CBAbench: An AutoCAD-based Dynamic Geometric Constraint System

Xiong Gong[*], Bo-xing Wang and Li-ping Chen

National CAD Support Software Engineering Research Center, Huazhong University of Science and Technology, Wuhan 430074, P.R.China

Abstract

In this paper, an integration framework of Geometric Constraint Solving Engine and AutoCAD is presented, and a dynamic geometric constraint system is introduced. According to inherent orientation features of geometric entities and various Object Snap results of AutoCAD, the proposed system can automatically construct an under-constrained geometric constraint model during interactive drawing. And then the directed constraint graph in a geometric constraint model is real-time modified in order to produce an optimal constraint solving sequence.

Due to the open object-oriented characteristics of AutoCAD, a set of user-defined entities including basic geometric elements and graphics constraint relations are defined through derivation. And the custom-made Object Reactor and Command Reactor are also constructed. Several powerful characteristics are achieved based on these user-defined entities and reactors, including synchronously processing geometric constraint information while saving and opening DWG files, visual constraint relations, and full adaptability to Undo/Redo operations. These characteristics of the proposed system can help the designers more easily manage geometric entities and constraint relations between them.

Key Words: Geometric constraint solving, Constraint relation decomposition, Geometric constraint graph, Object Snap, Constraint display

1. Introduction

The traditional CAD systems such as AutoCAD have few powerful capabilities in representing and processing constraint relations between geometric entities. Most systems are actually drafting-boards, and lack of efficient product design functions for the designers. The abilities to represent, solve and maintain geometric constraints have become a distinct characteristic to distinguish modern and traditional CAD systems [23]. Despite all that, these traditional CAD systems have been applied widely in industry community. Therefore, it is very important to extend constraint solving abilities to traditional CAD systems.

Geometric Constraint Solving (GCS) is one of the key technologies of parametric CAD systems, which enables the designers to make modifications to existing designs by changing parametric values. Many practical problems in engineering design fields can be regarded as Geometric Constraint

[*] Corresponding author:
Tel: +86-(27)-87463786
Fax: +86-(27)-87453986
E-mail: gongx@hustcad.com

Satisfaction Problem (GCSP), including parametric design and drawing, automatic assembly, planar or spatial mechanism analysis, layout problem and so on.

There are four major approaches to geometric constraint solving: the numerical solving approach [14, 19], the symbolic computing approach [6, 12], the graph-based analyzing approach [7, 8, 9, 15, 21], and the rule-based reasoning approach [1, 5, 10, 13, 16]. These methods have their own advantages and drawbacks about applied scope and solving speed, therefore, they are often combined to obtain the best results [2, 4, 11, 17, 18].

Nowadays, the 2D geometric constraint solving technology used in existing CAD systems has been developed perfectly, and applied successfully. However, in 3D geometric design fields, due to their own diversity and complexity, many problems about constraint solving have been not settled well. In fact, the underlying essence of geometric constraint problems in 2D and 3D fields are consistent, and the differences between them consist in that whether the constraint solvers are easy to be achieved. Moreover, for the multi-body system analysis problems in engineering design fields, they actually have the same geometric characteristics with those of geometric constraint systems.

Over the past years, most researches about geometric constraint satisfaction problems have been processed separately according to their own spaces and problem types. For instance, the solution to planar drafting is much different from that to spatial assembly. Essentially, all these applications are concerned with constraint matching, constraint sorting and constraint decomposing, which are regarded as common problems of constraint solving.

Based on above considerations, we have carried out the researches on general geometric constraint solving engine. The major goal is to construct an efficient and united geometric constraint solver, called Constraint Broadcasting Automation (CBA) [22]. It is hoped that CBA can be applied to matching, sorting and decomposing of 2D/3D constraints, and furthermore to mechanism kinematics and dynamics analysis. Moreover, CBA should have enough platform-independent compatibility and transplantability.

This work follows our previous researches on general geometric constraint solving engine. Using the existing open CAD system as a geometric engine, we try to study the integration framework of CBA and the geometric engine, and develop a new application system. In the existing parametric systems based on AutoCAD [2, 23, 24], following drawbacks should be attended.

1. Due to excessive dependence of geometric constraint systems on the given CAD platform, the transplantability of constraint solver was restricted.

2. Most systems often used separate data files to store geometric constraint information. As a result, the geometric constraint models would possibly be inconsistent with engineering drawings.

3. Various Object Snap results were simply discarded. Therefore, the descriptive geometric constraint models could not be automatically constructed during interactive drawing.

4. Invisible constraint information would possibly result in complex actions in managing and maintaining constraints.

5. Undo/Redo operations were not adapted well all along.

The thinking and relevant algorithms about geometric constraint solving will not be detailed in this paper. Instead, we present a new 2D computer-aided drafting system based on geometric constraint solving, called CBAbench. The proposed system demonstrates a seamless integration framework of independent geometric constraint solving engine and AutoCAD, and those problems will be solved well in this system.

