Mississippi State University
Scholars Junction

Theses and Dissertations Theses and Dissertations

8-3-2002

G+: A Constraint-Based System for Geometric Modeling

Joseph Britto Lawrence

Follow this and additional works at: https://scholarsjunction.msstate.edu/td

Recommended Citation

Lawrence, Joseph Britto, "G+: A Constraint-Based System for Geometric Modeling" (2002). *Theses and Dissertations*. 2368.
https://scholarsjunction.msstate.edu/td/2368

This Graduate Thesis - Open Access is brought to you for free and open access by the Theses and Dissertations at Scholars Junction. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholars Junction. For more information, please contact scholcomm@msstate.libanswers.com.

G+: A CONSTRAINT-BASED SYSTEM FOR GEOMETRIC MODELING

By

Joseph Britto Lawrence

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computational Engineering
in the College of Engineering

Mississippi State, Mississippi

August 2002

G+: A CONSTRAINT-BASED SYSTEM FOR GEOMETRIC MODELING

By

Joseph Britto Lawrence

Approved:

Michael L. Stokes
Associate Research Professor of
Computational Engineering
(Major Professor)

Edward A. Luke
Assistant Professor of
Computer Science
(Committee Member)

Seth F. Oppenheimer
Associate Professor of
Mathematics and Statistics
(Committee Member)

Boyd Gatlin
Associate Professor of
Aerospace Engineering
(Graduate Coordinator)

A. Wayne Bennett
Dean of the College of Engineering

Name: Joseph Britto Lawrence

Date of Degree: August 3, 2002

Institution: Mississippi State University

Major Field: Computational Engineering

Major Professor: Dr. Michael L. Stokes

Title of Study: G+: A CONSTRAINT-BASED SYSTEM FOR GEOMETRIC MODELING

Pages in Study: 56

Candidate for Degree of Master of Science

Most commercial CAD systems do not offer sufficient support for the design activity. The reason is that they cannot understand the functional requirements of the design product. The user is responsible for maintaining the functional requirements in different design phases. By incorporating constraint programming concepts, these CAD systems would evolve into systems which would maintain the functional requirements in the design process, and perform analysis and simulation of geometric models. The CAD systems incorporated with constraint programming concepts would reduce design time, avoid human fatigue and error, and also maintain consistency of the geometric constraints imposed on the model. The G+ system addresses these issues by introducing a constraint-based system for geometric modeling by object-oriented methods. The G+ is designed such that available specialized algorithms can be utilized to enable handling of non-linear problems by both iterative and non-iterative schemes.

ACKNOWLEDGMENTS

I would like to express my gratitude to my major professor and mentor, Dr. Michael Stokes, for his guidance, encouragement and also his continuing patience. With his interest in human-computer interaction and computer technology, Dr. Stokes has continously provided me with motivations and helped to put my work in a wider perspective.

I would also like to thank my commitee member, Dr. Edward Luke for his guidance, particularly in the earlier stages of my research work on constraint and logic programming systems. Dr. Luke provided many valuable comments on the earlier stages of my research work. I would also like to thank my commitee member, Dr. Seth Oppenheimer for his valuable comments and suggestions on the numerical methods used in my research work. Dr. Oppenheimer urged me on by way of his untiring support and seemingly unlimited belief in me throughout my research.

I would also like to thank Dr. Bharat Soni, Director, Center for Computational Systems for providing financial support to this research effort. I would also like to thank Dr. Boyd Gatlin and the Engineering Research Center for providing facilities for the present work.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENT ii

LIST OF TABLES v

LIST OF FIGURES vi

NOMENCLATURE viii

CHAPTER

I. INTRODUCTION 1

II. RELATED WORK 5
2.1 Constraint Representation 5
2.2 Geometric Constraint Solving 6
2.2.1 Algebraic Constraint Solvers 6
2.2.2 Constructive Constraint Solvers 6
2.2.2.1 Rule-Based Constructive Constraint Solver 7
2.2.2.2 Graph-Based Constructive Constraint Solver 7
2.2.3 Propagation Method Based Constraint Solvers 8
2.3 Constraint-Based Systems 8

III. GEOMETRIC CONSTRAINT SPECIFICATION AND SATISFACTION MECHANISM 10
3.1 Representation of Geometry and Functions 10
3.1.1 Representation of Geometric Functions 11
3.2 Constraints and Graph Structure 11
3.3 Geometric Constraint Solving 14
3.3.1 Constraint Satisfaction 14
3.3.2 Consistency Techniques 14
3.3.2.1 Arc-consistency 15
3.3.2.2 Path-Consistency 15
3.3.3 Constraint Propagation 15
3.3.4 Over Constrained Models 16
3.4 Object-Oriented Constraint Model 16
3.4.1 Consistency maintenance in Object-Oriented Systems 18
3.4.2 Dynamically asserted Constraints 18
3.5 Constraint Model to Capture Design Intent 18

CHAPTER Page

IV. THE STRUCTURE AND IMPLEMENTATION OF G+ 21
4.1 The G+ System Architecture 21
4.1.1 G+ Graph Component 22
4.1.2 AutoCAD Runtime Extension 25
4.2 The G+ Constraint Classes 28
4.2.1 Pre-defined Constraint Class 28
4.2.2 Generic Constraint Class 30
4.2.3 Solution for Nonlinear Constraint Systems 31
4.3 Control Flow of G+ 32
4.4 G+ Application Example 33

V. RESULTS 41
5.1 Issues 41
5.1.1 Multiple Solutions 42
5.1.2 Framing Generic Constraints 43
5.1.3 Computation Time for Non-linear Problems 44
5.2 Constraint Consistency Validation 44

VI. CONCLUSION 51
6.1 Future Work 52
6.1.1 Constraint-Based Computational Environment 52
6.1.2 Multi-User Environment 52
6.1.3 Expert System 53

REFERENCES 54

LIST OF TABLES

TABLE Page

4.1 Application Message Code 26

4.2 Pre-defined Constraints 29

LIST OF FIGURES

FIGURE Page

1.1 Geometric constraints in floor plan design 1
1.2 Geometric constraints in aircraft wing design 2
1.3 Geometric constraints in simulation of robotic arm 2
3.1 Graph with constraint representation 12
3.2 A cyclic graph 13
3.3 Local Inconsistencies 15
3.4 Constraint arc between *line_A* and *line_B* 18
4.1 The G+ system architecture 22
4.2 The G+ graph component 23
4.3 AutoCAD's partial database 27
4.4 Entity access to constraint solver 28
4.5 G+ control flow 33
4.6 Constraint graph for floor plan example 36
4.7 Floor plan 36
4.8 Floor plan: Moving the wall 37
4.9 Floor plan: Window A overlapped by room A 37
4.10 Floor plan: Movement of window A 38
4.11 Floor plan: Movement of window B 38
4.12 Floor plan: Movement of window C 39
4.13 Floor plan: Width adjustment of windows B and C 39
4.14 Constraint consistent floor plan 40
5.1 Multiple solutions along an ellipse 42

FIGURE Page

5.2 Solution by minimization 43

5.3 Graph representation of three lines model 46

5.4 Geometric constraint model of three lines 46

5.5 Three lines: *lineLength* constraint 47

5.6 Three lines: *lineLength* and *lineParallel* constraints 47

5.7 Geometric model 48

5.8 Constraint graph of the geometric model 48

5.9 *Angle* constraint of the geometric model 49

5.10 Geometric model: Four bar 49

5.11 Constraint graph representation of four bar 50

5.12 Four bar: Some solutions 50

NOMENCLATURE

Identifiers:

X	Set of variables
x_i	Variable
D_i	Finite set domain
$ext(C)$	Set of tuples
N_i	Graph arc representation
V	Graph vertex
G	Number of vertices
M	Computation mode
A	Value domain of a model
$*A$	Subset of A except empty set
R	Set of relations

Relations:

$\in$	In
$\forall$	For all
$\exists$	Exists
$\neq$	Not equal to

CHAPTER I

INTRODUCTION

Conventional CAD systems provide powerful tools for geometric modeling. Constructing geometric objects that have to meet the given geometric constraints still requires much work and experience with the tools provided by a CAD system. The geometric constraints represent the functional relationships between geometric objects. A geometric modeling CAD system would be much desired if the geometric objects met geometric constraints automatically rather than explicitly constructing them. Such kind of constrained-based system is useful in the design and simulation of geometric models. Some applications of such constraint-based systems include floor plan design, aircraft wing design, and simulation of a robotic arm. Geometric constraints in a floor plan design could include the area of a room, position of a door/window as shown in figure 1.1. Likewise in the case of aircraft wing design the positions of the ailerons and flaps, and the angle of the wing tip are represented by geometric constraints as shown in figure 1.2. The simulation of angular movements of the robotic arm are represented by geometric constraints as shown in figure 1.3. Conventional geometric modeling tools present in CAD systems do not offer a designer any support in time-consuming and error-prone calculations of coordinates and dimensions.

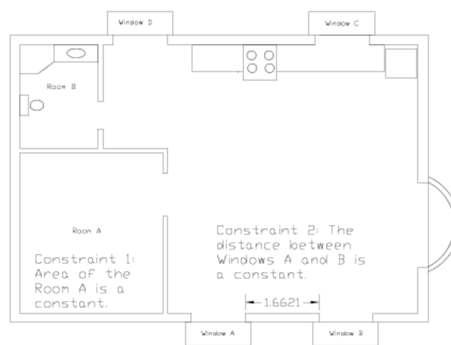

Figure 1.1: Geometric constraints in floor plan design

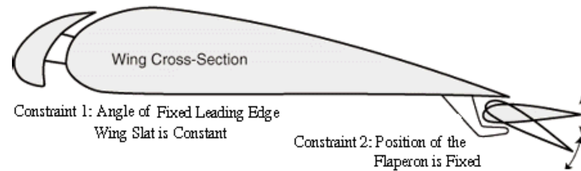

Figure 1.2: Geometric constraints in aircraft wing design

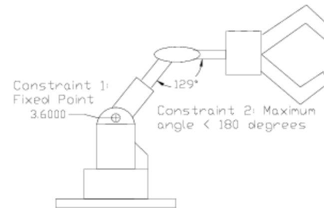

Figure 1.3: Geometric constraints in simulation of robotic arm

For example the CAD system would be able to draw a rectangle but would not be able to draw the rectangle by satisfying the constraints like the area of the rectangle be 20 square units and the length of one of the sides is 5 units. To meet these constraints, the designer has to perform additional calculations extending the design time and often presenting new sources of errors.

A geometric model can be described by defining constraints among geometric elements. The geometric constraints should maintain constraint consistency. Algebraic modeling and rule-based methods are used to maintain constraint consistency. In algebraic modeling of geometric systems the relationship between geometric elements are represented as equations, which are solved for the characteristics of the model [36] [33] [1]. Characteristic points and dimensions are taken as the basic elements to form the constraint equations, which will be solved simultaneously in order to guarantee the consistency of the geometric model after the model has been modified.

In rule-based methods the geometric relationships are defined as facts or rules [14] [5]. To maintain consistency of the model, these facts or rules are used to perform the inferences on the symbolic level. The method of parametric design defines construction sequences for sample

models. These construction sequences store the dimensional and topological information of the geometric models. The geometric model is created by these construction sequences.

To model geometric constraints, a constraint-based system is well suited. A geometric constraint is a logical relation among several unknowns, each taking a value in a given domain [4]. The constraints specify the relationship that must hold without specifying a computational procedure to enforce that relationship. The constraint solver does automatic solving of geometric constraints. This mechanism represents an interface between declarative description for the designer and a procedural description for the constraint solver [10]. The description of constraints in constraint-based systems fall into two categories:

1. Constraints that have distinguished output variable, in which the constraint solver is only allowed to change that variable.

2. Constraint solver is free to change any variables of the constraint.

Along with the classification of constraints, the constraints are associated with constraint hierarchy. The hierarchy of constraints is represented by the strength factor of the constraints. In this case the constraints represented with maximum strength must be satisfied and the other constraints are preferences. The constraint hierarchies represent state and change over time. We can view the geometric constraints and geometric entities forming a graph [9] [10] [44].

The geometric entities are represented by the nodes and the constraints are represented by the edges of the graph. These methodologies have been explored by a number of researchers, and used in a variety of systems and also provide good balance between expressiveness and efficiency.

The G+ system introduces contraint-based techniques into CAD systems to make geometric modeling more efficient and precise. The G+ system combines the advantages of the above mentioned methodologies. At the same time, conventional constraint-based systems gives rise to number of problems. The use of constraints may make the user interfaces complicated and hard to understand. The reason for this is that the specification of constraints has persistent effects on future operations. The effect of operations on mutually related geometric entities is harder to predict. The major difficulty is the conflict between two ways of constructing the programs, constraint programming and object-oriented programming. Object-oriented programming is a common technique for construction of interactive programs, because of its features of abstraction and modularity [40]. Functional operations are mutually applied by the objects to determine

the dynamic behavior of the entire system. Object-oriented concepts are suitable for modeling reactive systems. Objects can be created dynamically during run-time and may change their state. In constraint programming the system is modeled as a graph. The geometric objects maintain consistency by the use of a constraint solver. Since objects are related to each other through constraints, changes in objects have a global effect on the entire system. The G+ system integrates constraint programming techniques and object-oriented programming for applications, that particularly require event-based communication with end-users. The work is done considering the concepts, constraint solver mechanism, constraint specification and user interface.

Extensive research has been done in the field of constraint programming. Numerous efforts has been done in the field of geometric modeling in conventional CAD systems. This work takes the better of these two extensive research areas and integrates them to form a constraint-based geometric modeling system, which successfully interfaces with a commercial CAD package. This enables designers to take advantage of their existing CAD package along with the enhanced geometric modeling, based on constraint programming. The constraint satisfaction of the G+ system is not merely viewed as a system that takes a set of constraints as input and produces a solution as output. Constraint satisfaction is viewed as a system for maintaining consistency between objects. In this view the G+ system represents an object-oriented constraint system as a graph, to efficiently maintain the constraint consistency between geometric objects.

CHAPTER II

RELATED WORK

Recent research in the constraint logic community has been towards systems that perform automated design. A constraint-based system developed by Ivan Sutherland known as Sketchpad [39], paved way for the research on many constraint-based systems. Recent research in the area of constraint-based systems have employed constraints, which have a distinguished output variable [21] [22]. It has been observed that the research intent for these systems is to enhance the design capability, capture the design intent [37], focus on the solution of the specified problem and not on the procedure to arrive at the solution. This chapter discusses some of the current technology that is presently being used in constraint representation and constraint solving.

