A Case Study in Geometric Constructions

Étienne Schramm and Pascal Schreck[⋆]

Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection
UMR CNRS - Université Louis Pasteur 7005
Pôle API, Boulevard Sébastien Brant
F-67400 Illkirch - France
etienne@axis.u-strasbg.fr, schreck@dpt-info.u-strasbg.fr

Abstract. The geometric constructions problem is often studied from a combinatorial point of view: a pair data structure + algorithm is proposed, and then one tries to determine the variety of geometric problems which can be solved. Conversely, we present here a different approach starting with the definition of a simple class of geometric construction problems and resulting in an algorithm and data structures. We show that our algorithm is correct, complete with respect to the class of simply constrained polygons, and has a linear complexity. The presented framework is very simple, but in spite of its simplicity, this algorithm can solve non-trivial problems.

1 Introduction

Handling geometric objects declaratively described by a system of geometric constraints remains an important issue in CAD, even in the 2D case. To make feasible this manipulation, a geometric solver is always needed to find one, some, or all the solutions of a constraint system. Geometric constructions in CAD have been studied by various authors (see e.g.[2, 6, 3, 4, 9, 11, 10, 12, 7]). To this date, the most famous methods are based on a combinatorial analysis of the problem to be solved: the geometric side is quickly forgotten by giving a graph structure which is used to do constraint propagation [6, 1], maximal flow search [7] and/or problem decomposition [6, 10, 7]. In these approaches, different notions of correctness are defined and studied (e.g. in [6, 7]). We consider here "semantic" correctness, that is: the discovered figures are solutions of the constraint system.

We present a complete study in a very simple framework: we consider *simply constrained polygons* where the constraints can only concern the lengths of the edges and the angles between two edges of the polygon. We give an algorithm allowing to solve all the constraint schemes in this framework. We prove that our algorithm is correct and complete (*all* the solutions of the constraint system are found, in the degenerate cases zero solutions are found and explanations can be given). Moreover, we show that the class of complexity is $O(n)$ where n is the number of vertices. Our method is very simple and this paper can be

[⋆] To whom correspondence should be addressed regarded as a simple case study introducing the domain of geometric constructions.Nevertheless, the class of the solved problems is not trivial (e.g. see the example given on Figure 4) Furthermore, it seems possible to use this proper method in a multi-agent approach like the one described in [4]. Please, note that simply constrained polygons constitute a subclass of the problems solved by the Sunde method (see [13]), but, as far we know, there is no efficient implementation of this method.

In Sections 2 and 3 of this paper, we expose the geometric framework of the simply constrained polygons. In Section 4, we present and justify our resolution algorithm. In Section 5, we show how that algorithm can be efficiently implemented. And, finally, we give some concluding remarks in section 6.

2 Definition of Polygons by Lengths and Angles

A polygon is classically defined as a finite sequence of points in the Euclidean space with a fixed reference, that is , a polygon P is a function $P : [\![1, C]\!] \to \mathbb{R}^2$ where $[\![1, C]\!]$ is a finite interval of integers, $C > 2$, and C is the number of vertices of the polygon. Let us recall some usual notations: $|P| = C$ is the number of vertices, which is also the number of edges of P, $P(n)$ or simpler P_n denotes the n^{th} point of polygon P, $E(n)$ denotes the n^{th} oriented edge of P (i.e the oriented segment $[P(n), P(n+1)]$ if $n < |P|$ and $[P(n), P(1)]$ if $n = |P|$) and $\|E(n)\|$ its length. At last, we note $\mathbb{P}$ the set of the polygons of the plane.

For the purposes of this paper, we consider two functions families over the polygons namely the *polygon constructive functions* and the *permutation functions* which are defined as follows:

Definition 1. *The set of the polygon constructive functions is the set* $CF = \{f_{n,a,l} | n \in \mathbb{N}^*, a \in \mathbb{R}, l \in \mathbb{R}_+^*\}$ *where each function* $f_{n,a,l}$ *is defined by:*

$$f_{n,a,l} : \mathbb{P} \to \mathbb{P}$$

$$\forall m \in [\![1, |P|+1]\!] \qquad f_{n,a,l}(P)(m) = \begin{cases} P_m & \text{if } m < n \\ Q & \text{if } m = n \\ P_{m-1} & \text{if } m > n \end{cases}$$

with point Q defined by:

$$\begin{pmatrix} x_Q \\ y_Q \end{pmatrix} = \frac{l}{P_{n-1}P_n} \begin{pmatrix} \cos(a) & -\sin(a) \\ \sin(a) & \cos(a) \end{pmatrix} \begin{pmatrix} x_{P_n} - x_{P_{n-1}} \\ y_{P_n} - y_{P_{n-1}} \end{pmatrix} + \begin{pmatrix} x_{P_{n-1}} \\ y_{P_{n-1}} \end{pmatrix}$$

