ELSEVIER

Artificial Intelligence 105 (1998) 47–75

Artificial Intelligence

Empirically-derived estimates of the complexity of labeling line drawings of polyhedral scenes

P. Parodi *, R. Lancewicki, A. Vijh, J.K. Tsotsos

University of Toronto, Department of Computer Science, 6 King's College Road, Room 283, Toronto, Ontario, Canada M5S 3H5

Received 10 September 1996; received in revised form 16 December 1997

Abstract

Several results have been obtained in the past about the complexity of understanding line drawings of polyhedral scenes. Kirousis and Papadimitriou (1988) have shown that the problem of labeling line drawings of trihedral scenes is NP-complete. The human brain, however, seems to grasp at a glance the 3D structure associated with a line drawing. A possible explanation of this discrepancy, offered by Kirousis and Papadimitriou themselves, is that the worst-case complexity does not reflect the real difficulty of labeling line drawings, which might be far less in the average or in "typical" cases. However, no statistical analysis has ever been carried out to test this conjecture.

The core of this paper is an algorithm for the generation of random instances of polyhedral scenes. Random instances of line drawings are then obtained as perspective projections of these scenes, and can be used as an input to standard labeling algorithms so as to derive experimental estimates of the complexity of these algorithms. The results indicate that the median-case complexity is linear in the number of junctions. This substantiates the conjecture that "typical" instances of line drawings are easy to label, and may help explain the ease by which the brain is able to solve the problem.

Keywords: Line drawings; Computational complexity; Random polyhedral scenes

1. Introduction

The problem of understanding the three-dimensional structure of an object from a concise two-dimensional description (e.g., a line drawing) of it has intrigued researchers in computer vision and artificial intelligence since the early seventies, when the first attempts to tackle the problem from a computational standpoint were independently put

* Corresponding author. Email: parodi@sissa.it.

0004-3702/98/$ – see front matter
PII: S0004-3702(98)00077-0

forward by Huffman [10] and Clowes [4]. They both showed that an important necessary condition for a line drawing to represent the perspective or planar projection of an actual arrangement of polyhedral objects was *labelability*, that is the consistent assignment to the segments of the line drawing of a label $(+, -, \rightarrow, \leftarrow)$ describing such 3D properties as convexity, concavity, occlusion. The work of Huffman and Clowes was extended in several directions [11,16,29,34], but it was not until the work of Sugihara [30–32] that a necessary *and sufficient* condition for the realizability of a line drawing was found. Sugihara showed that, given a *labeled* line drawing, the realizability problem could be translated into an instance of Linear Programming.

More recently, several efforts have been concentrated on complexity issues. As our brain is very efficient at reconstructing the 3D structure of a scene from a single image with no texture, color or shading, one might be led to conclude that there is an efficient (i.e., polynomial-time) algorithm that interprets line drawings, at least qualitatively (by labeling their segments). Kirousis and Papadimitriou [13], however, have proved that this is unlikely to be the case, by showing that both the labeling problem and the realizability problem are $\mathcal{NP}$-complete even for the simple case of trihedral, solid scenes. This unexpected result has stimulated much research in order to find special cases for which the labeling problem was polynomially solvable. In the same paper [13], Kirousis and Papadimitriou proved that the labeling problem has polynomial complexity for line drawings of Legoland scenes, i.e., scenes made of objects whose 3D edges can only have one of three possible orthogonal directions. This result was extended in [23] to show that once the location of the vanishing points of the line drawing of a trihedral, solid scene is known, the labeling problem becomes solvable in polynomial time. These results suggests that the brain may exploit geometrical regularities in order to find a 3D reconstruction of a scene from a line drawing. It was also shown [21] that the information on vanishing points is not sufficient to break the NP-completeness of labeling line drawing of Origami scenes, although this information drastically reduces the number of legal labelings associated with a line drawing.

Thus, a possible explanation for the discrepancy between the NP-completeness result might be that the brain uses geometric information which is often found in natural scenes. Another possible explanation, which was offered by Kirousis and Papadimitriou themselves, is that the distribution of natural scenes might be such that the average-case complexity for the set of line drawings extracted from real scenes is polynomial, unlike the complexity for general line drawings. A third possibility exists also. For a related visual problem, [33] proved the NP-completeness of unbounded visual search and that visual search becomes linear in the image size when a target is used to guide the matching process. The claim there is that the brain can optimize visual processing by using known appearance of objects and thus for the set of well-known objects, visual processing is efficient.

As it is well known for other problems (such as SAT [19,25] and CSP [3,9,35]), hard instances of a problem are often elusive and can only be found with a careful tuning of some characteristic parameters. Furthermore, heuristics have been presented—such as the relaxation procedure devised by Waltz [34]—which can be used in conjunction with tree-search methods and allegedly provide an efficient way to deal with line drawing labeling.

The objective of this paper is to provide a method to generate random instances of line drawings with a useful distribution, so as to shed light on several questions related to the complexity of understanding images of polyhedral scenes: what is the average-

case (either mean-case or median-case) complexity of labeling? How much do we gain by pre-processing our line drawings by relaxation techniques which achieve some kind of local consistency before performing tree-search? More generally, are the available computational techniques satisfaction techniques competitive with the performances of the brain? And can these performances be assessed more precisely, beyond the general notion that images of natural scenes are perceived at a glance by humans? Is this still true when scenes display a more random character?

The paper is organized as follows. Section 2 provides a general introduction to the labeling problem. Section 3 addresses the problem of generating random line drawings. A method is devised to generate random instances of polyhedral scenes; the random line drawings are obtained by projecting these scenes on an arbitrary image plane. These line drawings are then used in Section 4 to estimate the complexity of some tree-search methods for labeling line drawings and to assess the efficiency of relaxation and other heuristics. Section 5 draws the conclusions of the work and discusses the relation of this paper to previous works in line drawing analysis.

2. The labeling problem

The first mathematical results about the interpretation of line drawings date back to the works of Huffman [10] and Clowes [4], who independently introduced an important necessary condition (*labelability*) for the realizability of a line drawing as the 2D projection of a polyhedral scene.

Labeling a line drawing means assigning a label to every segment describing the properties of the corresponding 3D edge. The label "+" means that the segment is the projection of a visible convex edge, "−" means that it is the projection of a visible concave edge, and "→" means that it is the projection of a convex edge such that only one face (the one at the right of the arrow) is visible.

Huffman and Clowes focused on the case of trihedral scenes (exactly 3 faces meeting at every vertex; see, for example, the line drawing of Fig. 1(A)). They found that only junctions of a few different shapes (Y, E, L, T) are possible and that only a few combinations of labels are allowed at junctions if they were to be realizable as the projection of 3D vertices; these legal labelings form the so-called *junction dictionary* (see Fig. 1(B)).

A line drawing is said to be labelable iff a label can be assigned to every segment so that every junction is labeled according to this dictionary.

Several methods have been proposed in the past for labeling a line drawing. Huffman [10] and Clowes [4] proposed a reduction to SAT. Waltz [34] devised a filtering algorithm which reported good average running time (roughly linear in the number of segments). The algorithm achieves local consistency in this way: given a junction, rule out all legal labelings of the junction for which there is no labeling of the neighbor junctions which is compatible with it. Repeat this procedure until no further progress can be made. To label a line drawing, first achieve local consistency and then achieve global consistency by tree searching with depth-first backtracking.

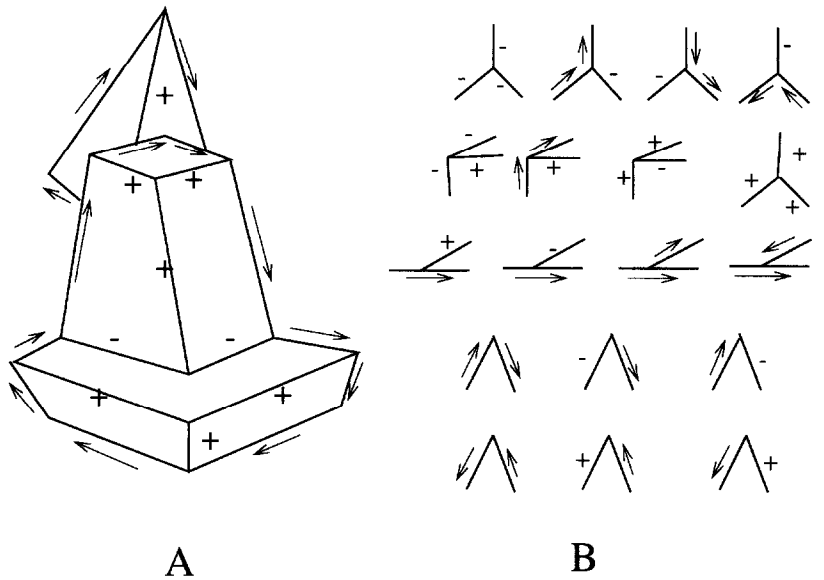

Fig. 1. (A) An example of a trihedral scene. (B) The Huffman–Clowes catalogue of legal labelings for trihedral vertices.

The work on line drawings has stimulated the analysis of a more general problem, the Constraint Satisfaction Problem (CSP; see Mackworth and Freuder [15] for a retrospective): we have a set of variables $X_1, \ldots, X_n$ each of which may assume a finite set of values, and there is a set of constraints of the kind "value a of variable X_i is not compatible with value b of variable X_j". The problem is to find a set of assignments for each variable that satisfies all constraints. This problem has been shown to be $\mathcal{NP}$-complete. Labeling is a special case of CSP. Waltz' procedure can be used to simplify this problem and is called, in this framework, "arc consistency".

The reason why the algorithm performed with linear average running time is that the arc consistency algorithm used by Waltz takes indeed linear time in the number of segments (see Mackworth [14]), and it is often the case that at the end of the relaxation procedure only a few segments are not already uniquely labeled, at least in the trihedral world with cracks and shadows studied by Waltz. Even this approach to labeling, however, is worst-case exponential.

The CSP approach is now the standard one used for labeling line drawings. It has been applied to line drawings of polyhedral as well as piecewise smooth curved line objects (see Malik [16]). Although this approach is in general worst-case exponential, in several meaningful cases the constraints are of a kind which allows unique propagation rules for the constraints and efficient sequential and parallel algorithms (Kirousis [12]).

In this paper we will use both the basic trihedral world introduced by Huffman and Clowes and a slight extension of it which is both more realistic and makes the problem less trivial in the case of random line drawings.

