Rapid Incremental Architectural Modeling From Imprecise Perspective Sketches And Geometric Constraints

Alex Sosnov[1] Stéphane Huot[1] Pierre Macé[2]
Gérard Hégron[3]
[1]Département Informatique, Ecole des Mines de Nantes, France
[2]Tornado Technologies, Nice, France
[3]CERMA UMR CNRS 1563, Ecole d'Architecture de Nantes, France

Abstract

We present a system for rapid reconstruction of accurate 3D models of architectural sites from freehand perspective sketches and geometric constraints that describe their spatial structure. This system is based on new approach to resolution of such constraints: they are separated from preferences — that are provided by sketches — and resolved formally in serialized manner. The reconstruction process is separated into two independent phases: first a sketch is corrected to ensure that affine and certain metric constraints would be satisfied in space and next a 3D model is elevated by resolution of only projective constraints. Usage of a geometric algebra allows to represent solutions in a simple coordinate-free form and to perform computations with any precision. The obtained formal solutions may be reused to accelerate succeeding reconstructions when the user incrementally modifies the scenes.

Input of geometric constraints is simplified greatly by their automatic inference, user-validate suggestive analysis of sketch, usage of extensible set of primitives and interaction with a knowledge base. Furthermore, the system examines automatically structural and numerical consistency of imposed constraints to avoid contradictions.

Keywords: *Modeling, resolution of constraints, perspective sketches, interactive reconstruction, projective geometric algebra*

1. INTRODUCTION

Use of new computer-aided instruments in architecture still causes certain problems. Namely, because most of CAD systems are not suitable for conceptual design of architectural sites, this stage is mainly done on the paper. Thus, once spatial structure of a building is conceived, the designer should again model it with a CAD system and hence loses his time. Moreover, architects often need to put already existing or even destroyed buildings in new virtual contexts; hence they need to reconstruct 3D models from photographs or even engravings, where it is not possible to make direct measurements of features of a scene. Therefore, the problem of reconstruction of 3D shapes from freehand imprecise drawings or uncalibrated views is receiving increased attention.

However, reconstructing 3D models of architectural sites from imprecise conceptual drawings still remains a challenge. Indeed, free-form 3D sketching systems [6] are not suitable because they are not able to deal with well-constrained polyhedral forms widely occurring in architecture and do not provide accurate solutions. The common way to increase accuracy is to introduce spatial structure of a depicted scene that is described usually by 3D *geometric constraints*. One can infer such constraints by analyzing a sketch [4, 8] automatically. However, completely automated analysis causes misinterpretations. Furthermore, most of such approaches involve only axonometric projections, while perspective ones are rarely used [13] despite of the fact that they provide more information. On the other hand, spatial constraints can be explicitly declared by the user, who recognizes 3D elements and their relations. However, most of user-centered model-based systems [3, 7] are not flexible to describe complex scenes, sensitive to input errors and cannot be applied to imprecise drawings. All the above methods involve heavy numerical computations that are time consuming and subject to instabilities.

In this paper, we present a new system that allows to reconstruct rapidly accurate 3D models from *single* perspective views, which may be either imprecise freehand conceptual sketches or photographs, and geometric constraints that describe spatial structure of depicted scenes. We do not try to infer such a structure by analyzing a view, that is, constraints are imposed *explicitly*. Thus, the real value and convenience of such an *interactive* reconstruction system depend heavily on the following properties. First, it should allow to flexibly describe complex scenes without overloading the user with a lot of complicated work. Second, the system should allow to repeat reconstruction rapidly when the user incrementally enriches a scene with new details. Furthermore, such incremental reconstructions should provide predictable results, whereas it is often not a case if numerical optimization methods [3, 8] are used. Third, fast response time should be ensured, while the obtained model should be accurate, that is, all the imposed geometric constraints should be satisfied with high precision. Finally, inconsistent constraints should be detected to alert the user. All these goals are achieved by *uniform* representation of constraints, their *serialized formal* resolution and usage of a *geometric algebra* and *projective geometry*.

The paper is organized as follows. In section 2 we start with a brief description of our base principles of serialization of constraints and their formal resolution. Section 3 presents the Grassmann-Cayley algebra that is our main reasoning tool. In section 4 we describe how geometric constraints are represented uniformly using this algebra and projective geometry and how their structural consistency is ensured. Section 5 presents how input of such constraints is simplified and how they are inferred to reduce user work. In sections 6, 8 we discuss a rapid 3D geometric constraints solver that first constructs a general formal solution of an imposed reconstruction problem and next evaluate a concrete numerical solution as well as ensures numerical consistency of constraints. Such a resolution is possible only if a sketch is a true perspective projection of a scene described by these constraints. Methods that allow to transform any imprecise freehand sketch so that it would be such projection are described in section 7. Sections 9, 10 present our approach to repeated and incremental reconstruction. Finally, we discuss implementation and present several examples.

2. THE APPROACH

To allow the formal resolution of *constraints* that describe spatial structure of a 3D scene, they are separated from *preferences* that are provided by the scene projection — that is, by the corresponding perspective sketch. All the 3D constraints are *strictly* satisfied, whereas preferences are treated in *relaxed* manner. Namely, we first find the *correct* sketch that is merely close to original one while strictly satisfies all the projective consequences of the imposed constraints. Certainly, such consequences may not hold due to user errors and imprecisions. Next, we elevate formally a 3D model from the obtained true perspective projection by using the Grassmann-Cayley geometric algebra.