Figure 1 a) illustrates such a construction. It is easy to verify that point Q is the unique point such that $P_{n-1}Q = l$ and $\angle(\overrightarrow{P_{n-1}P_n}, \overrightarrow{P_{n-1}Q}) = a$.

Let us define the other family of functions over the polygons:

Definition 2. *The set of permutation functions is the set* $PF = \{g_n | n \in \mathbb{N}^*\}$ *where each function* g_n *is defined by:*

$$g_n : \mathbb{P} \to \mathbb{P}$$

$$\forall m \in [\![1, |P|]\!] \qquad g_n(P)(m) = \begin{cases} P_m & \text{if } m \neq n+1 (mod|P|) \\ S & \text{otherwise} \end{cases}$$

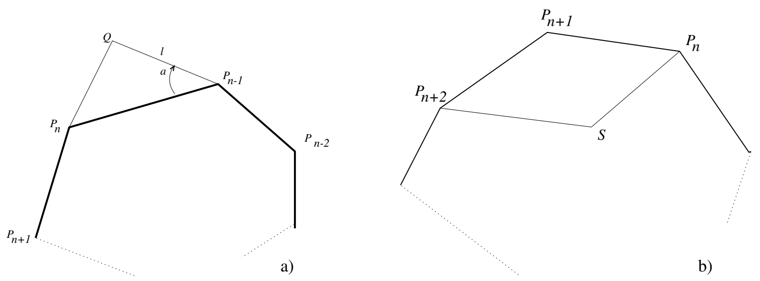

Fig. 1. Action of a polygon constructive function (a) and a permutation function (b)

with point S defined by:

$$\begin{pmatrix} x_S \\ y_S \end{pmatrix} = \begin{pmatrix} x_{P_{n+2}} + x_{P_n} - x_{P_{n+1}} \\ y_{P_{n+2}} + y_{P_n} - y_{P_{n+1}} \end{pmatrix}$$

Figure 1 b) shows the effect of the permutation function g_n over a polygon. Note that S is the unique point verifying $\overrightarrow{P_nS} = \overrightarrow{P_{n+1}P_{n+2}}$.

It is easy to see that any polygon P can be generated by the polygon constructive functions applied successively to the triangle (P_1, P_2, P_3) (see Figure 2 for a "graphic idea" of the proof). This assertion can be transformed into: let $C = |P|$, knowing triangle (P_1, P_2, P_3), the distances P_3P_4, ..., $P_{C-1}P_C$ and the oriented angles $\angle(\overrightarrow{P_3P_2}, \overrightarrow{P_3, P4})$, ... $\angle(\overrightarrow{P_{C-1}P_{C-2}}, \overrightarrow{P_{C-1}, P_C})$, one can re-construct the polygon P using the functions $f_{a,n,l}$

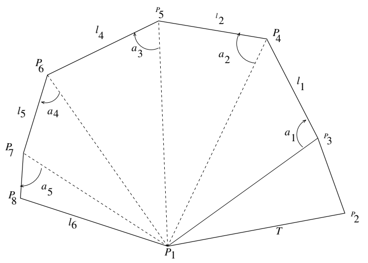

Fig. 2. Construction of a polygon using the polygon constructive functions

We will see below that, thanks to the permutation functions, this constraint scheme can be enlarged to the simply constrained polygons. But beforehand, we have to consider the way to get the triangle (P_1, P_2, P_3) from lengths and angles.