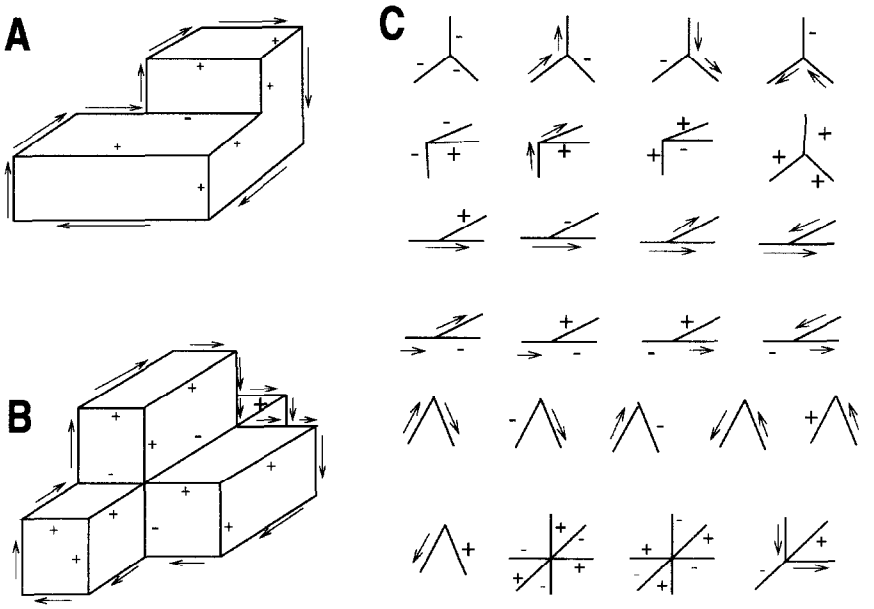

Fig. 2. (A) The natural interpretation of the only T-junction in this image is not as an occlusion but as an actual vertex of the polyhedron. This interpretation is not allowed by the Huffman–Clowes table of Fig. 1(B). (B) An object with an AST-junction (the one with six incident segments) and with a Ψ-junction (the one with four incident segments). This object does not have an interpretation in the Huffman–Clowes trihedral world, which only includes junctions of degree 2 and 3. (C) The dictionary of the legal labelings for extended trihedral scenes. Both (A) and (B) can be labeled according to this catalogue.

It will be called *extended trihedral world* and will include all objects which are (i) solid (ii) opaque (iii) such that only three planes meet at a vertex. To this definition the Huffman–Clowes world adds that only three planar faces meet at a vertex. This unduly eliminates common objects like the one depicted in Fig. 2(A) and the one depicted in Fig. 2(B).

The T-junction in Fig. 2(A) is the projection of a 3D vertex at which three different planes and four different planar faces (two of which are parallel) meet.

The complete dictionary for this world is given in Fig. 2(C), where it can be noticed that there is also a 6-degree junction, that we will call AST (after Kanade [11]). Also notice that there are no 5-degree junctions.

Since a line drawing of an extended trihedral scene can now be defined to be labelable iff it is possible to assign a label to every segment so that all junctions are labeled according to Fig. 2(C), it is maybe useful to point out that the NP-completeness result of Kirousis and Papadimitriou [13] also applies to this case:

Proposition 1. *It is NP-complete to determine whether a line drawing is labelable according to the extended Huffman–Clowes table of Fig.* 2(C).

Proof. First, notice that the NP-completeness proof in [13] for the basic trihedral world (BTW) still holds if the set of the possible labelings for a T-junction is extended to

include all labelings permitted in the extended trihedral world (ETW). This is because the components used in the proof do not make use of the fact that the head-segments of a T-junction must labeled as an arrow leaving the foot of the T on the left side. Let us call MTW (modified trihedral world) the problem of labeling a line drawing according to this scheme. Secondly, notice that MTW is the special case of ETW where no four-degree junctions and six-degree junctions appear. Therefore, since MTW is NP-complete, ETW is also NP-complete by restriction (see, for example, [7] for a more detailed explanation of this kind of reduction). □

3. Line drawings of random instances of scenes

In the previous section we have discussed the worst-case complexity of labeling line drawings. We are now going further to explore the structure of the labeling problem by studying the average-case complexity. From a theoretical standpoint, this is an extremely difficult problem. It is arguably more feasible to tackle it experimentally by producing large samples of instances of the labeling problem and performing a statistical analysis on these samples so as to extract useful statistics such as the mean, the median, and the estimated errors on these quantities as a function of the size of the line drawing. In order to do so, we need a method to generate random instances of line drawings.

Random line drawings are a very special case of planar graphs. In the case of line drawings of scenes from the basic trihedral world of Huffman and Clowes, they can be viewed as embeddings of planar graphs such that all arcs are straight line segments and nodes can only have two or three incident segments.

There exist several methods which generate random *graphs* according to a uniform probability distribution. However, no method has so far been exhibited that achieves the same result for the special case of planar graphs. The even harder problem of generating them so that they can be drawn on the plane by means of straight lines is beyond the scope of this paper.

A possible way out is to analyze the complexity of labeling planar graphs with a nonuniform distribution. Another possibility, which is perhaps more interesting, is suggested by the observation that the labeling problem is essentially combinatorial. Proper labels must be assigned to the arcs of a graph so that certain junction constraints are satisfied, and geometry only enters the picture in defining the specific junction constraints. It seems therefore meaningful to analyze the case in which the problems of embedding are ignored and only the combinatorial structure is kept. The problem of labeling a line drawing is replaced with the problem of labeling the arcs of a random graph whose nodes have degree two or three and whose 3-degree nodes are classified at random as T, Y, or E. It is to be noticed that random planar graphs with a specified degree sequence *can* be generated with uniform distribution. In a completely analogous fashion, we can generate random graphs having the same combinatorial properties as the line drawings of extended trihedral scenes, with nodes of degree 2, 3, 4, or 6.

This analysis, however, leads to rather unsatisfactory results: most of the line drawings generated in this fashion have local inconsistencies even if the most trivial inconsistencies (e.g., pairwise inconsistent junctions) are eliminated; as a consequence, the average-case

complexity remains roughly constant as the number of junctions of the line drawings increase. A more detailed discussion of the experiments showing these results can be found in [22].

The results on the complexity of labeling random graphs with the same combinatorial properties as the line drawings we are interested in give insight in the best of cases on the labeling problem considered as an abstract combinatorial problem but not on the more relevant problem of labeling line drawings which are *known* to be the projection of polyhedral scenes. These line drawings are labelable by definition; the labeling problem is not a decision problem anymore, but a *search* problem: an actual solution is required. In this section we adopt the following approach to the generation of random line drawings: first, we will construct random instances of polyhedral scenes; secondly, we will construct perspective projections of these scenes to obtain instances of line drawings.

Before describing the random scene generator, we briefly discuss the properties that this generator should possess (Section 3.1). We then describe a simpler algorithm, which generates random instances of polygons (Section 3.2), so as to prepare the ground for the description of the algorithm to generate random trihedral scenes from the extended trihedral world and line drawings of these scenes, to which Section 3.3 is devoted. Section 3.4 mentions some facts about the distribution of some properties (angle, length, location) of edges and vertices of random polyhedral scenes. Section 3.5 describes how a line drawing can be obtained as a projection of a random scene. Section 3.6 shows some examples of line drawings constructed by this method. Section 3.7 shows how this method can be modified so as to generate random instances of Legoland (also called *Manhattan*, or *orthohedral*) scenes.

3.1. Properties of a good random scenes generator

An ideal algorithm $\mathcal{A}$ for the generation of random trihedral scenes should have the following properties:

(a) *completeness*: every possible polyhedron $\mathcal{P}$ (or set of polyhedra) can be generated by $\mathcal{A}$, that is, every $\mathcal{P}$ is a possible output of $\mathcal{A}$;
(b) *uniformity*: all possible polyhedra appear with the same probability;
(c) *polynomial complexity*: the cost of generating a random polyhedron should be polynomial in the number of vertices of the polyhedron.

Property (a) is a minimal requirement for any procedure aiming at generating random scenes. It is analogous to require that a random generator of integer numbers between 1 and M be able to generate any number in that range with nonzero probability.

Property (b) is trickier. It is clear what "uniformity" means for a generator of random numbers: every integer number between 1 and M is equally likely to occur. It is not obvious, however, how to generalize this property to define uniformity for generators of more complex objects, such as polyhedra. Work on random graphs [2,20] suggests that many different definitions may be adequate depending on our needs. As to polyhedra, it is first of all necessary to limit the region of space into which the polyhedra must be generated, therefore limiting also their size. We cannot do without that more than we can think of generating random integer numbers with no limitations of range. Secondly, it is not clear what features we are supposed to extract to check against randomness. Some obvious

requirements are that each direction for the planar faces should be equally likely, that the length of the edges and the areas of the planar faces should have a wide spectrum of values and so on, but these are merely necessary conditions. See Section 3.4 for some additional comments and a sketchy presentation of some properties of our random scenes generator. It should be noted that this difficulty with generating useful random instances is very general and affects many other problems in AI, such as SAT and CSP (see, for example, [19] for the problem of generating genuinely random instances of SAT).

Finally, if property (c) were not satisfied, creating samples for any kind of statistical analysis of complexity or other features would be unbearably difficult for a large enough number of vertices. Notice that it does not matter whether the procedure for generating random scenes is sometimes aborted as long as (i) the probability that the procedure be successful is finite for all input sizes and (ii) if this probability $p = p(n)$ depends on the input size n (e.g., the number of vertices of the polyhedral scene), there is a lower bound $p_{\min}$ such that $p(n) \geqslant p_{\min}\ \forall n$. If this condition is satisfied, and the time complexity of the procedure is $T_s(n)$ (that is, it either terminates successfully after $T_s(n)$ steps or it is aborted within the same time limit), than the *expected* time $E(T(n))$ to generate one *successful* element of the set is given by

$$E(T(n)) = \frac{1}{p(n)} T_s(n) \leqslant \frac{1}{p_{\min}} T_s(n), \quad \forall n.$$

3.2. *Generating random instances of polygons*

The description of the algorithm for generating random scenes will perhaps be clearer if prefaced by the description of the algorithm which solves the analogous problem in two dimensions, namely that of generating random instances of polygons. This section describes a way of generating random instances of polygons. A 'polygon' is defined as a closed sequence of pairwise adjacent straight line segments such that:

(i) There are exactly two incident segments for every vertex of the polygon.
(ii) Segments never intersect unless at their endpoints. That is, they do not cross or touch one another and they do not overlap.

The case of two segments which cross each other and that of vertices with more than two incident segments will both be referred to as *knots*.

For some applications it may be useful to include polygons with polygonal holes in the definition. That is easily achieved by allowing more than one closed sequence. We must also add the condition that these sequences do not intersect, and that there exists a sequence which encircles all the others. Polygons inside polygonal holes are admitted but they are not considered part of the polygon within which they are enclosed.