2.1 Constraint Representation

The functional or semantic representation in the geometric environment enables the model to function consistently with the constraints [38]. The semantic representation models like Boundary representation (BREP) and Constructive Solid Geometry (CSG) [36] were developed. These semantic representations did not yield the desires capabilities [15]. In his research Ioannis Fudos [15] has developed Editable REPresentation (EREP) to overcome the problems caused by BREP and CSG.

Another method for constraint representation is known as the *feature* method. In this approach, parts of the geometric model are represented by predefined geometric objects. The parametric representation of constraints is primarily based on this *feature* approach [3] [42]. The same concept of features is described as *base parts* and as *modules* [25]. The constraint-based graphics system known as Juno was developed by Nelson [32] in the year 1985. The constraints were internally represented as nonlinear equations. Nelson goes on to state that representing geometric constraints as nonlinear equations may slow down the system. In recent research it

is found that designer would be able to represent the design intent in the form of constraint representations. Geometric constraints representing design intent of the geometric model needs to be solved, to obtain a consistent solution.

2.2 Geometric Constraint Solving

Research in the field of geometric constraint solving has devised several methods and approaches for solving geometric constraints. In equational constraint solving method, geometric relationships and functions represented by constraints are be mapped to a system of nonlinear equations [24]. Aoudia [2] in his research on reduction of constraints suggests, that system of equations can be solved by graph decomposition. Some authors approach the geometric constraint solving problem by propagation techniques [35] as discussed in section 2.2.3. The drawback of this approach according to Arinyo [24] is the risk of losing domain specific information of the geometric constraint system. This section describes the methods used for solving geometric constraints.

2.2.1 Algebraic Constraint Solvers

In the algebraic method of constraint solving the geometric constraints are translated into algebraic expressions. The algebraic expressions are solved to obtain the desired solution. The algebraic expressions can be solved by Newton-Raphson [32], Relaxation [39], Homotopy [26] and Symbolic algebraic [28] methods. The constraint solvers based on Newton-Raphson method require good initial values. The initial geometric model must almost satisfy all constraints already. The algebraic constraint solvers when using the numerical approach require the initial values of the geometric model to be correct. A problem may occur when using the numerical approach when the solver arrives at a solution, which is not suitable for the present geometric model, and no alternative is available.

2.2.2 Constructive Constraint Solvers

Basic constructive steps are used to satisfy the constraints in the case of constructive constraint solvers [16]. An example of constructive constraint solvers is a simple engineering drawing, where the drawing is constructed by ruler, compass and protractor by a sequence

of steps. Constraint propagation and logic inference are the building blocks of constructive solving of constraints. This approach was first used for defining shapes by horizontal and vertical dimensions. The geometric entities like circles and arcs were later included to this approach. Multiple solutions, which resulted from the solving process, were approached by directionality and geometry triplets [33]. This approach was later refined to handle over and under constraint systems [16]. Researchers focused on developing this approach into an expert system and solving constraint system by global constraint propagation. Constructive constraint solving is classified into rule-based and graph-based methods.

2.2.2.1 Rule-Based Constructive Constraint Solver

The constructive constraint solver has to determine the sequence for reaching the solution. The solution should be reached in minimum sequence of steps and loops should be avoided. The rule-based constructive constraint solvers use rewrite rules and tables to identify and execute new steps [11]. Some rule-based constraint solvers consider non-unique solutions. According to Bouma [10] logic programming approach for constraint solving is good for prototyping and experimentation, however he considers extensive computations in searching and matching rewrite rules constitute a liability.

2.2.2.2 Graph-Based Constructive Constraint Solver

Constraint dependencies are represented by a graph and are used to obtain a valid solution. The sequence of construction steps is derived from this graph representing the constraints. An edge-vertex graph is used to represent a geometric modeling problem. The vertices of the graph are used to represent the parameters of the geometric entities and also represent geometric constraints. The edges of the graph are used to represent the connection between these geometric parameters and constraints. Another graph-based method for constructive constraint solver would be to represent the geometric entities as vertices and the constraints by the edges of the graph [10]. The graph-based construction enables deriving the properties of the problems without solving the constraints. In his research Owen [33] detects tri-connected components, and solves them separately and represents them as a single component in the graph representation. The constraint dependencies can be decomposed into smaller sets of equations. This concept is used

by Miller and Ramachandran [31] in their algorithm. Relevant graph-based methods can be found in the research of Venkatraman *et al* [41], Hsu *et al* [20], and Hanrahan [18].

2.2.3 Propagation Method Based Constraint Solvers

The constraints are represented by a system of equations. This system of equations is represented by a uni-directional graph [8]. The vertices of the graph represent the simultaneous equations and the edges represent variables and constants of the equations. The set of equations are solved from the graph edges. The solving process, proceeds in the direction of the uni-directional edges. In the process of solving the set of constraint equations, if the same set of constraints are reused, it leads to a cyclic pattern in the graph. Propagation method based constraint solvers did not yield the desired solution in the case of cyclic constrained problems.

2.3 Constraint-Based Systems

There has been a long history of constraint-based systems. We will briefly survey some of the constraint-based systems. Ivan Sutherland's Sketchpad [39] system was the key to constraint-based systems in the field of design, user interfaces and interactive systems. Most of the recent constraint based systems are based on one-way constraints or local propagation algorithms. Pfefferkorn [34] has developed a system for space planners. This system uses brute-force search algorithms. Borning [6] developed a system that uses object-oriented techniques for constraint representation and satisfaction. Graf [17] has developed a layout for multimodal presentations. In his research Graf addresses the incorporation of artificial intelligence aspects in the visual software design process. A knowledge based system for generation of floor plans was developed by Charman [12]. In this system new partial consistency suited to problem of semi-geometric arc-consistency is defined. A constraint-based system, PLS, for layout design of chemical process plant is designed by McBrien *et al* [30].

Extensive research was done and systems were developed to enhance the capability of geometric modeling in CAD systems with user interfaces. But these systems were seldom used because of the limitations of their design for specific applications. The G+ system attempts to define the constraint mechanism in a generic approach and successfully integrate this system to commercial CAD package. The G+ system uses a constructive constraint solver, which is

graph-based. The G+ system uses object-oriented approach to overcome the limitations of the constraint solver and graph component. This enables the G+ system to have simple access features to the user.

CHAPTER III

GEOMETRIC CONSTRAINT SPECIFICATION AND SATISFACTION MECHANISM

Geometry is represented by a hierarchy of geometric structures, which represent some under constrained set of geometric elements. Models are represented by a constraint network that is generated by instantiation and unification by a set of geometric elements. Large systems of algebraic equations are created during the process of geometric modeling by constraints. The graphs make it possible to polynomially decompose these systems of equations into constrained systems.

3.1 Representation of Geometry and Functions

The design mechanism is divided into two stages, conceptual design and detail design. The conceptual design process includes the functional description of the system being designed. The system being designed is decomposed into simpler functional representation. The detailed design process involves representing the exact geometric representation of the system components. The computer-aided design systems have long been used for detailed design process. By the constrained-based systems these computer-aided design systems can be used in conceptual design process.

A constrained-based system should have flexibility in representing geometry. Geometric design models without all the details of the geometric design are known as incompletely specified design structures. The design system should be capable of handling incompletely specified geometric structures. This is based on the fact that not all details of the geometric design are critical to the function of the geometric system. The two dimensional representation of a room may be important to the design, but the thickness of the walls may not be so important as the representation of the two dimensional room itself. Incompletely specified design structures have many advantages. These incompletely specified design structures could be reused more

widely than completely specified structures. This is because details pertaining to a specific design can be omitted. So incompletely represented geometric structures have the elasticity, so multiple constraints can be overloaded on the same structure. These basic incompletely specified structures can be spared from the details of a design, which can later be specified. Lower level structures such as an edge, parallel lines or intersecting lines are used for incompletely specified structures. Geometric models are generated by merging these incompletely specified structures. These incompletely specified structures can be deformed because of their elasticity and combined with other structures.

3.1.1 Representation of Geometric Functions

The main motivation behind using the incompletely specified structure based geometric model is the ability to attach functional constraints that may occur during the design. The functional constraints may describe the relative motions allowed among parts. This may include the motions that may occur during the assembly of the mechanism or during the operation of the mechanism.

The obvious approach to representing functional constraints is by a constraint language. This however can be avoided by using a slightly different view of the problem. The functional constraint can be represented by defining a constraint keyword that may use the predefined structures or describe a geometric function in this routine. Then if the functional constraint needs to be imposed on a geometric structure this keyword can be called in order to impose the functional constraint. By using this approach to represent functional constraints a more abstract function can also be represented. A new functional constraint can be a collection of already existing geometric functional constraints. A new constraint keyword can represent this new functional constraint.

3.2 Constraints and Graph Structure

A new level of abstraction to describe geometric objects is derived by the application of geometric constraints. To support incremental design each constraint has to be solved as soon as possible. For this reason local propagation of known states is used in constraint solving. A graph structure is used for constraint solving. The advantages of the graph structure are that

it is independent of the order in which the constraints are inserted and ease in replacing the constraints.

When the values of the input parameters are presented to the constraint solving system the constraint is solved, and the output parameters are determined. This property triggers constraint-solving mechanism. This in known as constraint propagation. The constraint propagation is managed by the constraint description graphs.

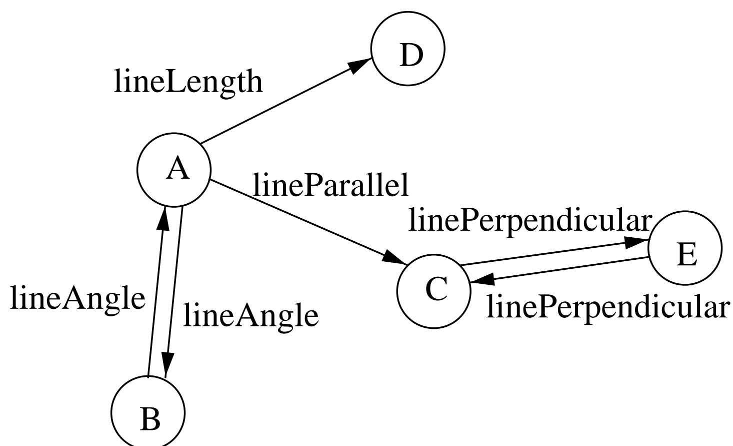

Figure 3.1: Graph with constraint representation

The geometric entities are represented by the nodes of the graph and the geometric constraints are represented by the edges of the graph. The nodes of the graph are created for each of the geometric entities used in the geometric model. The edges of the graph are either uni-directional or bi-directional based on the direction of constraint represented by the edges. Consider figure 3.1 where nodes A and C represent two lines. The edge connecting these geometric entities represent a constraint that these two lines should remain in parallel. The key word *lineParallel* is used to represent the constraint. The direction of the edge is from A to C. The constraint is imposed on the geometric entity represented by the node C, when the entity represented by node A is perturbed from the entity's present state. When the line represented by node A is moved to a new location, the line represented by node C is moved to a new location by the system in order to maintain the consistency of the constraint represented by the edge.

Consider another case between the nodes A and B, in which *lineAngle* constraint is represented by each of the two edges, one in each direction. Each of the nodes A and B represent two different lines. The nodes are connected by a bi-directional edge meaning that constraints are imposed

on both the nodes by the corresponding geometric entity. If the line represented by the node A is perturbed then the line represented by node B is moved in order to maintain the constraint imposed. This is also true when the line represented by the node B is perturbed. The bi-directional graph enables the designer to specify different constraints on each of the geometric entities based on the direction in which the constraints need to be specified.

To define the shape and position of a geometric object, enough geometric constraints have to be inserted. If a geometric object is already constrained the graph component issues a warning to the designer about the existence of constraints on the geometric entities. If the edges of the graph are cyclic the system issues a query to the designer, if the same entity that originated the constraint propagation needs to be modified. Consider the graph shown in figure 3.2, where the nodes represented by A, B and C are cyclic.

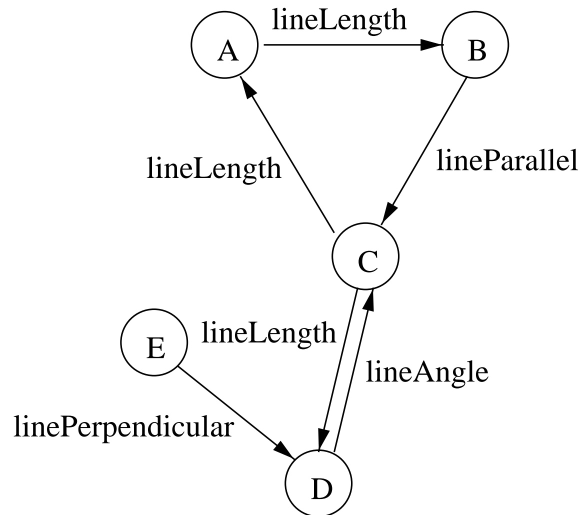

Figure 3.2: A cyclic graph

If the geometric entity represented by the node A is perturbed, then the constraint between nodes A and B is executed and imposed on the entity represented by node B. Since node B is perturbed from the previous operation entity represented by node C is imposed by the constraint between B and C. Then entity represented by node A needs to be modified since node C was modified. At this point the system issues a query to the designer if the entity represented by node A needs to be modified. Based on the designer's decision the constraints propagate in the graph. However the entity represented by node D is modified by the constraint imposed between nodes C to D, since C is not the originating node for constraint propagation. This approach

is favored since it decouples the constraint solving problem into groups of smaller systems of equations which can be solved independently and merged, rather than framing the problem as a single, large algebraic system to be solved (refer section 2.2).

3.3 Geometric Constraint Solving

Constraint programming is broadly classified into constraint satisfaction and constraint solving. Problems that vary over a finite domain are classified as constraint satisfaction. Problems that are defined by a set of constraints, which are solved to get the desired solution, are classified as constraint solving. Constraint satisfaction algorithms use combinatorial methods, whereas the constraint solving algorithms use algebraic methods.

3.3.1 Constraint Satisfaction

Constraint satisfaction problems are defined by set of variables $X=\{x_1, \ldots, x_n\}$. For each variable x_i, a finite set D_i representing its domain, and a set of constraints restricting the values that the variables can simultaneously take.

The solution of a Constraint Satisfaction Problem (CSP) is defined by a value from the domain for every variable represented by the problem. Each variable should be satisfied in such a way that all the constraints are satisfied. The solution to the CSP can be found by searching the domain for the right value of the variables.

