Implicit handling of geometric relations in an existing modeler

Salim Belblidia, Emmanuel Alby

IMPLICIT HANDLING OF GEOMETRIC RELATIONS IN AN EXISTING MODELER

S. BELBLIDIA, E. ALBY
MAP-CRAI CNRS Research Unit – School of Architecture of Nancy
2 rue Bastien Lepage, 54000 Nancy, France.
{belblidia,alby}@crai.archi.fr

Abstract. This paper presents a constraint-based modeling system, integrated into a widely used CAD modeler. Using a notification mechanism, the system records the precision functions called by the user in order to maintain geometric relations between points locations and source objects. These relations are stored in a directed graph which allows an automatic update of the model.

1. Introduction

The construction of constrained drawings is a widely discussed research topic, particularly in the field of architectural design. The related works aim to integrate into the drawing the intentions of the designer concerning layout and dimensioning, in order to facilitate further modifications.

Our objective is to develop a simple and efficient constraint-based modeling system, which helps produce and modify sketches during the design process, by migrating some of the designer intentions into the model. We have focused on two conditions that the system must satisfy: 1) the constraint setting must be intuitive without any extra manipulation from the designer, 2) the model modification must be efficient enough not to generate conflict situations that require the user intervention.

In order to be actually used in the design process, our system is integrated into an existing CAD modeler[1] which includes – like most modelers – drawing aids, also called snap modes, to acquire precise points relative to existing objects (midpoint, center, intersection, etc.). Our system records the user calls to these drawing aids to create geometric relations and to gradually build a relational graph that links the objects of the drawing.

[1] AutoCAD (Autodesk): http://www.autocad.com

Consequently, when modifications occur in the drawing, a top-down traversal of this graph allows the model update.

2. Related works and tools

Many research works have been carried out to provide modeling tools to create and manage models which can be modified interactively once they have been created. These contributions include very different approaches like parametric modeling, shape grammars or constrained-based modeling.

According to the review of these approaches made by J. Monedero (2000), our work can be classified under the heading of "history-based modeling" or "constructive parametric design". Unlike constraint-based systems, it does not use global solving algorithms nor evaluate if the model is underconstrained or overconstrained.

Rather than satisfying **constraints** on a set of parameters, our system maintains **relations** between points and source objects. The model also contains free points – at least one, the first created – which allow the user to introduce changes during the design activity.

Other research works have previously used similar point-based graphs (Martini, 1995) to develop prototypes such as the ReDRAW project (Kolarevic, 1997) or the CadLab Java applet (Medjdoub, 1999).

Besides, some commercial software like Cabri Géomètre[2] (dedicated to the interactive learning of geometry) and Solid Edge[3] modeler, have capabilities to maintain relations between objects by using a set of snap modes or geometric functions.

The history-based modeling system we present here is similar to these last tools in regard to the interactivity and functionalities even if it has major differences in some situations like object removal[4] or model modification[5]

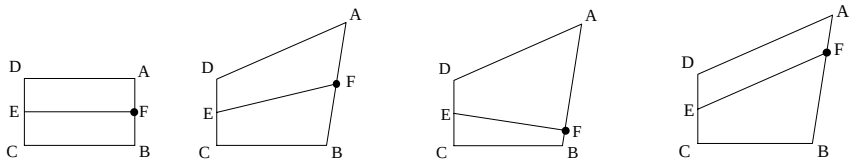
3. Implicit setting of geometric relations

Each time the user requests one of the available snap modes, the modeler computes a point position using one or two existing entities. This can be interpreted (if desired by the user) as a design intention and thus a new geometric relation is created without any manipulation from the user.

---

[2] Cabri Géomètre (Cabrilog) : http://www.cabri.com
[3] SolidEdge (EDS): http://www.solidedge.com
[4] In Cabri Géomètre, when an object is removed, all dependent objects are erased. In our system, objects are kept but only relations are removed.
[5] Solid Edge considers the geometric relations as bidirectional (that is the source object reacts to the target object). That sometimes generates unpredictable results.

So, during the construction process, the system builds, beside the modeler database, a custom graph of relations between point locations and graphic entities.
Consequently, when a point position can be the result of several construction methods, the user must choose the most relevant snap mode which matches its design intention and ensures the point will be updated in an adequate way. In a sense, the choices made by the user have more consequences than in a standard modeler.
In Figure 1, when point A is moved, point F is updated to different positions (b to d) according to the snap mode initially used (a).



a. Initial state b. F **midpoint** of AB c. EF **perpendicular** to AB d. EF **parallel** to DA

Figure 1. Different updates of a given point according to its construction method

4. Creation and management of a relational graph

Constrained points depend on source entities to which they are attached by "relation links". The points are also used in the geometric definition of target entities to which they are connected by "construction links". The constrained point is consequently the central object in the system. Its basic behavior is to react to the changes of its source entity, to re-compute its position accordingly and to transmit this new location to the target entity (Figure 2).

