to the other is governed by geometric reasoning alone, it does not require a matrix computation of equilibrium from a temporary non-equilibrated state.

3. Geometric data

Five types of data are used: points, primitive constraints, Boolean constraints, forces and rods. Each of them is defined in the following sub-sections. The term "geometric constraint" refers to either a primitive constraint or a Boolean constraint.

3.1. Points and their domains

Points are either stored in the set $\mathbf{P_{FORM}}$ or in the set $\mathbf{P_{FORCE}}$, depending on whether they belong to the form diagram or to the force diagram. They are all characterized by two coordinates and seven geometric constraints. Each of these geometric constraints can be seen as a particular sub-domain – *i.e.* a graphical region – in which the point must be in order to fulfill certain conditions:

- the explicit domain
- the strict domain
- the force domain
- the reading cycle domain
- the topological domain
- the consistency domain
- the solution domain.

The explicit domain is the Boolean intersection of all the constraints explicitly created and applied by the user. $\mathbf{A_{ED}}$ is here defined as the array of sets containing the explicit domain of all the points.

The strict domain represents the region of positions for which no other point moves. Its construction is given in Section 4.3. The force domain is the intersection of all the geometric constraints that the point must hold in order to define forces correctly (see Section 4.1). The reading cycle domain gives the positions for which the current reading cycle of forces (adjacent in force polygons) remains valid (Figs. 6 and 7). The topological domain gives the positions for which the set of rods crossing each other in the form diagram remains unchanged (Fig. 8). The need for and the construction of these last two domains is not explained in this paper. Detailed information about them can be found in [17].

The consistency domain is the intersection of all the geometric constraints that the point must satisfy in order to ensure consistency with the domain of all the other points. Its general construction and limits of its applications are explained in Section 4.4.

Finally, the solution domain is the Boolean intersection of the explicit domain, the force domain and the consistency domain.

3.2. Primitive constraints

Primitive constraints are one of the eight following types: a half-plane, the inside of a disc, the outside of a disc, a unit compass and their respective inversions. These constraints only take points as parameters and are stored in the array of set $\mathbf{A_{PC}}$.

HalfPlane$[p_0p_1p_2]$ is the closed region, *i.e.* with the boundary included, on the left of p_0 according to the direction going from p_1 to p_2 (Fig. 9). The inversion of the HalfPlane constraint, written \HalfPlane$[p_0p_1p_2]$, is then an open region – *i.e.* with the boundary excluded – to the right of p_0 according to the direction going from p_1 to p_2. If p_1 and p_2 are coincident, HalfPlane$[p_0p_1p_2]$ comprises the entire plane of p_0 and its inversion does not exist.

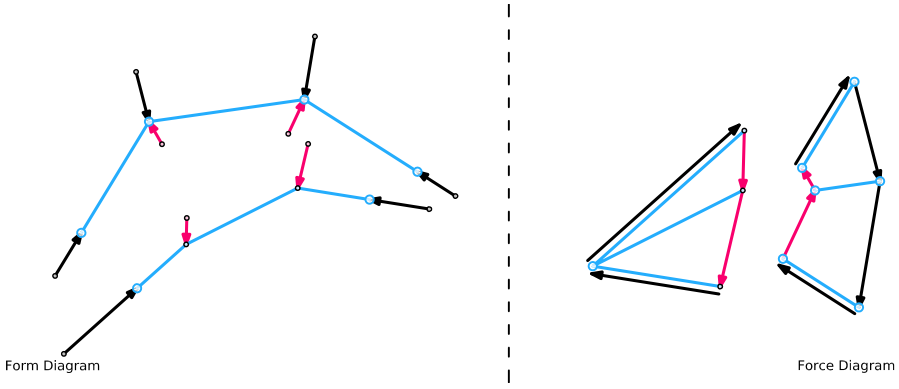

Fig. 3. Two strut-and-tie sub-networks prior to being combined.

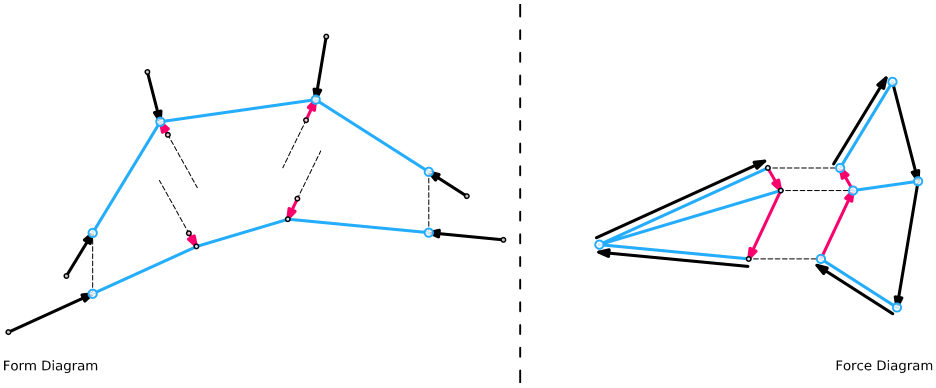

Fig. 4. The points of the previous sub-networks are dragged or geometrically constrained in order to make the combination feasible.

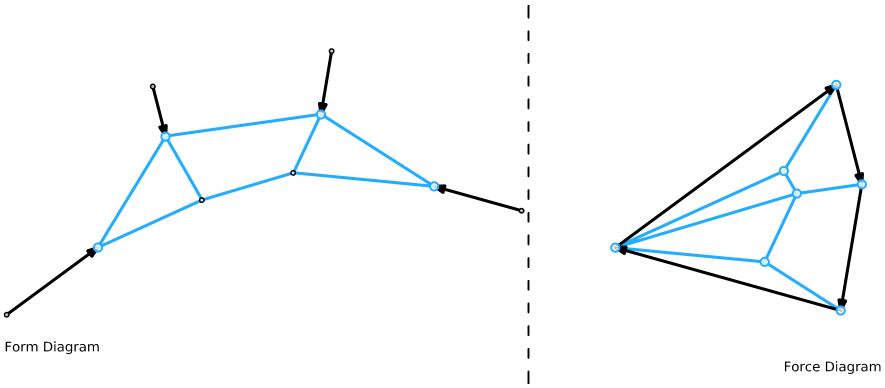

Fig. 5. The resulting strut-and-tie network after being combined.

DiscInside$[p_0p_1p_2]$ is the closed region inside the circle of center p_0 and radius p_1p_2 (Fig. 10). Its inversion, written \DiscInside$[p_0p_1p_2]$, is then an open region outside the circle of center p_0 and radius p_1p_2.

DiscOutside$[p_0p_1p_2]$ is the closed region outside the circle of center p_0 and radius p_1p_2 (Fig. 11). Its inversion, written \DiscOutside$[p_0p_1p_2]$, is then an open region inside the circle of center p_0 and radius p_1p_2. If p_1 and p_2 are coincident, the inversion of DiscOutside$[p_0p_1p_2]$ does not exist.

Finally, UnitCompass$[p_0]$ is a circle of center p_0 and of radius equal to the unit length of the diagram in which p_0 is applied.

As the HalfPlane constraint and its inversion depict the affine nature of geometry with relative directions, the DiscInside and DiscOutside constraints depict the metric nature of geometry with inequalities of distances. Furthermore, these regions are appropriate for use with two separate diagrams because the points that define the orientation of the half-plane or the radius of the disc can be distinct from their point of application.

An essential property of these primitive constraints is their symmetry. For instance, if the statement "$p_0 \in$ HalfPlane$[p_1p_2p_3]$" means that the point p_0 belongs to a HalfPlane$[p_1p_2p_3]$ constraint, then the four following statements are always equivalent (Figs. 12–15):

$p_0 \in$ HalfPlane$[p_1p_2p_3]$

$\Leftrightarrow p_1 \in$ HalfPlane$[p_0p_3p_2]$

$\Leftrightarrow p_2 \in$ HalfPlane$[p_3p_1p_0]$

$\Leftrightarrow p_3 \in$ HalfPlane$[p_2p_0p_1]$.

The same is valid with DiscInside and DiscOutside constraints and their inversions. For instance:

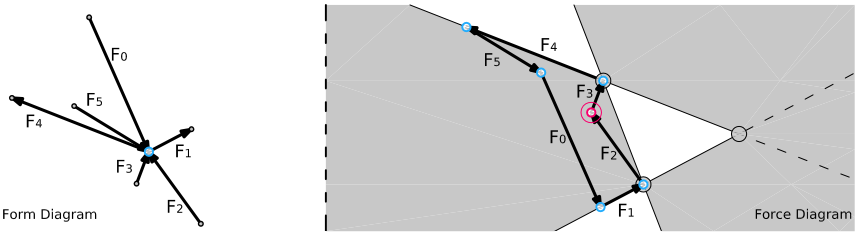

Fig. 6. Six forces in equilibrium. The shaded region is the domain that the highlighted point must hold in order to ensure that F_1, F_2, F_3 and F_4 are read in that order in the form diagram.

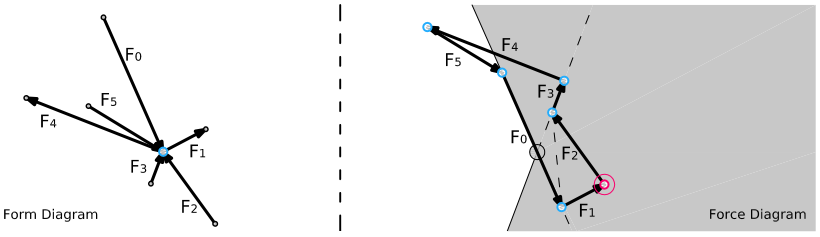

Fig. 7. The shaded region is the domain that the highlighted point must hold in order to ensure that F_0, F_1, F_2 and F_3 are read in that order in the form diagram.

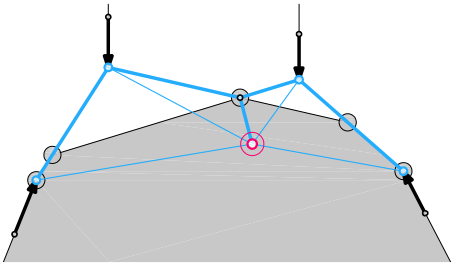

Fig. 8. Provided that the highlighted point remains within the shaded region, its move does not change any reading cycle of forces.