3.3.2 Consistency Techniques

An approach to solve CSP problems is to eliminate inconsistent values from the solution domain till the accurate solution is obtained. Consistency techniques are basically like the deterministic search algorithms. The CSP is represented by a graph component. The nodes correspond to the variables and the edges represent the constraints. The simplest consistency technique is referred to as node consistency. This technique eliminates the inconsistent values assigned to variables. A value is decided as inconsistent if the unary constraints on the respective variable are not of the specified domain.

3.3.2.1 Arc-consistency

The definition of arc-consistency states that if C be a constraint on n variables X_1, ..., X_n, *ext(C)* be the set of tuples of the values X_1, ..., X_n for which the constraint is satisfied, then the domains D_1, ..., D_n are said to be consistent with the constraint C iff:

$$\forall X_i \in \{X_1, \cdots, X_n\}, \forall v_i \in D_i, \forall j \neq i,$$
$$\exists v_j \in D_j \text{ such as } (v_1, \cdots, v_n) \in ext(C) \tag{3.1}$$

A graph is said to be arc-consistent, if the domains are consistent with all the constraints. Consider the arc represented by (N_i, N_j) in the figure 3.3. The arc is consistent only if the values in N_i and N_j are permitted by the binary constraint between N_i and N_j.

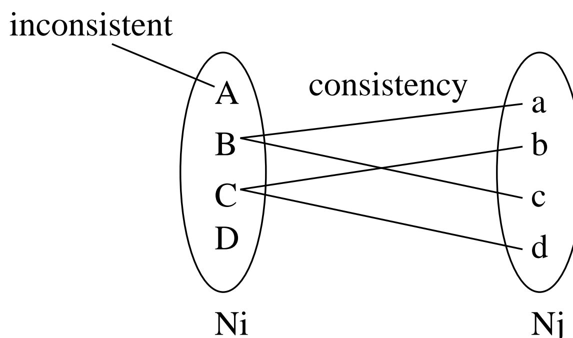

Figure 3.3: Local Inconsistencies

3.3.2.2 Path-Consistency

Path consistency requires for every pair of values of two variables X, Y satisfying the respective binary constraint, that there exists a value for each variable along some path between X and Y, such that all binary constraints in the path are satisfied [4].

3.3.3 Constraint Propagation

Systematic search in the solution domain and consistency techniques are used to solve CSP. In the case of already instantiated variables, consistency checks are performed. This method is known as *look back*. *Backtracking* is a method of solving CSP. The consistent values of some of the variables are found by extending a partial solution. The partial solution is extended towards

a complete solution by a value form another variable. This variable is consistent with the values in the current partial solution. *Backtracking* is an example of *look back*. The *look back* schemas solve the inconsistency when they occur but they do not prevent inconsistency from occurring. *Look ahead* schemas are proposed to prevent future inconsistencies. *Forward checking* is an example of *look ahead* schema. In the case of *forward checking* schema, when a value is assigned to a variable it is checked if it has a conflict with a value in the domain of a variable in the next steps.

3.3.4 Over Constrained Models

A model, which is represented by a large set of constraints, may not be able to find a solution that satisfies all the constraints presented in the model. These models are known as over constrained. The partial constraint satisfaction and constraint hierarchies are some of the approaches to handle over constrained models.

The partial constraint satisfaction involves finding values for a subset of the variables that satisfy a subset of constraints [43]. The principle of partial constraint satisfaction is that of reducing the constraint precedence from a strong constraint to a weak constraint by expanding the solution domain. Another method for handling over constrained problems is by constraint hierarchies [7]. The constraints are weakened explicitly by specifying precedence of constraints. The hierarchy does not permit weakest constraints to influence the result of the solution if the weakest constraint poses inconsistency over a stronger constraint.

3.4 Object-Oriented Constraint Model

The object-oriented constraint model is an extension of the conventional constraint object model. The object-oriented constraint model has classes, methods and inheritance. The concept of object provides a mechanism for extending the computational domain of the system. The object-oriented mechanism provides both primitive objects and the means for the designer to create new application specific objects. An object is viewed as a container of a collection of variables. These variables are associated with a set of operations that can be applied on the object. Each object has unique pointer or reference for the objects. The implementation of the objects is referred to as classes. These classes have the variables and methods for the

geometric entities. The class describes both, variables which are contained in the instances and how operations are carried out. In the object-oriented constraint model, the objects for the geometric entities are created. The geometric constraints are added to the constraint-based system to establish a connection between the objects of the geometric entities. The constraints are solved to achieve initial consistency. If any of the geometric entities are moved from its present state then the constraints are solved again to maintain consistency between the constraints.

A constraint is a logical relation between variables or objects. The objects refer to geometric entities. The constraints are referred to by constraint keywords. These keywords are function names, which take arguments, which have the implementation of the solving methods.

```
<object>::=<geometric_entity_class>
<constraint>::=<constraint_name>
<constraint_name>::=<solving_methods>|<consistency_techniques>
```

A constraint involving variables of basic types is considered as a relation between the variables while a constraint involving pointer variables is considered as a relation between the objects that the variables refer to. The former is realized by call-by-name in argument passing while the latter is realized by call-by-reference. The constraint added to the constraint-based system is dynamic. The constraint is independent of the object. The objects, which refer to the geometric entities, are chosen during the run time.

```
[line_B <sameLength> line_A]
```

The constraint *sameLength* imposes a constraint on *line_B*, that the length of *line_B* should be same as that of *line_A*. This means that when the length of *line_A* is changed, the length of *line_B* should also change in order to satisfy the constraint. In function form the constraint would be of the form,

sameLength(line_A, line_B)

The lines *line_A* and *line_B* are objects of the *line* class. The *line* class would have variables and methods for the function and representation of lines. Graphically the constraint can be represented as an arc between two objects *line_A* and *line_B* as in figure 3.4.

When the constraints are added it will reside in a constraint store. When the rector mechanism for triggering the constraints is called the constraints would be fetched from this constraint store.

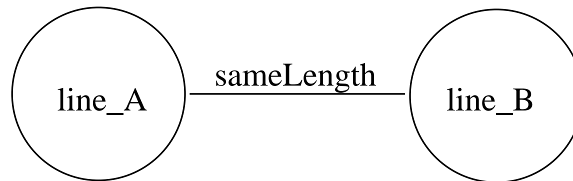

Figure 3.4: Constraint arc between *line_A* and *line_B*

3.4.1 Consistency maintenance in Object-Oriented Systems

In object-oriented constraint models, constraint satisfaction is achieved by maintaining consistency between objects. The solution obtained by solving the constraints is viewed as a new object binding which maintains new set of consistencies. The solution from the constraints is part of an object. The value of the solution lies in the variables of a class instantiated by an object. The object-oriented constraint system switches between consistent and inconsistent states. When an object is changed the system may switch from consistent to inconsistent state. The constraints from the constraint class have to be solved by the constraint solver to regain consistency. The objects that may be inconsistent during a change may be small as compared to the large number of objects that are present in the geometric model.

3.4.2 Dynamically asserted Constraints

Constraints are mostly asserted during the creation of the objects of geometric classes. By object-oriented systems it is possible to dynamically add and remove constraints during the run-time. An example of dynamically asserted constraints is by selecting the objects during run-time and adding constraints to the selected objects. Consider the case where the constraint *sameLength* has to be imposed between lines *line_A* and *line_B*. The designer during run-time enters the constraint, *sameLength*. The system then prompts for the designer to select the geometric entities. The designer selects the geometric entities, which have objects corresponding to each of the entities.

3.5 Constraint Model to Capture Design Intent

Geometric models have certain key aspects of design intent. The design intent of the geometric model is represented by constraints, design history, and features [37]. A static design provides no

mechanism to capture design intent. The static design is a barrier between product information and mechanical design applications. The geometric model rather than being a drawing file in CAD systems, by capturing the design intent, enables the designer to modify or regenerate the design from explicit information. The constraints capture the logical inter-relationship as well as the inter-dependence of design/shape data. These interactions of the design/shape and constraints shown the design intent.

The design history of the geometric model is a record of the design activities or events in the creation of the design. The design activities or events are signified by the creation of geometric entities and imposing constraints on these entities for a specific geometric design [19]. The design history is recorded as a class that has the entity classes, which are accessed by the objects, and also the constraint imposed on these objects. The new design classes that are created point to existing entity classes and constraints to make the new design class upward compatible. The new design class is self-sufficient without any new entities accessing it.

The geometric specification is broken into low-level constraints through a sequence of design activities. Thus, for a design to satisfy even a single specification it may require the validation of mathematical equations, engineering operations, or logical conditions at different phases of the design cycle. This design activity is represented by the design class. The design class acts like a skeleton structure with the sufficient constraint information. The skeleton structure of the design leaves out minor attributes. Minor attributes are those set of constraints and design features that do not completely change the design intent. When the design class is instantiated these minor attributes can be added in order to customize the design to the new geometric model features. An example for the design class would be a class for designing the carburetor for a car. This carburetor design class would have all the constraints and variables for the carburetor design. However, details like the specification for diameter of valves and length of adjustment screws are left with a default value. The designer could modify the specifications according to the car's engine design.

Identifying the elements that control the design outcome and the constraints facilitates modification of a design with limited intervention while enabling the preservation of the design intent. Few attribute values are changed as long as the consistency of the constraints is maintained. A pre-defined constraint for the design class could be added. The purpose of such a

design is to capture the designer's intent of the model, so as to maintain certain characteristics of its shape if any modifications are made.

CHAPTER IV

THE STRUCTURE AND IMPLEMENTATION OF G+

The G+ constraint-based system is designed with an interface to AutoCAD2000, to offer support for the design activity, by understanding the functional requirements of the design product, by imposing constraints on the geometric model. The G+ system is designed to be simple from the designer's perspective for adding and deleting constraints on geometric entities representing the geometric model in AutoCAD2000. The G+ system architecture is built to handle the high-level constraint representation of the designer to low-level constraint solving mechanism. Software modules are developed to handle the internal representation of the constraints and geometric entities, solving the constraints, capture design intent and interface to CAD system by object-oriented methodologies.

4.1 The G+ System Architecture

The designer enters the constraints, which represent the functional description of the geometric model, as *keywords* into system. These constraint definitions are of two different types.

1. Pre-defined constraints,
2. Generic constraints.

The pre-defined and generic constraints are explained in section 4.2. The Geometric entities created in AutoCAD2000 are internally accessed by the G+ system through the entity classes. These geometric classes have the variables and functions of the corresponding geometric entity. When an entity is created in AutoCAD2000 by the designer the G+ system gives the designer the choice of choosing between the pre-defined constraints or generic constraints. The pre-defined and generic constraints are each derived classes of the constraint class.

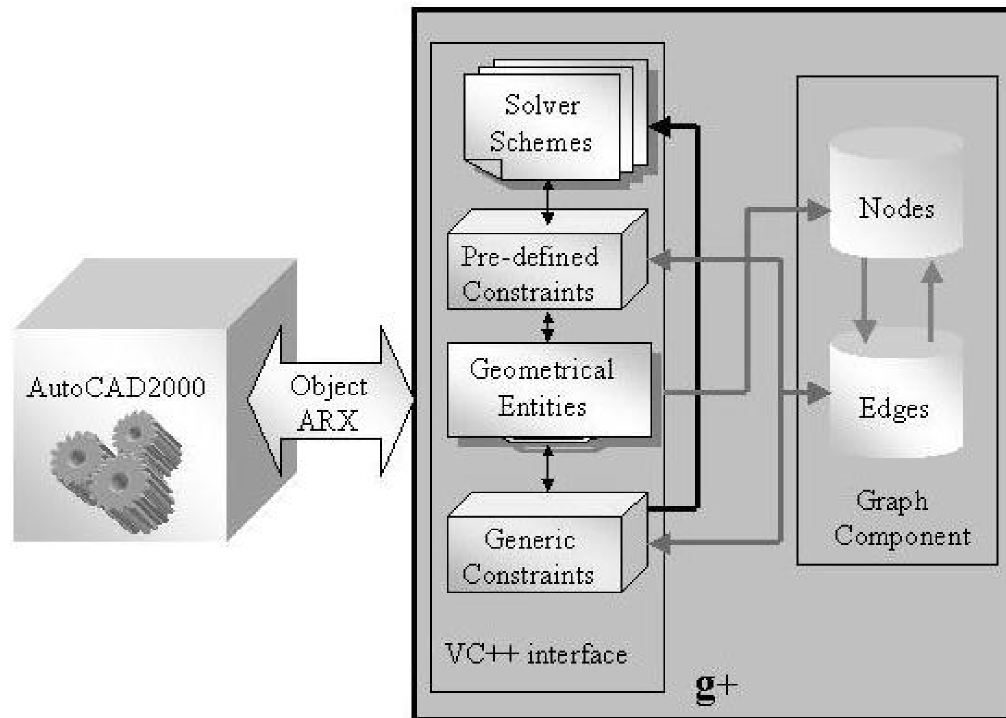

Figure 4.1: The G+ system architecture

Both the pre-defined constraints and generic constraints have access to solver schemes (refer section 4.2.2). The geometric entity classes, pre-defined, generic constraints and solver schemes are all developed in VC++. The VC++ interface communicates to the graph component of G+. The pre-defined and generic constraints are represented by the edges of the graph. The graph component of G+ is developed with the Generic Graph Component Library (GGCL) [27]. The nodes of the graph represent the objects of the geometric entity classes. On constraint *activation* the nodes of the graph get the constraints represented by the edges. The G+ constraint solver fetches the constraints and geometric entities from the G+ graph component and the solution of the problem are imposed on the geometric entities represented by the nodes. The architecture of the G+ system is shown in the figure 4.1.

4.1.1 G+ Graph Component

The graph component of G+ uses GGCL [27] now part of the C++ Boost library [13]. The GGCL offers a generic programming framework for graph data structures and graph algorithms. The objects of the geometric classes are placed in the nodes of the graph. The constraints are

represented by constraint keywords in the edges of the graph. The GGCL offers abstract interface that serves vertex iteration, adjacency iteration and property access. The G+ system uses the graph data structures to store the object identifiers of the geometric entity classes. The graph algorithms use the graph data structures provided by GGCL.