The rest of this paper is organized as follows. In Section 2, we state the notations and conventions used in this paper, and give the software architecture of CBA. Next, in Section 3, the integration framework of the proposed system is presented. And, Section 4 detailed explains the implementation methods of major functions, followed by several illustrative examples in Section 5. Finally, we offer some conclusions and discuss future works in Section 6.

2. CBA overview

2.1 Decomposition of constraint relation

Generally, the geometric constraints in a plane fall into two groups: structural constraints and dimension constraints. The structural constraints can be implicit or explicit. The implicit structural constraints describe the inherent orientation features of geometric entities, such as Point-On-Line, horizontality and verticality. The explicit structural constraints describe the relative positional and connective relations between geometric elements, such as parallelism, perpendicularity and tangency. And, the dimension constraints are made through dimensioning, including distance and angle.

From the viewpoint of granularity, the geometric constraints can be also classified into two types. One is the macro-definition for end users, called Constraint Relation, and the other is the micro-definition used in internal solving, called Constraint Component. Each constraint relation is equivalent to one or more constraint components. For example, the constraint relation produced by a distance dimension between two parallel lines can be decomposed into two constraint components, that is, Line-Line parallelism and Line-Line distance.

The two important notions, that is, Degree of Freedom (DOF) and Degree of Constraint (DOC), are often used to quantify geometric entities and constraint relations between them. DOF is defined by the number of independent movement variables of geometric entities, denoted by $DOF(e)$, and e is a given entity. The DOF of points, straight lines of infinite length, circles of variable radius in a plane is equal to 2, 2, 3 respectively.

DOC is defined by the reduced number of the DOF of geometric entities due to geometric constraints placed on them, denoted by $DOC(c)$, and c is a given constraint relation. A constraint relation is actually equivalent to a group of independent algebraic constraint equations, which are quadratic in the coordinates of the geometries [21]. Due to various constraints placed on them, the movements of geometric entities are restricted, and their DOF reduces consequently [1].

According to above explanations, the DOC of a constraint relation should be equal to the number of relevant constraint components, and to the number of equivalent independent algebraic constraint equations. The constraint relations whose DOC is greater than 1 should be decomposed, therefore, the constraint solver can consistently maintain the decomposed constraint components whose DOC is always equal to 1. Table 1 shows several general constraint relations the DOC of which is greater than or equal to 1.

Table 1. Constraint relations and relevant constraint components

Constraint Relation	DOC	Constraint Component(s)
Point-fixed	2	X-coordinate-fixed Y-coordinate-fixed
Point-Point coincidence	2	X-coordinate-equal Y-coordinate-equal
Point-Line symmetry	2	Line-Line perpendicularity Point-Line equidistance
Line-Line coincidence	2	Line-Line parallelism Line-Line distance = 0
Distance dimension between two parallel lines	2	Line-Line parallelism Line-Line distance
Line-Line perpendicularity	1	Perpendicularity between two lines

It can be seen from the following segments that the decomposition of constraint relations is helpful for representation and solving of geometric constraint models. And the persistent storage of constraint relations is more easily achieved.

From now on, we will not distinguish constraint relations from constraint components, and call them constraints instead, provided that no mistake arises.

2.2 Representation of geometric constraint model

In this work, Geometric Constraint Graph (GCG) is introduced to describe geometric constraint models. In a directed constraint graph, $GCG = G(V, A)$, V is a set of vertices denoted by $\{v\}$, where v is denoted by a geometric element; A is a set of directed arcs denoted by $\{a\}$, where a is denoted by a constraint component between two geometric elements.

The directed arc a is made with the following steps. Let constraint c be matched with vertex v; v_h and v_t represent the head and tail of a directed arc respectively. If c is matched with v_h, a is created by using v_t to represent its tail vertex and v_h to denote its head vertex. The other definitions concerning the structural properties of the constraint graph are provided in Ref. [17].

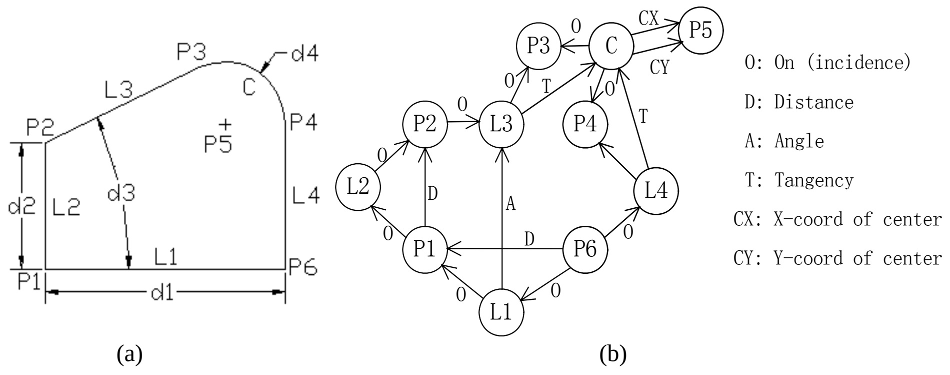


Fig. 1. A geometric constraint model and its optimized GCG. (a) a geometric constraint model, and (b) the optimized geometric constraint directed graph.