The key principle is the *serialization* of constraints. Each of them represents a projective (collinearity or coplanarity), affine (parallelism) or metric (orthogonality) relation. The idea is to elevate a 3D model from a true perspective projection by resolving only projective constraints that are easy to satisfy formally. On the other hand, if we consider a sketch only as a source of preferences and correct it to satisfy one by another projective consequences of affine and metric relations, constraints of the corresponding types would be satisfied *in space* without changing the formal solution. It allows to reduce 3D nonlinear problems to simple 2D linear or 2D minimization problems, which converge in few steps, and to represent a solution of reconstruction problem in the simple form of a set of *coordinate-free* constructive operations of the geometric algebra, and hence to provide numerical solutions of any desired accuracy.

We do not solve arising system of constraints globally. Instead, constraints are resolved formally by *local* methods. On the other hand, some *global* information is used to control resolution processes and hence to avoid well-known problems of local methods. Namely, we establish *operation priorities* to choose the most stable and computationally effective solution. Degenerate solutions are rejected formally by search for linear-dependent constraint configurations.

3. THE GRASSMANN-CAYLEY ALGEBRA

The Grassmann-Cayley algebra allows to express geometric constructions in space by projectively invariant coordinate-free algebraic statements [9]. Therefore, we choose it as the main reasoning tool for elevation of 3D models from their perspective projections. Furthermore, this algebra involves usage of projective geometry. Thus, we do not need to treat many special cases of parallelism and orthogonality of geometric objects; it simplifies our 3D geometric constraint solver and improves significantly its efficiency.

The Grassmann-Cayley algebra is a double algebra defined in projective space $P(\mathbb{R}^4)$. It provides two operators on projective subspaces of $P(\mathbb{R}^4)$ — that is, on points, lines and planes. The *join* operator $\vee$ generates a union of disjoint subspaces, the *meet* operator $\wedge$ generates intersection of subspaces[1]. Any k-dimensional subspace is represented by an *extensor* of step k — that is, an exterior product $a_1 \vee \ldots \vee a_k$ of vectors $a_1,\ldots,a_k$ of P. For instance,

$$\begin{aligned} \text{point } P_1 \vee \text{point } P_2 &= \text{line } P_1P_2 \\ \text{plane } \pi \wedge \text{line } L &= \text{point } \pi \cap L. \end{aligned}$$

The algebras implied by the *join* and the *meet* are related by the *duality* operator * as $(x \vee y)^* = x^* \wedge y^*$, where x, y are extensors of any step. The duality allows to express *joins* as matrix products

$$(x \vee y)^{*T} = x^*.y.$$

For any projective subspace, the duality operator generates a pseudo-orthogonal dual subspace and hence allows to match any given line with an orthogonal line and any given point with a plane.

Once any plane π is chose to represent the plane at infinity π_∞, it is possible to represent constructions that imply parallelism and orthogonality constraints [9]. For instance, a line L that is parallel to a given line L_1 and incident to a given point P can be constructed as $L = P \vee (L_1 \wedge \pi_\infty)$.

To perform computations from formal expressions of the Grassmann-Cayley algebra, 3D points are represented by 4D vectors of homogeneous coordinates, 3D lines are represented by matrices of their Plücker coordinates, and planes are represented by duals of points. All the computations are reduced to evaluations of exterior products for joins and matrix products for meets.

[1] However, for lines it generates a scalar, whereas its generalized version $\sqcap$ solves this problem [9].

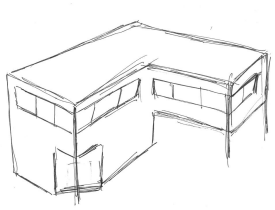


Figure 1: A scene that is hard to describe without low-level objects

4. REPRESENTATION OF CONSTRAINTS

We represent any 3D scene with only elementary 3D objects (*points*, *lines* and *planes*) and constraints (*collinearity* and *coplanarity* of points and lines, *parallelism* and *orthogonality* of lines and planes) [12]. The ability to manipulate such low-level objects straightforwardly allows the user to create easily models that otherwise would be hard or tedious to describe (Figure 1). On the other hand, the user is not burdened with a lot of complicated work due to automatic inference of many constraints (Section 4.1), suggestive analysis of a sketch, usage of extensible set of high-level primitives and possible application of the knowledge base (Section 5).

Using projective geometry and the Grassmann-Cayley algebra, we introduce very simple homogeneous representation of spatial structure of scenes. Indeed, any projective 3D constraint (e.g., collinearity or coplanarity) can be decomposed into a set of **incidences**. Incidence ε is a projectively invariant relation of containment of subspaces. For instance,

$$\text{points } A, B, C \text{ are collinear } \textbf{iff } \exists \text{ line } L\{A \,\varepsilon\, L, B \,\varepsilon\, L, C \,\varepsilon\, L\}.$$

It is also holds for affine constraints:

$$\text{lines } L_1 \parallel L_2 \textbf{ iff } \exists \text{ point } I \ \{L_1 \,\varepsilon\, I, L_2 \,\varepsilon\, I, I \,\varepsilon\, \pi_\infty\},$$

where π_∞ is the plane at infinity — that is, the projective completion of the 3D affine space. Finally, it also holds for metric orthogonality constraints. Indeed, let we call $\mathrm{orthinf}(A)$ the set of all the infinity points in directions orthogonal to A. For instance, for any line L $\mathrm{orthinf}(L)$ is a line at infinity, for any plane π $\mathrm{orthinf}(\pi)$ is a point at infinity. Then

$$\text{line } L \perp \text{ plane } \pi \textbf{ iff } \mathrm{orthinf}(\pi) \,\varepsilon\, L \textbf{ or } \mathrm{orthinf}(L) \,\varepsilon\, \pi.$$

Therefore, we can represent an arbitrary complex scene in the form of the **constraint graph**, which vertices represent elementary 3D objects, points at infinity and *orthinfs* while edges represent established and inferred incidence constraints.