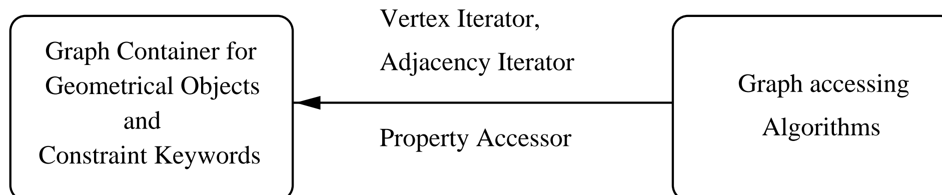

Figure 4.2: The G+ graph component

The G+ system uses a bi-directional graph for representing the constraints. The number of vertices of the graph is based on the number of geometric entities in the model. Based on the number of vertices the graph is internally generated. The generic programming paradigm of GGCL enables the graph component of G+ to be flexible and reusable. The independent graph components can be generated based on the geometric model. The sub graph would represent pieces of the geometric model that can function independent of the other parts of the model. The generic constraints can be designed based on sub graphs. Thus the entire sub graph can represent a node in the global constraint graph representation. This property of the graphs let the algorithms and data structures to freely interoperate.

The terminology used in the GGCL library is similar to that of the SGI STL. In the case of generic graph library like GGCL, any algorithm is written only once and used with any graph data structure. This enables the edge and vertex properties to be associated with vertices and edges. One method to store the vertex properties is to store the properties in an indexed vertex ID of an array. Another method to store the vertex properties is explicit storage inside the vertex data structure. The generic graph library provides a generic means for accessing the properties of a vertex or edge, regardless of the manner in which the properties are stored.

When the constraint reactor mechanism is triggered by moving any geometric entity, the node of the constraint graph gets the property associated with the vertex and sends it to the constraint solver. The property stored in the vertex of the graph in this case is object of the geometric entity class. The constrained-based system checks for the edges associated with this vertex. The direction of the edges is considered and the constraints associated with the edges are

retrieved. Any other vertex associated with the vertex, which triggered the constraint reactor, is also retrieved. The objects of the entity class stored in these vertices are also retrieved and sent to the constraint solver.

The constraint property that needs to be stored in the edges of the graph needs to be initialized by the Boost library. Also the template for the constraint-based geometric model is defined.

```
BOOST_INSTALL_PROPERTY(edge, constraint_keyword)
template <class Graph, class Model>
```

The vertex property is accessed by the vertex iterator. Since the constraints are represented by bi-directional graphs they are accessed by two iterators. One iterator for the in-edge and another for out-edge.

```
boost::graph_traits<Graph>::vertex_iterator Viter
boost::graph_traits<Graph>::out_edge_iterator OutEdgeIter
boost::graph_traits<Graph>::in_edge_iterator InEdgeIter
```

For example if the G+ constraint solver needs to access the list of edges leaving the vertex V, the constraint solver will be able to achieve this by the statement `v.out_edges()`. The GGCL graph data structure is constructed in a layered manner to provide flexibility and reuse. The layered architecture also provides several different points of customizability. If the graph properties are stored in `std::vector, std::list` the GGCL offers to build the graph framework out of these standard containers. Tuples for the edges, vertices are created with the `boost::tie` function template. Separate tuples are created for the out-edges and in-edges of the graph. The tuples are actually iterators pointing to the edges and vertices.

```
boost::tie(ui, uiend) = vertices(G)
boost::tie(out, out_end) = out_edges(*ui, G)
boost::tie(in, in_end) = in_edges(*ui, G)
```

The adjacency structure of the graph is configured in the `adjacency_list` class. The vertex, edge type and the properties for the edge and vertex are configured in the adjacency structure. For a model of `adjacency_list` the inner container must be variable sized whose `value_type` is the `size_type` for a vertex if the graph has no extra edge-associated data. The graph is generated based on the vertices and edges that are added to the graph by using `add_edge()`.

```
typedef adjacency_list<vecS, vecS, bidirectionalS, property, Flow> Graph
add_edge(0, 1, Edge("constraint_keyword"), G)
```

The G+ graph component with the GGCL implementation framework centers around the main *graph* interface class and the *graph representation* concept. The *graph representation* concept is basically a container. The *graph* interface class constructs the full graph interface based on the minimized interface exported by the *graph representation* concept. This allows *graphs* to be constructed out of standard container components with very little work.

By generic programming and performance optimization techniques used by GGCL, a great deal of efficiency can be gained. For many efficient graph data structures in GGCL, vertex and edge objects that model the GGCL interface concepts are not explicitly stored. In this case only partial information is stored. The interface layer constructs full vertex and edge objects on the fly from this information. The G+ graph component using generic GGCL algorithms allows basic algorithm patters to be applied in different ways, resulting in significant code reuse.

4.1.2 AutoCAD Runtime Extension

The G+ system interfaces with AutoCAD 2000 through a comprehensive Application Programming Interface (API) called as ObjectARX [29]. The ARX part of ObjectARX stands for AutoCAD Runtime Extension. The ObjectARX API contains some 220 classes over 3000 unique member functions. Many of the classes and functions are grouped by category.

Early versions of AutoCAD introduced AutoLISP. AutoLISP is a subset of LISP programming language. LISP is an interpreted programming language, so it does not require compilation. Because it is an interpreted programming language, it is slower than ObjectARX based on C++. AutoCAD 2000 includes Visual Basic in the form of Visual Basic for Applications. The limitation with VBA is that you cannot create custom AutoCAD objects. The AutoCAD Development System (ADS) was introduced in AutoCAD R10 for OS/2. ADS consisted of programming with C. ADS has been integrated into ARX in the form of ADSRX.

The G+ system was initially written in C++ in UNIX environment and later ported to Visual C++ in Windows 2000 environment. In the Visual C++ environment it is possible to use Microsoft Foundation Classes (MFC) to create dialog boxes using Class and CDialog. MFC provides Dialog Data Exchange (DDX) and Dialog Data Validation (DDV) to control various elements of your dialog. Commonly used functions and indeed dialog boxes can be stored in a

type of library called a Dynamic Link Libraries (DDL). DDLs are loaded at run time when your application calls them and they are independent of the code of the calling application. This is the approach AutoCAD takes with respect to application that are created using ObjectARX. The AutoCAD's internal code does not need to be changed to use a DLL. ObjectARX uses .arx extension rather than .dll extension.

A DLL is not an application in the sense that it will run on its own. The application will call on the service provided by the DLL. This means that AutoCAD must know the entry point into the service. The ARX entry point works on the basis of the *application message code*. The *application message code* messages are shown in table 4.1.

Table 4.1: Application Message Code

Application Message Code	Description
kInitAppMsg	Application is loaded to open communication with AutoCAD
kUnloadAppMsg	Application is unloaded
kLoadDwgMsg	Drawing is opened. Makes ADSARX function library available
kUnloadDwgMsg	User quits a drawing session and unloads ADS function library

The `acrxEntryPoint` has user-defined helper functions like `initApp()` and `unloadApp()` for initialization and unloading of the ARX application. The entry point for the ARX routine is shown in below.

```
AcRx::AppRetCode
acrxEntryPoint(AcRx::AppMsgCode msg, void* appId)
{
    switch (msg) {
    case AcRx::kInitAppMsg:
         acrxDynamicLinker->unlockApplication(appId);
         acrxDynamicLinker->registerAppNotMDIAware(appId);
         initApp();
         break;
    case AcRx::kUnloadAppMsg:
         unloadApp();
         break;
```

```
    }
    return AcRx::kRetOK;
}
```

The G+ system will be able to access the geometric entities of the AutoCAD by using entity handles. When the user selects an entity, the entity's handle can be retrieved and the appropriate record in the external database is found. An object ID is a unique identifier attached to the every object in AutoCAD. With an object ID, a pointer to the actual database object can be obtained. With this pointer the operations on the object can be performed. The object ID is from the class *AcDbObjectId*. AutoCAD 2000 can work with multiple databases with a single AutoCAD session. The AutoCAD's partial database is shown in 4.3. Each object in the session has a unique object ID for all entity databases loaded at that time. Whereas the entity handle is unique only within the scope of the particular *AcDbDatabase*. Thus G+ system uses the object ID in its graph component.

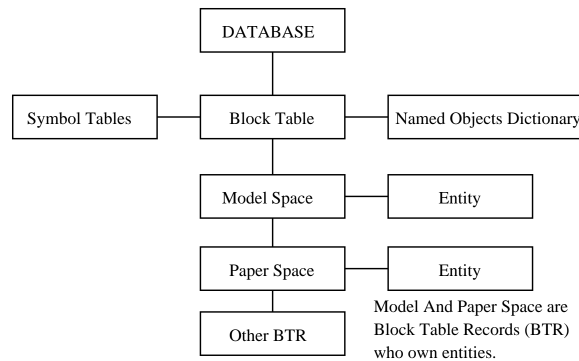

Figure 4.3: AutoCAD's partial database

The each geometric entity has a class, with all the variables and functions for the class. The geometric entity class can be instantiated by a pointer object. A G+ system accesses the variables and functions of the geometric entity class by *acdbOpenAcdbEntity*.

```
AcDbLine *<pointer object> = <memory allocation>;
acdbOpenEntity((AcDbEntity*&)<pointer object>, Entity ID, Opening Mode);
```

4.2 The G+ Constraint Classes

The class *constraint* serves as base class for group of derived classes. The base *constraint* class has the member functions and variables, which support communication between the geometric entity classes and the derived constraint classes. The base *constraint* class has the *algebraic solver* class objects. This enables the derived constraint classes to have access to these algebraic solver schemes. The G+ constraint solver has access to the geometric entity classes. The geometric entities are passed as arguments to the constraint functions as shown in figure 4.4.

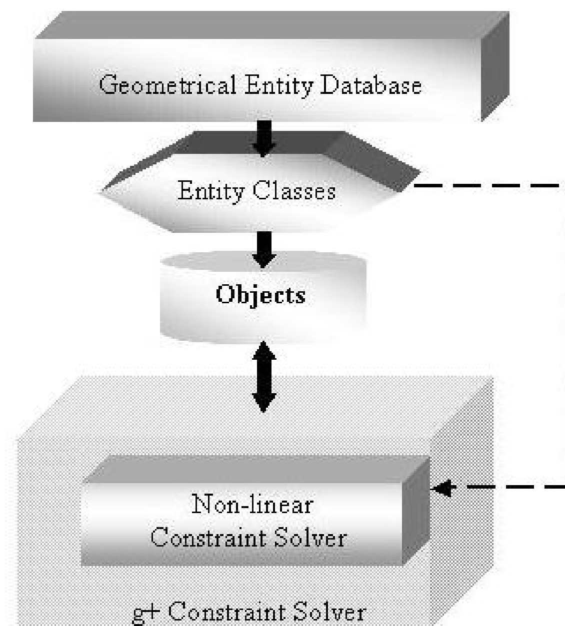

Figure 4.4: Entity access to constraint solver

4.2.1 Pre-defined Constraint Class

The *pre-defined* constraint class provides member functions, which define pre-defined constraints. These pre-defined constraint member functions, take the geometric entity arguments and impose the constraint function definition on the resulting geometric entity. Some of the pre-defined constraints are shown in table 4.2.

These pre-defined constraints are placed between geometric entities by selecting the entities in AutoCAD 2000 using the mouse pointer and entering the constraint keyword in the command prompt of AutoCAD 2000. The constraint definitions recognize the object ID's of the entities.

Table 4.2: Pre-defined Constraints

Constraints	Geometric Entities and Constants	Explanation
LineLength	Line l_1, Line l_2	The length of l_2 follows the length of l_1.
LineParallel	Line l_1, Line l_2	The line l_2 is made parallel to line l_1.
Angle	Entity e_1, Entity e_2, Angle θ	The angle θ is maintained between e_1 and e_2.
LinePerpendicular	Line l_1, Line l_2	The line l_2 is made perpendicular to line l_1.
Length	Line l_1, Length l	The line l_1 maintains length l.
Four Bar	Sides s_1, s_2, s_3, s_4, Points p_1, p_2, p_3, p_4	A four bar is created and maintains the side length and the property of the points either fixed or floating.

The object ID's are passed in the *constraint* class. Based on the solution obtained by the pre-defined constraint definition the result is imposed on the geometric entities. The pre-defined constraint class checks for the correct entity before applying the constraint definition. If the entity is not appropriate to the constraint definition the designer is asked to reselect the geometric entity. The object ID is cast to a line entity pointer object and then this pointer object is checked for the line entity. Consider the example for the *LineLength* constraint. The length of the line is obtained from the line drawn by the designer. Then this length is imposed by the *LineLength* constraint. The geometric entity is checked for the correct constraint argument.

```
AcDbLine *pLine = AcDbLine::cast(pObj);
if (!pLine) {
   const char* cstr = pObj->isA()->name();
   acutPrintf("This is a %s.\n", cstr);
   acutPrintf("Expecting line entity for this constraint.\n");
   return;
}
```

The length of the existing line is obtained.

```
AcGePoint3d p = pLine->startPoint();
AcGePoint3d q = pLine->endPoint();
```

```
AcGeVector3d v = q-p;
double len = v.length();
```

The length obtained from the previous line is imposed on the line on with the constraint is imposed.

```
p = pLine2->startPoint();
q = pLine2->endPoint();
v = q-p;
v = len * mFactor * v.normal();
pLine2->setEndPoint(p+v);
```

4.2.2 Generic Constraint Class

The *generic* constraint class is a template for user-defined constraints. The template framework has access to the constraint solvers and all the geometric entities in AutoCAD. The designer can have a new constraint that can be added to the constraint list with this framework. The designer will have to assign a constraint keyword that can be used to call the constraint in AutoCAD. The constraint keyword is registered as a command by using objectARX, so that the constraint can be called in the command prompt of AutoCAD.

```
acedRegCmds->addCommand(<constraint keyword>, <constraint function>);
```

The generic constraint function definition includes the functions for accessing the geometric entities and gets the values associates with the entities. The generic constraint would also have functions to set the solution values from the constraint solver routines. The generic constraints can have a collection of pre-defined constraints. If the designer needs to have a constraint over a parallelogram with length of one side being twice the length of another side. A general constraint for ratio of lengths between any groups of lines could be set with the generic constraint. Consider the following example to set the ratio between two lines by generic constraints.

```
v = len * <ratio_factor> * v.normal();
pLine2->setEndPoint(p+v);
```

The generic template would provide the variables that provide the co-ordinates of the lines. The *ratio_factor* variable value is provided by the designer to establish the desired ratio constraint

between the two lines. If the designer decides to use numerical schemes for constraint satisfaction and finding the solution, G+ system offers the algebraic solver class. The algebraic solver class has the following methods,

1. Lagrange Method of Multipliers,
2. Method of Steepest Descent,
3. Newton Raphson Method.

These numerical methods are used for finding the solution for constraints with minimization and non-linear problems. The designer uses the desired numerical scheme in his generic constraint specification. Usually the numerical method is chosen based on the order of accuracy and total number of iterations. The Lagrange method of multipliers is the default, which uses the method of steepest descent. This has been chosen considering the high non-linear nature of the problem particularly with geometric problems. However the designer can change to any other scheme based on requirement.