The work in Ref. [3] provides more detailed algorithms about how to construct, match and optimize a directed constraint graph. As an example, the GCG in Fig.1(b) is constructed through recognizing the constraint relations in the geometric drawing in Fig.1(a), and it has been dynamically optimized in order to decrease the constraint solving scale (explained below). As shown in Fig.1(b), the GCG clearly represents geometric entities and constraint relations between them in the geometric constraint model.

Practically, the vertex v can be classified into two types: basic vertex denoted by $v_b(e)$ and compound vertex denoted by $v_s(G_s)$. Here, v_b encapsulates basic geometric elements, such as points, straight lines and circles; v_s encapsulates a directed sub-graph G_s, which represents a condensed strong component of the directed graph G. The analogous methods about how to condense strong components to compound vertices in a directed graph are reported in Ref. [8].

A directed constraint graph G has the following characteristics. First, the directed arc a represents one and only constraint c, where c is matched with the head vertex of a. Second, the number of the constraints matched with $v(e)$, which is equal to the in-degrees of v, should be less than or equal to the DOF of e.

Thus it can be seen, the key point to construct a geometric constraint graph is to search a vertex v to match the new constraint c. If no appropriate v is found, c is redundant. The analogous conclusion can be also seen in Ref. [3].

2.3 Dynamic solving planning

In a directed constraint graph, the arc's direction indicates the dependence of its head vertex on its tail vertex. Then, the vertices whose out-degrees are equal to zero are independent on any other vertices. As shown in Fig.1(b), P6 is independent on P1, L1 and L4. The condensed directed acyclic graph indicates a partial-order between the set of vertices, and a global-order solving sequence can be produced with Topological Sort method.

For example, the directed acyclic graph in Fig.1(b) has no strong components. Therefore, the solving sequence, {P6, L1, L4, P1, L2, P2, L3, C, P3, P4, P5}, can be made by using Topological Sort.

Since the global solving is too slow and consumes too much space to satisfy real-time interactive operations, certain dynamic planning strategies must be taken to decrease the constraint solving scale. As a matter of fact, the arc's direction in a directed graph implies the broadcasting scope of the variations caused by the constraint's change, called Constraint Broadcasting Scope (CBS) [3]. In a directed acyclic graph, the CBS for vertex v_0 and the constraint matched with v_0 is defined by a set of vertices denoted by $\{v\}$, which can be produced through forward Depth-First-Search from v_0.

As shown in Fig.1(b), if the radius dimension d4 matched with the arc C is modified, only the vertices in {P3, P4, P5}, which are equivalent with the CBS of vertex C and d4, must be re-solved.

2.4 Software architecture of CBA

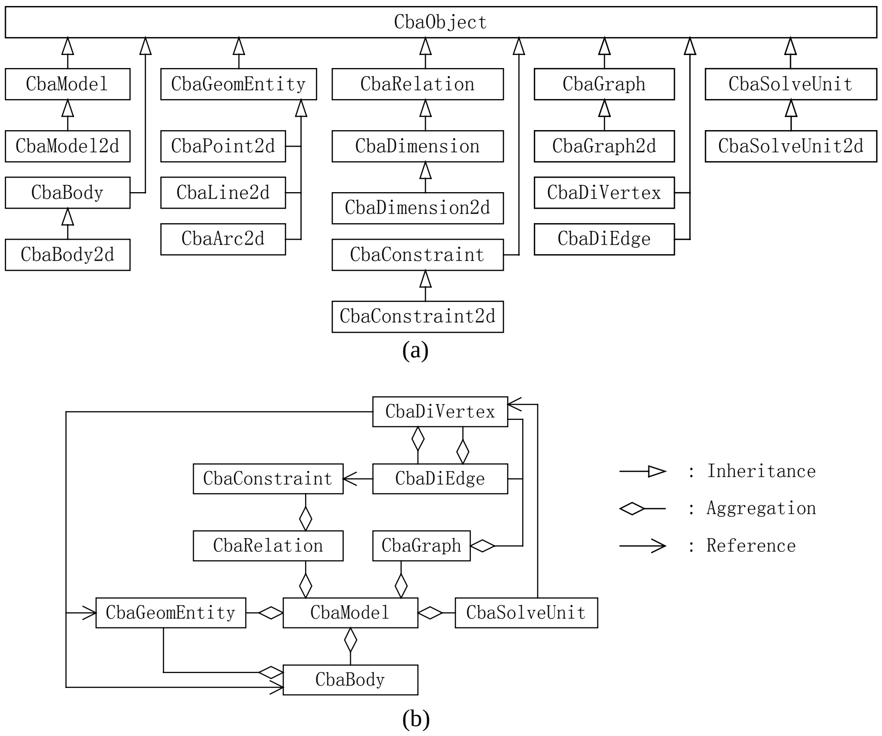


Fig. 2. Software structure of CBA. (a) object hiberarchy of CBA, and (b) aggregation and reference relations between CBA objects.