A possible strategy to generate random instances of polygons of 'size' n, or rather sets of polygons, is the following:

(1) Choose the *support* S of the generating algorithm, that is the closed region of the 2D plane where the polygon is bound to lie. So that distortions of uniformity are not introduced among the directions in space, the support region S should be a circle of radius M:

$$S := \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \leqslant M^2\}.$$

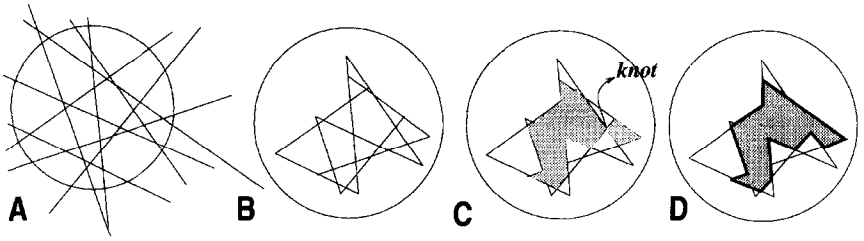

Fig. 3. (A) The support region and a certain number of lines drawn at random. (B) The graph after the intersection points falling outside of the support S have been removed, along with the edges incidents to them. (C) Filling elementary polygons at random may cause the creation of knots. (D) The polygon after the elimination of the knot.

(2) Draw n lines on the plane at random (Fig. 3(A)). This can be done by choosing uniformly at random n points (x_i, y_i) $(i = 1, \ldots, n)$ inside the support region, and by choosing uniformly at random n angles $\theta_i \in [0, \pi)$ $(i = 1, \ldots, n)$. The angle θ_i gives the orientation of line r_i with respect to the x-axis of a Cartesian coordinate system arbitrarily chosen.
It will then be possible to write the equations of the lines as

$$\sin\theta_i(x - x_i) + \cos\theta_i(y - y_i) = 0, \quad i = 1, \ldots, n \tag{1}$$

(observe that $\theta = 0$ yields a horizontal line).
We need to discretize both the support region and the possible angles. We must therefore choose:
(a) a quantity δl such that $x_i = n_i \delta l$, $y_i = m_i \delta l$ for some integers n_i and m_i, and
(b) a quantity $\delta\theta$ such that $\theta_i = k_i \delta\theta$ for some integer k_i.
The value of δl and $\delta\theta$ should be smaller and smaller as n increases, so as to prevent three different lines from meeting at the same point (see next step).

(3) Compute all the intersection points $r_i \cap r_j$ of the lines r_i and r_j created in step (2) (the maximum number of intersection points not at infinity is $n(n-1)/2$). The discretization of the grid should be fine enough as to prevent two intersection points from coinciding. If this should happen, however, the construction is aborted [22].
The computation of the intersection points takes $O(n^2)$ steps where n is the number of lines.

(4) For every line, order all the intersection points that belong to it by fixing an arbitrary direction on the line. Construct the graph such that:
(i) its nodes are all the intersection points inside the support region (all the other points will be discarded), and
(ii) there is an arc between two nodes iff the corresponding intersection points are adjacent to each other in one of the lines r_i (see Fig. 3(B)).
This graph is planar by construction, and we are interested in the specific embedding where the nodes occupy the same location in the 2D plane as the intersections points. In practice, this simply means that our graph is the collection of line segments and points inside the support region enhanced by an explicit graph structure.
The construction of the graph takes $O(n^2)$ steps where n is the number of lines.

(5) Find all the elementary polygons of the planar graph constructed in step (4). This can be done by standard techniques in time proportional to the number of nodes of the graphs, which is of order $O(n^2)$ where n is the number of generated lines. Also find the outer boundary of the graph: this will be called the *outer polygon.*
This step, too, takes time $O(n^2)$, where n is the number of lines.

(6) Every collection of elementary polygons that respects the definition stated at the beginning of this section is a random instance of a polygon, or of a set of polygons. It is understood that when two adjacent elementary polygons are included in the collection, the edge that they share is removed. We will say that an elementary polygon is *filled* iff it is included in the collection. If we fill polygons at random, the resulting construction may suffer in general from the presence of knots. We therefore need a procedure that is able to generate collections of elementary polygons with no knots. Many strategies can be adopted to this end. We have chosen the following simple strategy:
   (a) Fill each polygon π_i $(i = 1, \ldots, N_{\text{pol}})$ with probability p (we have chosen $p = 1/2$).
   (b) In the collection of elementary polygons thus constructed, which will be called $\mathcal{P}$, it may happen that four pairwise collinear segments meet at a point, forming a knot: see, for example, Fig. 3(C). Therefore, once $\mathcal{P}$ is constructed, all vertices of $\mathcal{P}$ are traversed to detect the presence of knots.

Once knots have been detected, they can be eliminated as follows. Notice that each knot divides two filled and two unfilled polygons. [1] We fill one of the two unfilled polygons at random. This may cause other knots to form, and they must be eliminated in the same way. However, it is easy to propagate this information and to ascertain whether the addition of an elementary polygon has caused the creation of other knots. It is not necessary to check the whole structure again after the elimination of a single knot. If it is not possible to fill elementary polygons so as to eliminate all knots, the procedure is aborted. This happens when both the unfilled regions meeting in the knot are part of the background, and is less and less likely as the number of generating lines increases. An example of this situation is shown in Fig. 4.

If the procedure terminates successfully, the polygon (or the set of polygons) is defined as

$$\mathcal{P}' = \mathcal{P} \cup \{\pi \text{ such that } \pi \text{ is filled by the knot-rejection procedure}\}.$$

The filling-in procedure and the knot-rejection procedure take time $O(n^2)$.
An example of output is given in Fig. 3(D).

Complexity. By summing all contributions, we obtain that the worst-case complexity of the algorithm described in steps (1)–(6) is $O(n^2)$ where n is the number of generating lines. $O(n^2)$ is the time that it takes to generate a random polygon or to abort the procedure. There is also experimental evidence that the expected time to generate a successful instance of

[1] This is true if the knot does not lie on the boundary. That case, however, is simpler than the general case: there is a single unfilled polygon, which must be filled to remove the knot.

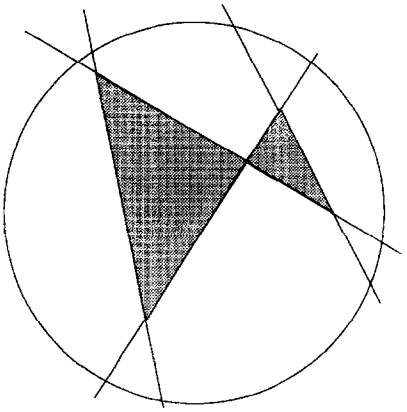

Fig. 4. In this simple example, there are only two elementary polygons. If both of them happen to be filled after step (6a), a knot is formed, and the knot cannot be removed because the two unfilled regions which meet at the knot both belong to the background.

random polygon is $O(n^2)$ (a more detailed discussion on this point can be found at the end of Section 3.3).

Is the algorithm for generating random instances of polygons described above *complete* and *uniform*? The completeness question can easily be answered by the following

Proposition 2. *The procedure to generate random instances of polygons described in steps* (1)–(6) *is complete.*

Proof. To prove completeness, one simply has to prove that, given an arbitrary polygon P with n edges, it is possible that the procedure described above generates P as an output. We will call P the *target polygon* and we will call the lines containing its edges the *target lines*. Suppose that the support region we use for generating the random polygons is a circle of radius M capable of containing the target polygon. When we generate the n random lines of step (2), there is a finite probability that these lines coincide with the target lines. Assume therefore that the randomly generated lines *do* coincide with the target lines. Let us now fill each elementary polygon of the tessellation produced by the generated lines. Once more there is a finite probability that such filling will reproduce the target polygon—either at the first attempt or after the knot-rejection procedure. This concludes our sketchy proof. (See [22] for a more detailed proof including estimation of a lower-bound on the probability of reproducing the target polygon which takes into account discretization.) □

The uniformity question is far more delicate to answer. We are unable to prove that every thinkable polygon whose vertices are inside the support region is equally likely to occur. In order to do so, one should be able first of all to provide an appropriate model for polygons, specifying what are exactly the quantities that should be distributed uniformly. Different choices lead to different distributions. There are some quantities which should certainly be distributed uniformly, such as the tilt angle of each edge of the polygon, whereas it would maybe be more appropriate that the length of the edges be distributed as $1/l^{\alpha}$,

so that the number of edges as a function of length keeps its form over different scales; alternatively, one might accept a world in which edge lengths are distributed exponentially, as when considering intersections of a Poisson line process (see [18,28] and the discussion in Section 3.4). This being the case, we might be content with the completeness property with the additional requirement that the distribution of polygons has good distribution properties, that is, that a number of selected quantities such as angles and lengths be distributed according to a reasonable probability distribution.

3.3. *Generating random instances of polyhedral scenes*

This section describes a way to generate random instances of scenes from the extended trihedral world defined in Section 2. We recall that the extended trihedral world includes all objects which obey these requirements:

(i) they are solid, i.e., there are no hanging edges or faces;
(ii) they are opaque;
(iii) exactly three planes meet at a vertex;
(iv) there are no isolated contact edges; exactly two planar faces meet at every edge;
(v) there are no knots, i.e., isolated contact points between different portions of the scene.

The definition above includes the possibility of having polyhedra with polyhedral holes.

The algorithm which generates random instances of scenes of 'size' n is similar to that which generates random instances of polygons. The number n is related to the number of vertices, edges and planar faces of the polyhedral scene.

(1) Choose the *support* S of the scenes generator, that is the closed region of the 3D space where the polyhedra must be contained. So that distortions of uniformity are not introduced among the directions in space, the support region S should be a sphere of radius M:

$$S := \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 \leqslant M^2\}.$$

(2) Pick n planes in the 3D space at random. To do this, we (a) choose uniformly at random n points $P_i = (x_i, y_i, z_i)$ inside the support region; (b) choose uniformly at random n unit vectors $\vec{n}_i$. Details on the generation of the P_i's and the $\vec{n}_i$'s, dealing with discretization and precision issues, can be found in [22].
The equations of the planes $p_1, \ldots, p_n$ will then be written as

$$\vec{n}_i \cdot (P - P_i) = 0, \quad i = 1, \ldots, n. \tag{2}$$

The complexity of generating the n random planes is $O(n)$.

(3) Compute all the intersection points $p_i \cap p_j \cap p_k$ of triplets of the planes p_i, p_j, and p_k created in step (2).[2] Discard those points that lie outside of the support region. As in the 2D case, discretization should be fine enough to prevent two intersection point from collapsing together—if that happens, however, the construction is aborted.
The computation of the intersection points takes $O(n^3)$ steps where n is the number of planes.

[2] The maximum number of intersection points not at infinity is $n(n-1)(n-2)/6$.

(4) Compute all the intersection lines $r_{ij} = p_i \cap p_j$ of the planes p_i, p_j created in step (2). For each line r_{ij}, consider the set of intersection points found in step (3) which belong to r_{ij}, and order them by assigning an arbitrary direction on r_{ij}.
This takes time $O(n^3)$ where n is the number of planes.