The construction of a triangle knowing the length of its three edges (resp. two lengths and one angle, or one length and two angles) is easy and well known, but this simplicity conceals two important problems of the geometric constructions in CAD. For instance, let us consider the function $t_1 : \mathbb{R}_+^* \times \mathbb{R}_+^* \times \mathbb{R}_+^* \to \mathbb{P}$ computing a triangle having the three given lengths. First problem, there is an infinity of triangles having these lengths: each of them can be deduced from one of two particular solutions using a direct isometry (also called a displacement or a rigid body motion). Such a triangle is told to be constructed modulo the displacements group. Here, we simply choose to fix point P_1 at $(0,0)^t$ and P_2 on the Ox axis. But, and this is the second problem, how to deal with the two or more particular solutions ? Some studies have been done on this subject considering some heuristics and/or similarities with a sketch given by the user ([6, 5]). In our simple framework, we have only, at most, two constructions to consider, so, we choose to keep both. The function t_1 is now defined by:

$$t_1 : \mathbb{R}_+^* \times \mathbb{R}_+^* \times \mathbb{R}_+^* \to \mathbb{P} \times \mathbb{P}$$

with $(P', P'') = t_1(l_1, l_2, l_3)$, $P_1' = P_1'' = (0,0)^t$, $P_2' = P_2'' = (l_1, 0)^t$ and

$$P_3' = \begin{pmatrix} \frac{l_1^2+l_3^2-l_2^2}{2l_1} \\ \sqrt{l_3^2 - \left[\frac{l_1^2+l_3^2-l_2^2}{2l_1}\right]^2} \end{pmatrix} \text{ and } P_3'' = \begin{pmatrix} \frac{l_1^2+l_3^2-l_2^2}{2l_1} \\ -\sqrt{l_3^2 - \left[\frac{l_1^2+l_3^2-l_2^2}{2l_1}\right]^2} \end{pmatrix}$$

Figure 3 shows the four cases of triangle construction (with, eventually some permutations of edges) and the construction of the two triangles in the cases of functions t_1 and t_3.

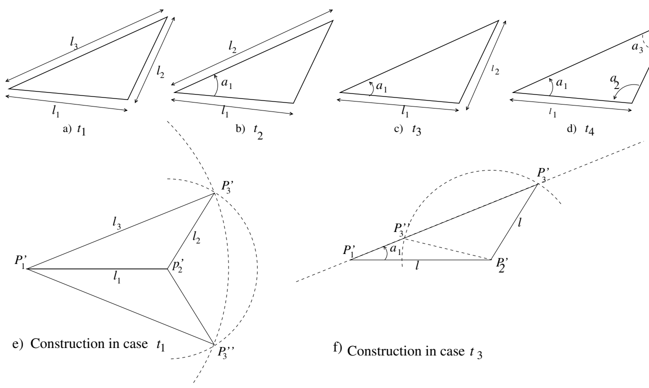

Fig. 3. Four triangle cases (a) to (d) and construction for t_1 (e) and t_3 (f)

The following definition makes the things clearer. In this definition, all the triangles (P_1, P_2, P_3) verify $P_1 = (0,0)^t$ and $P_2 = (l_1, 0)^t$.

Definition 3. *A triangle constructive function is one of the four partial functions t_1, t_2, t_3 and t_4 which are defined by:*

$$\begin{aligned} t_1 &: \mathbb{R}_+^* \times \mathbb{R}_+^* \times \mathbb{R}_+^* \to \mathbb{P} \times \mathbb{P} \\ t_2 &: \mathbb{R}_+^* \times \mathbb{R}_+^* \times \mathbb{R} \to \mathbb{P} \times \mathbb{P} \\ t_3 &: \mathbb{R}_+^* \times \mathbb{R}_+^* \times \mathbb{R} \to \mathbb{P} \times \mathbb{P} \\ t_4 &: \mathbb{R}_+^* \times \mathbb{R} \times \mathbb{R} \to \mathbb{P} \times \mathbb{P} \end{aligned}$$

– t_1 *is defined as before,*
– $t_2(l_1, l_2, a_1) = (P, P)$ *with* P *such that* $P_1P_3 = l_2$ *and* $\angle(\overrightarrow{P_1P_2}, \overrightarrow{P_1P_3}) = a_1$
– $t_3(l_1, l_2, a_1) = (P', P'')$ *where* P' *and* P'' *are the two polygons such that* $P'_2P'_3 = P''_2P''_3 = l_2$ *and* $\angle(\overrightarrow{P''_1P''_2}, \overrightarrow{P''_1P''_3}) = \angle(\overrightarrow{P'_1P'_2}, \overrightarrow{P'_1P'_3}) = a_1$ *(see Figure 3.f)*
– $t_4(l_1, a_1, a_2) = (P, P)$ *with polygon* P *such that* $\angle(\overrightarrow{P_1P_2}, \overrightarrow{P_1P_3}) = a_1$ *and* $\angle(\overrightarrow{P_2P_3}, \overrightarrow{P_2P_1}) = a_2$