Based on above explanations, we give the software structure of CBA in the object-oriented form, as shown in Fig.2. The object hierarchy of CBA presents most classes in the geometric constraint solver except those used in 3D constraint solving, which are beyond the scope of this work and have been ignored. The object inheritance relations are shown in Fig.2(a), and expressed with hollow arrows. Next, the aggregation and reference relations are shown in Fig.2(b) with rhombic nodes and arrows respectively. The brief descriptions for CBA objects are given as follows.

As the base-class of all the other objects, CbaObject provides necessary common interfaces and services, including object identification and debugging.

CbaModel encapsulates all objects of a geometric constraint system, including a set of virtual bodies denoted by {CbaBody}, a set of geometric entities denoted by {CbaGeomEntity}, a set of constrain relations denoted by {CbaRelation}, a directed constraint graph denoted by CbaGraph related to the geometric constraint model, and a dynamic solving sequence denoted by {CbaSolveUnit}.

Since the geometric constraint solving engine does not directly manage actual geometric elements, CbaBody in this work is defined as Virtual Body. The virtual body contains the DOF of a given geometric body and a set of basic geometric elements, where the basic geometric elements are attached to the virtual body and have various constraint relations with other entities. The virtual body links to the actual geometric elements through their unique handles from CAD systems.

Correspondingly, CbaGeomEntity encapsulates the common properties of all types of basic geometric elements. And, the basic geometric elements are constructed through derivation from it, such as points denoted by CbaPoint2d, straight lines denoted by CbaLine2d and circular arcs denoted by CbaArc2d.

CbaConstraint is used to describe a constraint component. As the minimal solving unit, it is associated with a unique algebraic constraint equation. And, CbaRelation encapsulates a group of (mostly, only one) constraint components, which can be seen through the user interface.

CbaDiGraph represents a directed constraint graph. It provides the operations of constructing, searching and traversing for directed graphs, and searching and condensing for strong components. The vertex and directed arc in a constraint graph are described with CbaDiVertex and CbaDiEdge respectively.

Finally, CbaSolveUnit is used to explain one step in a solving sequence, which is actually bound with a basic or compound vertex in a directed constraint graph.

CBA can be easily used to represent geometric constraint models, and many crucial algorithms for constructing, matching and decomposing constraint directed graphs are also developed for practical applications [3]. Among these classes, CbaBody is one of the most important objects for 2D/3D united constraint solver for its properties vary from different spaces and problems [22]. Furthermore, other classes and their sub-classes including CbaModel, CbaGraph, CbaConstraint and CbaSolveUnit will be used to deal with various problems in different fields.

3. Integration framework in CBAbench

3.1 Open characteristics of AutoCAD

Before introducing the integration framework of CBAbench, we begin to explain the object-oriented characteristics of AutoCAD. First, due to its open application architecture, AutoCAD can be regarded as a general geometric kernel. Then, a third-party application can be developed based on this system or transplanted to this platform, and seamlessly integrated with it.

Second, the object hierarchy of AutoCAD can be further extended. Besides providing various objects and functions in the object-oriented form, AutoCAD permits new objects to be constructed through derivation from existed classes, and provides relevant methods to manage them.

Third, AutoCAD provides powerful message mechanism. It reports various internal events (such as creations or deletions of entities, calling commands) to external applications. And the latter can process them in time.

Last, the graphics database of AutoCAD is open. Like existing internal objects, the new objects of external applications constructed through derivation can be saved into the graphics database, and managed and maintained by AutoCAD consistently.

3.2 Integration framework of CBA and AutoCAD

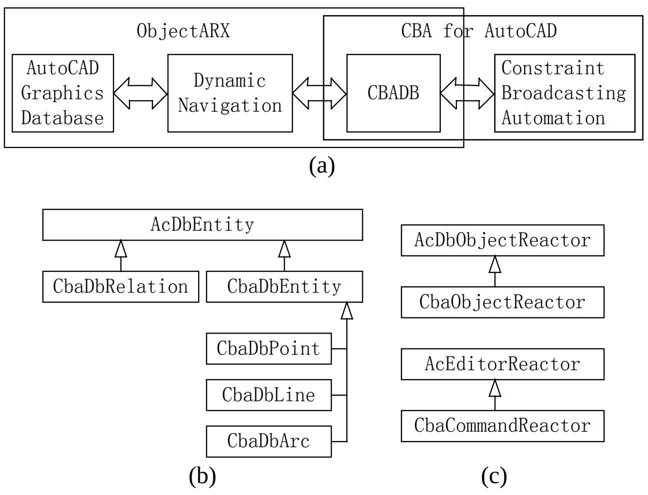


Fig. 3. Intergration framework in CBAbench. (a) integration framework of CBA and AutoCAD, (b) CBADB: run-time objects, and (c) object reactor and command reactor.