4.2.3 Solution for Nonlinear Constraint Systems

A mathematical problem solver Unicalc developed by Shetsov *et al* [23], to solve nonlinear, constraint systems and processing of partially known information (subdefinite). The solver is based on the combination of the following,

1. Interval Constraint Propagation,
2. Interval Mathematics,
3. Computer Algebra.

The Unicalc solver is based on the method of subdefinite models (or n-models) which is an approach based on representation and processing of incomplete knowledge. The problem being solved with the help of subdefinite models can be formulated as a constraint satisfaction problem. The subdefinite model is used to calculate the optimal parameters of the geometric model or to estimate its behavior on the basis of subdefinite information.

The value domain of the model variable is A. Let $*A$ denote subsets of A except empty set. The values other than $*a \in A$ which contain only one member is called subdefinite. When the

variable x is mapped uniquely from the set X into variable $*x$ which has value domain set $*A$. Thus the variables $*x$ are mapped onto set $*X$. These variables $*x$ are called subdefinite.

$$M = (X, R) \tag{4.1}$$

Consider a computation model M. The computation model M be determined by a set of variables X and set of relations R. Then the computation model can be represented as shown in equation 4.1. The linear and nonlinear problems posed by the constraint-based system can be solved by using Unicalc. The system of equations is moved to a file from the G+ constraint system. The Unicalc can solve this system of equations. Thus iterative schemes for nonlinear problems can be avoided. However at present the Unicalc has not been completely integrated into the G+ constraint-based system. The G+ system presently uses iterative schemes to solve nonlinear problems. On successful integration of Unicalc by the G+ constraint-based system, the system of nonlinear constraints can be solved by specifying the constraint keyword in AutoCAD 2000.

4.3 Control Flow of G+

The geometric objects imposed with constraints are monitored by a *reactor* mechanism. The *reactor* mechanism fires as soon as the geometric object is perturbed from its existing position. The G+ constraint based system captures the *reactor* signal and gets the corresponding graph node information. The object ID of the geometric entity stored in the graph node is retrieved. The constraint represented by the graph edge is also obtained and sent to the constraint solver as shown in figure 4.5.

The constraint represented by the edge of the graph is fetched from the constraint database. The constraint definition is sent from the constraint database to the constraint solver. The constraint solver fetches all the nodes associated with the node, that is moved. The corresponding geometric entity information is also retrieved from the nodes and sent to the constraint solver. The constraint solver then updates the values of the geometric entities.

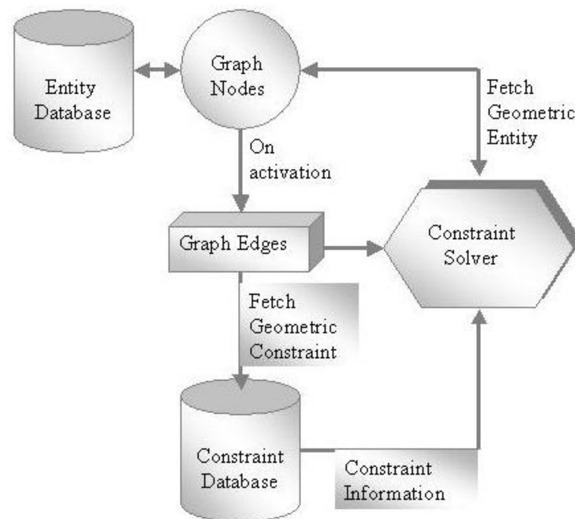

Figure 4.5: G+ control flow

4.4 G+ Application Example

In this section an application of the G+ constraint-based system is described. In this application factors such as sufficient support for the design activity, object-oriented nature of the system, constraint satisfaction are taken into consideration. This application is inspired by floor plan design by using computer aided design systems.

This application is about designing floor plan based on constraints set by the designer. The constraints are specified by selecting the geometric entity in the floor plan design and entering the constraint in the command prompt of AutoCAD 2000. When changes to the design are made the G+ system makes sure that the constraints are satisfied. In some cases all the constraints may not be satisfied so the designer is prompted to find an alternate to the current modification to the design. The designer can either find an alternate modification or change some of the constraints according to the design needs. Constraints are normally specified between the walls, rooms, window placement and position of the door. In this case each room, specific walls, windows, doors are specified as a geometric entities. In floor plan design by G+ constraint-based system, the following set of geometric entity objects and constraints may be used.

Geometric Entity Objects:

- Geometric Object:

  Walls, Rooms, Windows, Doors.

- Control Points:

  End points of walls, corners of rooms, edge points of windows and doors.

Geometric Constraints:

- Length Constraints:

  Length of Walls, breadth and width of doors and windows.

- Distance Constraint:

  Distance between two walls, distance between windows and/or doors, distance from a wall to a window or door.

- Area Constraints:

  Area of a room.

- Proportionality Constraint:

  Ratio of length between walls, windows and/or doors.

- Position Constraint:

  The position of the control points of rooms, windows, doors remain fixed.

Consider the floor plan as shown in figure 4.7. The designer imposes the following constrains on the design,

1. The area of room A is constant.
2. Windows A and D should be at a constant distance from the walls of room A and room B respectively.
3. The windows should be minimum 12 feet apart.
4. The windows B and C needs to maintain the proportional size and position between them.
5. The windows B and C should each be 6 feet from the front wall.

The figure 4.6 shown the constraint graph for figure 4.7. The designer decides to move the wall between rooms A and B to a new position as shown in figure 4.8. The new position of the wall is shown in green and old position by blue lines. This color convention is followed for all the figures. According to constraint 1 imposed on the room A the G+ system needs to maintain the area of the room as constant.

```
<room A>::=constant
```

The breadth of the room A is increased to maintain consistency of the constraint. The wall of room A overlaps the position of window A as shown in figure 4.9. The constraint 2 is imposed on window A.

```
<window A> distance::=constant <room A.wall>
```

The constant distance is maintained by the window A from the wall of room A as shown in figure 4.10. The distance between windows A and B is less than 12 feet. This violates the constraint 3 as shown in figure 4.10.

```
<window A> distance::=12 feet <window B>
```

The constraint 3 is activated and the G+ system moves the position of window B to maintain the constraint 3 as shown in figure 4.11. The constraint 4 is triggered and the position of window C is moved in order to maintain proportional position as shown in figure 4.12.

```
position, size::=proportional <window B><window C>
```

Both windows B and C are too close to the front wall violating the constraint 5. To maintain consistency of the constraints the width the windows D and C are shrunk as shown in figure 4.13.

```
<window C> distance::=6 feet <floor.wall_1>
<window D> distance::=6 feet <floor.wall_1>
```

The final design after maintaining consistency between geometric objects is shown in the figure 4.14 after moving the position of the wall between rooms A and B.