Thus, we extend classically our families of functions CF and PF to the ordered pairs of polygons in order to have the following result: *every polygon can be constructed thanks to the functions of CF, PF and* t_1*,* t_2*,* t_3 *and* t_4 *modulo the displacements group.* We can now examine the notion of constraint scheme.

3 Constraint Schemes and Simply Constraint Polygons

A length constraint (over a polygon) is an ordered pair (n, l) where n is a natural integer and l a non negative real. We say that a polygon P satisfies the constraint (n, l) if $n \leq |P|$ and $\|E(n)\| = l$.

Definition 4. *A set of length constraints* L *is* coherent *if* $(n, l) \in L$ *and* $(n, k) \in L$ *leads to* $k = l$*.*

We note $C_L(n)$ the predicate indicating if the constraint (n, l) is in L or not. When $C_L(n)$ is true, we note $\texttt{length}(n, L)$ the imposed value in L for the edge $E(n)$.

An angle constraint is a triple (n, m, a) where n and m are natural integers, with $n \neq m$ and a a real number. We say that a polygon P satisfies the constraint (n, m, a) if $n \leq |P|$, $m \leq |P|$ and the oriented angle between $E(n)$ and $E(m)$ is equal to a, i.e. $\angle(E(n), E(m)) = a$. Given a set of angle constraints A, we can define the *associated unoriented angle graph* $G(A)$ whose nodes are the segments $E(i)$ and whose unoriented edges correspond to the angle constraints between the segments *i.e.* $\{E(i), E(j)\} \in G(A)$ if there is a such that $(i, j, a) \in A$ or $(j, i, a) \in A$. The transitive closure of the angle graph corresponds to the Chasles relation ($\angle(\overrightarrow{OA}, \overrightarrow{OB}) + \angle(\overrightarrow{OB}, \overrightarrow{OC}) = \angle(\overrightarrow{OA}, \overrightarrow{OC})$). This fact justifies the definition:

Definition 5. *A set of angle constraints* A *is* coherent *if there is no circuit in the associated angle graph.*

As before, we note $C_A(n_1, n_2)$ the predicate indicating if the edges $E(n_1)$ and $E(n_2)$ are angle constrained or not. That is, $C_A(n_1, n_2)$ is true if and only if there is a path between $E(n_1)$ and $E(n_2)$ in the unoriented graph $G(A)$. Then, we note $\texttt{angle}(n_1, n_2, A)$ the value imposed by the angle constraints in A and computed using the Chasles relation.

A scheme of constraints is an ordered triple $S = (L, A, n)$ containing a set of length constraints, a set of angle constraints and the number n of edges considered (which is noted $N(S)$). We impose that n is greater than the maximal edge number appearing in the sets of constraints. A constraint scheme S is well-constrained if there is a finite number of polygons P such that P satisfies all the constraints in S, $|P| = N(S)$, $P_0 = (0,0)^t$ and P_1 is on Ox. This definition leads classically (e.g. see [6]) to the definition of a *structurally well-constrained* scheme of constraints which is *purely combinatorial* and which meets the previous definition when the constraints values are algebraically independent and the complex solutions are considered. Surprisingly, due to the simplicity of the framework, simply constrained polygons have always at most two solutions (possibly zero). So, if a constraint scheme is not a simply constrained polygon: either there are redundant constraints, or the system is not well-constrained. Then, we have the following theorem:

Theorem 1. *A constraint scheme* (L, A, n) *is structurally well-constrained if and only if* L *and* A *are coherent and either* $|L| = n$ *and* $|A| = n - 3$*, either* $|L| = n - 1$ *and* $|A| = n - 2$ *or either* $|L| = n - 2$ *and* $|A| = n - 1$*. We call such a constraint scheme a* simply constrained polygon*.*