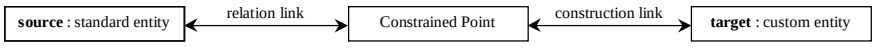


Figure 2. Basic dependency relation

The links are bi-directional. From left to right, they follow the creation and propagation order whereas the opposite direction figures the dependency.

4.1. DATA MODEL

The data model relies on two main object classes: the points and the entities. The class diagram below (Figure 3) shows inheritance links and reference links[6] which can be either “construction ” or “relation” links.

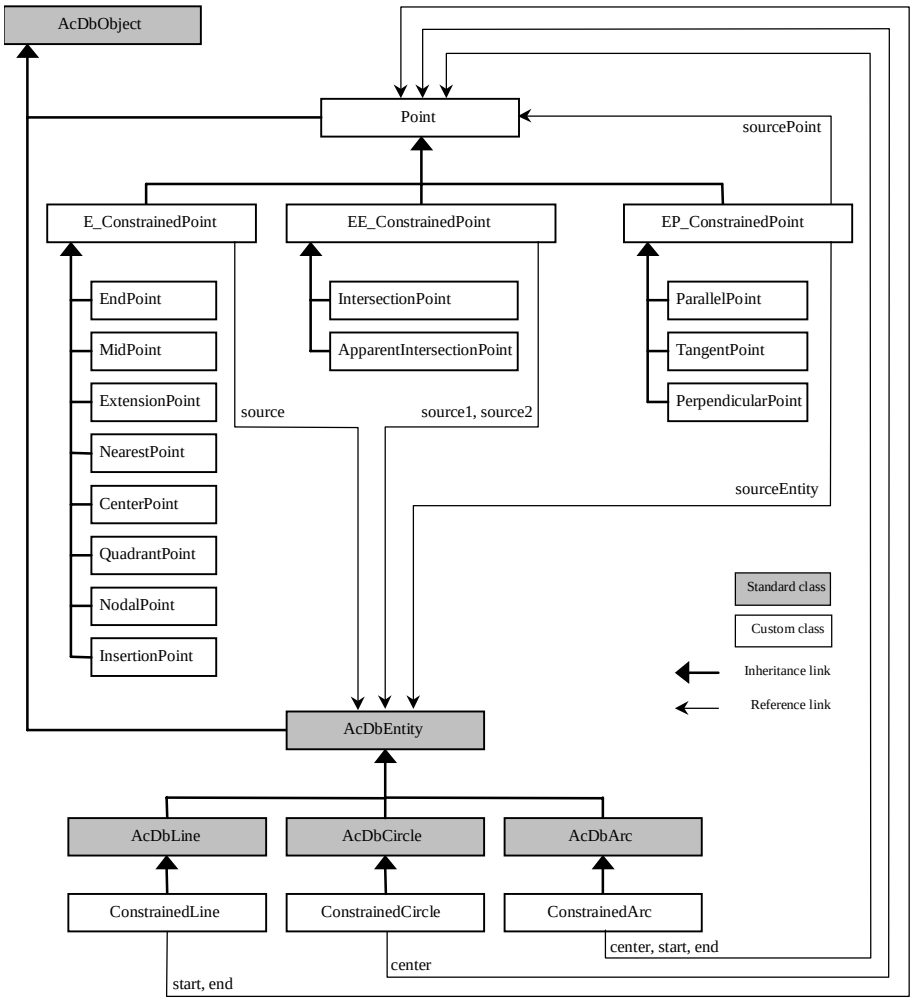


Figure 3. Model class diagram

[6] In fact, the reference links are bi-directional. For simplification purposes, we have only represented the dependency direction.

All theses classes derive from *AcDbObject* class so they are database-resident. Constrained entities need to derive from *AcDbEntity*. This is not the case of the Point class which has no graphic representation.

4.1.1. Points

The *Point* class is the base class for all free and constrained points. According to the type and number of its source objects, a constrained point derives from one of the following abstract classes:

– *E_ConstrainedPoint:* depends on a single entity (e.g. Center),
– *EE_ConstrainedPoint:* depends on two entities (e.g. Intersection),
– *EP_ConstrainedPoint:* depends on an entity and a point (e.g. Tangent).

These classes are then specialized into snap-mode-dependent point classes: *CenterPoint*, *IntersectionPoint*, *PerpendicularPoint*, etc.

4.1.2. Entities

Target entities are instances of custom classes (*ConstrainedLine*, *ConstrainedCircle*, etc.) which provide the functionality to react to point changes and update their own geometry. These classes are derived from built-in geometric classes (*AcDbLine*, *AcDbCircle*, etc.).

On the other hand, source entities are instances of standard classes so that any native entity can be the support of a geometric relation.