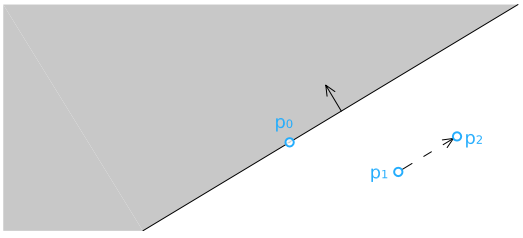

Fig. 9. A HalfPlane$[p_0p_1p_2]$ region.

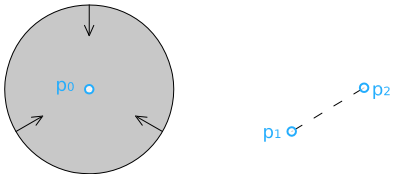

Fig. 10. A DiscInside$[p_0p_1p_2]$ region.

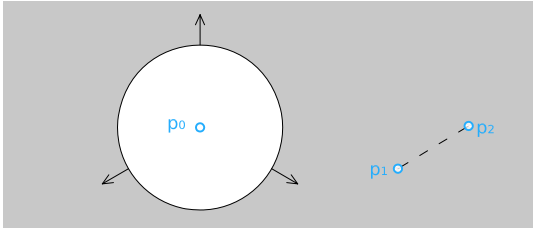

Fig. 11. A DiscOutside$[p_0p_1p_2]$ region.

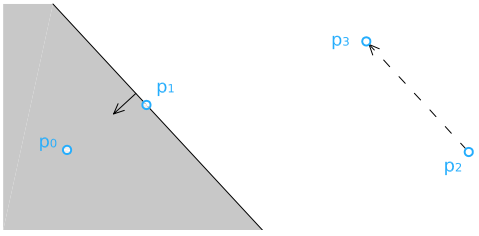

Fig. 12. Symmetric region to apply on p_0.

Fig. 13. Symmetric region to apply on p_1.

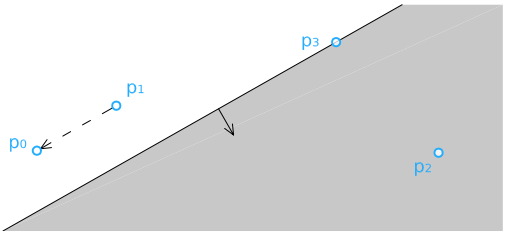

Fig. 14. Symmetric region to apply on p_2.

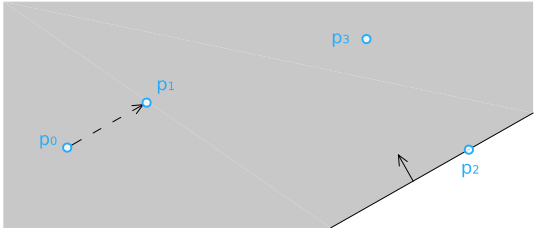

Fig. 15. Symmetric region to apply on p_3.

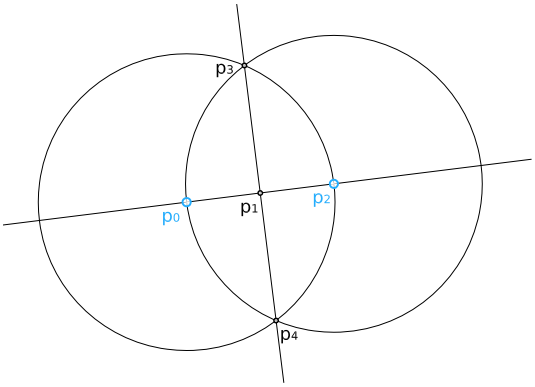

Fig. 16. Geometric construction of $p_1 \in \text{Midpoint}[p_0p_2]$.

$p_0 \in \text{DiscInside}[p_1p_2p_3]$
$\Leftrightarrow p_1 \in \text{DiscInside}[p_0p_2p_3]$
$\Leftrightarrow p_2 \in \text{DiscOutside}[p_3p_0p_1]$
$\Leftrightarrow p_3 \in \text{DiscOutside}[p_2p_0p_1]$.

3.3. Boolean constraints

Boolean constraints, stored in the array of set $\mathbf{A_{BC}}$, are of three types: the intersection constraint ($\cap$), the union constraint ($\cup$) and the inversion constraint ($\setminus$). The first two take any set of geometric constraints – *i.e.* primitive or Boolean constraints – as parameters and the third takes only one geometric constraint as a parameter.

Boolean operations are useful for creating more complex constraints. For instance, the following lines define a parametric circle (as drawn by the compass) of center p_0 and diameter p_1p_2, a parametric line (as drawn by the straightedge) passing through p_0 and parallel to p_1p_2, and a single position p_0:

- $\text{Compass}[p_0p_1p_2] := \text{DiscInside}[p_0p_1p_2] \cap \text{DiscOutside}[p_0p_1p_2]$
- $\text{Straightedge}[p_0p_1p_2] := \text{HalfPlane}[p_0p_1p_2] \cap \text{Halfplane}[p_0p_2p_1]$
- $\text{Position}[p_0] := \text{DiscInside}[p_0p_0p_0]$.

The following example uses these non-primitive constraints to define the point p_1 as the middle of the segment p_0p_2 (Fig. 16):

$p_1 \in \text{MidPoint}[p_0\ p_2]$
$\text{MidPoint}[p_0p_2] := \text{Straightedge}[p_0p_0p_2] \cap \text{Straightedge}[p_3p_3p_4]$
where : $p_3 \in \text{Compass}[p_0p_0p_2]$
$\cap\ \text{Compass}[p_2p_0p_2] \cap \text{HalfPlane}[p_0p_0p_2]$
$p_4 \in \text{Compass}[p_0p_0p_2] \cap \text{Compass}[p_2p_0p_2] \cap \text{HalfPlane}[p_0p_2p_0]$.

Point p_1 is consequently said to be a child of p_0 and p_2. From $\mathbf{A_{ED}}$, $\mathbf{A_{PC}}$ and $\mathbf{A_{BC}}$, the corresponding sets of children (out-neighbors), parents (in-neighbors), descendants, and ancestors of all the points can be deduced — descendants and ancestors sets being breadth-first sorted. These sets are illustrated in the graph of dependencies shown in Fig. 17.

3.4. Forces and rods

Forces are defined completely by four positions. A force F_0 determined by the sequence $[p_0\ p_1\ p_2\ p_3]$ (a) is applied on p_0 lying in the form diagram, (b) has an equal magnitude to the distance between p_2 and p_3 lying in the force diagram, (c) has the same direction as the one going from p_2 to p_3, and (d) pulls on p_0 if the directions going from p_0 to p_1 and from p_2 to p_3 are equal (Fig. 18) or pushes on p_0 if these directions are different (Fig. 19).

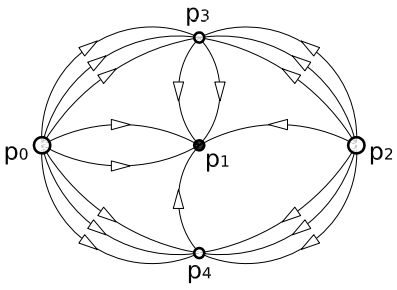

Fig. 17. Directed graph of dependencies for the construction illustrated in Fig. 16.

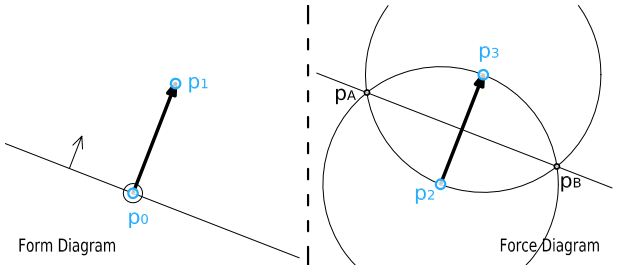

Fig. 18. Four points defining a pulling $\text{Force}[p_0p_1p_2p_3]$.

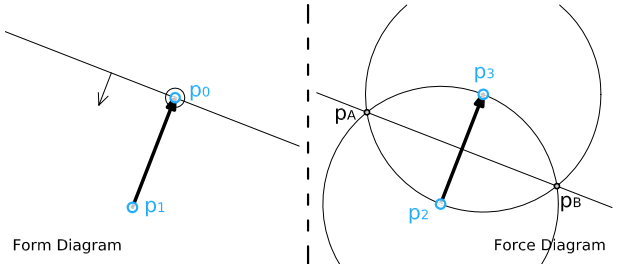

Fig. 19. Four points defining a pushing $\text{Force}[p_0p_1p_2p_3]$.

Rods are defined completely by the two opposite forces they replace. The rod is in tension or in compression depending on whether the two opposite forces are pulling or pushing. The geometric properties that these two opposite forces must hold are detailed in Section 4.1, Fig. 23.

The arrays of sets $\mathbf{A_F}$ and $\mathbf{A_R}$ contain all the forces and rods currently in use and their parameters.

4. Internal processing

This section initially lists native operations and then outlines the general functional flow whereby they are processed. The following sub-sections then highlight techniques that take full advantage of the high-level of abstraction of the fully geometric approach presented.

4.1. Native operations

The idea here is to identify a minimal set of basic operations that the user can use to constrain and transform any strut-and-tie network in equilibrium into another one. If algorithms exist to process these native operations, then they can be assembled in sequence in order to define and hence perform more complex operations (see Section 5). The following list explains how these native operations modify the arrays of data $\mathbf{P_{FORM}}$, $\mathbf{P_{FORCE}}$, $\mathbf{A_{ED}}$, $\mathbf{A_{PC}}$, $\mathbf{A_{BC}}$, $\mathbf{A_F}$ and $\mathbf{A_R}$. Other native operations should complement this list in order to obtain information on values, perform checks on values and select objects.