(5) Construct the graph such that:
   (i) its nodes are all the intersection points inside the support region;
   (ii) there is an arc between two nodes iff the corresponding intersection points are adjacent to each other in one of the lines $p_i \cap p_j$.

   We are interested in the specific embedding of the graph in 3D space in which the nodes occupy the same location in 3D space as the intersections points $p_i \cap p_j \cap p_k$. In practice, this simply means that our graph is the collection of line segments and points inside the support region enhanced by an explicit graph structure.
   This graph is in general not planar, but can be viewed as a collection of n planar graphs on n different planes. These planar graphs are interconnected: in the 3D space embedding specified above, the graphs G_i and G_j, respectively, associated with the two planes p_i and p_j share all the intersections points lying on the line $p_i \cap p_j$. This structure is more complicated than that of a planar graph but it will be useful in subsequent steps of the algorithm.
   The number of nodes of the graph and the number of arcs are of order $O(n^3)$, where n is the number of planes generated in step (2). Therefore, the construction of the graph, too, takes time $O(n^3)$.

(6) For every plane p_i independently, find all the elementary polygons of the planar graph constructed in step (4). A subset of these polygons will correspond to the faces of the polyhedra that we are constructing. This takes time $O(n^3)$ where n is the number of planes.

(7) Find all the elementary polyhedra of the graph constructed in step (4). This can be done by mimicking the standard techniques to find elementary regions in planar graphs, and requires time proportional to the number of nodes of the graph. Before sketching the procedure to find elementary polyhedra, we make the following observation.
   Observation. Each polygon found in step (6) is on the boundary of two different elementary polyhedra, unless it lies on the outer hull containing all elementary polyhedra, in which case it belongs to a single elementary polyhedron. We assume that the unit normal vector of a face of a polyhedron points inside the polyhedron itself. Therefore, given a polygon π_i, there are two unit normal vectors $\vec{n}_i$, $\vec{n}'_i = -\vec{n}_i$ associated with it, each of which correspond to a different elementary polyhedron.
   The procedure to find the elementary polyhedra is as follows:
   - We construct the graph such that:
     (i) its nodes are the pairs $(\pi_i, \vec{n}_i)$, where π_i is a planar face and $\vec{n}_i$ is one of its two unit vectors, and
     (ii) there is an arc between node $(\pi_i, \vec{n}_i)$ and node $(\pi_j, \vec{n}_j)$ iff
       (a) there exists a 3D edge e_k which belongs to both π_i and π_j;

(b) by rotating π_i around e_k according to the direction of $\vec{n}_i$, $\vec{n}_i$ being viewed as a pulling force applied to a point in the inside[3] of π_i, the first planar face which is met is π_j.

- We find the connected components of the graph constructed above. These connected components correspond to the elementary polyhedra.

This procedure also finds the *outer hull*, i.e., the boundary of the polyhedron containing all the elementary polyhedra. The outer hull corresponds, in fact, to the connected component which includes all nodes $(\pi_i, \vec{n}_i)$ such that $\vec{n}_i$ points into empty space.

The complexity of constructing the graph and finding its connected components is $O(n_\pi + n_e)$, where n_π is the number of planar faces and n_e is the number of edges. Since both n_π and n_e are of order $O(N)$, where N is the number of intersection points, the complexity of this step is $O(n^3)$, where n is the number of planes generated in step (2).

(8) Every collection of elementary polyhedra that respects the assumptions stated at the beginning of this section is a random trihedral scene. It is understood that when two adjacent elementary polyhedra are included in the collection, the planar face that they share is removed. We will say that an elementary polyhedron is *filled* iff it is included in the collection. If we fill polyhedra at random, the resulting construction may suffer in general from the presence of (a) X's, i.e., isolated contact edges of a special type, with four pairwise collinear planar faces meeting at an edge, and of (b) knots, i.e., isolated contact points. Also, we cannot fill *all* the elementary polyhedra, since the random generator would not have the completeness property introduced in Section 3.1. We therefore need a *complete* procedure that is able to generate collections of elementary polyhedra with no knots and no X's. To this end, we have chosen the following, simple strategy:

(a) Fill each polyhedron Π_i $(i = 1, \ldots, N_{\text{pol}})$ with probability p (we chose $p = 1/2$).

(b) The set of elementary polyhedra thus constructed, which will be called $\mathcal{P}$, is not necessarily a polyhedron nor a proper polyhedral scene. Two types of undesirable geometrical constructions are likely to appear:
- four planes, pairwise coplanar, meeting at an edge (X's);
- isolated contact points (knots).

Therefore, once $\mathcal{P}$ is constructed, we have to check for these irregular constructions. This can be done immediately by traversing all edges to check whether they are the crossing edges of an X, and all the vertices to check whether they are knots.

Once X's and knots have been detected, they can be eliminated as follows: as for the X's, notice that the crossing edge divides two filled polyhedra and at most two unfilled polyhedra. In case there are two unfilled polyhedra, we fill one of them at random; if there is only one unfilled polyhedron, we fill it. This may cause other X's to form, which must be eliminated in the same fashion. However, it is easy to propagate this information and to ascertain whether the addition of an elementary polyhedron has caused the creation of

[3] This is not ambiguous. The elementary polyhedra are all convex by construction, and therefore all points in the inside of π_i lie on the same side of e_k.

Table 1
The percentage of aborted scenes due to the knot- and X-rejection procedure, as a function of the number of planes. For each different number of planes a sample of 100 scenes was produced

Number of planes	10	15	20	25	30
Aborted	33%	23%	7%	4%	0%

other X's. It is not necessary to check the whole structure again after the elimination of a single X. We proceed analogously with knots: we pick one polyhedron at random among the ones that meet at the vertex, and we fill it. This creates an X, which is eliminated as explained above. Notably, the two procedures are intertwined, since the elimination of an X might cause the appearance of knots and *vice versa*. If it is not possible to fill elementary polyhedra so as to eliminate all knots and X's, the procedure is aborted.

If the procedure terminates successfully, we can define the new scene

$$\mathcal{P}' = \mathcal{P} \cup \{\Pi \text{ such that } \Pi \text{ is filled by the } X\text{- and knot-rejection procedure}\}.$$

The filling-in procedure and the knot- and X-rejection procedure take time $O(n^3)$.

Complexity. By summing over all contributions we obtain that the worst-case complexity of the algorithm described in steps (1)–(8) is $O(n^3)$, where n is the number of generating planes. This is the number of steps that it takes either to terminate the procedure successfully or to abort it. To show that $O(n^3)$ is also the *expected* time to generate a successful instance, we also need to show that the probability that the procedure is *not* aborted is bounded from below by a number different from zero and independent of the number of planes (see Section 3.1). We have estimated this probability by measuring the frequency of abortive instances for different values of n. The results are reported in Table 1. Observe that the percentage of aborted scenes decreases as the number of planes increases, as expected (see Section 3.2). Therefore there is strong experimental evidence that there exists a lower bound on the probability (around $100 - 33 = 67\%$ if we are interested in scenes with 10 planes or more) and we can state that the expected time is $O(n^3)$.

Analogously to the 2D case, we can state the following

Proposition 3. *The procedure to generate random polyhedra described in steps* (1)–(8) *is complete.*

Proof. The proof proceeds as in the 2D case: it is shown that given a polyhedron (or a polyhedral scene) it is possible for it to be the output of the random generator described in steps (1)–(8), with finite probability. □

3.4. Distribution properties

So far we have proved that our generator of random instances of scenes is *complete* and we have given experimental evidence that it has *expected polynomial complexity*. A good generator, however, should have the *uniformity* property as well, that is, all possible polyhedral scenes should have the same probability to occur. As we have previously

discussed, however, it is not clear what *uniform* means when referred to the set of polyhedral scenes. Nonetheless, it is interesting to analyze the distribution of several geometric elements of our random generator (e.g., vertices, edges, planar faces). We do not go into the details of this analysis here limiting ourselves to quoting a few results (see [22] for more details):

- The distribution of the directions of the edges of the generated line drawings is uniform, as expected.
- The distribution of the lengths of the edges—which can theoretically go from 0 to the diameter of the support region—is not uniform. Uniformity is not expected. An analysis of our data reveals that the distribution is well fitted by a function of the form $e^{(-\alpha l)}$ with $\alpha \sim 5.6 \times 10^{-4}$. This is consistent with the distribution of edge lengths for the intersections of a Poisson line process in a 2D space (i.e., the process formed by randomly drawing lines in the plane), which gives rise to a Poisson process on each line, and therefore to an exponential distribution of edge lengths (see Miles [18], Stoyan et al. [28]).
- The density of vertices in the support region is expected to be more or less uniform, except close to the boundary where there are fewer possible lengths of edges which terminate into it. The experimental results show that the density of vertices decreases more or less linearly as the distance from the center of the support region increases, up to about 85% of the radius, where the density starts falling more steepily.

3.5. *The line drawing*

For our experiments on complexity we are interested in obtaining line drawings of random instances of scenes rather than the scenes themselves. This can be easily achieved by projecting the scene onto an arbitrary plane by a perspective projection,[4] and then removing all the hidden edges so as to render the opacity of the polyhedra. In more detail, the procedure is as follows:

(1) Let $(O; X, Y, Z)$ be the coordinate system according to which the scene is described. The center of the support region is the point $O = (0, 0, 0)$ in these coordinates. The support region is a sphere of radius M centered in O. A center of projection O' is chosen at an appropriate distance D from the center of the support region. A focal length f and an *image plane* are also chosen. In the actual experiments, D was equal to $2M + f$, so that $O' = (0, 0, -2M - f)$. The image plane is defined by the equation $Z = -2M$.

(2) The scene is projected onto the image plane, regardless of visibility questions. Every vertex of the scene, $V = (V_x, V_y, V_z)$, is projected onto a 2D junction v whose coordinates on the image plane are

$$v = f\left(\frac{V_x}{V_z + 2M + f}, \frac{V_y}{V_z + 2M + f}\right).$$

[4] Most works on line drawings deal with orthographic projection, which can be viewed as a special case of perspective projection. Perspective projection is the correct modelization of the vision process. All the complexity results which are valid for orthographic projections of polyhedral scenes are also valid for perspective projections, and *vice versa*.

This can be done in $O(N)$ steps, where N is the number of vertices in the scene.

(3) All the hidden edges can be removed by one of the many existing techniques in computer graphics and computational geometry. We have used Appel's algorithm [1,6], which suits our case well. The algorithm has complexity $O(N \log N)$.

The overall complexity of the algorithm for obtaining the line drawings from the scene is therefore $O(N \log N)$, where N is the number of vertices in the scene. In terms of the number n of generating planes, the complexity is $O(n^3 \log n)$, since $N = O(n^3)$.