Practically, CBA is an independent calculation model, and CBA object information is temporary and cannot be persistently saved. The correlations between the object instances of CbaBody, CbaGeomEntity, CbaRelation and the platform-dependent entities can be built only by using unique handles, which helps to construct an integration framework of CBA and AutoCAD. In this work, a seamless integration framework shown in Fig.3 has been accomplished based on ObjectARX [20].

Among these sub-systems, Dynamic Navigation in Fig.3(a) is a set of interactive tools used to draw lines and dimension, which uses the Object Snap functions of AutoCAD to draw exact lines and extract accurate constraint relations between them. Moreover, in this framework, Dynamic Navigation is used as a bridge to communicate the geometric constrain solving engine with AutoCAD.

In this framework, CbaDbEntity and its sub-classes in Fig.3(b), including CbaDbPoint, CbaDbLine and CbaDbArc, are defined as actual geometric elements, and used in Dynamic Navigation to create geometric entities during interactive drawing.

Besides being the storage carrier of geometric constrains in AutoCAD, CbaDbRelation is used to really display the types and states of constraints. And that, the function of synchronously processing constraint information while saving and opening DWG files is also achieved based on it.

The two reactors, that is, CbaObjectReactor and CbaCommandReator in Fig.3(c), are defined to implement Object Reactor and Command Reactor respectively. And, these reactors are combined to achieve the full adaptability to Undo/Redo operations. This will help the system automatically update geometric constraint models.

In a geometric constraint graph, the vertices and directed arcs are associated with different types of graphics entities. In this work, those actual geometric elements, defined by CbaDbPoint, CbaDbLine and CbaDbArc, are associated with the basic vertices, and the constraint entities defined by CbaDbRelation are related to the directed arcs.

Through unique handles, called AcDbHandle in ObjectARX [20], the correlations between the temporary run-time objects in CBA (such as CbaGeomEntity and CbaRelation) and the persistent user-defined entities in AutoCAD (such as CbaDbEntity and CbaDbRelation) can be created conveniently.

4. System implementation

4.1 Construction of geometric constraint model

As mentioned above, Dynamic Navigation has two major sets of functions, that is, drawing lines and dimensioning. And it plays an important role in automatically constructing geometric constraint models during interactive drawing.

The functions of drawing lines in Dynamic Navigation are similar to those existing commands in AutoCAD. During drawing lines, Dynamic Navigation completes two major works (explained below) and outputs necessary results to the constraint system in an appropriate form.

The first is to create actual geometric elements, such as points, straight lines and circular arcs, which are the instances of CbaDbPoint, CbaDbLine and CbaDbArc respectively.

The second is to collect snap types and target entities of Object Snap during interactive operating. The typical snap types include end-point, perpendicular-point and tangent-point and so on.

Subsequently, the relevant constraints are extracted from the Object Snap results, and then exported into the geometric constraint system. For example, the Point-On-Line constraint is obtained from the end-point snap. The same is true for the perpendicularity constraint from the perpendicular-point snap, and the tangency constraint from the tangent-point snap.

The dimensioning functions in Dynamic Navigation have some inferential characteristics. According to the selected entities and mouse's movement, the dimension types can be automatically determined, such as Point-Point distance, Line-Line angle, Radius and Diameter. Subsequently, the dimensions and their associated entities are converted into relative constraints, which are submitted into the geometric constraint system finally.

Through analyzing the results submitted by Dynamic Navigation, the system obtains geometric entities and various types of constraints, including implicit structural constraints, explicit structural constraints and dimension constraints. Then, an under-constrained geometric constraint model is automatically constructed by using the methods in Section 2.2. Finally, the GCG is optimized to meet the requirement of decreasing the constraint solving scale.

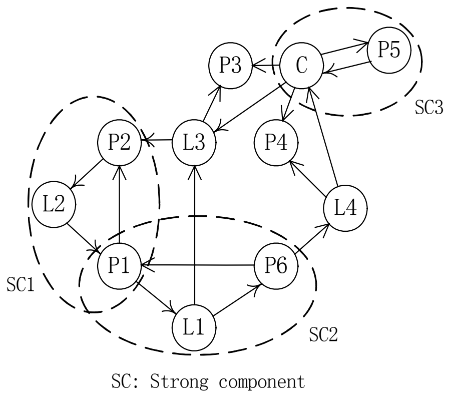


Fig. 4. Initial GCG of the geometric constraint model in Fig.1(a).

As an example, the initial GCG of the geometric constraint mode in Fig.1(a) is shown in Fig.4, and the optimized GCG is shown in Fig.1(b). The differences between the two figures present the GCG's transition in various states.

Apparently, more than one strong component exists in the GCG shown in Fig.4. Therefore, a global solving sequence cannot be obtained only by using simple Topological Sort method. Furthermore, for the modifications of some constraints, the local solving sequence cannot be made only by using the CBS method in Section 2.3.

As a coupled problem, the strong component in a GCG can be solved only through the numerical computing methods, such as Newton-Raphon's iteration or its variants [22].

Comparatively, the GCG in Fig.1(b) is more applicable to produce a serial solving sequence, which can be orderly solved with algebraic analysis methods [21].