4.1 Structural Consistency of Constraints

Uniformness of the above representation allows to define easily the notion of *structural consistency* of 3D geometric constraints [11]. Indeed, for geometry of incidences there are only five basic contradictory configurations, such as a pair of distinct lines that are incident to the same pair of points. A constraint graph is contradictory if it involves any of these configurations. The idea is to prevent creation of contradictory configurations during input of constraints. It is achieved by usage of **geometry laws** and **graph update procedures** [11, 12]. Such procedures are called automatically once new incidence constraints are established and infer new constraints according to the corresponding geometry laws. Thus, substantial part of geometric constraints is created by the system. If such a procedure infers a contradictory configuration, the created constraint is not consistent with already established ones. In this case, the user is alerted to correct the problem, while the system suggests possible solutions. Therefore, it is possible to create only **structurally consistent** constraint graphs that are not contradictory and contain all the geometric consequences of all the imposed constraints.

4.2 Projection of Constraints

Once we have a perspective sketch of a scene, projections of all the visible scene points and of all the points at infinity involved by parallelism and orthogonality constraints are known (the last ones are computed during the sketch correction as described in section 7). We add new constraints that relate these projections with their originals in 3D space and the center of projection (*eye*). Then, once positions of certain 3D objects of the scene are known, one can determine the eye position and "elevate" the scene from its projection as described in section 6.

5. INPUT OF CONSTRAINTS

5.1 Analysis of a Sketch

Whereas we use only elementary 3D geometric objects and constraints, our goal is to allow to describe spatial structure of complex 3D scenes in a simple, natural and intuitive manner. Thus, we need well-designed computer human interaction tools. As we have already mentioned, to reconstruct a 3D model the user should sketch its shape and specify constraints on its elements. It is a typical *Post-WIMP* interaction paradigm [14]. Thus, efficiency of such a system interaction can be improved greatly by anticipating user behavior and hence by analyzing his sketches in real-time to determine forms and constraints that the users means to do. The user may then concern himself to design objects in a creative way. However, completely automated analysis of sketches often causes misinterpretations, especially for perspective views (Section 1). Thus, we need user validations, whereas without disturbing the subject and stopping his creativity. We hence plan to use sketching and deduction tools in a gestural and suggestive interface [5].

According to imposed requirements, the first step of our approach to system interaction was to conduct a study of architect sketches — precisely, of strokes composing their.The conducted experiment was aimed to understanding architect gestures and strokes. The hypothesis is that studying those drawings and making statistics may enable us to isolate invariants and typical tasks of the sketching process. Our experiment consisted in several drawing tasks. Participants were architect students and architects. Their tasks were to draw buildings in single perspective views. The drawings were captured with a digitizing tablet. To analyze these drawings, we introduced a taxonomy of architectural strokes — that are, construction, primary, detail, and style ones. By analyzing on this corpus with this taxonomy, we identified three drawing contexts — that are, constructive, completion, and style. Actually, we work on autodetection of these contexts and their transitions; it allows to adapt the system to users behavior and facilitate knowledge extraction from sketches. For instance, we cannot ask the user to validate a system suggestion during the constructive phase because it will distract him from a creative work.

The obtained results allow to simplify low-level sketch processing and extraction of geometric properties. We implemented input filters to transform original digitized pen strokes to most structure data. First, we use a segmentation filter that is based on advanced vertices and corners detection [10]. This filter produces segments that are then merged with the approximation filter. The latter uses heuristics to combine certain close segments in one, because several strokes may define one segment in freehand sketches. We use results of the preceding study and user-dependant data to make this filter more effective. The last low-level processing step is to link segments at connection endpoints, because users tend to place stroke endpoints innacurately. The problem of detection of geometric constraints can be divided in two tasks. First, we use basic object recognition methods to extract simple 2D properties. Next, we have to focus on advanced techniques of sketch recognition. Note that extracted properties are only *suggested* to the user [5] and hence problems of completely automated analysis are avoided.

5.2 High-Level Primitives

While the system infers automatically substantial part of constraints, either by ensuring structural consistency or by analyzing sketches, describing complex scenes with only elementary 3D objects such as points and lines still may be a tedious task. Thus, it is possible to group elementary 3D objects and constraints into high-level **primitives**. For instance, the *rectangle* primitive represents a 3D rectangle and contains four coplanar points, incident to four lines that are pairwise orthogonal.

Unlike to traditional "black-box" shapes that have fixed set of properties [3], our primitives are similar to macros. Namely, a primitive is represented by a constraint graph (Section 4) that describe its constituent geometric objects and constraints. Once the user instantiates a primitive, the corresponding graph is simply added to a scene constraint graph. Such a representation provides significant advantages. First, a primitive may be an object of any degree of complexity. Second, the user is able to create easily his own primitives. Thus, it is possible to reuse scene elements that are tedious to describe every time they are required. For instance, consider the *balcony* primitive (Figure 2), which is a rectangular polyhedron with two faces removed. Although such elements appear often in architectural scenes, many modeling systems do not provide this primitive. On the other hand, it is difficult to model it every time, because creation of four rectangles and three constraints on their sides is required. Contrariwise, because of our representation, the user is able to describe this primitive only once and next simply instantiate it. Finally, because primitives are just instances of constraint graphs, it is possible to establish additional constraints on

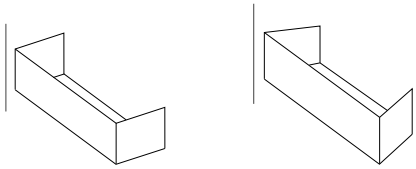


Figure 2: Two instances of the *balcony* primitive. Constraints that describe the primitive may be removed by the user (right)

their constituent elements or even *remove* them. For instance, the user can remove the constraint that sets parallelism of two lateral faces of an instance of the *balcony* (Figure 2, right).