Sketch of the proof: The necessary condition comes immediately from the facts $|A| + |L| = 2n - 3$, $|A| \leq n$ and $1 \leq |L| \leq n$. It is relatively easy, but too long for this paper, to prove by recurrence that this cardinality condition is sufficient: the idea is to use on a constraint scheme the equivalent operations than the ones used on the polygons constructive functions and the permutation functions). This idea is also used in the analysis of a constraint scheme given below. □

Figure 4 shows, graphically, an example of a simply constrained polygon $S = (L, A, n)$ where $L = \{(1, k_1), (2, k_2), \dots, (6, k_6)\}$ and $A = \{(1, 4, a_1), (2, 5, a_2), (3, 6, a_3)\}$. It becomes a well constrained scheme when the formal parameters k_1, k_2 $\dots k_6$ and a_1, $\dots a_3$ are instantiated with appropriate values.
The objective is then to take a well constrained schema S in order to yield one (or two) polygons P which satisfies all the constraints in S.

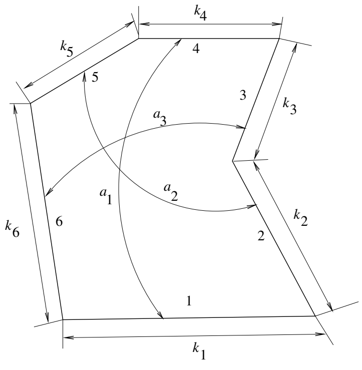

Fig. 4. A structurally well-constrained polygon

4 Analysis and Construction

From the Theorem 1 of the previous section, the following result can be deduced:

Theorem 2. *Given $S = (L, A, n)$ a simply constrained polygon with $n > 3$, there are two edges $n_1 \in \{1, 2, 3, 4\}$ and $n_2 \in \{1, 2, 3, 4\}$, with $n_1 \neq n_2$, such that $C_L(n_1)$, $C_L(n_2)$ and $C_A(n_1, n_2)$.*

Proof. Since $G(A)$ has no circuit, each addition of an edge decreases the number of connected components by 1. So, in case $|A| = n - 3$, there are exactly three connected components in $G(A)$. The pigeon hole principle shows that $\exists n_1 \in \{1, 2, 3, 4\}$ and $n_2 \in \{1, 2, 3, 4\}$ such that $C_A(n_1, n_2)$ and $n_1 \neq n_2$. Since $|L| = n$ in this case, the result is obvious. The other two cases ($|A| = n-2$ and $|A| = n-1$) can be treated by the same use of the pigeon hole principle. □

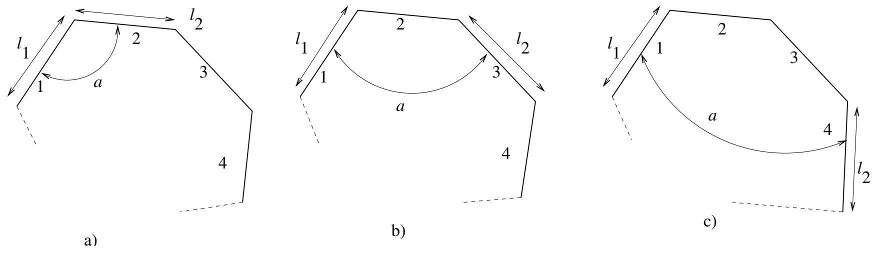

Fig. 5. The three cases

Thanks to Theorem 2, we can see that there are fundamentally three cases examining the constraints over the first four edges: the angle and length constrained edges are consecutive, or separated by one edge, or separated by two edges. These cases are graphically represented on Figure 5 .Let us examine the c) case and explain how to simplify it. The c) case corresponds to the constraint scheme $L = \{(1, l_1), (4, l_2), \ldots\}$ and $A = \{(1, 4, a), \ldots\}$. This constraint scheme can be transformed in a way that is similar to the permutation function: $S = (L, A, n)$ becomes $S' = (L', A', n)$ where L' and A' are obtained from L and A replacing 3 by 4 and vice-versa. Next, we apply the same operation exchanging 3 with 2 (See Figure 6) giving the scheme S''. It is easy to demonstrate that P is a solution of S'' if and only if $g_2(g_3(P))$ is a solution of S. Then, we are in the situation described on Figure 7 left: edges 1 and 2 are constrained in length and in angle. The construction of triangle (ABC) is quite usual and requires only standard formulae: so the distance AC and the angle b can easily be computed. Then, we obtain a new constraint scheme S''' computed from S'' by forgetting edges 1 and 2, adding a new edge (say $1'$) and transferring all the angle constraints between edge 1 or 2 and the other edges to the new edge $1'$ using computed angle b and, finally, by adding the length constraint $(1, l'_1)$ to L''. It is still easy to demonstrate that polygon P satisfies the scheme constraint S''' if and only if $f_{1,b,l_1}(P)$ satisfies S''. Note that the resulting scheme is still structurally well constrained and has one edge fewer than S.