3.6. Examples

Examples of line drawings generated by the algorithm described above are given in Figs. 5 and 6. For every line drawing a legal labeling of it is also given. Since the labeling was generated automatically, it was necessary to represent labels by a different code than the usual one consisting of putting a label (chosen in the set $\{+, -, \rightarrow, \leftarrow\}$) close to the segment. Here, a "+"-segment is drawn as a dotted segment, while a "−"-segment is drawn as a dashed segment. Segments labeled with an arrow are simply drawn as arrows.

3.7. Generating random instances of Legoland scenes

The generator of random instances of scenes described in Section 3 can be easily modified so that its output be a Legoland scene rather than a general trihedral scene. Legoland is a world of trihedral polyhedra whose planar faces are oriented according to one of three orthogonal axes. It is a very simple world but it is often powerful enough to capture the main features of simple scenes in a man-made world. Legoland scenes have been widely discussed in the literature [13,26,27], because of their simplicity and practical relevance. From the theoretical point of view, they represent one of the few examples for which the labeling problem is polynomial [13].

To generate a random Legoland scene of size n, we first choose an arbitrary triplet $(\vec{n}_1, \vec{n}_2, \vec{n}_3)$ of orthogonal unit vectors. The algorithm then proceeds as for general

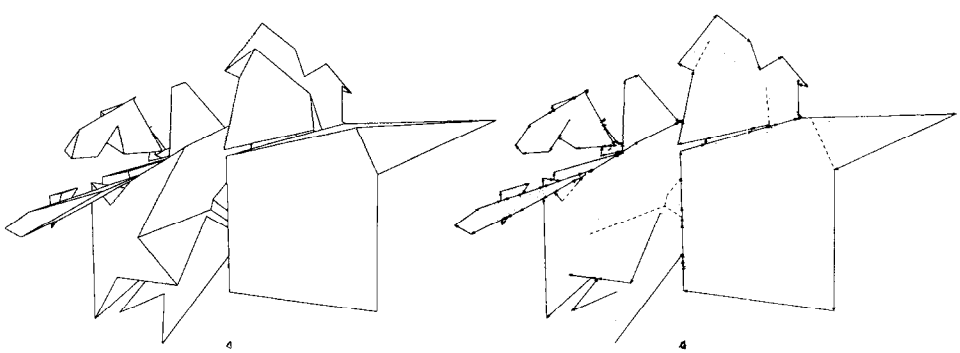

Fig. 5. An example of a line drawing of a random trihedral scene (left), and a legal labeling (right). The number of junctions is 121. Notice that it takes a while (and a sort of Necker's reversal limited to the top-right part of the picture) to a human observer to understand that this line drawing can be interpreted as a solid (as opposed to Origami) object.

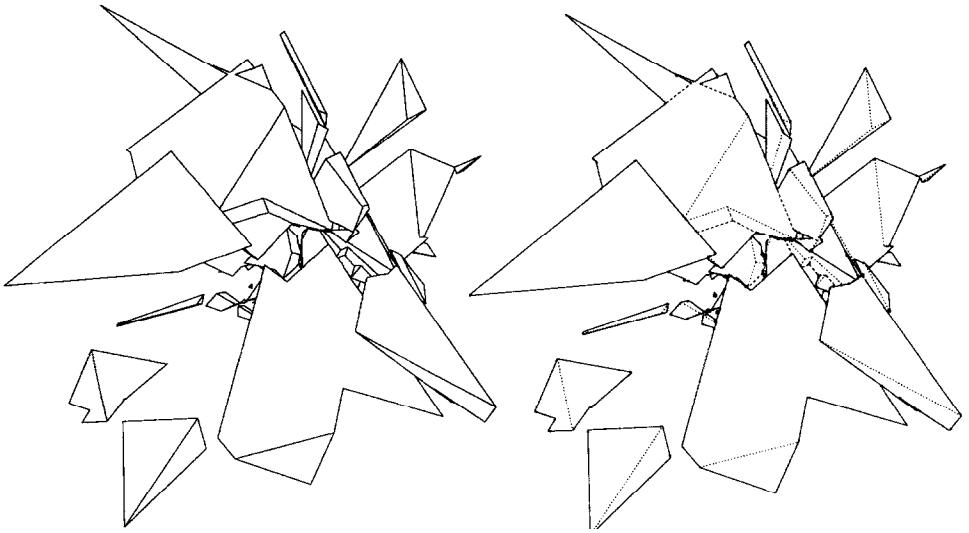

Fig. 6. An example of a line drawing of a random trihedral scene (left), and a legal labeling (right). The number of junctions is 282. The labeling has been performed using the labeling rule for boundaries.

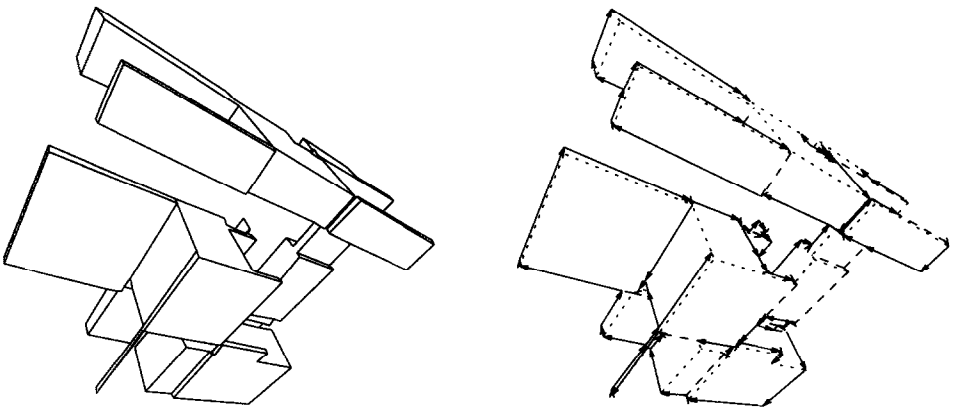

Fig. 7. An example of a line drawing of a random Legoland scene. The number of junctions is 127.

trihedral scenes, except that the unit vector of each of the n planes is chosen at random among $\vec{n}_1$, $\vec{n}_2$, and $\vec{n}_3$. An example of a line drawing of a random Legoland scene is shown in Fig. 7.

4. Experiments on the complexity of labeling

In this section we use the line drawings generated as explained in Sections 3.3 and 3.5 to perform experiments on the complexity of labeling line drawings and to assess the

efficiency of some well-known heuristics. Labeling is performed by standard techniques for solving constraint satisfaction problems. The core of these techniques is a tree search, usually preceded by a relaxation stage during which local consistency is achieved, as in Waltz [34]. We will consider two kinds of tree search: 'blind' depth-first search with backtracking and a more sophisticated best-first search informed by knowledge on the structure of the problem.

As for the relaxation stage, we will use the simple algorithm referred to as AC-1 by Mackworth and Freuder in [14]:

> For each pair of adjacent junctions J_1 and J_2, remove all labelings of J_1 for which there is no labeling in J_2 which is compatible with it, and *vice versa*. Repeat the operation for all pairs of junctions until we have gone through all the pairs of junctions once without deleting a single labeling.

This procedure takes in the worst case time $O(n^2)$, where n is the number of segments (or of junctions) in the line drawing; it does not take advantage of the fact that it is only necessary to check again junctions adjacent to junctions whose set of legal labelings is modified. Better techniques exist: one of them is Waltz' procedure (sometimes referred to as AC-2), which has complexity $O(n)$; other techniques are described in [14]. In our case the difference is of no consequence (most of the time is spent during the depth-first backtracking anyway), and we have therefore implemented the simpler and slower version.

The experiments on the complexity of labeling that we are going to describe have been carried out by producing a number of line drawings for different numbers of planes. This gives a number of samples each of which refers to a fixed number of planes. It is, however, more useful to refer to a fixed number of junctions. Therefore we create *transformed samples* each of which contains instances of line drawings having approximately the same number of junctions. If, e.g., a line drawing constructed out of 16 initial planes has 122 junctions and is labeled in 1338 steps, we add the number 1338 to the sample associated with a number of junctions equal to 120, which might actually include instances having, e.g., from 115 to 124 junctions. This is called *pooling*. Before associating a median computational time to each number of junctions N we require that there is an accumulation of 100 line drawings which have a number of junctions in the range $[N - \frac{1}{2}\Delta N, N + \frac{1}{2}\Delta N - 1]$ (usually $\Delta N = 10$, or $\Delta N = 5$, in which case we take intervals of the type $[N - 3, N + 2]$). After completing the set for a certain range of values for N we do not accept any more entries. This indeterminacy must be taken into account when assessing the error by which the various statistical quantities are computed. This allows us to pass from the number of planes to the number of junctions, which is a more useful variable since it is a property of the line drawing and not of the way we produce it. After the experiment is over, we compute the median number of steps for each interval and the error on the median, and these are the quantities that appear in the plots. Estimates of the median-time complexity are more reliable than those of the mean-time complexity, and reflect more clearly what are the time requirements of the "typical" case. A single item which is unusually hard can throw off the mean-case estimate, while it has little impact on the median. The error on the median can be estimated by the bootstrap procedure [5].

We now consider several methods to label line drawings and we analyze their performance.

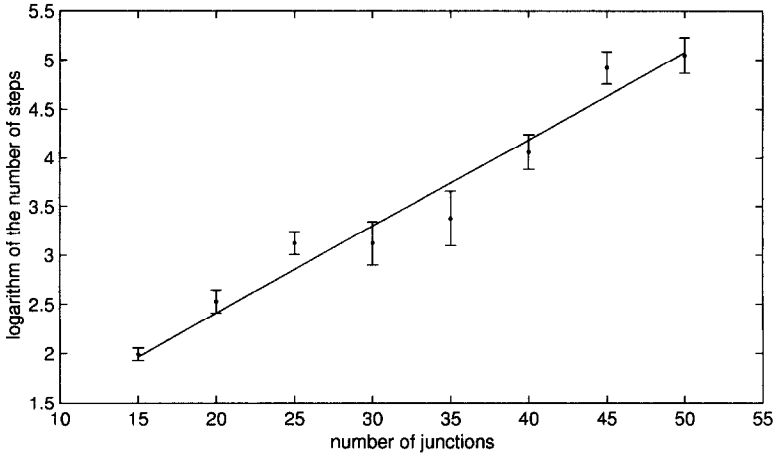

Fig. 8. The base-10 logarithm of the median-case complexity of labeling line drawings of random trihedral scenes as a function of the number of junctions. The error bars give the error on the median estimated by the bootstrap procedure (±1 standard deviation). Labeling is performed by a simple depth-first search. The plot is fitted reasonably well by the line $\log_{10} T(N) = aN + b$ where $a = 0.089 \pm 0.003$, $b = 0.6 \pm 5.9$ ($\chi^2 = 11.2$, $P(\chi^2 > 11.2) = 0.082$). We have not plotted the indeterminacy on the values of N on the x-axis since that is systematic and independent of N, but it gives a decisive contribution to the computation of the χ^2-statistic.

