Rapid Algebraic Resolution of 3D Geometric Constraints and Control of their Consistency
Extended Abstract

Alex Sosnov[1] and Pierre Macé[2]

[1] Ecole des Mines de Nantes, Département d'Informatique, 44307 Nantes Cedex 3, France
[2] Tornado Technologies Inc., 7–11 av. Raymond Feraud, 06200 Nice, France

1 Introduction

CAD systems still remain modeling tools instead of being desing ones. Therefore, the stage of conception is usually still done on the paper; once spatial structure of an object is conceived, the designer should again model it with a CAD system and hence loses his time. Therefore, the problem of conception really aided by computer attracts attention of many researchers. The major trend is to improve human-computer interface of CAD systems by automating certain geometric constructions: The designer defines an object by describing its properties and the system infers an appropriate 3D form. The common way to describe properties of an object is to use *3D geometric constraints.*

Most of the geometric constraint solvers presented in the last years [2] use either expensive and imprecise numerical optimizations or slow symbolic term-rewriting or problem decomposition approaches. These solvers work usually in a 3D euclidean space and hence should treat many different cases of parallelism and orthogonality of geometric objects; it increases complexity and decreases efficiency. However, there exist tools that allow to improve significantly efficency and "intelligence" of geometric constraint solvers, that are *geometric algebras.* In this paper, we present a new method that allows to formally resolve systems of 3D projective, affine, and orthogonality constraints in a very efficient local constructive manner as well as detect rapidly logical and numerical contradictions. This method is based on uniform projectively invariant representation of constraints and usage of the Grassmann-Cayley geometric algebra.

2 The Gina project

This work is a part of the **Gina** project, which is aimed to reconstruction of 3D forms from freehand drawings and statements of known properties. Distinctive features of the **Gina** approach are *separation* of 3D constraints, which describe spatial structure of a scene, and 2D preferences, whicht describe the scene projection, and *serialized resolution* of constraints [3]. Because of the particularly important role played by the projective constraints for the reconstruction from a perspective view, we use the Grassmann-Cayley algebra, which allows to easily express projectively invariant geometric statements by coordinate-free algebraic ones. However, the resulted constraint solver [4] appears to be more general. In this paper, we present a generalization of this solver to any systems of 3D projective, affine and orthogonality constraints and improved techniques of maintenance of their consistency.

3 The Grassmann-Cayley Algebra

The Grassmann-Cayley algebra is a double algebra defined in the projective space $P(\mathbb{R}^4)$. It provides two operators, which operate on projective subspaces — that is, on projective points, lines and planes. The *join* operator $\vee$ generates a union of disjoint subspaces. The *meet* operator $\wedge$ (or $\sqcap$) generates an intersection of subspaces. The unary *duality* * operator generates a pseudo-orthogonal dual subspace and hence allows to match an orthogonal line with any given line, to match any given point with a plane, and to exchange meets and joins.

The Grassmann-Cayley algebra allows to simplify representation of constraints. In fact, any projective constraint can be decomposed into a set of incidences. Incidence is a binary 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 also holds for affine properties

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

where π_∞ is the infinite plane — that is, the projective completion of the 3D affine space. It also holds for orthogonality properties. Namely, let we call $\mathrm{ortinf}(A)$ the set of all the infinite points in the direction orthogonal to the A. For instance, $\mathrm{ortinf}(L)$ is the infinite line, $\mathrm{ortinf}(\pi)$ is the infinite point. Then

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

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 their duals. All the computations are reduced to evaluations of exterior products for joins and simple matrix products for meets.

4 Representation of Constraints

We use only low-level elementary 3D objects (*points, lines* and *planes*) and constraints (*collinearity* and *coplanarity* of points and lines, *parallelism* and *orthogonality* of lines and planes). On the other hand, elementary objects can be grouped in primitives of any complexity. Such a representation provides flexibility and allows to describe easily arbitrary complex scenes. Since we use the geometric algebra that operates with invariant intersections and unions of subspaces, any constraint is expressed as a set of *incidences* as described in the section 3. Therefore, we can represent an arbitrary complex scene in the *simple* and *homogeneous* form of the **constraint graph**, which vertices represent elementary 3D objects while edges represent established and inferred incidence constraints.