5.3 Inference from a Knowledge Base

Because architectural sites are often designed following well-defined rules, it is possible to use domain-specific knowledge to simplify modeling of complex scenes. Namely, 3D geometric constraints can be inferred automatically from the knowledge base that stores information about spatial structure of buildings and their elements, their number and relations [1].

6. FORMAL RESOLUTION OF CONSTRAINTS

To reconstruct a scene is to determine positions in space of all its constituent elementary objects a scene so that imposed constraints would be satisfied. Unfortunately, most of geometric constraint solvers use expensive numerical optimization or symbolic approaches. However, due to usage of the *constructive* geometric algebra and projective geometry it is possible to build a *formal solution* of a given scene reconstruction problem by very efficient *propagation of known data* and hence to ensure fast response time. Furthermore, formal solutions are reused to accelerate significantly succeeding repeated and incremental reconstructions.

A formal solution is a set of coordinate-free expressions of the Grassmann-Cayley algebra that determine each of the elementary 3D objects composing a scene so that constraints imposed on it would be formally satisfied. For instance, if a point P is constrained as incident to a line L and a plane π, the formal solution will contain the expression $P = L \wedge \pi$, because a meet of a line and a plane determines their intersection. Using the Grassmann-Cayley algebra, it is possible to *determine* — that is, to construct to satisfy imposed constraints — any elementary 3D object once there is a sufficient number of *known* 3D objects incident to it [12]. However, because our constraint graphs contain all the consequences of all the constraints, an object can be determined often in different ways. On the other hand, operations of the Grassmann-Cayley algebra differ in accuracy and computation cost. Furthermore, due to separation of constraints and preferences, any 3D point should be determined via its projection only if it is impossible to construct it from other incident 3D elements. Therefore, we establish the *operation priorities* [12]. Our choice of priorities ensures that a whole formal solution does not depend on order of establishing constraints that is not a case for many solvers. Once an object is determined with the highest possible priority, the obtained solution is checked to be non-degenerate. We can analytically reject degenerate solutions. Indeed, degeneracy is caused by linear dependency of solution elements, whereas any *structurally* linear dependent configuration can be found and hence rejected by rapid searching the constraint graph and hence without any computations [12]. Once the best non-degenerate solution of the current local problem of constructing a 3D object via its known neighbors is found, it is added to the formal solution. Elements of remaining solutions form a list of object's *alternative evaluators*, which are used to check numerical consistency.

To start the resolution of constraints, some elementary 3D objects of a scene should be known. Once we have a perspective view of a scene, projections of all visible scene points and points at infinity are known (Sections 4.2, 7). Position of the center of projection is known from orthogonality constraints (Section 7.3). Thus, our algorithm is called as "elevation" from a view. Because a single perspective view corresponds to an infinite set of 3D models, the user should set additional constraints to choose the interesting one. Namely, the user should set as known certain elementary 3D objects by specifying, for instance, their depths. The number of parameters that determine such objects required and sufficient to resolve unambiguously the system of constraints describing a scene is the *number of degrees of freedom* of this scene. This number is determined automatically [12]. If the system is under-constrained, the user is demanded either to add new constraints or set as known more objects. If the system is over-constrained, its consistency is checked during numerical evaluation.

The described approach provides advantages that improve convenience and efficiency of our interactive modeling system. First, it is possible to reconstruct certain hidden parts of a scene. The user should just describe their spatial structure as usual and declare them as hidden. Projections of hidden 3D points and lines are not created and hence not used. However, hidden points and lines are constructed via its incident elements if there are enough constraints on them. Second, for certain configurations a formal solution does not depend neither on order of creating constraints nor on choice of objects to constrain degrees of freedom or to be hidden. For example, this is a case for instances of the *box* primitive. Thus, for such primitives it is possible to have predefined solutions; it accelerates the resolution of constraints for scenes containing their instances.

6.1 Example of elevation

Let we have a corrected perspective sketch $abcd$ of a 3D parallelogram $ABCD$ and geometric constraints that describe its structure: points A, B, C, D are coplanar (i.e., incident to a plane π), lines $AB \parallel CD$ and $AD \parallel CB$ (Figure 3). Vanishing points i, j are computed during the sketch correction as well as the center of projection O (Section 7). To elevate a 3D parallelogram from the sketch, we have to know one of its points. Thus, the system has one degree of freedom; it is determined automatically and the user is demanded to set as known one of the points. Let he specifies a depth of the point A. Then, this point is determined as $A = (O \vee a) \wedge \pi_d$, where π_d is a plane that sets its depth. Points at infinity I, J of the pencils (AB, CD), (AD, BC) are determined immediately as $I = (O \vee i) \wedge \pi_\infty$, $J = (O \vee j) \wedge \pi_\infty$, where π_∞ is the plane at infinity, lines $A \vee I$, $A \vee J$ are constructed and the plane $\pi = A \vee I \vee J$ is determined. Next, points $B = (O \vee b) \sqcap (A \vee I)$[2], $D = (O \vee d) \sqcap (A \vee J)$ are constructed, because due to choice of priorities we prefer to intersect lines of sight with lines. Finally, whereas the point C can be constructed as $C = (O \vee c) \wedge \pi$, our choice of priorities implies usage of projections of 3D points only if there is no other ways to construct them; thus, the point C is constructed as $C = (B \vee I) \sqcap (D \vee J)$.