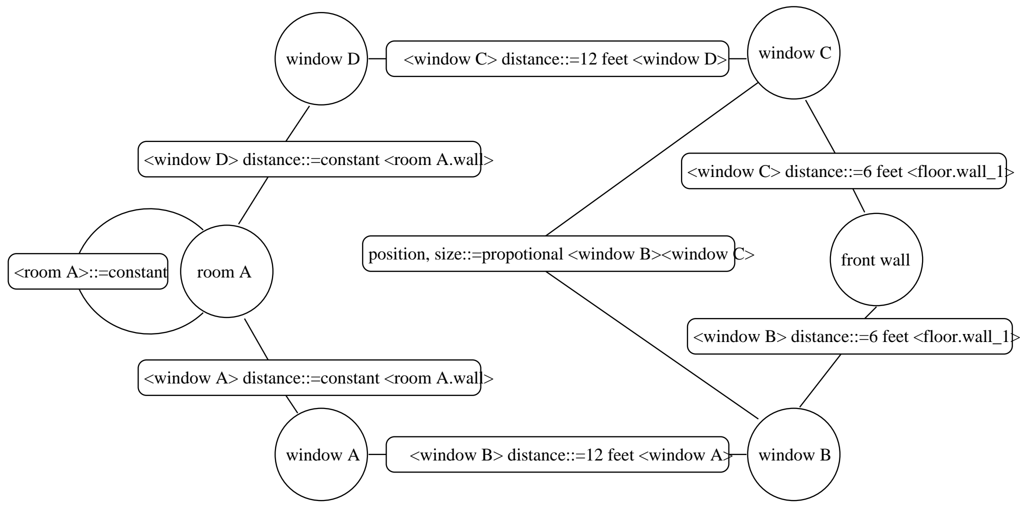

Figure 4.6: Constraint graph for floor plan example

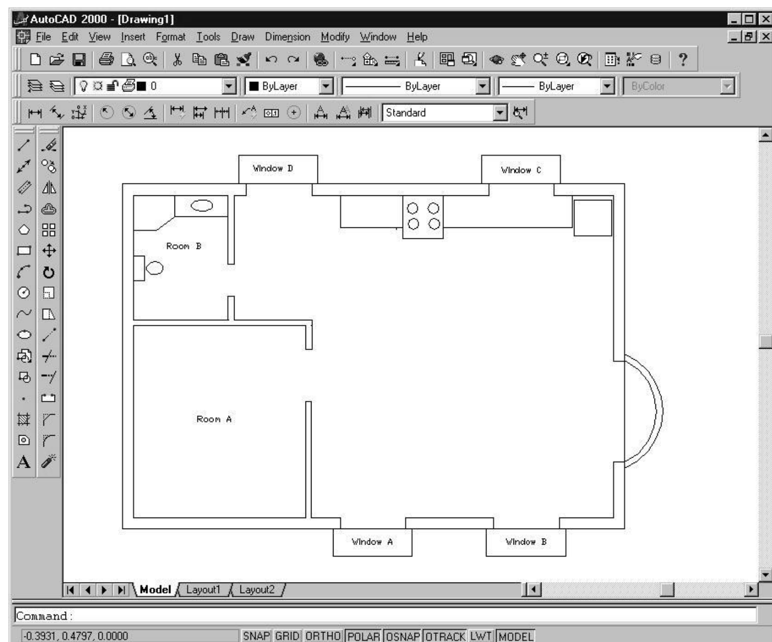

Figure 4.7: Floor plan

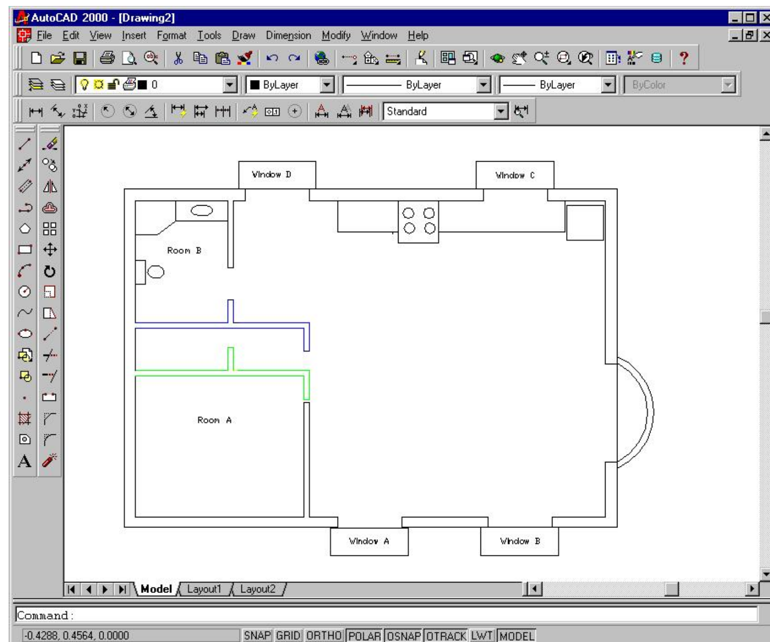

Figure 4.8: Floor plan: Moving the wall

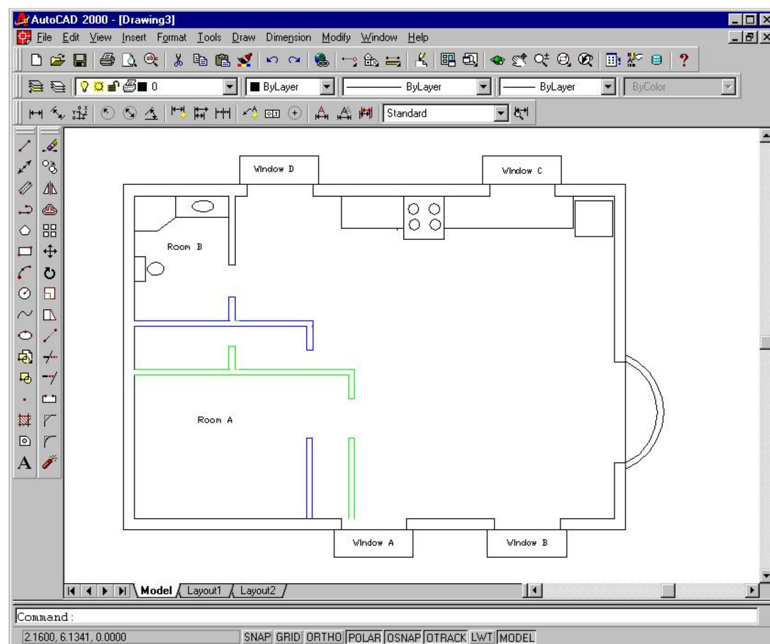

Figure 4.9: Floor plan: Window A overlapped by room A

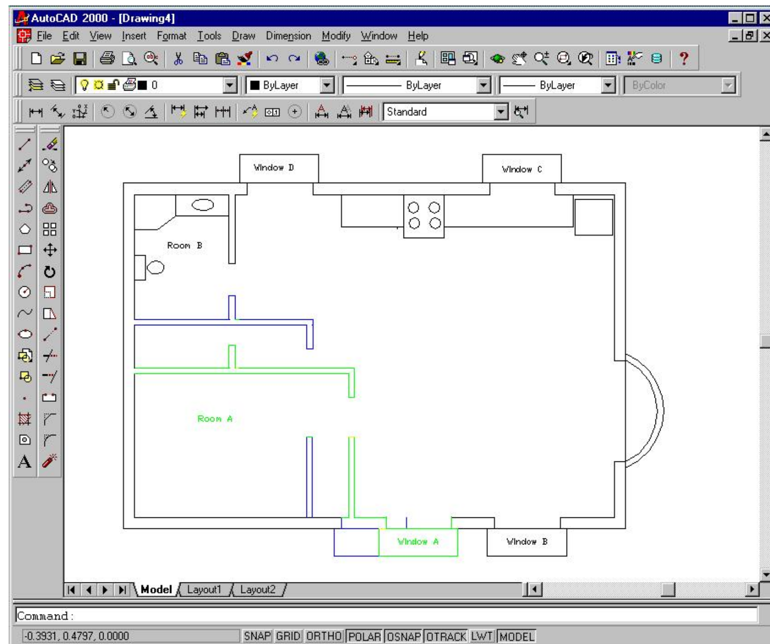

Figure 4.10: Floor plan: Movement of window A

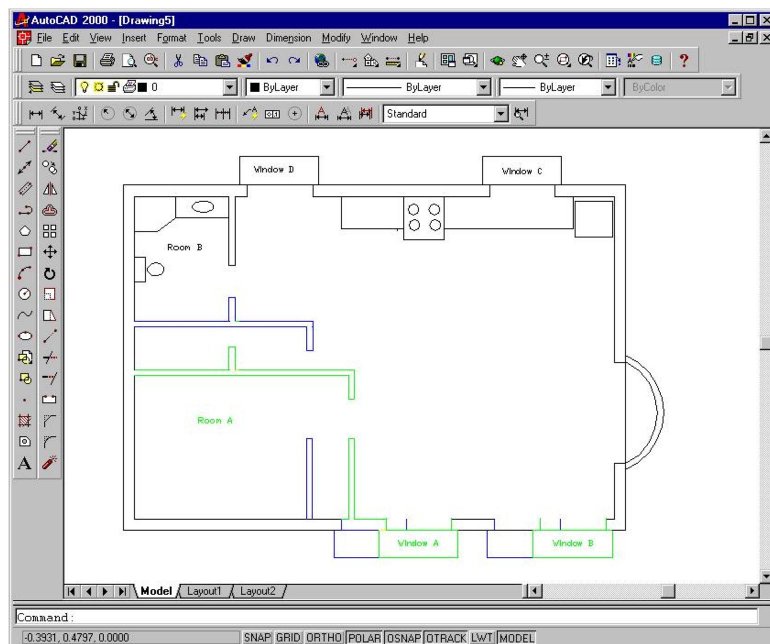

Figure 4.11: Floor plan: Movement of window B

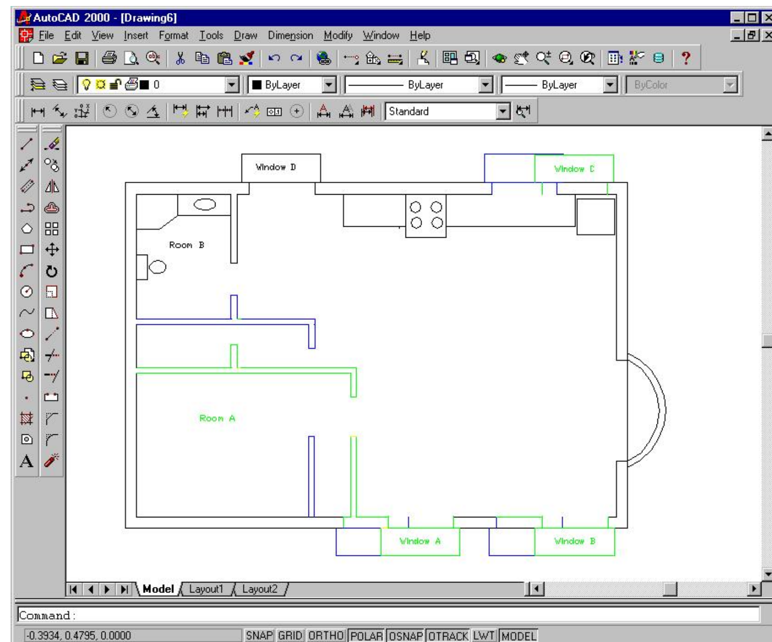

Figure 4.12: Floor plan: Movement of window C

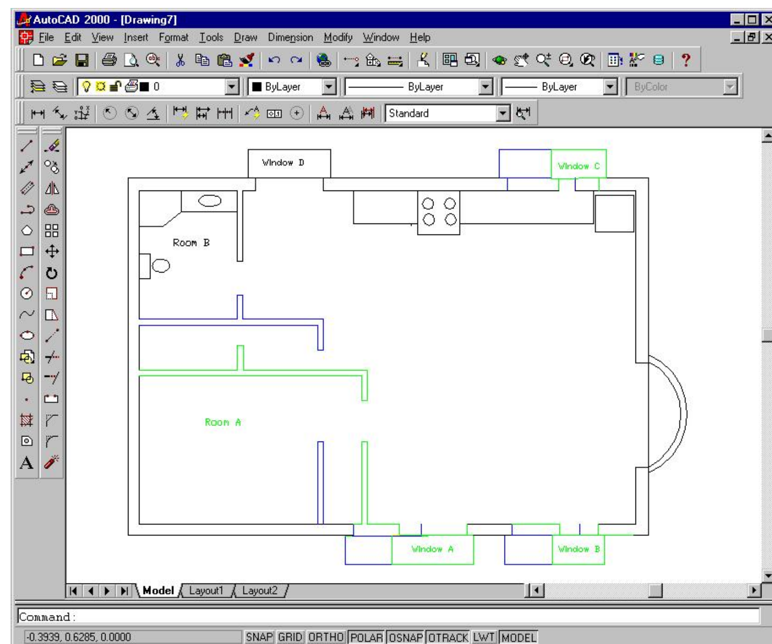

Figure 4.13: Floor plan: Width adjustment of windows B and C

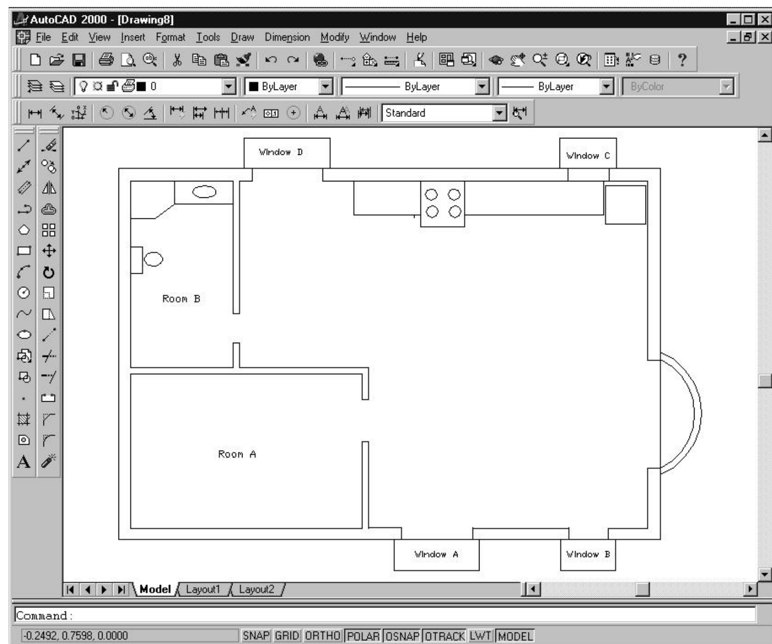

Figure 4.14: Constraint consistent floor plan

CHAPTER V

RESULTS

This chapter presents some of the constraint implementation on geometric objects by the G+ constraint based system. Some of the basic geometric objects are imposed with constraints and their behavior is studied under the influence of constraints. Cases have been considered for pre-defined constraints and non-linear solution of geometric models. The current G+ implementation was tested on AutoCAD 2000 on Windows 2000 platform. The G+ code was developed in Visual C++ version 6. The earlier version of the G+ constraint-based system was developed in C++ on Sun OS release 5.8. The code was ported from the C++ environment to Visual C++ environment for the ObjectARX interface with AutoCAD 2000. The system configuration in which the G+ constraint-based system was tested is an AMD Athlon, 1100 Mhz processor speed, and 256 KB cache. The operating system is a Linux version 2.4.9 with a VMware Workstation version 3.0 for virtual machine environment running Windows 2000.

The G+ system is classified based on the capabilities to support design activities for constraint models. The classification is as follows,

- Level 1: Solution for model represented by linear constraints,
- Level 2: Solution for model represented by non-linear constraints,
- Level 3: Capture design intent of the model,
- Level 4: Expert system for designing models.

5.1 Issues

Although the aim of this thesis is to provide support for the design activity in CAD systems, by maintaining the functional requirements in different design phases by the G+ constraint-based

system, some issues are yet to be resolved. The following section discusses some of these issues and the methods to resolve these issues.

5.1.1 Multiple Solutions

When a geometric model is imposed with constraints there are cases when multiple solutions may appear. In this case the G+ system relies on the constraints. Either one of the solution is chosen in the case of multiple solutions. Consider the case of two lines AC1 and BC1 as shown in figure 5.1. The constraint imposed on the lines are such that,

1. The points A and B are fixed.

2. The sum of the lengths of the lines AC1 and BC1 is a constant.

The point C1 is free to move. Based on the constraints imposed on this model the point C1 is free to move along an ellipse with the two points A and B as the foci of the ellipse. Few multiple solutions are shown as the points C2 and C3 in the figure 5.1.

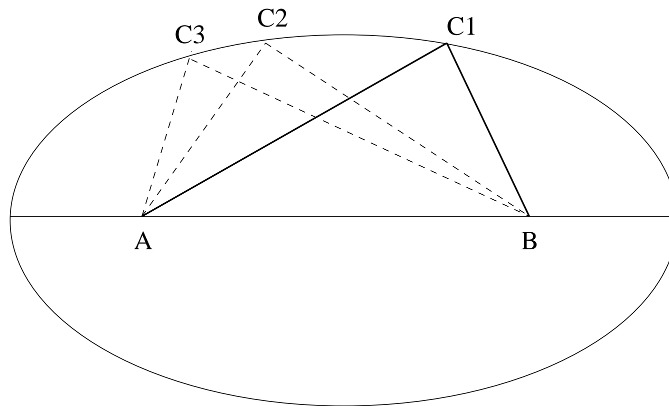

Figure 5.1: Multiple solutions along an ellipse

The G+ system at present offers possible solution to this problem by minimization constraints. In this particular geometric model the user may select a new point C as shown in figure 5.2. The minimization constraint may be used between the point C' and the point O, which is the center of the ellipse. By posing the geometric model by this method there will exist only one solution C to the problem as shown in figure 5.2.

However the issue needs to be resolved by the designer as to posing the problem as a minimization problem. The better method to resolve this issue would be to display in the

CAD environment all the possible solutions and the designer selects the desired solution. The multiple solutions can be displayed in the CAD environment by using dashed lines with a different color from the existing geometric model's color representation. In the case of AutoCAD 2000 the user may select the solution from the set of multiple solutions displayed by the G+ system. Computing all the possible solutions to the geometric model will increase the computation time. The advantage by this method of displaying all the possible solutions would be much more beneficial to the designer than the problem of increased computation time.

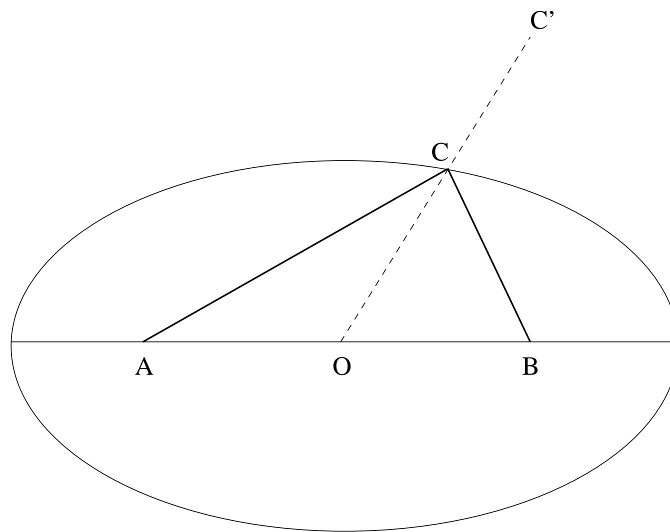

Figure 5.2: Solution by minimization

5.1.2 Framing Generic Constraints

The user-defined constraints are framed by using the generic constraint template. The generic constraint frame is filled with objects from the different methods available. There are objects for accessing the geometric entity classes and also objects for accessing the solver schemes. The designer needs to have knowledge in analytical geometry to formulate the user-defined constraints. The designer may need to frame mathematical equations and then translate the solutions to the geometric entities by using the geometric entity classes. Using the *CDialog* class and application wizard of MFC (Microsoft Foundation Class) in Visual C++ and creating a toolbox where the designer has the required mathematical and analytical geometry functions may overcome this issue. The designer can choose from this toolbox the desired functionality for the geometric model. This enhances the features of the G+ system for framing generic constraints.

5.1.3 Computation Time for Non-linear Problems

Those geometric models that have constraints that lead to models that are non-linear use iterative methods to find the solution. Some of these iterative methods take more computation time because of more iterations depending on the non-linearity of the problem. When using schemes like the Newton-Raphson the problem may not converge to a solution. Even thought the Newton-Raphson method offers quadratic convergence towards solution, a problem of not converging towards the solution may occur. Thus another scheme for iterative methods like the method of steepest descent is offered by the G+ system. By using the method of steepest descent for some problems the total number of iteration to arrive at a solution may be higher thus taking more computation time. But the method of steepest descent does provide convergence for most non-linear constraint problems.

This issue of increased computation time may be overcome by the successful integration of Unicalc (refer Section 4.2.3). The Unicalc solver provides non-iterative methods to find the solution to non-linear problems. The non-iterative schemes would reduce the computation time of the G+ system.

5.2 Constraint Consistency Validation

Consider the model shown in figure 5.4. Three lines are imposed with pre-defined constraints. The *lineLength* constraint is imposed between lines AB and BC. The direction of the constraint is imposed from AB to BC. Similarly the *lineLength* and *lineParallel* constraint is imposed between lines BC and DE, the direction of the constraint is from BC to DE.