Simple depth-first search with backtracking. In the first experiment we are going to use the following labeling method: (1) we achieve arc-consistency by the relaxation algorithm AC-1; (2) we perform a depth-first search on the constraint network. Time for the search stage is computed as the number of times that the depth-first-search stack containing all nodes which have been visited (that is, touched at least once) but not explored (that is, such that all the nodes adjacent to them have not been visited) is updated.

Fig. 8 shows the computational time employed by the depth-first search as a function of the number of planes. Each point in the plot refers to a sample of 100 line drawings and the reported time is the median number of steps for the sample. A first look at the data reveals a seemingly exponential behavior (that is, a linear dependence of $\log_{10} T(N)$ on N). This is consistent with the fact that the general form of the worst-case complexity for a depth-first search with backtracking when applied to our algorithm is $O(A^N)$, where A is a constant related to the maximum number of legal labelings of a junction and N is the number of junctions.

To test the hypothesis that the functional dependence can indeed be described by the equation $T(N) = B \cdot A^N$ where N is the number of junctions and $T(N)$ is the average number of steps (specifically, the median), we proceed according to standard statistical methods:

- We plot $Y = \log_{10}(T(N))$ against $X = N$ and we fit the results with a line: $Y = aX + b$. The parameters of the line are computed as those which minimize the quantity:

$$\chi^2(a,b) = \sum_{i=1}^{m} \frac{(Y_i - aX_i - b)^2}{\sigma_i^2},$$

where (X_i, Y_i) are the pairs of experimental data, m is the number of pairs, and σ_i is usually the estimated error on Y_i. In the present case, however, we must also take into account the error on the X_i variables, which is not neglectable. The average error on X_i is the same for all i and can be estimated as $\sigma_i^X = \Delta N/(2\sqrt{n_s})$, where ΔN is the interval between two successive values of N in the experiment ($\Delta N = 5$ in the present case) and n_s is the size of the sample. The errors on the Y_i's are computed by the bootstrap procedure, specifically by resampling the original data 100 times (this is sufficient under ordinary circumstances [5]), computing the median for each resampled set and then computing the standard deviation of the set composed by all the medians. See [5] for details on the bootstrap procedure sketched here. A workable way to combine the errors on X_i and Y_i is first to compute an estimate $\overline{a}$ of a and then to choose $\sigma_i^2 = (\sigma_i^Y)^2 + \overline{a}^2(\sigma_i^X)^2$ in order to compute the final values of a and b and to estimate the χ^2 (see next point). Once we have obtained the values a^* and b^* which minimize $\chi^2(a, b)$, we can compute A and B as $A = 10^{a^*}$, $B = 10^{b^*}$.

- Let $\chi^2(a^*, b^*)$ be the minimal value of $\chi^2(a, b)$. $\chi^2(a^*, b^*)$ obeys to the so-called χ^2-distribution which has an expected value equal to the number $\nu = m - 2$ of degrees of freedom of the distribution (which is in turn equal to the number of experimental points minus the number of parameters computed by using the data). From the χ^2 distribution we can directly compute the probability $P(\chi^2 > \chi^2(a^*, b^*))$ that the χ^2 is greater than the value found in the experiment. The number $P(\chi^2 > \chi^2(a^*, b^*))$ gives the probability that χ^2 is greater than the value we have found. A hypothesis is discarded if $P(\chi^2 > \chi^2(a^*, b^*))$ is below a certain threshold, usually around 5% (the fit is not good enough), or if it is above a certain threshold, e.g., 95% (the fit is *too* good and the errors have probably been overestimated).

In the specific case, we have found $a^* = 0.089 \pm 0.03$, $b^* = 0.6 \pm 5.9$. For these values, we have $\chi^2(a^*, b^*) = 11.2$, which means that the fit is not very good (the expected value is 7), but it is still acceptable, since $P(\chi^2 > 11.2) = 0.082$. (In the following we will omit the star and we will simply write a, b to indicate the parameters of the best-fit line.)

The procedure described above uses pooling (gathering all line drawings with similar number of junctions into the same sample) because it is difficult to obtain large samples of line drawings with exactly the same number of junctions. This can be annoying because it adds a source of error—the indeterminacy on the exact number of junctions—which reflects on the precise estimation of the coefficient of the regression line. An alternative to pooling which does not require using far larger samples is that of using robust regression techniques such as those described in [8,17]. As a result, we have employed one of these techniques, specifically *least median of squares* (LMS), to provide secondary confirmation of the conclusions reached in the preceding analysis. LMS consists of minimizing the quantity

$$\mathrm{MS}(a, b) = \mathrm{median}_i (Y_i - aX_i - b)^2,$$

where (X_i, Y_i) is a single experimental pair and not a sample of 100 elements as in the classical regression analysis discussed above. The minimum of $\mathrm{MS}(a, b)$ can be obtained by generating trial values for (a, b) and choosing the one, denoted as $(a_{\mathrm{LMS}}, b_{\mathrm{LMS}})$, that gives the smallest value of $\mathrm{MS}(a, b)$ (see [5]). Trial values can be generated by sampling a number (100 in our case) of random pairs of experimental points (X_i, Y_i), (X_j, Y_j) without

replacement and computing the coefficients (a, b) of the line passing through the different pairs of experimental points [5]. An estimate of the error on $(a_{\mathrm{LMS}}, b_{\mathrm{LMS}})$ can be obtained by repeating this procedure a number of times (50 in our case) by bootstrapping pairs (X_i, Y_i) from the original data set [5]. The method is computationally demanding even for a small number of bootstrap repetitions, and the errors on the estimated coefficients with this method are often quite high.

The regression coefficients obtained with LMS performed on the unpooled data for the experiment of Fig. 8 described above are: $a_{\mathrm{LMS}} = 0.08 \pm 0.01$, $b_{\mathrm{LMS}} = 0.9 \pm 0.3$. These values are consistent with those obtained by classical regression analysis (least sum of squares).

We conclude that the complexity of labeling line drawings by a simple depth-first search with backtracking is exponential in N, where N is the number of junctions: the median-case complexity has therefore the same form as the worst-case complexity. The use of more sophisticated search methods, however, yields drastically different results.

Best-first search with backtracking. In the depth-first search method with simple backtrack, junctions are chosen blindly as deeper and deeper layers of the depth-first search tree are explored. A more refined search method (see, for example, [24]) is the so-called *best-first search*, which exploits the fact that it is more convenient to label the junctions which have the smallest possible set of permissible labelings. Furthermore, the search is guided by the *a priori* knowledge about the structure of the problem, and labelings which are more "likely" for certain junctions are tried first. Specifically, this refined search uses the following heuristic rules:

- At a given point during the search, nodes can be subdivided into three classes: unvisited, visited, explored. *Unvisited* nodes are those that have yet to be considered by the search; *visited* nodes are those that have been touched but the nodes that they are adjacent to have not been visited yet; *explored* nodes are those that have been visited and the nodes adjacent to them have been visited as well. The nodes that have been visited have already been labeled. The set of those nodes which are adjacent to visited nodes is the set in which the next node to be tentatively labeled must be chosen. Best-first search chooses the node which has the smallest set of legal labelings, breaking ties arbitrarily.
- T-junctions have six possible labelings. Four of them are common to the basic and the extended trihedral world, the other two only appear in the extended trihedral world, and they appear more seldom. Therefore, it is convenient to try first the "conventional" labelings and only successively the remaining two labelings. For E-junctions, we try first the labeling with the middle-segment labeled as a convex segment and the remaining segments labeled as arrows.

In addition to these heuristic rules we adopt the usual relaxation stage before starting the search. Fig. 9(right) shows the median computational time versus number of junctions for the best-first search method. The error bars are computed by the bootstrap procedure. Fig. 9(left) shows the 50% (median), 75%, 90%, 95% percentiles of the distribution of the computational time, so as to give an idea of the scatter of the data. We see the first clear signs of the emergence of the exponential behavior at the 95% percentile.

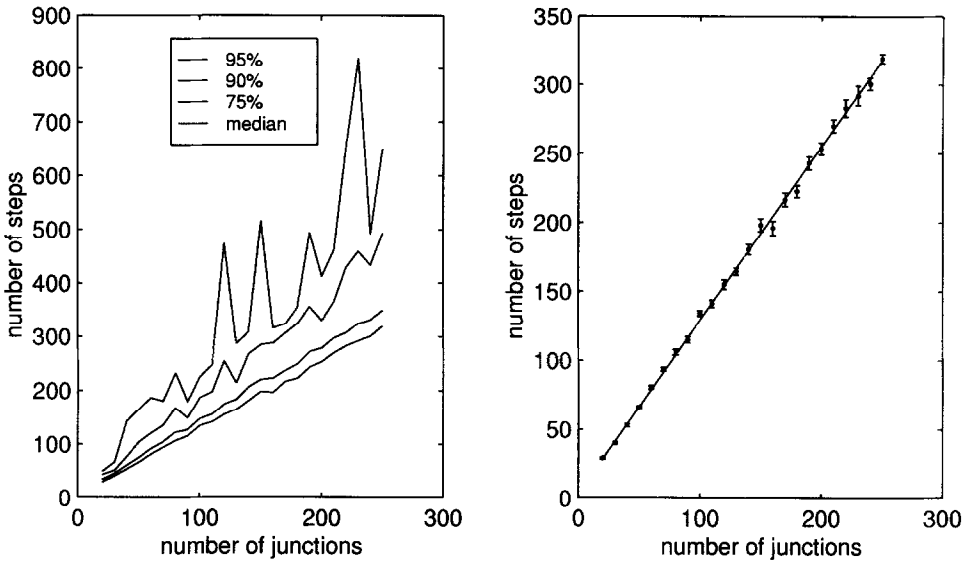

Fig. 9. (Left) Showing the 50% (median), 75%, 90%, 95% percentiles of the distribution of the computational time. (Right) Time (number of steps) versus the number of junctions for the best-first search. The graph is best-fitted by the line $T(N) = aN + b$, $a = 1.26 \pm 0.01$, $b = 3.3 \pm 0.4$, where N is the number of junctions. For these values we find $\chi^2 = 22.1$, $P(\chi^2 > 22.1) = 0.46$.

The computational complexity appears to be linear in the number of junctions; more precisely, the median number of steps—measured as the number of time the search stack is updated—is always very close to the number of junctions n. The best-fit line through the data is $T(N) = aN + b$ (N being the number of junctions) with $a = 1.26 \pm 0.01$, $b = 3.3 \pm 0.4$. where N is the number of junctions. The fit is good: we find $\chi^2 = 22.1$, $P(\chi^2 > 22.1) = 0.46$, corroborating the hypothesis that the number of steps grows linearly in the number of junctions. These values of the regression coefficients are to be compared with those obtained by least median of squares: $a_{\mathrm{LMS}} = 1.22 \pm 0.06$, $b_{\mathrm{LMS}} = 7.0 \pm 3.8$.