The other cases are very similar. This leads us to propose the following algorithm to solve a simply constrained polygon:

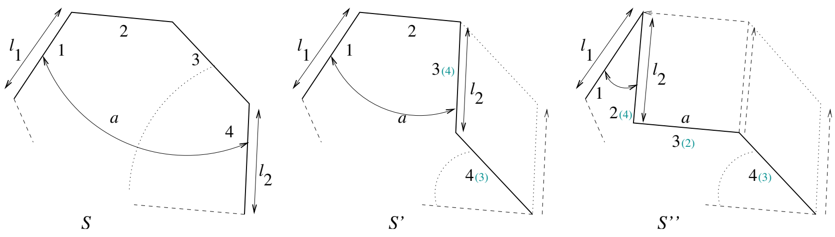

Fig. 6. Beginning of the analysis (1): permutations

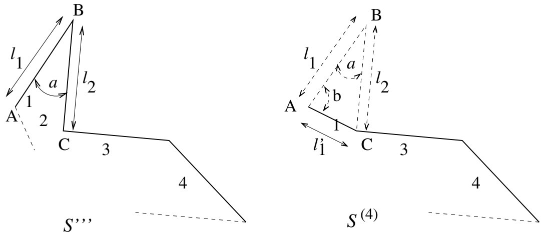

Fig. 7. Beginning of the analysis (2): triangle construction

```
input: simply constrained polygon
output: polygon
solve(S) = if triangle(S) then solve_triangle(S)
  elif case_a(S) then build_case_a(coefs_a(S),solve(simplify_case_a(S)))
  elif case_b(S) then build_case_b(coefs_b(S),solve(simplify_case_b(S)))
  elif case_c(S) then build_case_c(coefs_c(S),solve(simplify_case_c(3)))
fi
```

where functions `triangle()` and `solve_triangle()` are in charge of the triangle cases resolution, functions `case_`X() test if the constraint scheme is in case a), b) or c). Then, functions `simplify_case_`X() make the simplification following the description above given, while functions `coefs_`X() compute the corresponding dimensions in order to apply the correct construction and permutation functions thanks to *meta-functions* `build_case_`X(). We have then the following theorem:

Theorem 3. *The previous algorithm is correct and complete with respect to the simply constrained polygons.*

That means the algorithm finds all the solutions whatever the simply constrained polygon to be solved. In the degenerate cases, no solutions are found (this is correct) and explanations can be given to the user. Since, in this paper, the auxiliary functions, and hence the algorithm, are not formally described, it is impossible to prove rigorously here this theorem. Let us say that the correctness comes from the facts that, first, the functions `triangle()` and `solve_triangle()` are correct and, second, that if S is in the case X (with $X \in \{a, b, c\}$) then this is correctly discovered by the `case_`X() function and, `build_case_`X`(coef_`$X(S), P)$

is a solution of S if and only if P is a solution of `simplify_case_`$X(S)$. The description of the simplification above given should convince the reader of these facts. The completeness comes from Theorem 2 and the fact that all the cases are handled by the algorithm.

5 Implementation

Although we have functionally and recursively described our algorithm, we have carried out an imperative procedural implementation. We have shown that, with adequate data structures, all the auxiliary functions given in previous section have a $O(1)$ complexity in space and time.

Note first, that the algorithm can easily be put under a procedural form using a stack in order to keep in memory the construction functions to apply and which were discovered during the analysis of the constraint scheme. By this way, the transformed algorithm consists in two phases: the analysis of the constraint scheme and the re-construction of the resulting polygon.