4.2 Storage and restoration of constraint information

AutoCAD permits that the user-defined objects derived from AcDbObject can be persistently saved into the graphics database of AutoCAD through overriding the following virtual functions, including dwgInFields(), dwgOutFields(), dxfInFields() and dxfOutFields() [20]. And, as the AcDbObject is the base-class of AcDbEntity, therefore, above rules can be also applied to CbaDbRelaton.

Since the user-defined objects are accessed as a part of DWG files, the object instances of CbaDbRelaton can be the storage carrier of constraint relations in AutoCAD to preserve the consistency of constraint information with geometric drawing.

On the basis of the fact that the decomposition mode from constraint relations to constraint components is invariable in CBA, certain simplified measures can be taken to decrease the storage scale of constraint information in DWG files. For example, when a DWG file is saved, the whole GCG is ignored because it can be re-built from geometric entities and correlative constraint relations between them. The storage methods used in this work are given as follows.

First, all constraint relations in current geometric constraint model, including their types and geometric elements associated with them, are saved into DWG files.

Second, the states of each constraint component decomposed from the given constraint relations, for example, whether it is redundant or not, are also stored.

In this way, when a DWG file is saved in AutoCAD, all of necessary data about constraints are stored in the graphics database simultaneously. While the DWG file is opened, all the data about constraint relations and constraint components are extracted, and then a geometric constraint model is re-built as done during interactive drawing. If needed, the geometric constraint graph is optimized automatically. Subsequently, new design starts.

4.3 Display and deletion of constraints

AutoCAD permits that the user-defined graphics entities derived from AcDbEntity can be displayed according to given styles by overriding worldDraw() and other virtual functions [20]. And, by calling Erase command, the designers can directly select the displayed entities and delete them in AutoCAD’s graphics window.

Since the object instances of CbaDbRelation resident in the graphics database of AutoCAD can be displayed, the designers can learn the types and states of the constraint relations in current geometric model in AutoCAD’s graphics window, and directly select and delete them. This will help the designers maintain the constraint system conveniently.

In CBAbench, the normal constraints are displayed in green color, and the redundant constraints are displayed in red color. In Fig.5, the types and states of the normal constraints in the drawing of Fig.1(a) are displayed in green color, including one horizontality constraint, two verticality constraints, two Line-Circle tangency constraints. But, the L1-L2 perpendicularity constraint is displayed in red color because it is apparently redundant. Then the designers can easily learn the state of a given constraint according to its color.

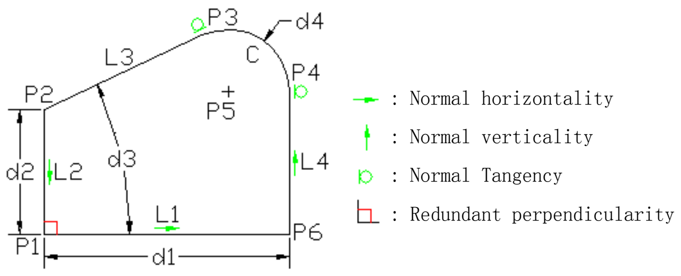


Fig. 5. Real display of constraints.

When a graphics constraint entity is deleted, the constraint engine can be notified due to Object Reactor (explained below), and some actions are automatically taken to deal with the delete event. On this condition, we establish the Truth Maintenance Scheme (TMS), which can be used to automatically maintain the remainder set of constraints placed on geometric entities, where the geometric entities are associated with the deleted constraint. The truth maintenance scheme is briefly explained as follows.

When a constraint entity is deleted, the system automatically checks the geometric element e matched with the deleted constraint. Assume that a redundant constraint c_r exists in the remainder set of constrains matched with e. If the number of the existing normal constraints matched with e is less than the degrees of freedom of e, the state of c_r can be turned normal, that is, c_r is activated.

As shown in Fig.5, if the verticality constraint placed on L2 is removed, the L1-L2 Perpendicularity constraint matched with L2 can be turned normal, and can be activated immediately.

4.4 Adaptability to Undo/Redo

In AutoCAD, AcDbObjectReactor and AcEditorReactor are used to create object reactors and command reactors for special applications respectively. The applications can make use of both the reactors to capture the user's intents, and automatically make responses.

As shown in Fig.3, the user-defined object reactors, called CbaObjectReactor, is defined through derivation from AcDbObjectReactor. The implementation of CbaObjectReactor needs overriding the following callback functions, such as erased(), modified(), modifyUndone(), reappended() and unappended(). After the object reactor is attached to the objects resident in the graphics database of AutoCAD, all actions operated on the given objects can be captured, and relevant notifications are sent automatically.

And, the command reactor, called CbaCommandReactor, is defined through derivation from AcEditorReactor. The implementation of CbaCommandReactor needs overriding the following callback functions, such as commandWillStart(), commandEnded(), commandCancelled() and commandFailed(). After the command reactor is registered in AutoCAD, one or more of the user's commands can be captured.

CBAbench combines the object reactor and command reactor to achieve the full adaptability to

Undo/Redo operations. The approximate steps are described as follows.