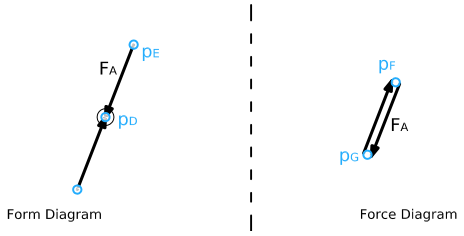

Fig. 20. Two forces in equilibrium.

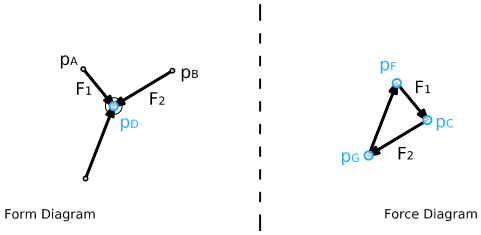

Fig. 21. Three forces in equilibrium after resolution of force F_0 (Fig. 20) into F_1 and F_2.

These native operations have here very limited purposes in order to ease their implementation. Their capabilities are expected to be extended with more general and intuitive higher-order procedures (see Section 5).

- **CreatePointInFormDiagram**
  adds a new point at the end of $\mathbf{P_{FORM}}$
- **CreatePointInForceDiagram**
  adds a new point at the end of $\mathbf{P_{FORCE}}$
- **DeletePoint**[$\mathbf{p_A}$]
  if p_A is not a member of $\mathbf{A_{ED}}$, $\mathbf{A_{PC}}$, and $\mathbf{A_F}$: removes p_A from $\mathbf{P_{FORM}}$ or $\mathbf{P_{FORCE}}$
- **MovePoint**[$\mathbf{p_A}$ **xy**]
  according to the method presented in Section 4.3:
  updates the x and y coords of p_A as close as possible to the desired position and,
  if required, updates the coords of all the descendants of p_A, their reading cycle domain and the topological domain of all the points referenced in $\mathbf{A_F}$
- **MergePoint**[$\mathbf{p_A p_B}$]
  if the coords of p_A and p_B are equivalent and
  if p_B belongs to $\mathbf{P_{FORCE}}$ or if it does not belong to $\mathbf{A_F}$:
  replaces every occurrence of p_B by p_A
- **CreatePrimitiveConstraint**[**type**][$\mathbf{p_A p_B p_C}$]
  adds a constraint and its given parameters at the end of $\mathbf{A_{PC}}$
  this constraint is of one of the eight primitive types given in Section 3.2
- **CreateIntersection**[$\mathbf{c_A}$ $\mathbf{c_B}$]
  adds an Intersection[c_A c_B] constraint at the end of $\mathbf{A_{BC}}$
- **CreateUnion**[$\mathbf{c_A}$ $\mathbf{c_B}$]
  adds a Union[c_A c_B] constraint at the end of $\mathbf{A_{BC}}$
- **CreateInversion**[$\mathbf{c_A}$]
  adds an Inversion[c_A] constraint at the end of $\mathbf{A_{BC}}$
- **DeleteConstraint**[$\mathbf{c_A}$]
  if c_A does not belong to $\mathbf{A_{ED}}$ or to $\mathbf{A_{BC}}$: removes c_A from $\mathbf{A_{PC}}$
- **ApplyConstraint**[$\mathbf{p_A c_A}$]
  checks whether the application of c_A empties the domain of p_A (similar to the method used to update positions in Section 4.3)
  if not, adds the membership "$p_A \in c_A$" to $\mathbf{A_{ED}}$
  updates the strict domain of p_A
  updates the consistency domain of the ancestors of p_A
- **CancelConstraint**[$\mathbf{p_A c_A}$]
  removes the constraint c_A from the set of constraints applied to p_A in $\mathbf{A_{ED}}$
  updates the strict domain of p_A
  updates the consistency domain of the ancestors of p_A
  SwitchDependencies[$\mathbf{p_A p_B c_A}$]
  if c_A is applied on p_A in $\mathbf{A_{ED}}$ and if p_B is a parameter of c_A in $\mathbf{A_{PC}}$:
  cancels the application of c_A onto p_A in $\mathbf{A_{ED}}$,
  adds c_B at the end of $\mathbf{A_{PC}}$, c_B is a new constraint that is symmetric to c_A according to the rules of symmetry described in Section 3.2,
  adds the membership "$p_B \in c_B$" to $\mathbf{A_{ED}}$,
  updates the strict domain of p_A and p_B
  updates the consistency domain of the ancestors of p_A and p_B
- **CreateZeroForce**[$\mathbf{p_A p_B}$]
  if p_A belongs to $\mathbf{P_{FORM}}$ and if p_B belongs to $\mathbf{P_{FORCE}}$:
  adds a new force [$p_A p_A p_B p_B$] at the end of $\mathbf{A_F}$
- **DeleteZeroForce**[$\mathbf{F_A}$]
  if F_A is a zero force in $\mathbf{A_F}$:
  removes the force F_A from $\mathbf{A_F}$
- **ResolveForce**[$\mathbf{F_A p_A p_B p_C}$] (Figs. 20 and 21)
  if p_A and p_B belong to $\mathbf{P_{FORM}}$, if p_C belongs to $\mathbf{P_{FORCE}}$ and
  if F_A is in the form of [$p_D p_E p_F p_G$] in $\mathbf{A_F}$:
  removes F_A from $\mathbf{A_F}$
  removes the constraint "Straightedge[$p_D p_F p_G$] ∩ Coincidence Condition[$p_D p_F p_G$]" from the force Domain of p_E
  adds a new force [$p_D p_A p_F p_C$] at the end of $\mathbf{A_F}$
  adds a new force [$p_D p_B p_C p_G$] at the end of $\mathbf{A_F}$
  adds the constraint "Straightedge[$p_D p_F p_C$] ∩ Coincidence Condition[p_D p_F p_C]" in the force Domain of p_A
  adds the constraint "Straightedge[$p_D p_C p_G$] ∩ Coincidence Condition[p_D p_C p_G]" in the force Domain of p_B
  selects all the forces in $\mathbf{A_F}$ that are applied on p_D and updates the reading cycle domain of their parameters
  updates the topological domain of all the points referenced in $\mathbf{A_F}$
  updates the strict domain of p_A, p_B and p_E
  updates the consistency domain of the ancestors of p_A, p_B and p_E
- **SwapForceCycle**[$\mathbf{F_A F_B}$] (Fig. 22)
  if neither F_A nor F_B is part of $\mathbf{A_R}$
  if F_A is in the form of [$p_A p_B p_C p_D$] in $\mathbf{A_F}$ and
  if F_B is in the form of [$p_A p_E p_D p_F$] in $\mathbf{A_F}$ and
  if there is no constraint applied on p_A, p_B, p_C, p_D, p_E and p_F in $\mathbf{A_{ED}}$:
  removes the constraint "Straightedge[$p_A p_C p_D$] ∩ Coincidence Condition[$p_A p_C p_D$]" from the force Domain of p_B
  removes the constraint "Straightedge[$p_A p_D p_F$] ∩ Coincidence Condition[$p_A p_D p_F$]" from the force Domain of p_E
  changes the parameters of F_A and F_B in $\mathbf{A_F}$ such that F_A is defined by the points [$p_A p_B p_D p_F$] and F_B by the points [$p_A p_E p_C p_D$]
  adds the constraint "Straightedge[$p_A p_D p_F$] ∩ Coincidence Condition[$p_A p_D p_F$]" to the force Domain of p_B
  adds the constraint "Straightedge[$p_A p_C p_D$] ∩ Coincidence Condition[$p_A p_C p_D$]" to the force Domain of p_E
  selects all the forces in $\mathbf{A_F}$ that are applied on p_D and updates their reading cycle domain
  updates the topological domain of all the points referenced in $\mathbf{A_F}$
  updates the strict domain of p_B and p_E
  updates the consistency domain of the ancestors of p_B and p_E
- **CreateRod**[$\mathbf{F_A F_B}$] (Fig. 23)
  if neither F_A nor F_B is already part of $\mathbf{A_R}$ and
  if F_A is in the form of [$p_A p_B p_E p_F$] in $\mathbf{A_F}$ and
  if F_B is in the form of [$p_C p_D p_F p_E$] in $\mathbf{A_F}$ and
  if constraints exist in $\mathbf{A_{ED}}$ such that $p_A p_C$ and $p_E p_F$ are parallels and
  if the following memberships exist in $\mathbf{A_{ED}}$ (for any points p_G and p_H)
  $p_B \in$ HalfPlane[$p_A p_G p_H$]
  $p_D \in$ HalfPlane[$p_C p_H p_G$]
  $p_G \in$ Compass[$p_A p_A p_C$] ∩ Compass[$p_C p_A p_C$] ∩ HalfPlane[$p_A p_A p_C$]
  $p_H \in$ Compass[$p_A p_A p_C$] ∩ Compass[$p_C p_A p_C$] ∩ HalfPlane[$p_C p_C p_A$]

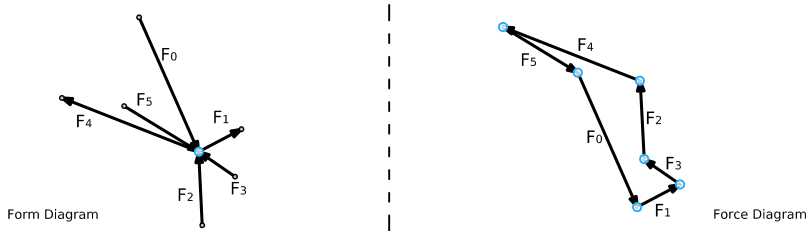

Fig. 22. Forces F_2 and F_3 from Fig. 6 have been switched to accommodate a new reading cycle.