5 Structural Consistency of Constraints

Simplicity and uniformity of representation allow to easily define the notion of *structural consistency* of geometric constraints. Logically, a system of constraints is **contradictory** if absurdity can be derived from it. For the geometry of incidences, there are only five basic contradictory configurations, such as two lines that are incident to the same pair of points. Thus, a constraint graph is contradictory if it has a subgraph corresponding to one of these configurations. The idea to ensure structural consistency is to prevent creation of contradictory configurations. To achieve it, we use a notion of **geometry law**, which is a function from a constraint to its **consequences**. For the geometry of incidences, there are only three basic laws. A constraint graph is **saturated** if it contains all the consequences of all the imposed constraints. During creation of such a graph it is possible to prevent logical contradictions. Namely, we define the *graph update procedures*, which represent the basic geometry laws and called automatically once new incidence constraints are estalished. An update procedure infers new constraints, which are consequences of the created one and the applied law. If such a procedure infers a basic contradictory configuration, the created constraint is not consistent with already established ones. Therefore, we use only saturated graphs. Thus, a constraint graph is **structurally consistent** if it is saturated and not contradictory.

6 Two Stages of Resolution of 3D Geometric Constraints

The process of resolution of constraints is decomposed into the independent *formal* and *numerical* stages. We first construct a *formal solution*, which is a set of 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. Second, it the user needs not only a formal solution and hence specifies numerical parameters (i.e., positions in space) of certain scene objects, we evaluate a *numerical solution* from the obtained formal one and provided parameters and check its *numerical consistency*. Such a decomposition allows to resolve constraints efficiently in the incremental manner, to construct solution rapidly when the user changes numerical parameters, to distinguish between inconsistencies and inaccuracies, which are caused by use of inexact arithmetics, and to recover such inaccuracies.

7 Stage I: Formal Resolution

Because our goal is efficiency, we do not use slow term-rewriting and problem decomposition methods to find a formal solution for a given constraint graph. The key principle of our approach is to use the *constructive* geometric algebra, which allows to build the solution by very efficient *propagation of known data*. Indeed, by using the Grassmann-Cayley algebra, it is possible to *determine* any elementary 3D object once there is a sufficient number of *known* 3D objects incident to it [3]. However, because we use saturated constraint graphs, an object may have a redundant number of known neighbors (in the graph) and hence can be determined in different ways. On the other hand, operations of the Grassmann-Cayley algebra differ in accuracy and numerical computation cost. Therefore, we establish the *operation priorities* [4]. Our choice of priorities ensures that the whole formal solution does not depend on the order of establishing constraints, that is not a case for many solvers. Once an object is determined with the highest possible priority (it is ensured by usage of the priority queue), the obtained solution is checked to be nondegenerate. We can *analytically* reject degenerate solutions. Indeed, degeneracy of a local solution is caused by linear dependency of its elements. Because we use only saturated graphs, which contain *all* the possible incidences, any *structurally* linear dependent configuration can be found and hence rejected just by searching the constraint graph and hence without any computations. Once the best nondegenerate solution of the current local problem of determining 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. The determined object becomes known and is propagated to its neighbors that are not known yet.

To start the resolution of constraints, the user should provide some known elementary 3D objects of a scene. The number of such objects required and sufficient to resolve the system of constraints of the scene is the number of *degrees of freedom* of the scene. This number is determined automatically. If the system is under-constrained, the user is demanded either to add some new constraints or set as known more objects, or the resolution process is stopped. If the system is over-constrained, its consistency is checked during numerical evaluation.

During the resolution of constraints, we use *serialized* geometric constructions. Namely, all the geometric constructions provided by the Grassmann-Cayley algebra are separated into three levels:

1. On the *projective level*, we use only the operations that does not involve the infinite plane. Therefore, projective problems can be solved.
2. On the *affine level*, we use all the operations from the projective level and can construct (determine) infinite points and lines by meets with the infinite plane. Therefore, affine problems can be solved.
3. On the *semi-metric level*, we use all the operations from the affine level and can construct (determine) orthogonal projective subspaces by using *ortinfs*. Therefore, orthogonality problems can be solved.

These levels are used depending on the type of a problem under consideration. For instance, when we solve problems of geometric constructions in 3D space (e.g., construct a box from given two vertices, a point incident to an edge, and a point incident to a face), all the levels are available.

7.1 Elevation from a Perspective Projection

Reconstruction from a perspective projection is a particular case of our problem of resolution of 3D geometric constraints. In this case, in addition to constraints that describe spatial structure of a scene, we have a *perspective view*, which may be either a freehand sketch or a photograph.