First, the object reactor captures the actions operated on geometric entities, and records operation types and target entities.

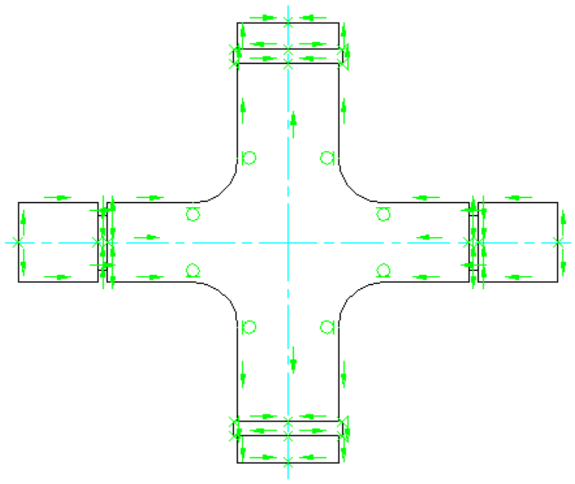
Second, the command reactor captures the command callings of Undo or Redo. When the command comes to an end, the geometric constraint model is automatically updated according to previous operating information.

Actually, while running Undo or Redo command, the objects of CBADB and constraint relations resident in the graphics database of AutoCAD might be modified due to the previous actions operated on them. Once the relevant events happen, the object reactors capture them, and record necessary information. When the command comes to an end, the system will automatically update the geometric constraint model. As an example, the system's response to an Undo command is given briefly as follows.

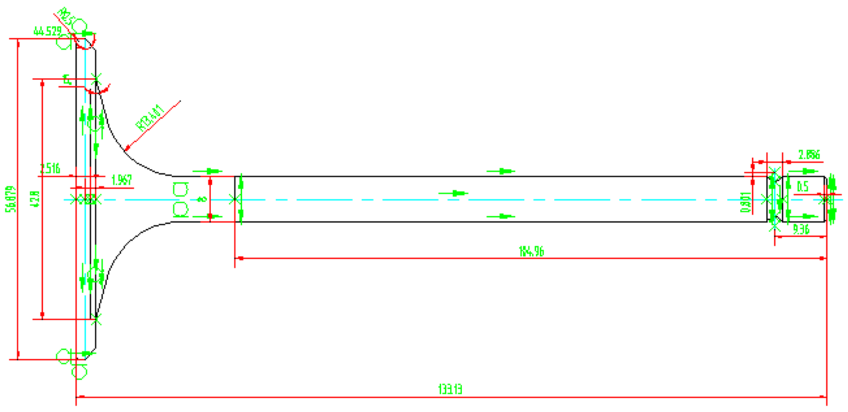
While running the Undo command, the new-created geometric elements and constraint entities are deleted from AutoCAD. When the command comes to an end, the relevant vertices and directed arcs are automatically removed from the GCG with necessary optimizations. These delete events are detected by the object reactors, and the operation types and target entities are detailed recorded. If a Redo immediately follows the Undo command, the effects of Undo will be reversed.

5. Illustrative examples

CBAbench introduced in this paper meets well the requirements of interactive designs, and can be applied to the routine product designs supported by geometric constraints. Two industrial models made with CBAbench are shown below.



(a)



(b)

Fig. 6. Industrial models made with CBAbench. (a) drafting of a cross gimbal, and (b) engineering drawing of a valve in the motor-engine.

In Fig.6, Fig.6(a) shows the drafting of a cross gimbal without dimension constrains, and Fig.6(b) shows the engineering drawing of a valve in the motor-engine with a complete geometric constraint model.

6. Conclusions and future works

It can be seen from this work, as a general geometric constraint solving engine, CBA is independent on special CAD systems and geometric modeling kernels. As shown in Fig.3, when CBAbench is transplanted to other systems with necessary capabilities summarized in Section 3.1, only the run-time classes of CBA, i.e. CBADB, need be re-constructed with minor modifications made to the other modules. Similar works have been carried out based on ACIS.

CBAbench introduced in this paper meets well the requirements of interactive designs, and can be applied to the routine product designs supported by geometric constraints. The advantages of the proposed system are given as follows.

1. Dynamic Navigation connects the run-time classes of CBA with AutoCAD efficiently. Based on it, the system succeeds in automatically constructing geometric constraint models in the course of interactive drawing. In fact, the geometric entities with more complex shapes, i.e. formed features, such as holes and key-slots, can be also defined through derivation. And their characteristic points for Object Snap can be defined and obtained by Dynamic Navigation. Moreover, their DOF and sub-entities can be consistently maintained with the Virtual Body. These aspects will help to extend this system to incorporate other applications beyond mechanical scope.

2. The constraint information is persistently stored as a part of DWG files. CBAbench preserves the consistency of constraint models with geometric drawings. And, the types and states of constraints in a geometric constraint model can be shown in a visual form.