Adding the labeling rule for boundaries. Under ordinary conditions, a scene is viewed through a "window": only those objects which are projected into a certain portion of the image plane, the field of view, are visible. However, if we may assume that the scene consists of a finite set of polyhedra all of which are inside the field of view, then there is a useful rule of thumb that we can use for labeling: segments which lie on the boundary can be labeled with arrows that leave the background on their left. This rule has been extensively used in the past in works on line drawings. It is not always justified, but we can certainly use it in our case because the support region of the polyhedra is fixed and known: therefore we can always choose the viewing window so as to include the support region.

Notice that a simple modification of the random scenes generator, which introduces a viewing window artificially, enables us to deal—if the circumstances demand so—with the most general case, and to generate line drawings for which the labeling rule for boundaries cannot be applied: an example is shown in Fig. 10(right).

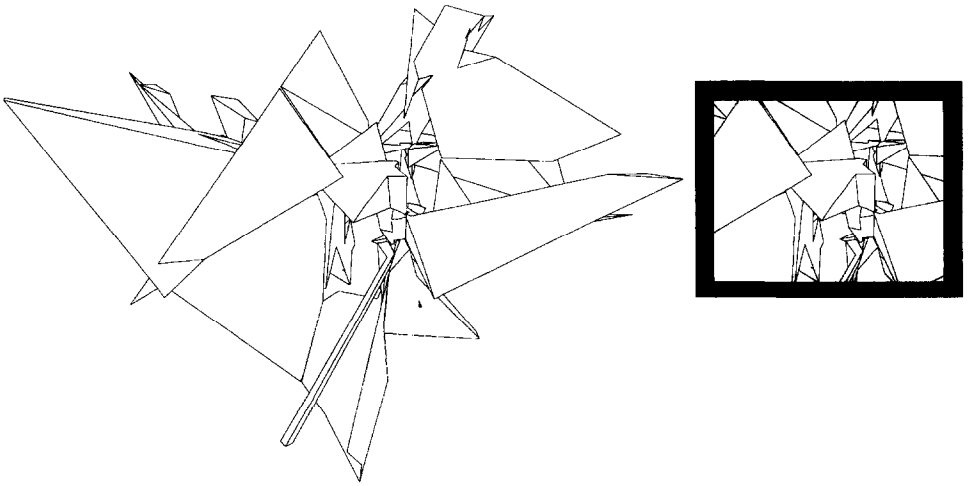

Fig. 10. (Left) An example of a line drawing of a random trihedral scene. (Right) The same line drawing seen through a viewing window. The labeling rule for boundaries can be used for the line drawing on the left but not for the line drawing on the right.

It is interesting to study the effect of this rule for labeling boundaries on the complexity of labeling a line drawing. Fig. 11 shows the computational time as a function of the number of junctions in case this rule is adopted. The slope of the best-fitting line $T(N) = aN + n$, $a = 1.22 \pm 0.01$, $b = 3.9 \pm 0.4$, is slightly smaller than that for the best-first search without the labeling rule for boundaries. For these values we have $\chi^2 = 29.2$, with a confidence value of $P(\chi^2 > 29.2) = 0.14$. The fit is therefore acceptable. These values are also to be compared with those obtained by least median of squares: $a_{\mathrm{LMS}} = 1.19 \pm 0.05$, $b_{\mathrm{LMS}} = 5.5 \pm 2.5$.

Observe that the labeling rule for boundaries helps narrow the distribution of values of the ratio r between the number of steps and the number of junctions around the optimal value $r = 1$, as an inspection of the 50% (median), 75%, 90%, 95% percentiles of the distribution of the computational time shows.

The role of relaxation. In the previous experiments on labeling we have used the relaxation algorithm AC-1 as a pre-processing stage to the search. We now want to determine whether the use of this pre-processing stage is crucial to the finding that the computational time is linear in the number of junctions.

We therefore perform another experiment on the complexity of labeling, again using best-first (with no labeling rule for boundaries), this time switching off the relaxation stage. The results are shown in Fig. 12: the number of steps still appears to be linear in the number of junctions, and can be best-fitted by the line $T(N) = aN + b$, $a = 1.34 \pm 0.02$, $b = 2.2 \pm 0.6$. For these values we have $\chi^2 = 19.2$, with a corresponding confidence value of $P(\chi^2 > 19.2) = 0.63$. These values are to be compared with those obtained by least median of squares: $a_{\mathrm{LMS}} = 1.28 \pm 0.08$, $b_{\mathrm{LMS}} = 5.7 \pm 5.6$.

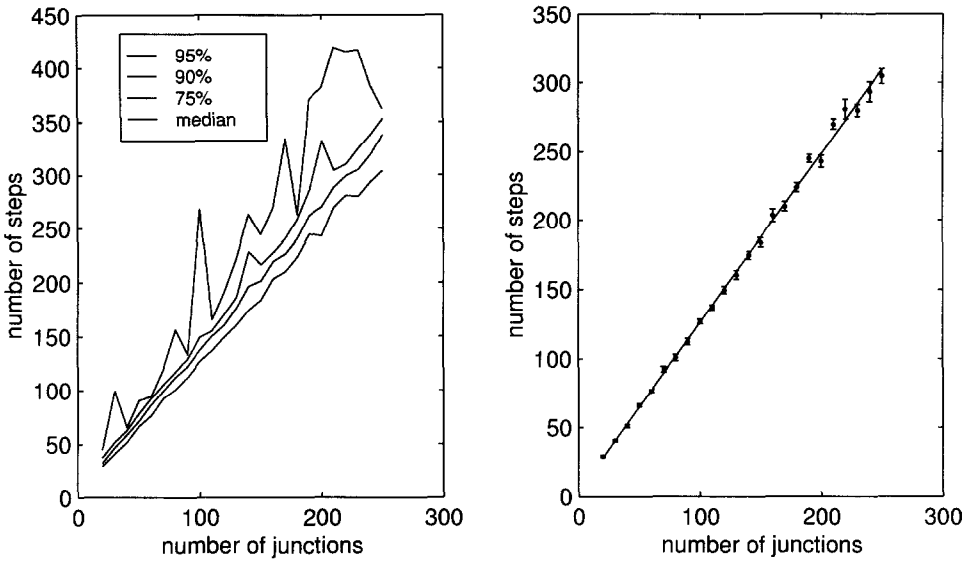

Fig. 11. (Left) Showing the 50% (median), 75%, 90%, 95% percentiles of the distribution of the computational time. (Right) Time (number of steps) versus the number of junctions for the best-first search, when the labeling rule for the boundaries is applied. The data are best-fitted by the line $T(N) = aN + b$, $a = 1.22 \pm 0.01$, $b = 3.9 \pm 0.4$, where N is the number of junctions. The fit is acceptable: $\chi^2 = 29.2$, $P(\chi^2 > 29.2)) = 0.14$.

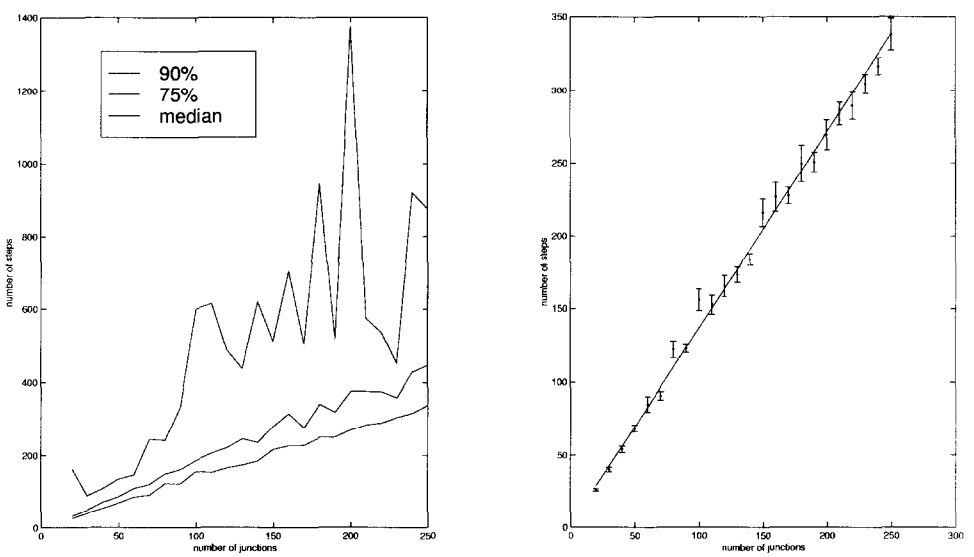

Fig. 12. (Left) Showing the 50% (median), 75%, 90%, 95% percentiles of the distribution of the computational time. (Right) Time (number of steps) versus the number of junctions for the best-first search, when the labeling rule for the boundaries is *not* applied, and *no relaxation* is performed in advance. The best-fit line is $T(N) = aN + b$, $a = 1.34 \pm 0.02$, $b = 2.2 \pm 0.6$, where N is the number of junctions. The fit is good ($\chi^2 = 19.2$, $P(\chi^2 > 19.2) = 0.63$).

The slope of the best-fit line for the present case is not much different from that for the case where relaxation is used. However, the data are far more spread, as an inspection of the percentiles of the distribution immediately shows (see Fig. 12(left): observe that in this case it was not possible to draw the 95% percentile, in that it was too irregular and too high-reaching). Relaxation is therefore indispensable to control the width of the distribution of the computational time and tame the 'bad minority' of cases.

The importance of the relaxation stage can also be evaluated by the amount of legal labelings of a junction, among those possible *a priori*, that have been discarded by the relaxation procedure. Suppose that each junction J_i has an initial set of $n_i^{(0)}$ labelings, and that after the relaxation stage this number has gone down to n_i. A measure of the gain in performing the relaxation procedure is given by the ratio

$$R = \frac{\prod_i n_i}{\prod_i n_i^{(0)}}.$$

The product of the number of labelings for each junction is related to the complexity of the worst-case complexity of the depth-first backtracking algorithm, which is $O(E\prod_i n_i) = O(N\prod_i n_i)$, where N is the number of junctions of the line drawing and $E = O(N)$ is the number of segments (see, for example, [14]). Also, $O(\prod_i n_i)$ is the worst-case complexity of the trivial algorithm which tries every possible combination of legal labelings for each junction and stops whenever it finds one for which all segments are uniquely labeled. Therefore, the ratio R is a measure of how much we gain in processing time (in the worst-case, or with the trivial algorithm) by pre-processing the line drawing with a filtering algorithm before performing the search.