The *lineParallel* constraint is imposed as a bi-directional constraint between BC and DE. This means when any of the lines BC or DE are moved, then the other line would also move to maintain the two lines in parallel. The model is shown in figure 5.4 and the graph representation is shown in figure 5.3. When line AB is changed in length the other lines BC and DE also changes the length as shown in figure 5.5. The change in length and position of the line DE to a new position makes the line BC also to move to a new position and change its length to maintain the consistency between the constraints as shown in figure 5.6. The geometric model and the constraints imposed on the model fall into the Level 1 category based on the constraint model.

Consider the geometric model shown in figure 5.7. The following constraints are imposed on this model.

1. The length of the lines AB, BC, CD, DE, EF, FG, GH and AH should remain fixed. This is implemented by the *lineLength constant* constraint.

2. The following pair of lines are grouped,

   - AB and BC
   - CD and DE
   - EF and FG.

   The angle when changed on any of these pair of lines should be replicated by the other two pairs. This is implemented by the *lineAngle* constraint. The constraint direction in the constraint graph is unidirectional, for example from AB to BC.

When the designer moves the position of line BC as shown in figure 5.9, the angle set by this pair of lines is followed by the other two pair of lines to maintain the consistency between the constraints. The geometric model and the constraints imposed on the model fall into the Level 1 category based on the constraint model. The constraint graph representation for the model in figure 5.7 is shown in figure 5.8.

Consider the four bar problem shown in figure 5.10, where the two points A and B are fixed and points D and C are free to move in two dimensional space. The length of all the four bars is constrained to be constant. When the point C is moved by the designer, the lengths of each of the four bars should remain as constants and the points A and B should remain fixed. This is a non-linear problem of Level 2. Some of the possible solutions to the problem are shown in figure 5.12. This problem uses the Lagrange method of multipliers to minimize the distance between initial guess by the user for the point C and the solution by solving the constraints. The constraint graph representation for the model in figure 5.10 is shown in figure 5.11.

The support for the design activity of the G+ system supports the Level 1 (linear constraint system) and Level 2 (non-linear constraint system). The Level 3 (capture of the design intent) can be partially achieved by abstraction by object-oriented principles (see section 3.5). The functional relationships are represented as constraints. From the functional relationship of the geometric model the design intent can be framed. The G+ system provides the basis for the

Level 3 (expert system). The necessary constraints for the knowledge base of the G+ system can be build from the principles and methods discussed in this thesis.