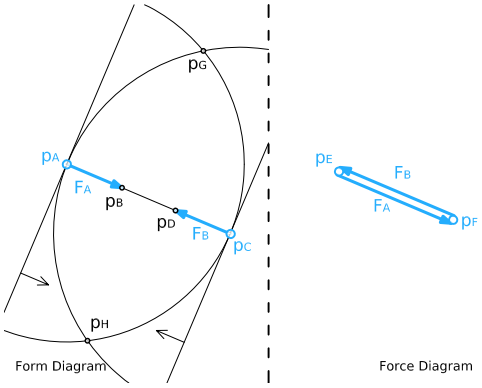

Fig. 23. Two forces equivalent to a tension rod.

adds a new entry $[F_A F_B]$ in $\mathbf{A_R}$
updates the topological domain of all the points referenced in $\mathbf{A_F}$

- **CancelRod**[$\mathbf{R_A}$]
  removes the rod $\mathbf{R_A}$ from $\mathbf{A_R}$
  updates the topological domain of all the points referenced in $\mathbf{A_F}$.

The non-fundamental constraint CoincidenceCondition is defined in Section 5.2.

The **SwapForceCycle** operation is intended to be called by the software itself as soon as a point is moved outside its reading cycle domain.

The **CreateZeroForce** operation actually inserts the smallest strut-and-tie network that is proved to be in static equilibrium whatever its parameter points, while the **ResolveForce** operation implements the rule of the force parallelogram in a way that guarantees the reciprocal rules between the two diagrams of graphic statics (see Section 2.1). These two operations suffice to create as many forces in equilibrium as desired. The sum of two forces is simply obtained by canceling a force in the force diagram — *i.e.* by superimposing the two points defining one of the two forces in the force diagram and merging them.

The constraints that are required by the **CreateRod** operation (Fig. 23) are also meant to guarantee the permanency of the reciprocal rules of graphic statics.

4.2. Functional flow

Since the structural designer modifies his or her design step by step, the resolution of the geometric constraints is performed in a similar way — *i.e.* in a sequential manner that builds on the previous results. Fig. 24 presents the functional flow diagram of the approach: boxes are computed data and arrows are algorithms or user commands. The construction plan holds a declarative list of all the native operations (Section 4.1) applied by the designer to the strut-and-tie network.

These native operations are first interpreted by a coordinate-free solver that updates the arrays of data $\mathbf{P_{FORM}}$, $\mathbf{P_{FORCE}}$, $\mathbf{A_{PC}}$, $\mathbf{A_{BC}}$, $\mathbf{A_F}$ and $\mathbf{A_R}$, the sets of explicit domains $\mathbf{A_{ED}}$, the sets of children, descendants, parents and ancestors of all points, the sets of strict domains (see Section 4.3), force domains (see operation **ResolveForce**), reading cycle domains, topological domains and certain sets of consistency domains (see Section 4.4). The conjunctive and disjunctive normal forms of these domains are also computed by this solver. Except for the reading cycle domains, these various domains offer the advantage of not being recalculated when points are dragged [17].

The update of the positions and hence of the actual shape of the domains of solutions is undertaken by a numerical solver whose job is primarily to compute orthogonal projections on lines or circles (see Section 4.3) and to complete the consistency domains that could not be created by the coordinate-free solver.

The user can then modify the strut-and-tie network by moving points or applying a new operation. Thanks to the nature of the native operations, a local treatment of the data is usually sufficient for processing these modifications in both the symbolic and numerical solvers.

4.3. Update of positions

If a constraint c^* is applied on a point p^*, it is expected that p^* remains inside c^* when one of its ancestors p_i (*i.e.* a parameter of c^*) is moving. To this end, the **MovePoint** operation (Section 4.1) updates the position p^* automatically to the closest position inside its solution domain as soon as one of its ancestors point p_i is dragged. Thanks to the nature of the three primitive regions, this new position is fairly quick to compute because it is either an orthogonal projection on a line or a circle, or the intersection between two lines, two circles or a line with a circle. It can be noticed that in most cases this action minimizes the disturbance of the model when it jumps from one solution to the other.

Interdependency occurs when constraints present a loop — meaning that each point in that loop is constrained by itself to some extent. The update of positions might then either (1) converge after a finite number of updates, (2) converge after an infinite number of updates — in which case it can be stopped as soon as the distance of the displacement is smaller than a fixed small value ε — or (3) never converge (in which case the application of the last constraint should be avoided).

The resulting iterative adjustment gives the ability to constrain a point directly on any algebraic or transcendental curve, something that is not feasible with non-interdependent compass and straightedge constraints. As an example, the following description constrains a point p_2 on a parabola whose focus is on p_0 and whose directrix passes through p_1 and is perpendicular to p_0p_1 (Fig. 25):

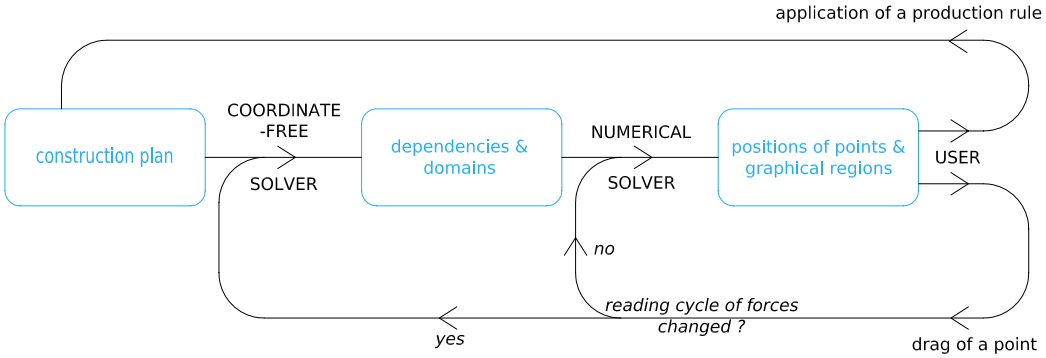

Fig. 24. Functional flow diagram.

$p_3 \in \text{HalfPlane}[p_0p_0p_1] \cap \text{Compass}[p_0p_0p_1] \cap \text{Compass}[p_1p_0p_1]$

$p_4 \in \text{HalfPlane}[p_0p_1p_0] \cap \text{Compass}[p_0p_0p_1] \cap \text{Compass}[p_1p_0p_1]$

$p_5 \in \text{Straightedge}[p_1p_3p_4] \cap \text{Straightedge}[p_2p_0p_1]$

$p_2 \in \text{Compass}[p_0p_2p_5]$.

The two interdependent cycles can be identified in the center of Fig. 26.

The region resulting from the explicit domain might in some cases be a unique position, as was the case for points p_1, p_3 and p_4 in Fig. 16. This means that the point can no longer be moved.

The general update of positions when a point p_0 is dragged is consequently formalized as follows:

- get T_1 = the set of all the descendants of p_0
- add p_0 at the beginning of T_1
- while every point p_j in T_1 is not marked "updated":
  - T_2 = the set of all the ancestors of p_j that are also in T_1
  - if T_2 is empty
    or if every point in T_2 is either already marked "updated" or is also a child or grandchild of p_j (interdependency) :
    - ∗ check (Boolean search) whether p_j belongs to its solution domain
    - ∗ if true: mark p_j as "updated"
    - ∗ if false:
      - · get T_3 = the set of positions representing all the orthogonal projections and all the intersections between the primitive lines and circles of its solution domain
      - · find the position p_C of T_3 that is closest to p_j
      - · get λ = the distance between p_C and p_j
      - · apply the coords of p_C onto p_j (if p_C is not inside the explicit domain because its boundary is excluded, then slightly shift its coords inside the boundary)
      - · if T_2 is empty or if every point in T_2 is already marked "updated" or if λ is less than or equal to ε (convergent interdependency):
        - mark p_j as "updated"
      - · else if λ is greater than the previous λ calculated for the same p_j:
        - the interdependency does not converge and the application of the last constraint that creates that interdependency should be canceled.

An empty set T_3 means that the region resulting from the solution domain of p_j is empty. This case is not taken into account here because the consistency domains of the ancestors of p_j are assumed to prevent the solution domain of p_j from being empty.

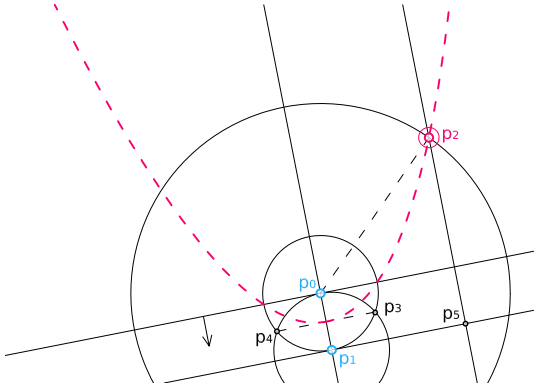

Fig. 25. Constraining p_2 on a parabola with focus p_0.

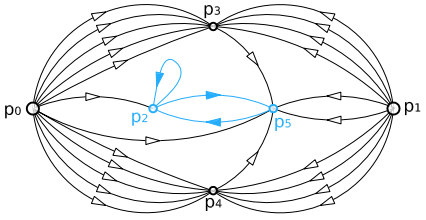

Fig. 26. Directed graph of dependencies for the construction illustrated in Fig. 25.

4.4. Ensuring consistency by means of coordinate-free geometric reasoning

This sub-section shows how the above-mentioned fully geometric approach can favor coordinate-free methods to ensure consistency between specific domains of solutions.

A domain of solution is consistent here if every point has a non-empty graphical region of possible positions. Owing to the constraint dependencies, this means that the set of possible positions of each father point must be restricted so that it can never be placed in a position that empties the domain of one of its descendants. This is called constraint propagation.

Fig. 27 presents an example in which points p_0 and p_1 are constrained as follows:

$p_0 \in \text{HalfPlane}[p_2p_2p_3] \cap \text{HalfPlane}[p_4p_4p_5]$

$p_1 \in \text{HalfPlane}[p_6p_6p_7] \cap \text{HalfPlane}[p_8p_8p_9] \cap \text{HalfPlane}[p_0p_{10}p_{11}]$.

If p_0 moves to the right, its child p_1 is then moved to the right too until the domain of p_1 becomes empty, meaning that the problem has no solution. In order to prevent this impasse, the domain of p_1 is propagated on the consistency domain of p_0.

The resulting solution domain of p_0 shown in Fig. 28 is the intersection between its explicit domain and its consistency domain.