Fig. 13 shows the behavior of the base-10 logarithm of the pruning ratio R as a function of the number of junctions. Each point corresponds to a sample of 100 line drawings and

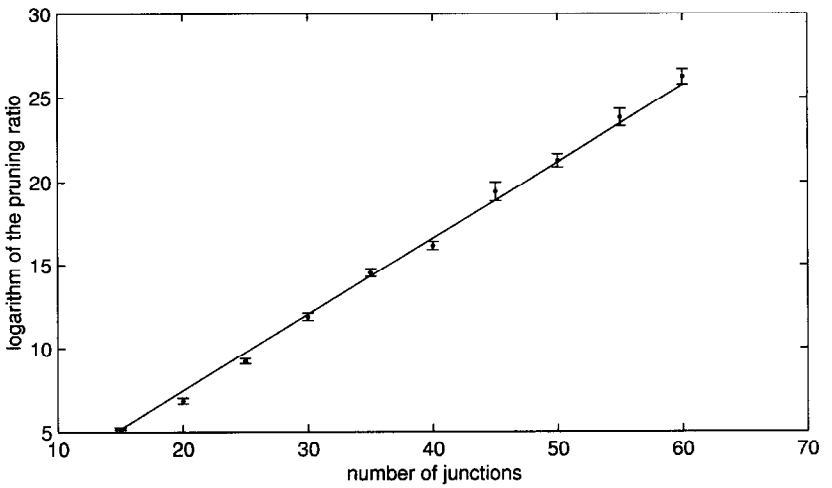

Fig. 13. The base-10 logarithm of the pruning ratio R, versus the number of junctions N. For each value of N, R is computed as the median ratio for a set of 100 line drawings. The plot is best-fitted by the line $\log_{10} R(N) = aN + b$ with $a = 0.46 \pm 0.01$, $b = -1.7 \pm 4.2$. The fit is acceptable: $\chi^2 = 12.4$, $P(\chi^2 > 12.4) = 0.13$.

reports the median of each sample. Of course this measure is independent of the successive methods by which one performs the search for a legal global labeling. The best-fit line $\log_{10} R(N) = aN + b$, with $a = 0.46 \pm 0.01$, $b = -1.7 \pm 4.2$. For these values we have $\chi^2 = 12.4$, with a corresponding confidence value of $P(\chi^2 > 12.4) = 0.13$, which makes an acceptable fit. The gain in the worst-case complexity is huge, and accounts for the far smaller number of hard cases that are found in the distribution if relaxation is used.

5. Discussion

In this paper we have analyzed the average-case complexity of labeling line drawings. To carry out this analysis, we have devised a method to generate random instances of polyhedral scenes. Such a method has enabled us to generate random line drawings which are guaranteed to be the projection of an actual scene. Afterwards, we have tested several different search methods to label the line drawings so constructed. (The labeling problem is in this case not the problem of deciding whether a labeling exists, but that of finding a particular labeling. This problem resembles more closely that which the brain solves well, namely the extraction of 3D information from a drawing.)

The experimental analysis carried out in this paper has allowed us to reach the following conclusions:

(1) The computational complexity of labeling line drawings is, in the median case, linear in the number of junctions, as long as an appropriate search method is used. This is probably due to the highly constrained nature of the labeling problem for trihedral scenes. Although it is possible to construct line drawings containing components which are difficult to label (see [13]), randomization in the construction of scenes makes these components unlikely to appear. This work substantiates the conjecture in [13] that the complexity of labeling line drawings might be polynomial for typical instances of the problem (the mean or median case).

(2) Relaxation plays an obvious role when the line drawing contains local inconsistencies: it is often the case that these inconsistencies are found out during the relaxation stage and this means a huge saving in the average computational time (see [22], where an analysis of the labeling problem for random graphs with possible inconsistencies has been carried out). In the case of line drawings derived as projections of (random) scenes, which by definition do not contain any inconsistencies, relaxation is still useful but for different reasons: it does not strongly affect the median-case complexity of the problem, but it is indispensable so as to narrow the distribution of the values of the computational time over the set of random instances so that most values are close to the median-case; also, it has a strong effect on the worst-case complexity, as clarified by the high values of the pruning ratio.

5.1. Relation to previous work

Our method for generating random instances of polyhedral scenes was inspired by Sugihara's work [30–32], in which realizability of a line drawing is checked by associating a plane to every polygon of the line drawing and solving a linear system where the

equations are the plane equations and the variables are the parameters of the planes. In this work the process is reversed: we generate the equations of the planes at random, and we construct a scene from them.

Our work should also be compared with that by Waltz [34], who reported linear-time performances for his labeling strategy, based on a number of examples generated manually. In that case, however, the linear complexity depended on the fact that in his examples relaxation was by itself usually enough to give an almost unique labeling. This might depend on the fact that the world considered in [34] includes shadows and cracks, has a far wider junction dictionary. Therefore the results cannot be immediately compared. In our case, the linear complexity of the median-case does not seem to depend on the relaxation stage, although the performances increase if relaxation is used.

Acknowledgements

We would like to thank David Mitchell, Michael Molloy, Stephen A. Cook, Hector Levesque, James McLean, Martin Held and Thomas Auer for helpful discussion. The referees have given us important advice on the statistical analysis of our data. We would also like to thank Mario Portoraro, Fernando Nuflo and Domenico Macrì for their technical help. This project was partially funded by the IBM Centre of Advanced Studies in Toronto and by the Institute for Robotics and Intelligent Systems, a Network of Centres of Excellence of the Government of Canada.

References

[1] A. Appel, The notion of quantitative invisibility and the machine rendering of solids, in: Proceedings of the ACM National Conference, Thompson Books, Washington, DC, 1967, pp. 387–393.
[2] B. Bollobas, Random Graphs, Academic Press, London, 1985.
[3] P. Cheeseman, B. Kanefsky, W.M. Taylor, Where the really hard problems are, in: J. Mylopoulos, R. Reiter (Eds.), Proceedings IJCAI-91, Sydney, Australia, 1991, pp. 331–337.
[4] M.B. Clowes, On seeing things, Artificial Intelligence 2 (1971) 79–116.
[5] B. Efron, R.J. Tibshirani, An Introduction to the Bootstrap, Chapman and Hall, London, 1993.
[6] J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes, Computer Graphics: Principles and Practice, Addison-Wesley, Reading, MA, 1990.
[7] M.R. Garey, D.S. Johnson, Computers and Intractability, W.H. Freeman, New York, 1979.
[8] E.M. Hampel, F.R. Ronchetti, P.J. Rousseeuw, Robust Statistics: The Approach Based on Influence Functions, Wiley, 1986.
[9] T. Hogg, C.P. Williams, The hardest constraint problems: a double phase transition, Artificial Intelligence 69 (1994) 359–377.
[10] D.A. Huffman, Impossible objects as nonsense sentences, in: B. Meltzer and D. Michie (Eds.), Machine Intelligence 6, Edinburgh University Press, Edinburgh, 1971, pp. 295–323.
[11] T. Kanade, A theory of Origami world, Artificial Intelligence 13 (1980) 279–311.
[12] L.M. Kirousis, Fast parallel constraint satisfaction, Artificial Intelligence 64 (1993) 147–160.
[13] L.M. Kirousis, C.H. Papadimitriou, The complexity of recognizing polyhedral scenes, J. Computer and System Sciences 37 (1988) 14–38.
[14] A.K. Mackworth, E.C. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, Artificial Intelligence 25 (1985) 65–74.
[15] A.K. Mackworth, E.C. Freuder, The complexity of constraint satisfaction revisited, Artificial Intelligence 59 (1993) 57–92.

[16] J. Malik, Interpreting line drawings of curved objects, Internat. J. Computer Vision 1 (1987) 73–103.
[17] P. Meer, D. Mintz, D.Y. Kim, A. Rosenfeld, Robust regression methods for computer vision: a review, Internat. J. Comput. Vision 6 (1) (1991) 59–70.
[18] R.E. Miles, Random polygons determined by random lines in the plane, in: Proceedings of the National Academy of Science 52 (1964) 901–907.
[19] D.G. Mitchell, H.J. Levesque, Some pitfalls for experimenters with random SAT, Artificial Intelligence 81 (1–2) (1996) 111–125.
[20] E.M. Palmer, Graphical Evolution, Wiley, New York, 1985.
[21] P. Parodi, The complexity of understanding of origami scenes, Internat. J. Computer Vision 18 (2) (1996) 139–170.
[22] P. Parodi, R. Lancewicki, A. Vijh, J.K. Tsotsos, Empirically-derived estimates of the complexity of labelling line drawings. Technical Report RBCV-TR-96-52, Department of Computer Science, University of Toronto, Toronto, September 1996.
[23] P. Parodi, V. Torre, On the complexity of labeling line drawings of polyhedral scenes, Artificial Intelligence 70 (1994) 239–276.
[24] J. Pearl, Heuristics, Addison-Wesley, Reading, MA, 1984.
[25] B. Selman, D.G. Mitchell, H.J. Levesque, Generating hard satisfiability problems, Artificial Intelligence 81 (1–2) (1996) 17–29.
[26] M. Straforini, C. Coelho, M. Campani, Extraction of vanishing points of images of indoor and outdoor scenes, Image Vision Computing 11 (2) (1993) 91–100.
[27] M. Straforini, C. Coelho, M. Campani, V. Torre, The recovery and understanding of a line drawing from indoor scenes, IEEE Trans. Pattern Analysis and Machine Intelligence 14 (2) (1992) 298–303.
[28] D. Stoyan, W.S. Kendall, J. Mecke, Stochastic Geometry and its Applications, Wiley, 1996.
[29] K. Sugihara, Picture language for skeletal polyhedra, Computer Graphics and Image Processing 8 (1978) 382–405.
[30] K. Sugihara, Mathematical structures of line drawings of polyhedrons, IEEE Trans. Pattern Analysis and Machine Intelligence 4 (5) (1982) 458–469.
[31] K. Sugihara, An algebraic approach to shape-from-image problems, Artificial Intelligence 23 (1984) 59–95.
[32] K. Sugihara, A necessary and sufficient condition for a picture to represent a polyhedral scene, IEEE Trans. Pattern Analysis and Machine Intelligence 6 (5) (1984) 578–586.
[33] J.K. Tsotsos, The complexity of perceptual search tasks, in: Proceedings International Joint Conference on Artificial Intelligence (IJCAI-89), Detroit, MI, 1989, pp. 1571–1577.
[34] D. Waltz, Understanding line-drawings of scenes with shadows, in: P.H. Winston (Ed.), The Psychology of Computer Vision, McGraw-Hill, New York, 1975, pp. 19–91.
[35] C.P. Williams, T. Hogg, Exploiting the deep structure of constraint problems, Artificial Intelligence 70 (1994) 73–117.