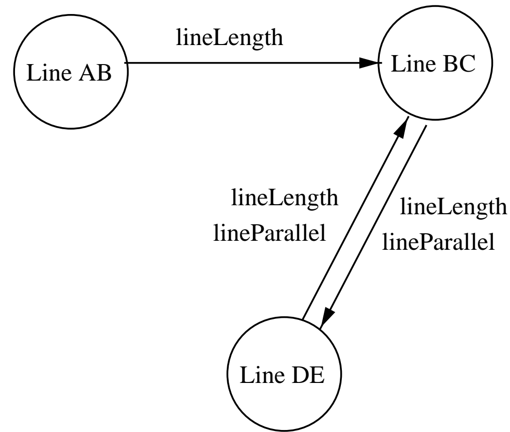

Figure 5.3: Graph representation of three lines model

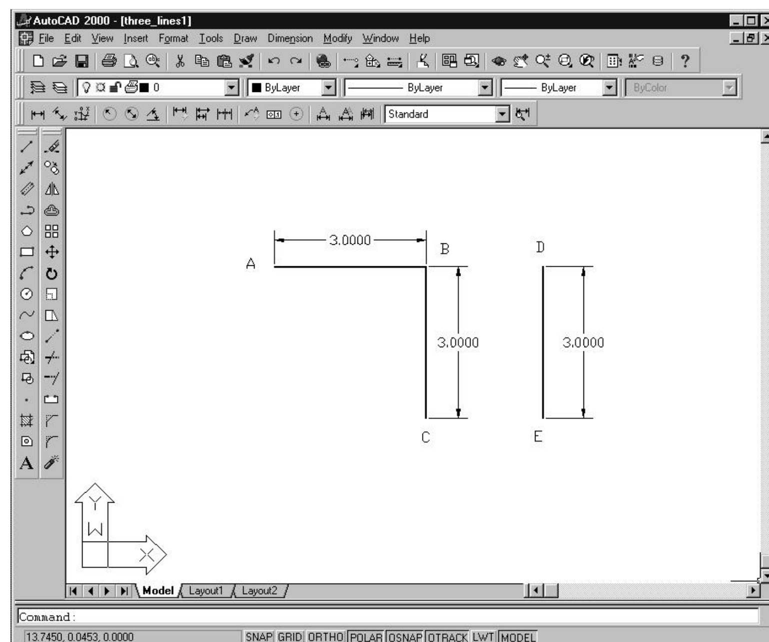

Figure 5.4: Geometric constraint model of three lines

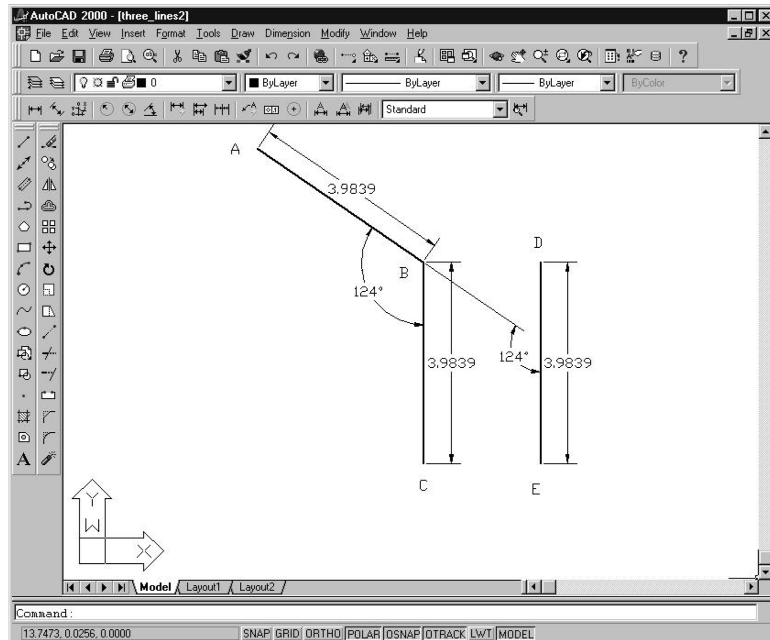

Figure 5.5: Three lines: *lineLength* constraint

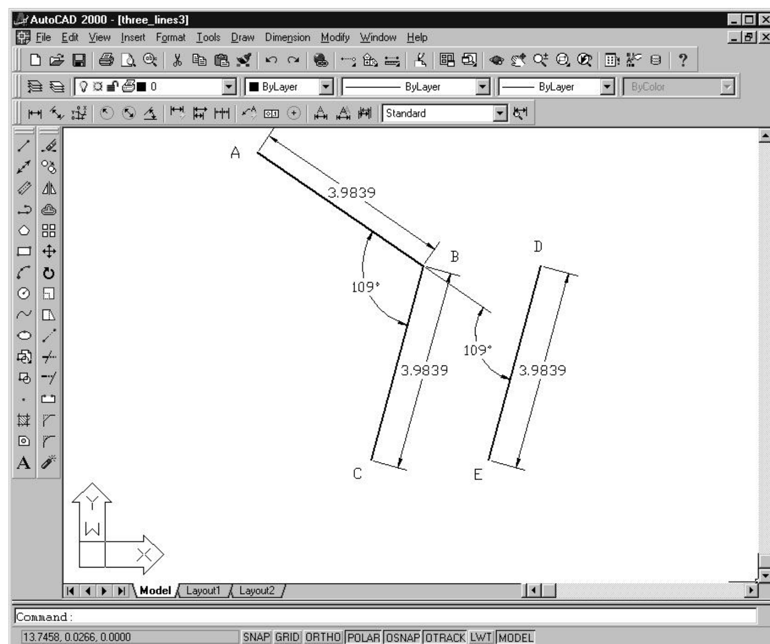

Figure 5.6: Three lines: *lineLength* and *lineParallel* constraints

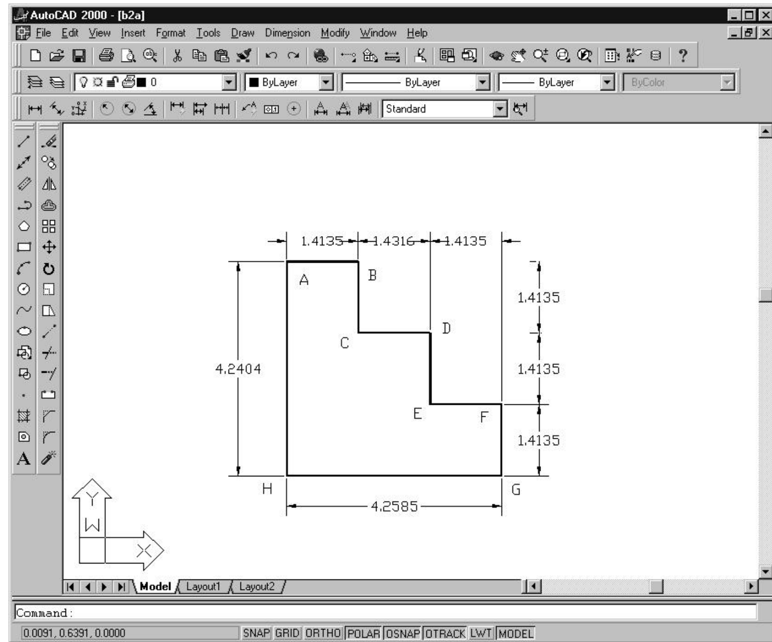

Figure 5.7: Geometric model

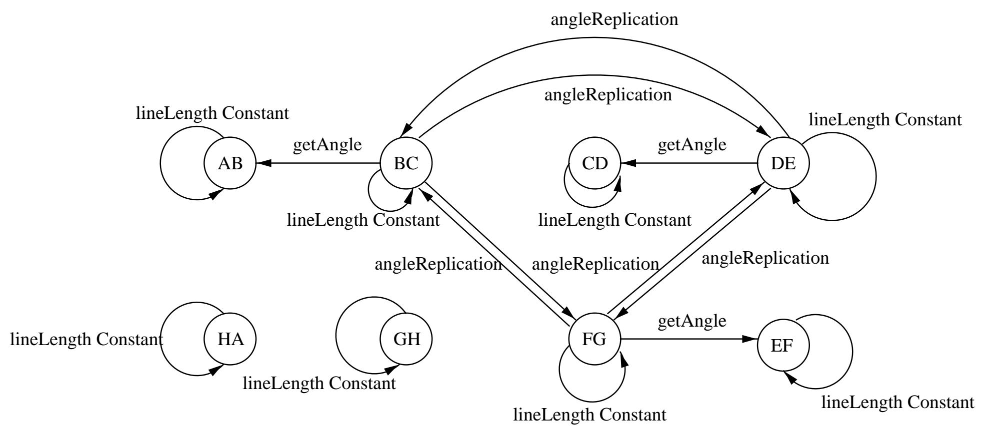

Figure 5.8: Constraint graph of the geometric model

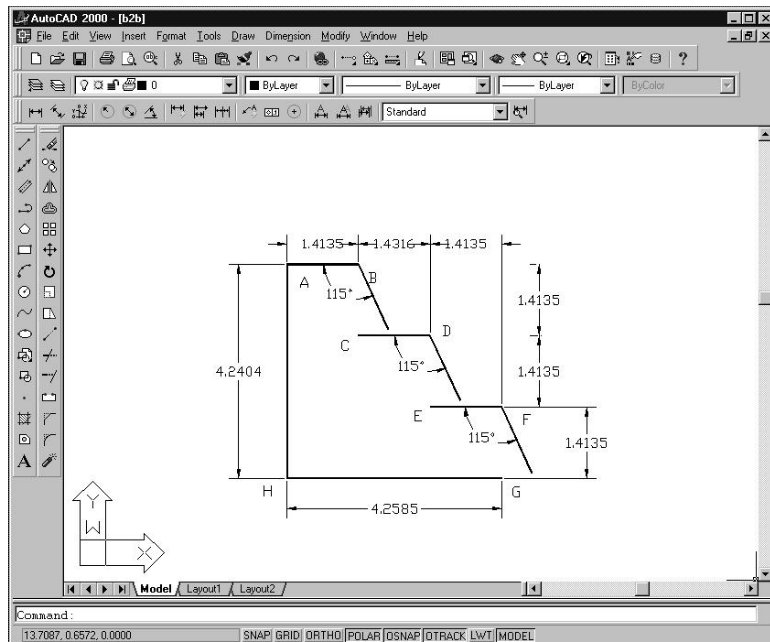

Figure 5.9: *Angle* constraint of the geometric model

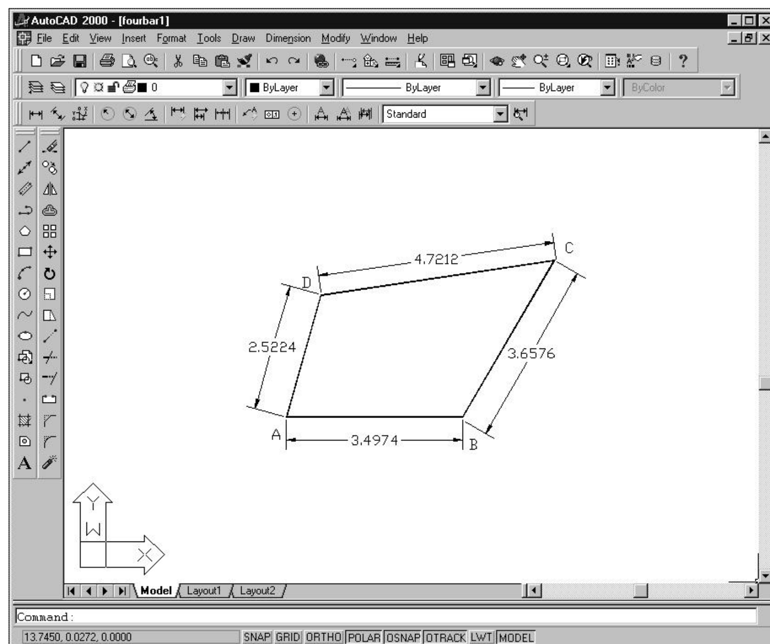

Figure 5.10: Geometric model: Four bar

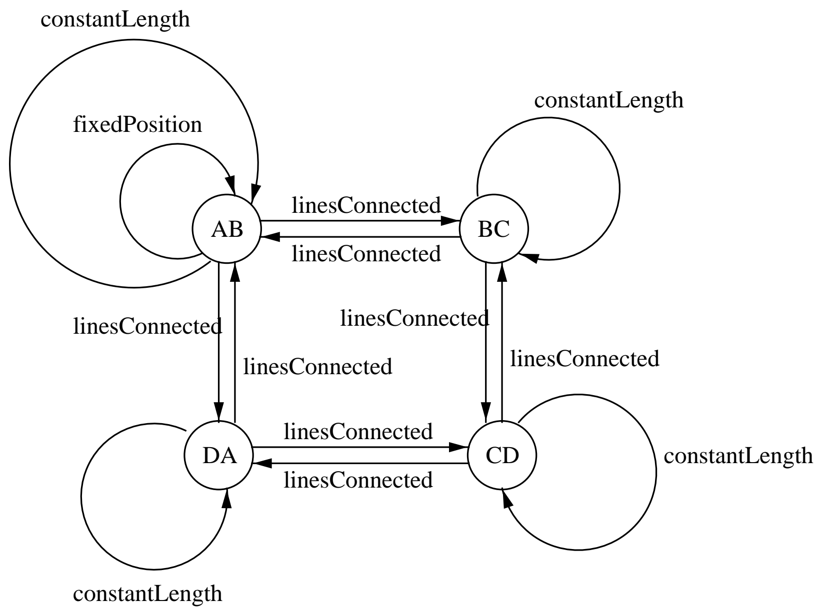

Figure 5.11: Constraint graph representation of four bar

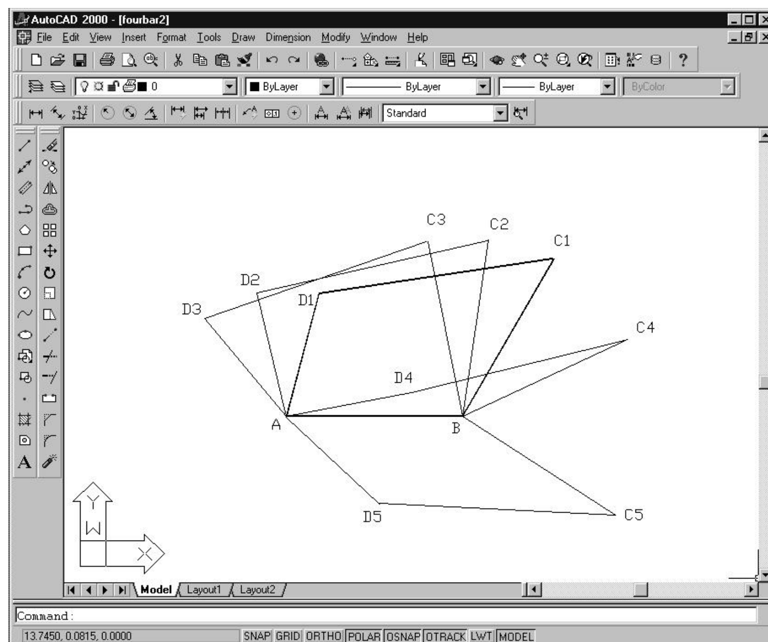

Figure 5.12: Four bar: Some solutions

CHAPTER VI
CONCLUSION

A constraint-based system was consructed, capable of offering support for the design activity of geometric models. The functional relationships of the geometric entities was maintained to obtain consistent solution. Modeling of geometric entities with constraints contains a high potential for the efficient working in all steps of the design cycle. The constraint-based geometric modeling finds applications in all phases of the product life cycle. This may include archiving and reuse of existing design solutions. In general users often have difficulty understanding how constraint-based system works, and therefore how best to utilize it. By having the constraint-based system capable of interfacing with commercial CAD system, which is widely used, and user friendly the designer will be able to use the CAD system with extensive design capabilities. The G+ constraint-based system has been successfully interfaced with AutoCAD 2000 with this intension. The designer may use all the existing geometric entities in AutoCAD 2000 for design activity in the G+ system.

The G+ system improves on general constraint-based systems by providing a specialized set of constraints, simple mechanisms for the designer to add and delete constraints, and intuitive methods for solving the constraints. The G+ system is based on object-oriented constraint programming framework. This enables the G+ system to combine the declarative aspects of constraint programming with abstraction and encapsulation of object-oriented languages, easing the task of incorporating into applications. The solver schemes for the G+ system for solving non-linear constraints have been systematically dealt, and efficient and accurate methods have been incorporated.

6.1 Future Work

6.1.1 Constraint-Based Computational Environment

The G+ constrained-based system is a part of the computational environment as a constraint-based control center. Advanced grid generation concepts may be included into the existing G+ system. These concepts may be introduced into the system as constraints. The object oriented nature of the G+ system enables grid generation software to be integrated into the system. The G+ system ensures the final grid generated to be consistent with the grid generation constraints. The grid generation constraints provide the definition for point placement, high aspect ratio element generation, connectivity issues, and multi-block grids.

The surface definition for the grids can be generated in AutoCAD 2000. The G+ system can generate a geometric solution consistent with geometric constraints imposed on the surface definition. The surface definition of the model and advanced grid generation concepts in the same constraint-based system with a well defined user-interface would enhance the design capabilities of the user.

6.1.2 Multi-User Environment

Designers in remote locations can each set their constraint. These constraints may be classified as *global* and *local* constraints. When a designer in location A sets a *global* constraint on a model the constraint is sent to the global constraint database for the model. Thus the designer in location B when making modifications to the model would not have to be responsible for the constraints set by the designer in location A. Since the G+ system works as objects, these objects can be synchronized between these locations for a multi-user environment.

The *global* constraints set by designers in remote locations would be imposed by the G+ system on the geometric model. The designer may set *local* constraints on the model for testing purposes. The *local* constraints will not be synchronized across remote locations. The multi-user environment of the G+ system would further assist the designer in remote locations with their designs.

6.1.3 Expert System

The G+ system lays the basics for an expert system. Customized constraint database designed for specific classes of geometric problems are used in constraint solving process. The specific classes of geometric problems may include aircraft design, floor plan design, and carburetor design. The constraint database has the definitions for the specific geometric problems. For example in the case of aircraft design, the constraint database can have constraints for wing design, body design, and flutter design. The constraint database has the functionality and specifications for the specific design. The geometric constraint database which has generic design pattern, enhances design capabilities of the G+ system.

REFERENCES

[1] H. Ait-Aoudia, S.and Badis and M. Kara. Solving geometric constraints by a hybrid method. In *Information Visualisation, 2001. Proceedings*, pages 749–753, 2001.

[2] S. Ait-Aoudia, R. Jegou, and D. Michelucci. Reduction of constraint systems. *Compugraphic*, pages 83–92, 1993.

[3] R. Anderl and R. Mendgen. Parametric design and its impact on solid modeling applications. In *Proceedings of the third symposium on Solid modeling and applications*, pages 1–12. ACM Press, 1995.

[4] R. Bartak. Constraint programming: In pursuit of the holy grail. In *Proceedings of the Week of Doctoral Students (WDS99)*, 1999.

[5] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Consistency checking in complex object database schemata with integrity constraints. In *Knowledge and Data Engineering, IEEE Transactions*, volume 10, pages 576–598, 1998.

[6] Alan Borning and Robert Duisberg. Constraint-based tools for building user interfaces. *ACM Transactions on Graphics (TOG)*, 5(4):345–374, 1986.

[7] Alan Borning, Robert Duisberg, Bjorn Freeman-Benson, Axel Kramer, and Michael Woolf. Constraint hierarchies. In *Conference proceedings on Object-oriented programming systems, languages and applications*, pages 48–60. ACM Press, 1987.

[8] A. Bossi, N. Cocco, and S. Dulli. A method for specializing logic programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(2):253–302, 1990.

[9] William Bouma, Xiangping Chen, Ioannis Fudos, Christoph Hoffmann, and Pamela J. Vermeer. *An Electronic Primer on Geometric Constraint Solving*, 1993. http://www.cs.purdue.edu/homes/cmh/electrobook/intro.html.

[10] William Bouma, Ioannis Fudos, Christoph Hoffmann, Jiazhen Cai, and Robert Paige. Geometric constraint solver. *Computer-aided Design*, 27(6):487–501, 1995.

[11] Beat Bruderlin. Constructing three-dimensional geometric objects defined by constraints. In *Proceedings of the 1986 workshop on Interactive 3D graphics*, pages 111–129. ACM Press, 1987.

[12] P. Charman. A constraint-based approach for the generation of floor plans. In *Sixth International Conference on Tools with Artificial Intelligence*, pages 555–561, 1994.

[13] Beman G. Dawes, John Maddock, Jens Maurer, Paul Moore, Mac Murrett, and all. *C++ Boost Libraries Document*, 1998. http://www.boost.org/libs/libraries.htm.

[14] Gene L. Fisher, Dale E. Busse, and David A. Wolber. Adding rule-based reasoning to a demonstrational interface builder. In *Proceedings of the fifth annual ACM symposium on User interface software and technology*, pages 89–97. ACM Press, 1992.

[15] I. Fudos. Editable representations for 2d geometric design. Master's thesis, Purdue University, W. Lafayette, IN., 1993.

[16] Ioannis Fudos and Christoph M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics (TOG)*, 16(2):179–216, 1997.

[17] W. Graf and S. Neurohr. Constraint-based layout in visual program design. In *Proceedings of the 11th International IEEE Symposium on Visual Languages*, 1995.

[18] Patrick M. Hanrahasn. Creating volume models from edge-vertex graphs. In *Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pages 77–84, 1982.

[19] Mark R. Henderson. Representing functionality and design intent in product models. In *Proceedings on the second symposium on Solid modeling and applications*, pages 387–396. ACM Press, 1993.

[20] C. Hsu, G. Alt, Z. Huang, E. Beier, and B. Brderlin. A constraint-based manipulator toolset for editing 3d objects. In *Proceedings of the fourth symposium on Solid modeling and applications*, pages 168–180. ACM Press, 1997.

[21] Scott E. Hudson and Ian Smith. *SubArctic UI Toolkit User's Manual*, 1996. http://www.cc.gatech.edu/gvu/ui/sub_arctic/sub_arctic/doc/users_manual.html.

[22] Scott E. Hudson and Ian Smith. Ultra-lightweight constraints. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 147–155. ACM Press, 1996.

[23] Shvetsov I., Semenov A., and Telerman V. Application of subdefinite models in engineering, 1997.

[24] R. Joan-Arinyo and A. Soto-Riera. Combining constructive and equational geometric constraint-solving techniques. *ACM Transactions on Graphics (TOG)*, 18(1):35–55, 1999.

[25] Timo Laakko and Martti Mntyl. A feature definition language for bridging solids and features. In *Proceedings on the second symposium on Solid modeling and applications*, pages 333–342. ACM Press, 1993.

[26] H. Lamure and D. Michelucci. Solving geometric constraints by homotopy. In *Visualization and Computer Graphics, IEEE Transactions on*, volume 2, pages 28–34, 1996.

[27] Lie-Quan Lee, Jeremy G. Siek, and Andrew Lumsdaine. The generic graph component library. In *Conference on Object-Oriented*, pages 399–414, 1999.

[28] D. Manocha. Solving systems of polynomial equations. In *Computer Graphics and Applications, IEEE*, volume 14, pages 46–55, 1994.

[29] Charles McAuley. *Programming AutoCAD 2000 Using ObjectARX*. Autodesk Press, 2000.

[30] A. McBrien, J. Madden, and N. R. Shadbolt. Artificial intelligence methods in process plant layout. In *Proceedings of the second international conference on Industrial and engineering applications of artificial intelligence and expert systems*, pages 364–373. ACM Press, 1989.

[31] G. Miller and V. Ramachandran. A new graphy triconnectivity algorithm and its parallelization. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 335–344. ACM Press, 1987.

[32] Greg Nelson. Juno, a constraint-based graphics system. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 235–243. ACM Press, 1985.

[33] J. C. Owen. Algebraic solution for geometry from dimensional constraints. In *Proceedings of the first symposium on Solid modeling foundations and CAD/CAM applications*, pages 397–407. ACM Press, 1991.

[34] Charles E. Pfefferkorn. A heuristic problem solving design system for equipment or furniture layouts. *Communications of the ACM*, 18(5):286–297, 1975.

[35] M. Rosendahl and R. Berling. Modelling of geometric constraints in cad-applications. *Geometric Constraint Solving and Applications*, pages 151–169, 1998.

[36] Jaroslaw R. Rossignac. Constraints in constructive solid geometry. In *Proceedings of the 1986 workshop on Interactive 3D graphics*, pages 93–110. ACM Press, 1987.

[37] Chia-Hui Shih and Bill Anderson. A design/constraint model to capture design intent. In *Proceedings of the fourth symposium on Solid modeling and applications*, pages 255–264. ACM Press, 1997.

[38] A. Sourin and A. Pasko. Function representation for sweeping by a moving solid. In *Proceedings of the third symposium on Solid modeling and applications*, pages 383–391. ACM Press, 1995.

[39] I. E. Sutherland. Sketchpad a man-machine graphical communication system. In *Papers on Twenty-five years of electronic design automation*, pages 507–524. ACM Press, 1988.

[40] Zahir Tari, John Stokes, and Stefano Spaccapietra. Object normal forms and dependency constraints for object-oriented schemata. *ACM Transactions on Database Systems (TODS)*, 22(4):513–569, 1997.

[41] Sashikumar Venkataraman, Milind Sohoni, and Vinay Kulkarni. A graph-based framework for feature recognition. In *Proceedings sixth ACM symposium on Solid modeling and applications*, pages 194–205. ACM Press, 2001.

[42] Jan Wolter and Periannan Chandrasekaran. A concept for a constraint-based representation of functional and geometric design knowledge. In *Proceedings of the first symposium on Solid modeling foundations and CAD/CAM applications*, pages 409–418. ACM Press, 1991.

[43] Ming-Hsuan Yang. Recognizing machining features through partial constraint satisfaction. In *IEEE International Conference on Systems, Man, and Cybernetics, 'Computational Cybernetics and Simulation'*, pages 1912–1917, 1997.

[44] B. Zalik, N. Guid, and A. Vese. Representing geometric objects using constraint description graphs. In *Proceedings of the 5th International Conference IEA/AIE-92*, pages 505–514, 1992.