Once we have a perspective view of a scene, projections of all the visible scene points and of all the infinite points involved by parallelism (i.e., vanishing points) and orthogonality constraints are known. We add new constraints that relate these projections with the camera and their originals in the 3D space. Therefore, once certain 3D objects of the scene are known, one can determine position of the camera and "elevate" the scene from the projection by using the propagation of known data described above [4]. Once the camera is known, all the infinite points are known, because they can be determined as meets of optical rays passing by the corresponding vanishing points with the infinite plane. On the other hand, vanishing points are subject to projective consequences of 3D affine and orthogonality constraints. Therefore, we cannot create infinite points directly in the 3D space and hence do not use the affine and the semi-metric levels of constructions[1]. However, spatial affine and orthogonality constraints will be satisfied only if their projective consequences are respected in the view. It is often not a case either for freehand sketches or photographs due to user errors or camera distortions, respectively. However, we have proved that it is possible to correct the view and find the position of the center of projection so that all the required projective consequences would be satisfied [4].

8 Stage II: Numerical Evaluation

If the user specifies numerical parameters of certain elementary 3D scene objects, we evaluate the *numerical solution* from the obtained formal one and the provided parameters. Namely, we first set Plücker coordinates of the specified objects according to the provided parameters and next evaluate Plücker coordinates of each of the remaining scene objects by computing a coordinate expression of its determing operation. To compute reliably and efficiently, we use the principles from floating point filters. Namely, all the computations are first performed in *doubles*, and only if a problem occurs, exact arithmetics [1] is used as backup. On the other hand, if extremely high precision is required, we apply rational arithmetics.

[1] except for orthogonality problems that do not involve infinite points, such as orthogonalities between lines that do not belong to any pencils of parallel lines

Because a formal solution is constructed by using only a constraint graph, it does not depend on the user specified numerical parameters. Therefore, once the user changes coordinates of any elementary 3D scene object, we *only* recompute objects that depend on it according to the formal solution. Thus, numerical solutions can be reevaluated very efficiently.

9 Numerical Consistency of Constraints

A numerical solution of a system of constraints 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 infinite plane, and (3) all the incidence constraints presented in the constraint graph are satisfied. Thus, numerical consistency differs from structural one because it depends on particular values of user specified numerical parameters of certain scene objects. For instance, let a plane is determined as a join of three points that are not *declared* or *inferred* as incident to the same line. The user may specify coordinates of these points so that they would be really collinear.

If a result of evaluation of any elementary 3D object is a null tensor (i.e., degenerate) or represent an object at infinity while should represent the usual one, the object is reevaluated by using solutions provided by its *alternative evaluators*, which are obtained during the formal resolution. Real arithmetics is used. If a number of coincident[2] *alternative results* is more than a certain threshold, there is just an error of precision and one of these results is returned as correct [4]. Otherwise, there is a contradiction.

Once an elementary 3D object is evaluated correctly, we test it to be incident to all of its *alternative evaluators* to satisfy the third criteria of numerical consistency. If one of these constraints is not respected, we reexamine it by using real arithmetics. If it is still not respected, there is a contradiction.

Therefore, use of independent notions of *structural* and *numerical* consistency allows to prevent easily creation of logical contradictions in constraints as well as to detect rapidly contradictions in numerical parameters specified by the user.

10 Conclusion

We presented the method that allows to construct rapidly consistent solutions of arbitrary complex systems of 3D geometric constraints by performing local elementary geometric constructions. Serialization of geometric constructions allows to treat projective, affine, and orthogonality constraints uniformly. Although we cannot treat now constraints on *distances*, many complex problems can be formulated in the projective manner by using only the supported types of relations. On the other hand, our future work is to add metric relations, namely, distances. The idea is to use again serialized resolution and hence to separate metric constraints from other ones, to add a new level of geometric constructions involving

[2] according to a given precision

distances, then to use our recent algorithm or a numerical solver to obtain a required order of resolution of distance constraints, and finally to set priorities to ensure that remaining projective, affine and orthogonality constraints would be resolved according to the obtained order. We check numerical consistency as usual by using alternative evaluators.

References

1. Burnikel, C., Fleischer R., Mehlhorn K., and Schirra S.: Efficient exact computation made easy. In: Proceedings of 15th ACM Symposium on Computational Geometry (1999) 341–350
2. Eggli, L., Hsu, Ch., Brüderlin, B., Elbert, G.: Inferring 3D models from freehand sketches and constraints. Computer-Aided Design, **29(2)** (1997) 101–112
3. Lhomme, O., Kuzo, P., and Macé, P.: A constraint-based system for 3D projective geometry. In: Brüderlin, B., Roller, D. (eds.): Geometric Constraint Solving and Applications. Springer Verlag, Berlin Heidelberg New York (1998)
4. Sosnov, A., Macé, P., and Hégron, G.: Semi-metric formal 3D reconstruction from perspective sketches. In: Proceedings of ICCS (2002) Part II 285–295