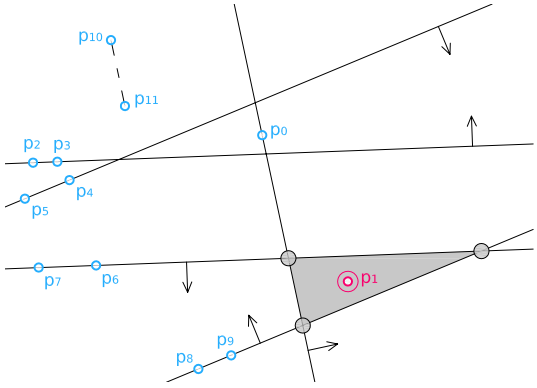

Fig. 27. The shaded triangle is the domain of p_1.

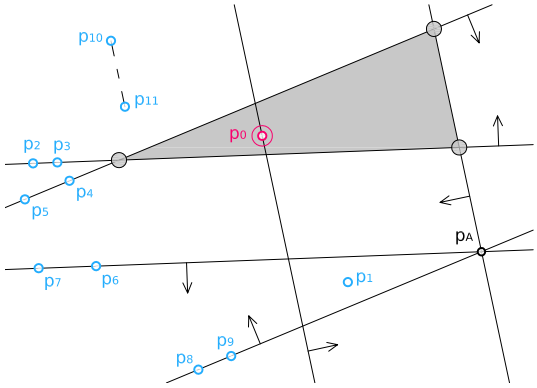

Fig. 28. The shaded triangle is the intersection of the explicit domain of p_0 with its consistency domain. It ensures that the domain of p_1 is never empty.

Propagation of inequalities constraints (like the HalfPlane, DiscInside and DiscOutside constraints) is a problem that is not yet successfully tackled by the scientific community and that requires extensive development. Possible approaches may perform numerical approximations to delimit the consistency domain with a desired accuracy or reproduce exact known loci of possible positions based on the given constraints and their dependencies. The following method exemplifies how the latter approach can in some cases be here done in a fully constructive way using coordinate-free geometric reasoning.

The scope of application of this method is limited to regions that can be described with HalfPlane, DiscInside and DiscOutside constraints. As a consequence, it is restricted to propagations where the path of paternal filiation between the child point and the father point is unique, *e.g.* if a point is the father point of another point, it can be its father several times (by several constraints) but it cannot also be its grandfather or its child. This restriction is intended for two reasons. Firstly, the description of the resulting propagated regions, *i.e.* the consistency domains, uses a coordinate-free grammar, meaning that any technique developed in this paper can be applied to it. Secondly, consistency domains can be defined by combinations of constraints in such a way that their geometrical behavior, *i.e.* the resulting region, remains valid for any new position of a point. They therefore do not have to be recalculated every time they are moved and the update of their coordinate-free description is only required when new constraints are applied or old ones canceled.

The propagation of Fig. 28 is relatively simple since it concerns two direct relatives whose domain comprises only an intersection of constraints. When the paternal filiation is greater, the following general procedure should be used to obtain the consistency domain of a point p_0. It is based on the facts that (1) no parallel paternal filiation exists and (2) the solution domain of a child point p^* cannot be propagated on its parents if the solution domain of its own children has not already been propagated onto its consistency domain:

- T_{child} = the set of all the descendants of p_0
- T_{child} is breadth-first sorted starting from childless points and ending with the direct children of p_0
- for each point p_i of T_{child} (starting from the first element):
  - if p_i is childless: its consistency domain is equal to the entire plane (not constrained)
    - consider the next point p_i
  - else, propagate the solution domain of p_i on its direct parents:
    - get $C_\cup$ = the disjunctive normal form of the Boolean combination of the primitive constraints in the solution domain of p_i
    - get T^* = the subset of "$T_{child} + p_0$" containing the direct parents of p_i
    - for every point p_j in T^*:
      - for every sub-intersection $C_\cap$ of $C_\cup$:
        - get C_k = the subset of $C_\cap$ that uses p_i as parameter
        - get $C_{\cap-k}$ = the subset of $C_\cap$ that does not use p_i as parameter
        - for every constraint C_m in C_k:
          - find the pattern of sub-propagation to be applied (see below) according to C_m and $C_{\cap-k}$
          - construct D_m = the consistency sub-domain corresponding to that pattern
        - get $D_\cap$ = the intersection of every D_m
      - get $D_\cup$ = the union of every $D_\cap$
      - intersect $D_\cup$ with the existing consistency domain of p_j
- the solution domain of p_0 is the intersection of its explicit domain with its force domain and its consistency domain.

Patterns of sub-propagation are specifications based on the constraint type of C_m and $C_{\cap-k}$ and their quantity. They are used to differentiate the geometric reasoning to be applied. The following paragraphs present three patterns among others. Their complete implementation goes beyond the purpose of this paper. The implementation of all the conceivable patterns is an area that requires additional research.

(1) A first pattern that is very easily automated comprises no more than one HalfPlane or \HalfPlane and an undetermined number of DiscOutside or \DiscInside constraints. Since the intersection of these constraints always includes points at infinity, the propagation domain resulting from this pattern is the entire plane.

(2) A second pattern is the one for which the constraint that propagates is a DiscOutside or a \DiscInside constraint and the intersection to be propagated (a convex shape) includes only HalfPlane, \HalfPlane, DiscInside and \DiscOutside constraints. The resulting domain is always a union of arcs of DiscOutside and \DiscInside constraints.

(3) A third pattern is concerned only with the convex primitive constraints (*i.e.* HalfPlane, \HalfPlane, DiscInside and \DiscOutside). Using Eduard Helly's theorem [18], it can be shown that the set $C_m \cap C_{\cap-k}$ is non-empty if each sub-intersection comprising a maximum of three constraints is non-empty, whatever the large number of constraints in $C_m \cap C_{\cap-k}$. Moreover, since this pattern considers only four types of primitive constraints there are only 60 types of such sub-intersections. This means that each propagation of these sub-intersections is predefined constraints that can be propagated on an individual basis.

Whether one of these typical intersections presents an empty domain or not actually depends on the current positions of the

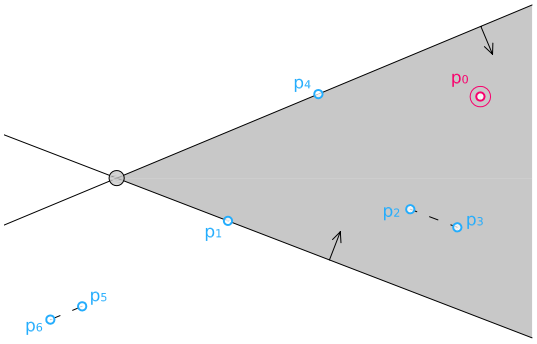

Fig. 29. The shaded area is the domain of p_0, i.e. the intersection of two half-planes.

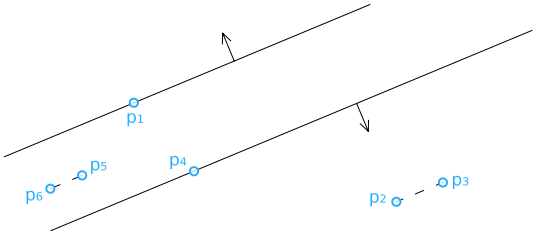

Fig. 30. The domain of p_0 is empty because the half-planes are parallel and opposite.

points defining it. However, it is here possible to construct these constraints in a coordinate-free way.

For instance, Fig. 29 presents one of the typical intersections, here involving two half-planes: HalfPlane$[p_1p_2p_3] \cap$ HalfPlane$[p_4p_5p_6]$. If (1) their borders are parallel, (2) they do not face one another and (3) their borders are distinct, they then describe an empty domain in Euclidean geometry (Fig. 30). To avoid this situation, a propagation constraint must be applied to each of these six points. For instance, the one applied to p_1 can be written as follows:

$$\text{HalfPlane}[p_Gp_Cp_4] \cup \text{HalfPlane}[p_Hp_Hp_4] \cup \text{Straightedge}[p_4p_5p_6]$$

where:

$$p_A \in \text{Compass}[p_2p_2p_3] \cap \text{Compass}[p_3p_3p_2] \cap \text{HalfPlane}[p_2p_2p_3]$$
$$p_B \in \text{Compass}[p_2p_2p_3] \cap \text{Compass}[p_3p_3p_2] \cap \text{HalfPlane}[p_3p_3p_2]$$
$$p_C \in \text{Compass}[p_5p_5p_6] \cap \text{Compass}[p_6p_6p_5] \cap \text{HalfPlane}[p_5p_5p_6]$$
$$p_D \in \text{Compass}[p_5p_5p_6] \cap \text{Compass}[p_6p_6p_5] \cap \text{HalfPlane}[p_6p_6p_5]$$
$$p_E \in \text{Compass}[p_4p_2p_3] \cap \text{Straightedge}[p_4p_5p_6] \cap \text{HalfPlane}[p_4p_Cp_D]$$
$$p_F \in \text{Compass}[p_4p_2p_3] \cap \text{Straightedge}[p_4p_5p_6] \cap \text{HalfPlane}[p_4p_Dp_C]$$
$$p_G \in \text{Compass}[p_4p_2p_3] \cap \text{Straightedge}[p_4p_2p_3] \cap \text{HalfPlane}[p_4p_Ap_B]$$
$$p_H \in \text{Compass}[p_4p_2p_3] \cap \text{Straightedge}[p_4p_2p_3] \cap \setminus\text{Position}[p_E] \cap (\text{HalfPlane}[p_4p_Bp_A] \cup \text{Position}[p_F]).$$

As previously mentioned, this constraint remains valid whatever the current positions are: if the orientation of the two half-planes are different, the domain of p_1 is not altered (Fig. 31), but if these orientations are parallel and opposite, the domain of p_1 is restrained (Fig. 32). This remains true even if the orientations are undefined — when p_2 and p_3 are coincident or p_5 and p_6 are coincident.