4.2. RELATIONAL GRAPH

Using the previous classes, the system builds a relational directed[7] graph (Figure 4). This graph is acyclic since it is based on the construction history. However, cycles could be introduced if the system allows an a posteriori relation-setting on existing points (which is not currently the case).

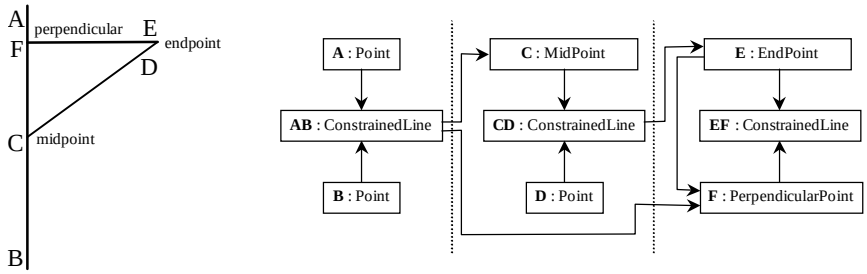


Figure 4. A drawing and its relational graph

[7] The direction of this graph follows the sense of creation and propagation.

4.3. MODIFICATION MANAGEMENT

When the end-user interactively requests an operation on a set of entities, the transformation only applies to the subset of free points which constitute these entities. From these points, a top-down traversal of the relational graph starts, following alternatively a construction link and a relation link.

This update process is more a data propagation than a solving procedure. It does not generate conflicts simply because each point is defined in a unique way, without redundancy.

5. Implementation

The first prototype of this system has been implemented using *Object ARX*, the *AutoCAD* object-oriented SDK (Autodesk, 1999). The following are some of the capabilities that were useful to this development.

5.1. CLASS SPECIALIZATION

When deriving a custom class from a standard one, most properties and methods concerning entity display, transformation and filing are inherited (unless overloaded) and new ones can be added.

For example, the class *ConstrainedLine* is a kind of *AcDbLine*. It has the same graphic characteristics but it also reacts to the changes that affect its start and end *Points*, checks if the modified point is free in which case it allows the operation and updates its own geometry.

5.2. EVENT NOTIFICATION

When an event occurs in the system, certain objects, called notifiers, automatically relay the event to other objects called reactors. This notification mechanism applies to various events related to a given object, to the database or to the graphic interface.

Database objects can be notifiers as well as reactors. In our case, a constrained point is the reactor of the entity it depends on (relation link) as well as the notifier of the entity it belongs to (construction link).

5.3. POINT MONITORING

This built-in notification mechanism is intended to relay the information relative to point acquisition. This work is achieved by a specific reactor called an “input point monitor”.

In this application, a custom input point monitor provides, for each point designation, the information required to add a relation: the snap mode used, the final computed point and the entities responsible for its location.

6. Discussion

The current prototype works with constrained lines only. We project to extend it to support other geometric types like arcs and circles. For some of these types, derived classes must take into account the construction method. For example, the abstract class *ConstrainedCircle* will have derived classes such as *CenterRadiusCircle*, *CenterPointCircle*, *ThreePointsCircle*, etc.

In addition, one of the objectives of this work is to deduct the geometric relations only from the interactive use of snap modes so the model is mainly based on the points. In a previous work (Alby, 2002), we have considered a data model which also relies on the angles and the distances, which values can be extracted from existing objects. The introduction of these new basic items allows to meet most of the modeling needs. However, their usage might be less intuitive since it is not directly supported by the modeler interface.

One other issue is the management of cycles in the relational graph. When dragging an existing point on an object snap point, the user makes this point depend on the object[8]. Therefore, a cycle can appear if this object was created after the point and indirectly depends on it. The propagation process can produce a result even with cycles but it requires to handle tolerance values. The cycles can also be detected beforehand and locally solved using numeric methods (Mathis, 1997) but we have not yet investigated this possibility.

References

Alby, E.: 2002, Assistance à la modification d'un dessin par mise en contrainte implicite, Master report, School of Architecture, Nancy.

Autodesk Inc.: 1999, ObjectARX developer's guide, online documentation.

Kolarevic, B.: 1997, Regulating Lines, Geometric Relations, and Shape Delineation in Design, *Proceedings of the 15th eCAADe Conference*, Vienna, 17-20 September 1997.

Martini, K.: 1995, Hierarchical geometric constraints for building design, *Computer-Aided Design*, 27(3), pp. 181-191.

Mathis P.: 1997, Constructions géométriques sous contraintes en modélisation à base topologique. PhD Thesis, Louis Pasteur University, Strasbourg, pp 5-34.

Medjdoub, B.: 1999, Interactive 2D Constraint-Based Geometric Construction System, *Proceedings of the 8th CAAD Futures Conference*, Atlanta, 7-8 June 1999, pp. 197-212.

[8] This function is not supported yet.

Monedero, J.: 2000, Parametric design: a review and some experiences, *Automation in Construction* 9(4), pp. 369-377.