In the first phase, the algorithm has to detect if the simply constrained polygon is a triangle or not. In the first case, the resolution is well known and is done in constant time. If it is not a triangle, the system searches, with the first four edges, what is the simplification case. This can be done easily by examining the data structures containing the constraints. We chose to implement the set of angle constraints with a forest where each tree has a depth 1 and each node contains a pointer to the root. The construction of such a structure is achieved in linear time and the search for an angle constraint between two edges is achieved in constant time. So, it is easy to see that the functions `case_`X are executed in constant time. The analysis and the simplification are implemented according to the description given above. It is obvious that each step can be performed in constant time. Since there are $n - 3$ steps, the complexity of the analysis/simplification phase is $O(n)$.

In the second phase, the system uses all the informations collected during the first phase and which are kept in the stack. These informations indicate which functions, with which parameters and in which order to call, in order to build the polygon from the "initial" triangle yet constructed. Each step of the construction consists in the application of one, two or three functions. So, it is achieved in constant time. Thus, the entire construction is done in linear time.

It is easy to see that with our data structures the space complexity is $O(n)$ as well.

This algorithm has been implemented under the form of a C++ prototype. It was experimented on small examples, *i.e.*: simply constrained polygons with about ten points. For each example, the resolution was immediate.

6 Conclusion

We presented in this paper a simple framework in the geometric constraint solving domain. Examining the family of the simply constrained polygons, we found a general algorithm to solve the structurally well-constraint scheme. This algorithm is correct, complete and has a linear complexity.

Of course, this framework is only a case study and the resulting algorithm is not very powerful. But it can solve difficult problems. Furthermore it can be used by other solvers which perform decomposition as the one described in [4]. One can also imagine to generalize this approach to "constrained complex cellular" consisting in an assembling of simply constrained polygons.

References

1. S. Ait-Aoudia, R. Jegou and D. Michelucci: Reduction of constraint systems. In: *Proceedings of the Compugraphics Conference*. Compugraphics Alvor, Portugal(1993) 83-92.
2. B. Aldefeld, H. Malberg, H. Richter and K. Voss: Rule-based variational geometry in computer-Aided Design. In: D.T. Pham (ed):*Artificial Intelligence in Design*. Springer-Verlag (1992) 27-46.
3. B. Brüderlin: Automatizing geometric proofs and constructions. In:*Proceedings of Computational Geometry '88*. Lecture Notes in Computer Science, Vol. 333. Springer-Verlag (1988) 232-252.
4. J.-F. Dufourd, P. Mathis, and P. Schreck: Formal resolution of geometric constraint systems by assembling. In:*Proceedings of the ACM-Siggraph Solid Modelling Conference, Atlanta*. ACM Press (1997) 271-284.
5. C. Essert-Villard, P. Schreck, and J.-F. Dufourd: Sketch-based pruning of a solution space within a formal geometric constraint solver. In: *Journal of Artificial Intelligence*, Num. 124. Elsevier (2000) 139-159.
6. I. Fudos and C. M. Hoffmann: Correctness proof of a geometric constraint solver. In: *International Journal of Computational Geometry and Applications* Num. 6. World Scientific Publishing Company(1996) 405-420.
7. C. M. Hoffmann, A. Lomonosov, M. Sitharam: Decomposition Plans for Geometric Constraint Systems, Part I: Performance Measures for CAD. In: *Journal of Symbolic Computation* Num. 31. Academic Press(2001) 367-408.
8. G.A. Kramer: A geometric constraint engine. In: *Artificial Intelligence* Num. 58. Elsevier(1992) 327-360.
9. H. Lamure and D. Michelucci: Solving constraints by homotopy. In: *Proceedings of the ACM-Siggraph Solid Modelling Conference*. ACM Press(1995) 134-145.
10. J. Owen: Algebraic solution for geometry from dimensional constraints. In: *Proceedings of the 1st ACM Symposium of Solid Modelling and CAD/CAM Applications*. ACM Press(1991) 134-145.
11. G. Sunde: Specification of shape by dimensions and other geometric constraints. In: *Proceedings of the Eurographics Workshop on Intelligent CAD systems*. Noordwisjkerout(1987).
12. I.E. Sutherland. Sketchpad: A man-machine graphical communication system. In: *Proceedings of the IFIP Spring Joint Computer Conference*.Detroit, Michigan(1963) 329-36.
13. A. Verroust, F. Schoneck and D. Roller: Rule-oriented method for parameterized computer-aided design. In: *Computer-Aided Design* Vol. 10 Num. 24. Elsevier(1992) 329-36.