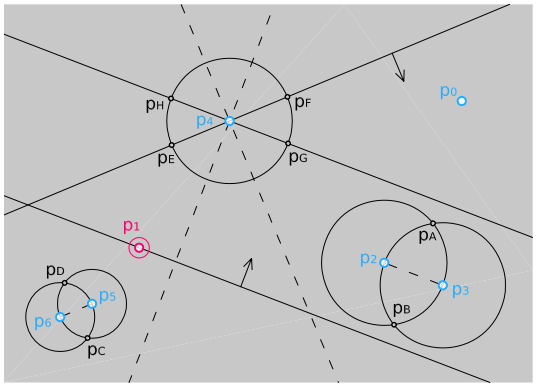

Fig. 31. The shaded area is the domain that p_1 must hold in order to ensure that the domain of p_0 is not empty — this domain is equal to the entire plane when the two half-planes of Fig. 29 are secant.

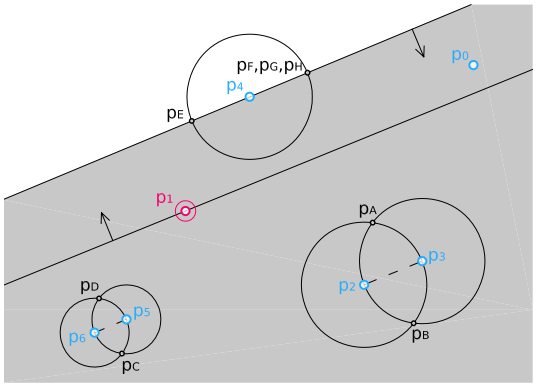

Fig. 32. The shaded area is the domain that p_1 must hold, when the two half-planes of Fig. 29 are opposite (Fig. 30), in order to ensure that the domain of p_0 is not empty.

5. Higher-order procedures

5.1. The role of higher-order procedures

The native operations presented in Section 4.1 are not very practical for direct use by the designer. An extensive library of higher-order procedures must be created in order to meet the user's intuitive workflows. These routines can be seen as declarations of successive native operations. A first set of routines can concern purely geometric statements — *e.g.* constraining a point on the middle of a segment, on a certain distance from a given segment or on the mirror of another constraint. Another set of routines can build the equilibrium of standard sub-structures — *e.g.* sustaining a set of forces with a catenary or a basic truss or refining a sub-structure to allow a denser discretization of a certain distributed load.

More specialized routines can be used in line with the nature of the current structural abstraction — *e.g.* graphically describing the bending moments in a given beam, defining new operations that are able to build and modify discontinuous stress fields, or graphically optimizing the given criteria of a strut-and-tie network.

Other routines can also be used to handle topological restrictions. For instance, Fig. 33 presents the domain that the highlighted point must satisfy in order for the rod to remain in compression. This domain can be deduced from the rules defining the type of application of a force (see Section 3.4).

Finally, routines can be used to apply boundary conditions to the strut-and-tie network. Fig. 2 shows one example of this, while Fig. 41 presents another.

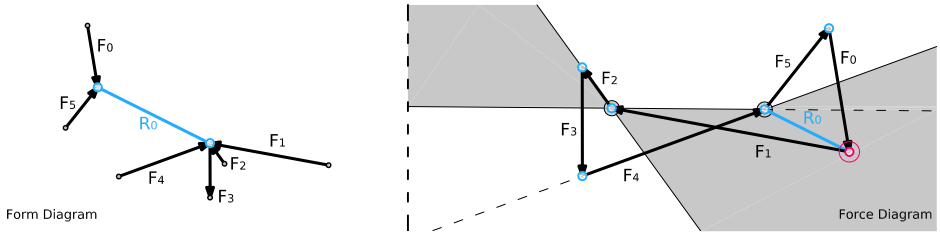

Fig. 33. The rod R_0 remains in compression as long as the highlighted point in the force diagram stays within the shaded region.

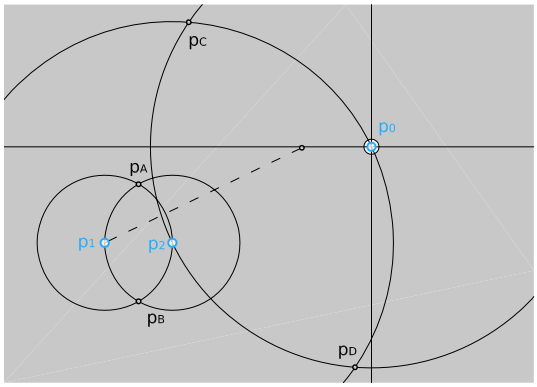

Fig. 34. The resulting domain is the position of p_0 if p_1 and p_2 are coincident, and the inverse if not.

5.2. Dynamic conditional geometric statements

An interesting consequence of the nature of the geometric constraints discussed in this paper is the ability to compute conditional statements graphically – such as If/Else conditions – by basic combinations of primitive constraints. Although these constraints are constructed symbolically as usual, their inner behavior produces conditional results that are continually updated when points are moving. The conditional result – *i.e.* the graphical region of this constraint – is chosen to be the unique position if the condition is satisfied and its inverse – *i.e.* the entire plane excluding the unique position – if the condition is not satisfied. A complete set of conditional expressions can be obtained with five non-fundamental constraints: the first checks the coincidence of two positions, the second and third check if four points presently verify a membership with a HalfPlane or with a DiscInside constraint, and the final two compare two simultaneous coincidences either conjunctively or disjunctively.

By way of introduction, the ConcidenceCondition$[p_0p_1p_2]$ constraint is presented. It returns a domain equal to the position of p_0 if p_1 and p_2 are coincident, and the inverse if not. This constraint can be defined using the following combinations (Fig. 34):

$$\begin{aligned}&\text{CoincidenceCondition}[p_0p_1p_2]\\&\quad := \setminus(\text{Straightedge}[p_0p_1p_2] \cap \text{Straightedge}[p_0p_Ap_B])\\&\quad \cup (\text{Compass}[p_1p_2p_0] \cap \text{Straightedge}[p_1p_2p_0]\\&\quad \cap \text{HalfPlane}[p_1p_Cp_D])\end{aligned}$$

where:

$$\begin{aligned}p_A &\in \text{Compass}[p_1p_1p_2] \cap \text{Compass}[p_2p_2p_1] \cap \text{HalfPlane}[p_1p_1p_2]\\p_B &\in \text{Compass}[p_1p_1p_2] \cap \text{Compass}[p_2p_2p_1] \cap \text{HalfPlane}[p_2p_2p_1]\\p_C &\in \text{Compass}[p_2p_2p_0] \cap \text{Compass}[p_0p_0p_2] \cap \text{HalfPlane}[p_2p_2p_0]\\p_D &\in \text{Compass}[p_2p_2p_0] \cap \text{Compass}[p_0p_0p_2] \cap \text{HalfPlane}[p_0p_0p_2].\end{aligned}$$

5.3. Switch of dependencies hierarchy

A second interesting technique is the ability to switch the dependencies of constraints between themselves — that is to say, to make a child point its father's father and vice versa. This feature is highly convenient for the user because it allows an inversion of the parametric hierarchy at any time, without having to rebuild the entire problem. It is as if the user first analyzes the behavior of a certain result by varying the terms of the problem and then decides to alter this result explicitly to see how the terms of the problem would behave.

If the dependencies hierarchy between two points p_A and p_B is to be switched, the following procedure applies:

- if p_B is a descendant of p_A
  - T_1 = the set of all the descendants of p_A that are also ancestors of p_B
  - T_1 is breadth-first sorted starting from childless points and ending with orphans (parallel dependencies may occur)
  - add p_A at the end of T_1 and p_B at the beginning of T_1
  - for every p_i and p_{i+1} of T_1:
    - from $\mathbf{A_{ED}}$ get T_2 = the set of all the constraints applied onto p_i
    - from $\mathbf{A_{PC}}$ get T_3 = the subset of T_2 that uses p_{i+1}
    - for every constraint c_j in T_3:
      - apply the native operation SwitchDependencies$[p_ip_{i+1}c_j]$ (see Section 4.1).

This procedure is significant only when there is no loop of interdependency between p_A and p_B. Moreover, this procedure does not create superfluous interdependency because all the parallel dependencies – *i.e.* when a point is the parameter of another point through several constraints – are switched together.The following construction is used as a basic example (Figs. 35 and 36):

$$\begin{aligned}p_7 &\in \text{Straightedge}_A[p_4p_1p_2] \cap \text{Straightedge}_B[p_6p_0p_1]\\p_8 &\in \text{Straightedge}_C[p_5p_2p_3] \cap \text{Straightedge}_D[p_7p_0p_2]\\p_9 &\in \text{Straightedge}_E[p_8p_0p_3].\end{aligned}$$

This description can be seen as the geometric skeleton of a simply-connected structure passing through p_6 and bearing two forces that are applied on p_7 and p_8 and that have magnitudes equal to the distances p_1p_2 and p_2p_3 respectively.

The point p_6 can be moved anywhere in the plane while p_8 is constrained on the intersection of two straightedges. The example consists of switching those two degrees of freedom. Since p_8 is a grandson of p_6, the algorithm must perform two successive changes on the graph of Fig. 36: the first between p_8 and p_7 (S_D), and the second between p_7 and p_6 (S_B). After the switching, the resulting dependencies are as follows (Fig. 37):

$$\begin{aligned}p_6 &\in \text{Straightedge}_F[p_7p_0p_1]\\p_7 &\in \text{Straightedge}_A[p_4p_1p_2] \cap \text{Straightedge}_G[p_8p_0p_2]\\p_8 &\in \text{Straightedge}_C[p_5p_2p_3]\\p_9 &\in \text{Straightedge}_E[p_8p_0p_3].\end{aligned}$$

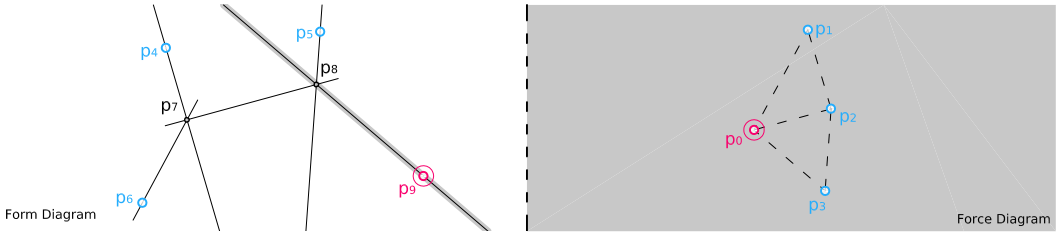

Fig. 35. Prior to switching (smaller points are directly movable and bigger points must remain at the intersection of two lines; the shaded area in the force diagram is the domain of p_0).