[2] $\sqcap$ is a generalized join operator that allows to construct intersections of lines [9].

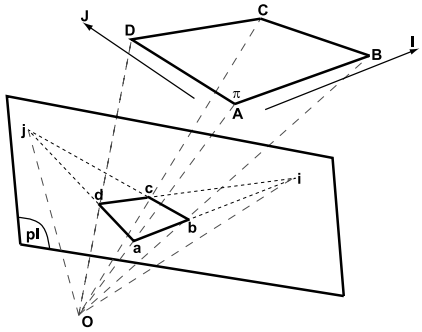


Figure 3: An example of elevation of a 3D parallelogram

7. SERIALIZED SKETCH CORRECTION

Whereas the formal elevation described in the previous section uses only projectively invariant operations of generation and intersection of subspaces, it allows to satisfy affine and even certain metric (orthogonality) constraints *in space* without changing the formal solution. Indeed, such constraints imply **projective consequences** — that is, constraints on elements of a sketch that should be respected to ensure that the spatial ones hold. For instance, images of 3D parallel lines should intersect at the same vanishing point. However, such consequences do not hold due to errors in freehand sketches (Figure 4). Moreover, it is quite impossible to draw a perspective sketch without such errors. Thus, the idea is to correct a sketch to respect these consequences and to find a center of projection so that all the affine and orthogonality constraints would be exactly satisfied in space once we evaluate a formal solution (Section 6).

A sketch is corrected in the *serialized* manner. First, we construct a **formal correction plan** that expresses all the sketch elements (images of 3D points and lines) so that all their incidence relations, which are projectively invariant, would be satisfied once the sketch will be redrawn from vanishing points and certain *free points*. Then, we compute vanishing points to satisfy projective consequences of affine constraints. Next, vanishing points are corrected and a center of projection is found so that 3D orthogonality constraints would be satisfied. Finally, we redraw the sketch from these vanishing points by using the formal correction plan. Thus, a true perspective projection is obtained.

7.1 Formal Correction for Incidence Constraints

The idea to satisfy incidence constraints between sketch points and lines and construct a formal correction plan is to redraw the sketch from the most constrained to less constrained points. The required order can be obtained by *propagation of degrees of freedom*. Namely, we erase step by step sketch points and their incident sketch lines according to degrees of freedom (DOF) of these points. A number of DOFs of a sketch point is defined as the number of its incident lines that are not erased yet. The point is constructed as the intersection of these lines, while all the already erased incident lines are constructed from the point. To express such constructions, we use the 2D Grassmann-Cayley algebra. Because it is possible to compute *exactly* only intersections of pairs of lines, over-constrained configurations are possible (imagine a corridor). However, we treat such configurations formally as described in [12]. At each step, we erase a point with the minimal number of DOFs. The reversed order of these operations is the required one.

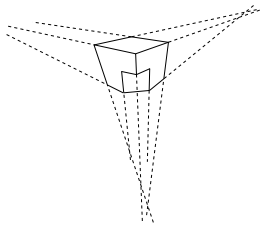


Figure 4: An incorrect sketch: projective consequences of affine constraints are not respected

7.2 Affine Constraints

To satisfy 3D affine constraints, images — that are, lines on a sketch — of 3D parallel lines should intersect at the same points, while vanishing points of images of 3D lines such that certain of are coplanar should be aligned along vanishing lines. We compute vanishing points of images of 3D parallel lines so that it would be close as possible to these images by using weighted linear least squares. Next, vanishing lines are evaluated by linear regression on the computed vanishing points. Finally, vanishing points are projected on the computed horizon lines. Thus, once we redraw a sketch from these points, it will be distorted minimally.

7.3 Orthogonality Constraints

3D orthogonality constraints determine a position of the center of projection (*eye*) such that they would be satisfied in space after the formal elevation from a sketch. Namely, if we have three pencils of 3D parallel lines declared as pairwise orthogonal, their vanishing points determine the principal point — that is, the intersection of the optical axis with the sketch plane — as well as the focal length [2]. For instance, the principal point is the orthocenter of a triangle built on these vanishing points. Thus, is we have several triples of such orthogonal pencils, we correct the corresponding *vanishing triangles* so that their orthocenters would coincide and they would define the same focus distance [12].

Once the eye position is computed, the image of the absolute conic is known. Vanishing points of remaining pairs of orthogonal pencils of 3D parallel lines should be conjugate with respect to this image [7]. Thus, we correct them to satisfy this constraint by solving simple minimization problem, which converges in few steps.

Because a 3D line is orthogonal to plane iff it is incident to the plane's *orthinf* (Section 4), an image of the line on a sketch should be incident to a projection of the *orthinf* of the plane. We can correct a sketch to satisfy this constraint if degrees of freedom of a plane involve only translation, because in this case it is possible to compute the projection of its *orthinf* without elevation of the plane.