3. The user-defined geometric and constraint entities can be managed in a manner analogous to

that for other types of existing entities in AutoCAD. And, the full adaptability to Undo/Redo operations gives a good help to end users in interactive designs.

In actual design process, the engineering constraints expressed with formulas or equations are often used to satisfy the given functional, manufacturable and structural design requirements. If the interpretive design languages, such as InteBasic [24] or VBScript, are introduced to convert complex engineering constraints into parametric geometric constraints, the proposed system can become more flexible. As one of leading CAD systems, AutoCAD has been widely applied in various industrial fields. The secondary developmental modules with parametric capabilities will improve the designers' work efficiency further.

7. Acknowledgements

This work was supported by the National High Technology Research and Development Program of China (863 Project, #2003AA411044) and the National Natural Science Foundation of China (NSFC, #60503069). The authors are grateful to all the CAD staff at the National CAD Support Software Engineering Research Center, Huazhong University of Science and Technology.

References

1. Aldefeld B. (1988), Variation of geometries based on a geometric reasoning method, *Computer-Aided Design*, 20(3), 117-126.
2. Chen L.P., Tu Z.B., Luo H. and Zhou J. (1996), A new method about constraint management of parametric drawing, *Journal of Software*, 7(7), 394-399.
3. Chen L.P., Wang B.X., Peng X.B. and Zhou J. (2000), An optimal method of bipartite graph matching for under-constrained geometry solving, *Journal of Computers*, 23(5), 523-530.
4. Durand C. and Hoffmann C.M. (2000), A systematic framework for solving geometric constraints analytically, *Journal of Symbolic Computation*, 30(5), 493-529.
5. Gao X.S. and Chou S.C. (1998), Solving geometric constraint systems I: A global propagation approach, *Computer-Aided Design*, 30(1), 47-54.
6. Gao X.S. and Chou S.C. (1998), Solving geometric constraint systems II: A symbolic approach and decision of re-constructibility, *Computer-Aided Design*, 30(2), 115-122.
7. He W., Tang M., Dong J.X. and He Z.J. (2003), A constraint solving approach based on graph decomposition, *Journal of Image and Graphics*, 8(8), 926-931.
8. Hoffmann C.M., Lomonosov A. and Sitharam M. (1997), *Finding solvable subsets of constraint graphs*, Berlin: Springer, 463-477.

9. Hoffmann C.M. and Vermeer P.J. (1995), Geometric constraint solving in R2 and R3. In: Du D.Z. and Huang F., eds. *Computing in Euclidean Geometric*, Singapore: World Scientific Publishing, 170-195.

10. Joan-Arinyo R. and Soto A. (1997), A correct rule-based geometric constraint solver, *Computers & Graphics*, 21(5), 599-609.

11. Joan-Arinyo R. and Soto A. (1999), Combining constructive and equational geometric constraint-solving techniques, *ACM Transactions on Graphics*, 18(1), 35-55.

12. Kondo K. (1992), Algebraic method for manipulation of dimensional relationships in geometric models, *Computer-Aided Design*, 24(3), 141-147.

13. Kramer G.A. (1992), Solving geometric constraints systems: a case study in kinematics, Cambridge MA: MIT Press.

14. Lamure H. and Michelucci D. (1996), Solving geometric constraints by homotopy, *IEEE Transactions on Visualization Computer Graphics*, 2(1), 28-34.

15. Latham R.S. and Middleditch A.E. (1996), Connectivity analysis: a tool for processing geometric constraints, *Computer-Aided Design*, 28(11), 917-928.

16. Lee J.Y. and Kim K. (1998), Geometric reasoning for knowledge-based design using graph representation, *Computer-Aided Design*, 28(10), 831-841.

17. Lee K.Y., Kwon H., Lee J.Y. and Kim T.W. (2003), A hybrid approach to geometric constraint solving with graph analysis and reduction, *Advances in Engineering Software*, 34(2), 103-113.

18. Li Y.T., Liu S.X., Hu S.M. and Sun J.G. (2002), Geometric constraint solving techniques based on symbolic algebra and graphical reduction, *Journal of Tsinghua University* (Sci. & Tech.), 42(10), 1410-1413.

19. Lin V.C., Gossard D.C. and Light R.A. (1981), Variational geometry in computer-aided design, *Computer & Graphics*, 15(3), 171-177.

20. ObjectARX 2002 (July 10, 2002), http://www.autodesk.com/objectarx.

21. Owen J. (1991), Algebraic solution for geometric from dimensional constraints, *ACM symposium*, New York: ACM Press. 397-407.

22. Peng X.B. (2002), Study of principles and methods of 2D/3D united geometric constraint solver, PhD dissertation, Wuhan: Huazhong University of Science and Technology.

23. Wang, B.X., Chen L.P. and Zhou J. (1998), Study and practice of key technologies of geometric constraint driven function in traditional drafting systems, *Computer Research & Development*, 35(10), 935-940.

24. Zhou J.L., Chen L.P., Wang B.X. and Zhou J. (1999), The total solution for parametric engineering drawing, *Computer Engineering and Design*, 20(3), 39-41.