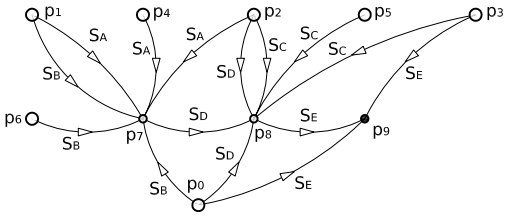

Fig. 36. Directed graph of dependencies for the construction illustrated in Fig. 35.

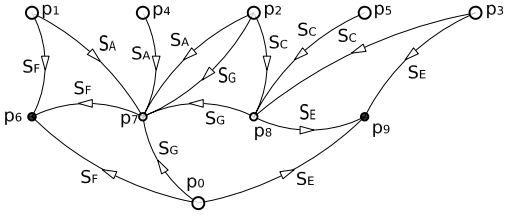

Fig. 37. Directed graph of dependencies after switching p_6 and p_8.

Point p_8 is now the one that controls the structure geometry. It is important to note that no computation of solutions is required by these changes of dependencies because the positions remain unchanged. Another example will be introduced at the end of Section 6.

5.4. Preventing two line segments from intersecting one another

Many practical applications are strut-and-tie networks that are required to remain inside a given area (Fig. 41). In order to ensure consistency, these areas are propagated on every domain of solutions associated with the strut-and-tie network. Although it is easy to constrain a point inside a given region by direct application of constraints, to constrain an entire rod inside a given region is less straightforward. Fig. 38 shows a rod that is not entirely inside a region although its extreme points are constrained inside that region. This issue can here be settled by applying particular constraints on both extreme points of the rod and the boundary segment it crosses. In concrete terms, the two line segments p_0p_1 and p_2p_3 never cross if the point p_0 is constrained as follows (Figs. 39 and 40):

$p_0 \in \text{HalfPlane}[p_1p_1p_A] \cup \text{HalfPlane}[p_Bp_Bp_1] \cup \text{HalfPlane}[p_Bp_Bp_A]$

where : $p_M \in \text{MidPoint}[p_2p_3]$

$p_A \in (\text{Position}[p_2] \cup \text{Position}[p_3]) \cap \text{HalfPlane}[p_1p_1p_M]$

$p_B \in (\text{Position}[p_2] \cup \text{Position}[p_3]) \cap \text{HalfPlane}[p_Mp_Mp_1]$.

This definition remains valid when p_1, p_2 and p_3 are aligned and when p_2 and p_3 are coincident.

6. Case study

The following figures briefly simulate the reconstruction of the structural skeleton of Robert Maillart's Chiasso sheds (1924).

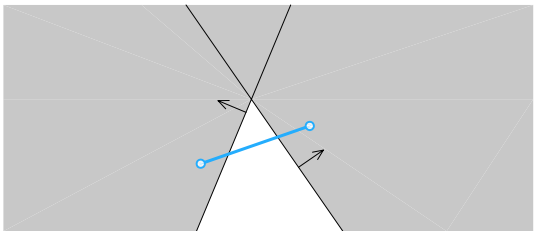

Fig. 38. A rod that is not totally inside a given region.

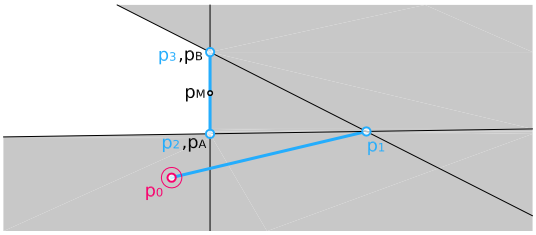

Fig. 39. The domain that p_0 must hold in order to prevent the intersection of the two line segments; first example.

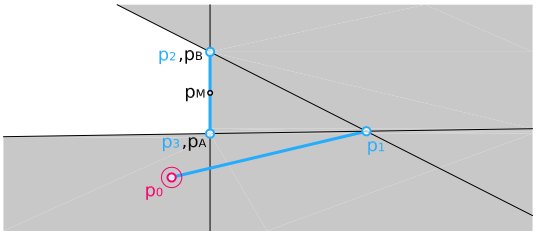

Fig. 40. Second example.

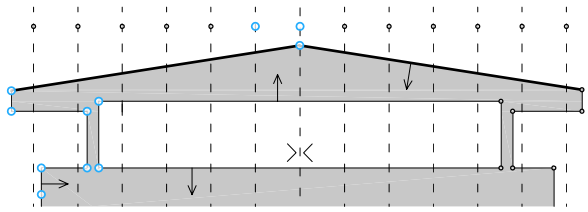

Fig. 41. Initial buildable volume.

Other applications may be found in [19]. First the buildable volume is defined as a Boolean combination of HalfPlane constraints (Fig. 41) and a first load case is discretized and applied on a Straightedge constraint representing the roof (Fig. 42) using the operations CreateZeroForce, ResolveForce and MovePoints. Unlike Maillart's original design process [20], the goal here is not to find a way to close the force polygon by successive adjustments in the form diagram. Rather the role of the designer is to focus on what operations should be applied to transform the initial equilibrium until the only remaining forces are the loads applied

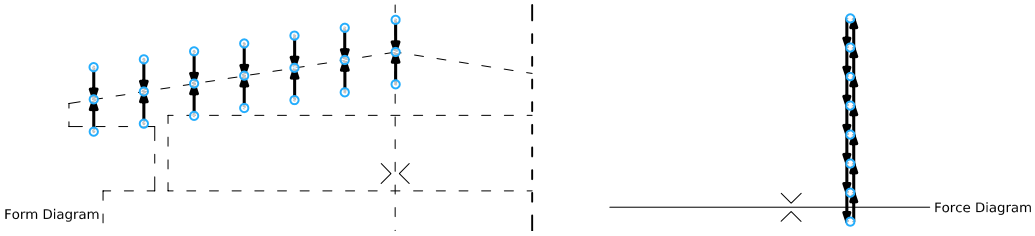

Fig. 42. Application of initial loads on one half of the shed.

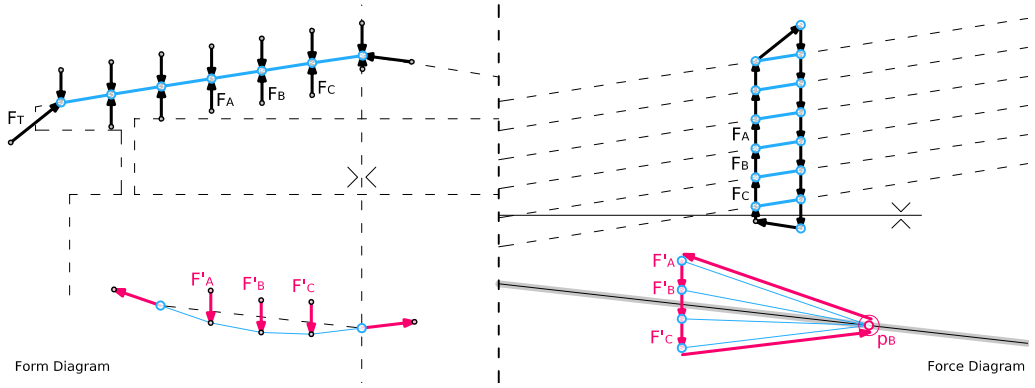

Fig. 43. Two sub-networks prior to combination.

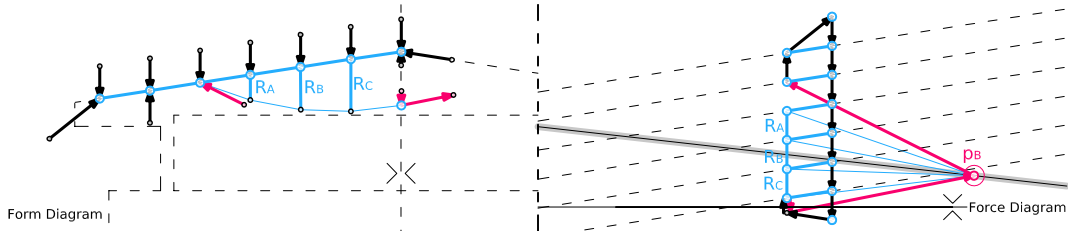

Fig. 44. The resulting network after combination.

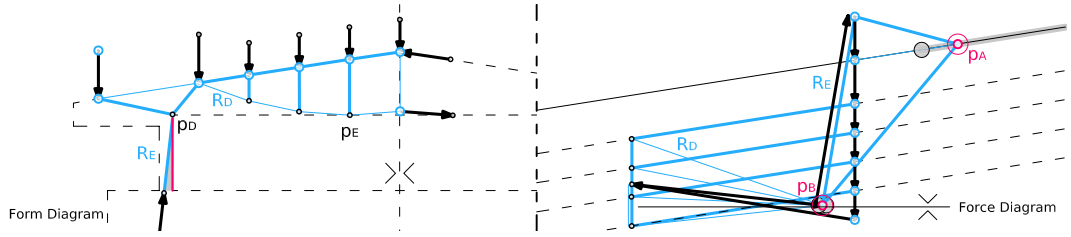

Fig. 45. Final network; the linear domains of p_A and p_B are highlighted.

on the structure or the reactions at the supports. For instance, Fig. 43 shows two sub-networks being combined together in Fig. 44 with the operations CreatePrimitiveConstraint, ApplyConstraint and CreateRod: the pairs of forces $F_AF'_A$, $F_BF'_B$ and $F_CF'_C$ form the rods R_A, R_B and R_C. Fig. 45 shows the final strut-and-tie network. In some cases, rods (*e.g.* R_D) and their magnitudes directly provide the orientation and width of the effective structural members. In other cases, the eccentricity between a rod (*e.g.* R_E) and the corresponding line of centroids multiplied by the magnitude of the rod measured in the force diagram provides the bending moment occurring in the structural member.