8. NUMERICAL EVALUATION

Once we have constructed a formal solution of a scene reconstruction problem, computed position of the center of projection and corrected all the sketch points, we compute a **numerical solution**: we evaluate Plücker coordinates of all the elementary 3D objects composing the scene. First, the user is demanded to specify numerical parameters of 3D objects that were required to be known during the formal resolution of constraints (Section 6) — for instance, their depths or distances from the eye. Then we put a sketch

and the computed center of projection into 3D projective space and evaluate Plücker coordinates of each of remaining scene objects by computing a coordinate expression of its determining operation. To compute reliably and efficiently, we use principles from floating point filters: all the computations are performed in *doubles*, and only if a problem occurs, exact arithmetics is used as backup. On the other hand, we can achieve extremely high precision by using real or rational arithmetics.

A numerical solution of a scene reconstruction problem is **consistent** iff: (1) all the elementary 3D scene objects are not degenerate, (2) all the points and lines that are not inferred by affine or orthogonality constraints are not incident to the plane at infinity, and (3) all the incidence constraints presented in the constraint graph are satisfied. Thus, the **numerical consistency** differs from structural one (Section 4.1) because it depends on particular values of user specified numerical parameters — for instance, consider a plane determined by a join or three points that are not *declared* or *inferred* as collinear. However, the user may specify coordinates (depths) of these point so that they would be in fact collinear. We ensure numerical consistency by using alternative evaluators (Section 6) obtained during the formal resolution [11, 12]. If contradictions are found, the user is alerted to change provided parameters.

9. REPEATED RECONSTRUCTION

Once a 3D model is reconstructed, it is possible to rapidly obtain the new one when the user changes certain scene parameters. In fact, the user may move certain points of a drawing, or change spatial positions of certain elementary 3D objects that were required to be known (Section 6). Such changes do not affect formal solutions neither for scene reconstruction nor for sketch correction problems.

Thus, in the first case, vanishing points of parallel lines incident to the moved points are affected by the these points and hence are recalculated (Section 7.2). Because the center of projection is determined by vanishing points, the user is demanded to choose whether its position has to be changed. In this case, the process of resolution of orthogonalities and eye calibration is repeated from scratch (Section 7.3). Otherwise, affected vanishing points are corrected so that it would determine the previous eye position. Finally, the numerical evaluation is performed as usual, though the user is not demanded to set known objects, since their positions are retained from the previous reconstruction pass (Figure 5).

In the second case, positions of the view points and hence the vanishing points and the center of projection are not affected. Therefore, we just repeat the numerical evaluation (Section 8) by using changed spatial positions of known elementary 3D objects.

Because formal solutions are reused, our system allows to rapidly repeat reconstructions and produce accurate and *predictable* results.

10. INCREMENTAL RECONSTRUCTION

Once a 3D model is reconstructed, it is possible to rapidly obtain the new one when the user enriches a scene with new details. In fact, new elements may have no any relations with the already reconstructed ones, or may be linked with the old ones by constraints.

In the first case, the algorithm for formal constraint resolution (Section 6) is applied *only* to the *isolated* subgraph of the scene constraint graph that corresponds to the new elements. Next, new part of the drawing is corrected as an independent view (Section 7) and its vanishing points are estimated (Section 7.2). If new elements impose orthogonality constraints, these vanishing points affect the center of projection. Thus, the user is demanded to choose whether the eye position has to be changed. In this case, resolution of orthogonalities and eye calibration (Section 7.3) are repeated for the *whole* view. Therefore, the view would be completely recorrected and hence *all* the elementary 3D objects would be reevaluated. Thus, the user is advised to choose this alternative only if complexity of the new part of the scene is comparable with the old one. Otherwise, new orthogonal vanishing points are corrected so that they would determine the previous eye position. Finally, only the *new* elementary 3D objects are evaluated and hence the already reconstructed part of the scene remains unchanged).

In the second case, if spatial structure of the old part of the scene is changed after establishing new constraints and inferring their consequences, we just repeat the whole reconstruction process from scratch. Otherwise, old elementary 3D objects are declared as known and the algorithm for formal resolution of constraints is applied. This algorithm takes into account only the new part of the scene constraint graph and constructs the formal solution only for the *new* elements so that it would satisfy to constraints linking them with the already reconstructed ones. The same approach is applied to the sketch correction. Next, new vanishing points are estimated and the user is demanded to choose whether the old part of the view has to be changed. Certainly, the user is advised to choose this alternative only if the new part is as complex as the old one. Otherwise, new vanishing points are corrected so that it would determine the already calculated eye position while points from the old part of the view would intact. Finally, only the obtained *new* parts of the formal correction and reconstruction plans are evaluated. Thus, while the old part of the scene remains unchanged, constraints that relate it with the new reconstructed part are satisfied (Figure 6).

Because new parts of a scene constraint graph are usually simpler than the whole one and the previously obtained parts of a 3D model remain unchanged, our approach allows to rapidly perform incremental reconstructions and produce *predictable* results.

11. IMPLEMENTATION AND RESULTS

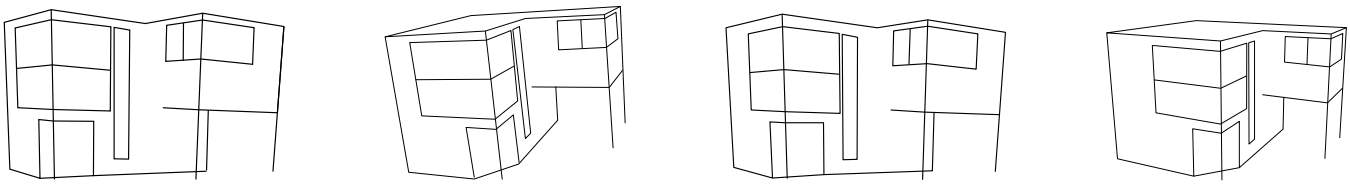
Our system has been implemented on PC workstations using C++ for the kernel and Java for the GUI.

11.1 Reconstruction from Sketch

The figure 7 represents an example of reconstruction from a single imprecise hand-drawn perspective sketch. We used a drawing 7(a) that was created by one of the authors of the paper, who is not a professional architect. The most time-consuming part of the acquisition was the description of spatial structure of the scene. The scene was represented by 12 primitives and 6 elementary objects (pillars and windows), there were 12 constraints specified explicitly. The user-assisted drawing and constraining phase took around 6 minutes.Once all the constraints were explicitly imposed or inferred, the scene constraint graph contained 46 points and their 43 projections, 43 lines and 43 lines of sight, 9 planes and set approximately four hundreds elementary incidence relations. Once the graph was constructed, the model 7(c) was obtained in 1 second on the usual PIII 733 MHz workstation.

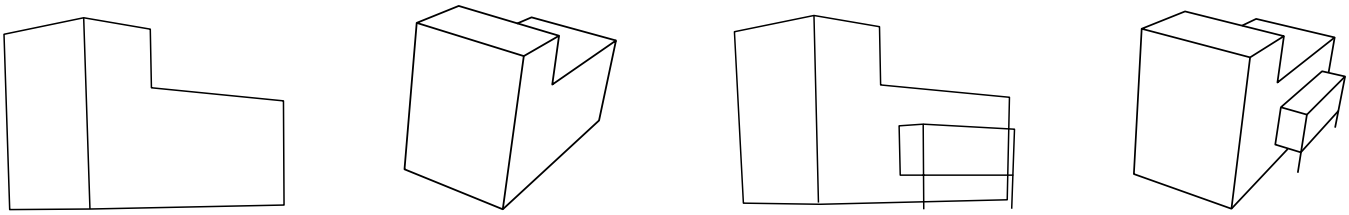
11.2 Application to Photographs

Our approach can be applied to photographs as well as to freehand perspective sketches. For instance, the figure 8 represents an exam-



(a) (b) (c) (d)

Figure 5: Repeated reconstruction. (a) The initial view. (b) The initial 3D model. (c) The modified view. The left wall is enlarged. (d) The corresponding changed 3D model. The model changes in a predictable manner.



(a) (b) (c) (d)

Figure 6: Incremental reconstruction. (a) The initial view of a building. (b) The initial 3D model. (c) The modified view. The annex with pillars and walls that are parallel to the building is added. (d) The obtained 3D model. The old part of the scene remains unchanged, while the parallelism constraints are satisfied.

ple of reconstruction from a single photograph. Because we have only one image of the building 8(a), there is no any information on its back face. On the other hand, it is possible to reconstruct hidden parts of a scene once there are enough constraints on them (Section 6). Thus, we may describe the back face and other elements of the building that are not visible on the photograph as we imagine (Figure 8(b), light grey lines). Note that because projections of hidden 3D points are not used for the reconstruction, the user may set hidden points in arbitrary convenient positions. The 3D model (Figure 8(c)) was obtained in three incremental stages. First, the user described the basic shape of the building. Next, the front entrance was detailed. Finally, windows in the front wall were created. Three user-assisted phases took around 12 minutes in total. The scene constraint graph finally contained 70 points and their 57 projections, 82 lines and 57 lines of sight, 14 planes and several hundreds elementary incidence relations. On the first stage, the 3D model was obtained in 1 second, two enriched models were obtained in 1 second in total on the usual PIII 733 MHz workstation.

12. CONCLUSION

We have presented a system to accurate and rapid reconstruction of 3D models of architectural sites from perspective views, which may be either freehand sketches or photographs, and constraints that describe their spatial structure. Whereas we use only elementary objects and constraints to ensure homogeneous internal representation, input of constraints is greatly simplified. Furthermore, their consistency is ensured automatically. Thus, our system delivers simplicity of use as well as flexibility.

The usage of formal solutions allows to rapidly perform incremental reconstructions with predictable results and to provide any desired accuracy. The reconstruction is simplified by separating correction of sketches to recover input errors and elevation of 3D models from true perspective projections. Serialization of constraints and usage of local methods allow to avoid expensive numerical computations and hence improves stability and ensures fast response time.

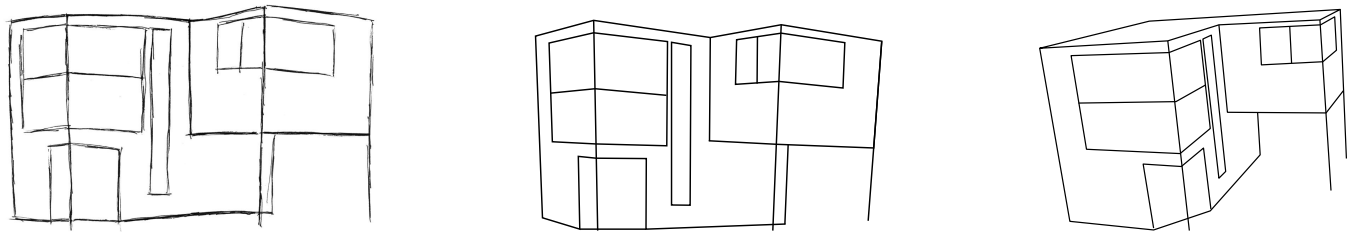
The principle of serialization of constraints can be extended to other metric relations, for instance, to constraints on ratios of distances. For certain configurations, it is also possible to correct a sketch so that such metric constraints would be satisfied in space. Because metric constraints may be subject of contradictions with projective or affine ones, they should be resolved in a specific order. We have developed an algorithm to determine required order. Furthermore, many constraints that seems to be metric and occur often in architectural scenes — for instance, transitions, central or plane symmetries — in fact can be decomposed into projective ones and hence resolved with only the Grassmann-Cayley algebra.