Since every rod in the form diagram is constrained to remain inside the buildable volume, this buildable volume is propagated onto each movable point of the final network. The solution domains of p_A and p_B for example (Fig. 45) consequently synthesize the remaining degrees of freedoms with which this structural shape can be altered (Fig. 46).

The impact of these alterations can be studied simultaneously in two ways: according to spatial considerations and mechanical considerations. For instance, p_C (defining the buildable volume) can be dragged to see how the buildable volume modifies the extreme value p_A^* of the domain of p_A and hence to see what the allowable orientations are that minimize the stresses inside R_F and R_G. As a consequence, specific design objectives can be simply achieved by dragging points (*e.g.* the smaller the force diagram, the less volume of material is needed) or by applying new geometric constraints (*e.g.* compelling p_A to remain inside a disc centered in p_F and whose radius is given by the maximum allowable tensile strength in R_H, Fig. 47).

Moreover, other load cases may be studied by varying the points that define the loads in the force diagram. For instance the position of p_E can be dragged (Fig. 48) to see how the bending moments in the column R_E fluctuate when the load F_Y increases or has another orientation.

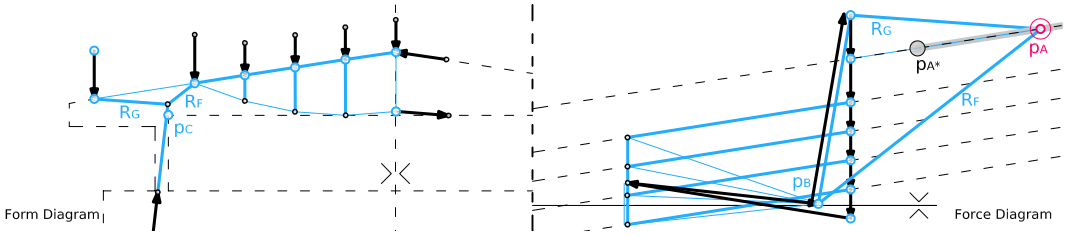

Fig. 46. Alternative obtained after displacement of p_A.

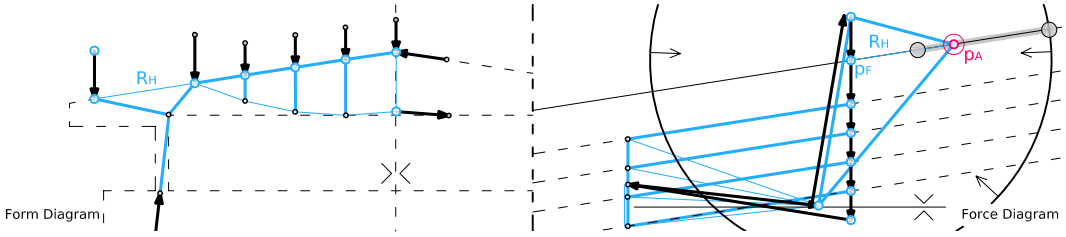

Fig. 47. Constraining p_A to control the tensile strength inside R_H.

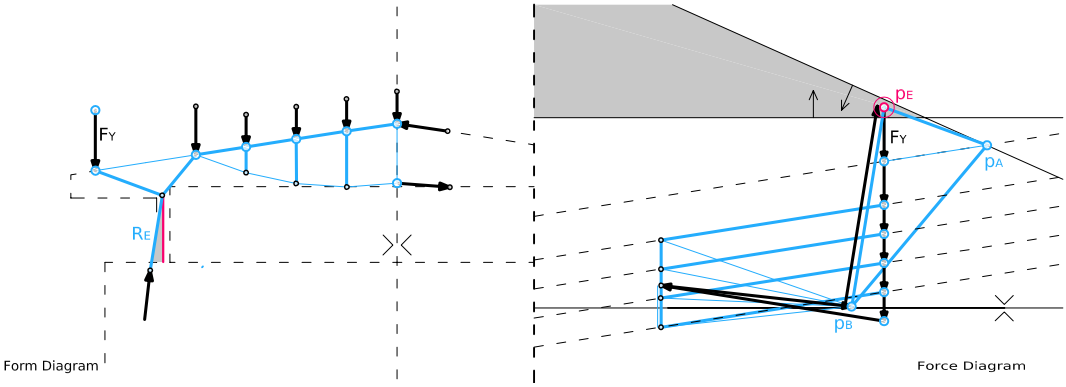

Fig. 48. Alternative obtained after displacement of p_E.

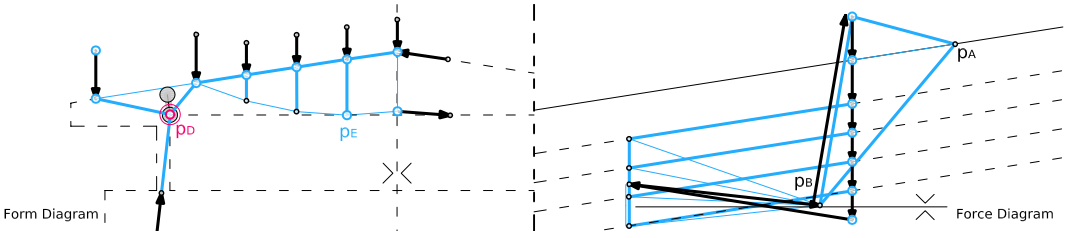

Fig. 49. Domain of p_D after switching the hierarchy between p_A and p_D.

In the current construction, dragging p_A (respectively p_B) has the effect of changing the position of p_D (respectively p_E) (Fig. 45). That means that two points in the force diagram (p_A and p_B) control the geometry of the structure. If the user now wishes to control the structure by moving points in the form diagram in order to see how the force diagram reacts, he simply has to switch the dependencies between p_A and p_D and between p_B and p_E (Fig. 49). This operation is fully automated (Section 5.3) and no reconstruction of the parameterization of both diagrams is needed.

7. Conclusion

This paper has introduced computational techniques capable of supporting the interactive definition of any plane structural equilibrium within two reciprocal diagrams, in a fully geometric environment.

Following an introductory section and an overview of the main concepts, the core functioning has been described in three sections. Section 3 introduced the fully geometric nature of the various data. Section 4 presented native operations whose sequential call can be made by the designer to modify and create any strut-and-tie network in equilibrium. Some algorithms were developed in Section 4 in order to outline how the described environment allows constructive and coordinate-free propagations to be performed. Section 5 illustrated how native operations can be combined to offer more intuitive and specific purposes. Finally, a brief case study illustrated some practical benefits of this approach in Section 6.

Applications comprise all those described by finite models of struts and ties. Although specific parts of the internal solver need more in-depth research, the fully geometric approach already demonstrates interesting practical capabilities such as the coordinate-free switching of the dependencies hierarchy, the

execution of dynamic conditions by means of static constraints, the design-oriented alteration of load-cases and the propagation of boundary conditions. Other capabilities, not presented in this paper, are also within reach. They involve the control of structural indeterminacy with graphical regions, the direct alteration of bending moments and beam deflections or the fully graphical execution of certain optimizations.

As a result, this computer-aided environment supports the renaissance of a forgotten paradigm of structural design. Indeed, the common "shape sketching > analysis > sizing" scheme can be mixed up by giving the structural designer the ability to explore the structural behavior interactively, in a chronology-free process, while controlling both the form and the forces simultaneously.

References

[1] Muttoni A, Schwartz J, Thürlimann B. Design of concrete structures with stress fields. Basel, Boston and Berlin: Birkhäuser; 1997.
[2] Heyman J. The stone skeleton: structural engineering of masonry architecture. Cambridge University Press; 1995.
[3] Heyman J. Navier's straitjacket. Architect Sci Rev 1999;42(2):91–5.
[4] Ochsendorf J. Practice before theory: the use of the lower bound theorem in structural design from 1850–1950, In: Huerta S, Instituto Juan de Herrera editors, Essays: the history of the theory of structures, 2005, p. 353–366.
[5] Maxwell JC. On reciprocal figures and diagrams of forces. Phil Mag 1864;27(4): 250-12.
[6] Culmann C. Die graphische statik. Zürich: Meyer und Zeller; 1866.
[7] Greenwold S, Allen E. Active statics. Cambridge: MIT; 2003.
[8] VanMele T, Rippmann M, Lachauer L, Block P. Geometry-based understanding of structures. JIASS 2012;53(4): 285-11.
[9] Dohmen M. A survey of constraint satisfaction techniques for geometric modeling. Computers & Graphics 1995;19(6): 831-15.
[10] Hoffmann CM, Joan-Arinyo R. A brief on constraint solving. Comput Aided Des Appl 2005;2(5):655–9.
[11] Bettig B, Hoffmann CM. Geometric constraint solving in parametric CAD. J Comput Inf Sci Eng 2011;11(2):26.
[12] Veltkamp RC. A quantum approach to geometric constraint satisfaction. In: Laffraet C, et al., editors. Object-oriented programming for graphics, eurographics. The European Association for Computer Graphics; 1995.
[13] Richter-Gebert J, Crapo H, Kortenkamp UH. Cinderella—the interactive geometry software. www.cinderella.de; first released in 1998.
[14] Laborde JM, Marcadet M. Cabri Geometry II Plus. Cabrilog Company, www.cabri.com, first released in 2002.
[15] Hohenwarter M, Borcherds M. Geogebra, free mathematics software for learning and teaching. www.geogebra.org, first released in 2002.
[16] Zaleswki W, Allen E. Shaping structures: statics. Wiley & Sons; 1997.
[17] Fivet C. Constraint-based graphic statics [Ph.D. Thesis]. Belgium: UCLouvain; 2013.
[18] Radon J. Mengen konvexer Körper, die einen gemeinsamen Punkt enthalten. Math Ann 1921;83(1–2): 113-3.
[19] Fivet C, Zastavni D. Constraint-based graphic statics: new paradigms of computer-aided structural equilibrium design. JIASS 2013;54(4): 271-10.
[20] Zastavni D. The structural design of Maillart's Chiasso shed (1924): a graphic procedure. Struct Eng Int 2008;18(3): 247-6.