REFERENCES

[1] Boucard, D., Huot, S., Colin, Ch., Siret, D., and Hégron, G. *An image-based and knowledge-based system for efficient architectural and urban modeling. To appear in: Proceedings of ACADIA (2002)*

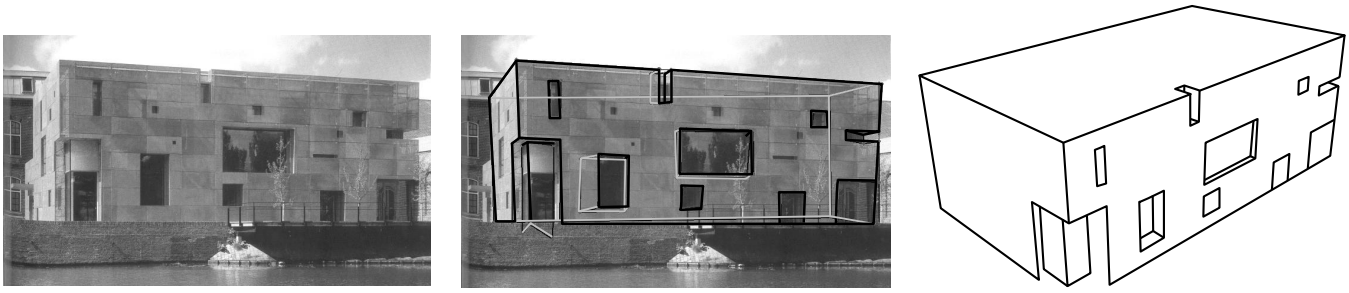
[2] Caprile, B., Torre, V. *Using vanishing points for camera calibration. International Journal of Computer Vision 4(2) (1990) 127–140*

[3] Debevec, P.E., Taylor, C.J., and Malik, J. *Modeling and rendering architecture from photographs: a hybrid geometry-*



(a) (b) (c)

Figure 7: 3D Reconstruction from a single sketch. (a) An imprecise perspective hand drawing of an office building. (b) The corrected sketch satisfying to all the projective consequences of imposed constraints. (c) The obtained 3D model.



(a) (b) (c)

Figure 8: 3D Reconstruction from a single photograph. (a) An image of the *maison verte*. (b) The corrected drawing (in *black* lines) and hidden parts of the scene (it *light grey* lines). (c) The obtained 3D model.

and image-based approach. In: Proceedings of ACM SIGGRAPH (1996) 11–20

[4] Eggli, L., Hsu, Ch., Brüderlin, B., and Elbert, G. *Inferring 3D models from freehand sketches and constraints. Computer-Aided Design 29(2) (1997) 101–112*

[5] Igarashi, T., Hughes, J.F. *A suggestive interface for 3D drawing. In: Proceedings of 14th Annual Symposium on User Interface Software and Technology (2001) 173–181*

[6] Igarashi, T., Matsuoka, S., and Tanaka, H. *Teddy: A Sketching Interface for 3D Freeform Design. In: Proceedings of ACM SIGGRAPH (1999) 409–416*

[7] Liebowitz, D.,Criminisi, A., and Zisserman, A. *Creating architectural models from images. In: Proceedings of EuroGraphics (1999) 39–50*

[8] Lipson, H., Shpitalni, M. *Optimization-based reconstruction of a 3D object from a single freehand line drawing. In: Computer-Aided Design 28(8) (1996) 651–663*

[9] Macé, P. *Tensorial calculus of line and plane in homogenous coordinates". In: Computer Networks and ISDN Systems (1997) 1695–1704*

[10] Sezgin, T., Stahovich, T., Davis, R. *Sketch based interfaces: early processing for sketch understanding. In: Proceedings of Perceptive User Interfaces Workshop (2001)*

[11] Sosnov, A., Macé, P. *Rapid algebraic resolution of 3D geometric constraints and control of their consistency. To appear in: Proceedings of ADG (2002)*

[12] Sosnov, A., Macé, P., and Hégron, G. *Semi-metric formal 3D reconstruction from perspective sketches. In: Proceedings of ICCS (2002) Part II 285–295*

[13] Ulupinar, F., Nevatia, R. *Constraints for interpretation of line drawings under perspective projection. In: CVGIP: Image Understanding 53(1) (1991) 88–96*

[14] VanDam, A. *The Human Connection: Post-WIMP User Interfaces. In: Communications of The ACM 40(2) (1997) 63–67*

About the authors

Alex Sosnov is a PhD student at Ecole des Mines de Nantes, France. E-mail: **sosnov@emn.fr**

Stéphane Huot is a PhD student at Ecole des Mines de Nantes, France. E-mail: **huot@emn.fr**

Pierre Macé is a scientific consultant at Tornado Technologies, Canada. E-mail: **Pierre.Mace@emn.fr**

Gérard Hégron is director of CERMA laboratory of CNRS at Ecole d'Architecture de Nantes, France.
E-mail: **gerard.hegron@cerma.archi.fr**