USING 3D GEOMETRIC CONSTRAINTS IN ARCHITECTURAL DESIGN SUPPORT SYSTEMS

B. de Vries [a], A.J. Jessurun [a], R.H.M.C. Kelleners [b]

[a] Department of Architecture, Building and Planning, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

[b] VDO Car Communication, Navigation Lab. Eindhoven, Glaslaan 2, 5616 LW Eindhoven, The Netherlands

ABSTRACT

To support 3D architectural modeling, geometric constraints are introduced. Explicit and implicit geometric relations between building elements can be expressed by the designer and imposed on the 3D geometry of the design. A complete set of constraint types is defined. Constraints are satisfied through Numerical Methods to achieve that the model responds to the designers' actions in a user-intuitive way. Two prototype systems demonstrate the application of geometric constraints in the architectural domain.

Keywords: CAD, Geometric constraints, Design systems

1 INTRODUCTION

Most traditional CAD systems developed from 2D drawing systems to 3D drawing systems, lacking appropriate tools for 3D design. One would expect a 3D architectural design system to have a 3D-design environment with knowledge about the designed entities. Such a 3D-design environment supports the creation and manipulation of 3D shapes. The design environment should essentially be 3D as opposite to the well-known Windows environment with its menu-structured interface. Navigating and manipulating in 3D requires 3D geometrical primitives, but also a set of 3D design tools. It seems obvious that these tools differ from traditional CAD aids such as grid and snap.

This article introduces geometric constraints as a solution for adding geometric design knowledge to an architectural design support system*. The kind of geometric design knowledge that is required is directly deduced from the design operations that are performed. The design operations are limited to conceptual design. Geometric constraints can be expressed as part of the design model on a semantically high level. The concept of geometric constraints has been tested in two prototype design systems. The theoretical background for constraint management as well as the application in architectural design is reported in this article.

The outline of the article is as follows. In the next section geometric constraints are introduced and elaborated. In the third section the constraint solving process is considered. In the fourth section two prototype systems are presented that demonstrate the possible use of geometric constraints. In the final section some conclusions are draw from the research and proposals are made for future research.

2 GEOMETRIC CONSTRAINTS

In general, geometric relationships can be expressed using some formal specification languages like Express [ISO/TC184], UML [Odell98] or the so-called Feature-Type definition language [Leeuwen99]. In architectural design user requirements are often described as functional relations (e.g. the wall must bear the floor). Many functional relations can be expressed as geometric relations (e.g. bottom of the floor should coincide with top of the wall).

* The research presented on geometric modeling is executed within the framework of the development of a Virtual Reality – Design Information System [Vries97]

In this article we argue that if we can formalize geometric relationships of building elements then behavior of the design system is more natural. Natural behavior in this context means that the model responds to the designer's expectations. Moreover, geometric constraints should be accessible using a vocabulary that is close to the designers' natural language.
In the following we will develop formally a system for expressing and solving geometric constraints.

2.1 Interval constraints

The formal geometric specifications we developed are based on Allen's temporal interval algebra [Allen93]. Allen recognizes five basic relationships: ahead, front-touch, in, back-touch and behind. Assuming that a geometry can be projected on each of the x-, y- and z-axis, the projection is called an interval. From that thirteen interval-interval relations can be derived: x before y, x meets y, x overlaps y, x finished-by y, x contains y, x started-by y, x equals y, x starts y, x contained-in y, x finishes y, x overlap-by y, x met-by y and x after y. At this point we assume that a building element is represented by a bounding box (see Figure 1). The consequences of this assumption will be discussed in the last section.

As a result the spatial relationship of two building elements can be expressed as interval-interval relations. In turn, these relations can be expressed as numerical equations.
To introduce the use of interval constraints on bounding boxes an example will be worked out in 2D to express that two bounding boxes B1 and B2 should meet each other.

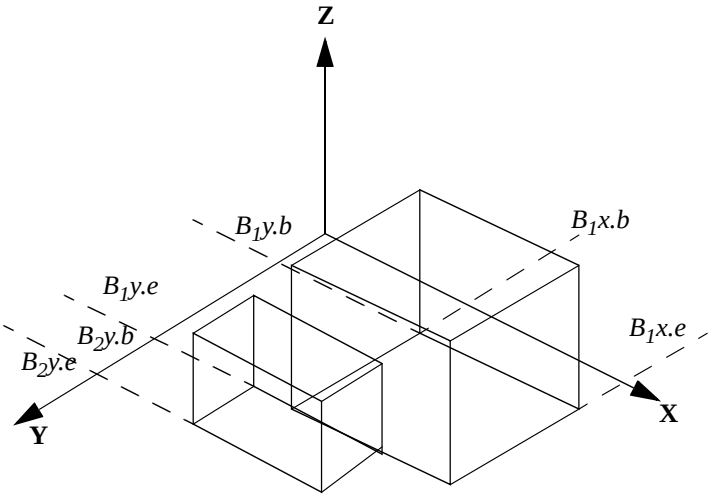


Figure 1: Interval constraints

Specification of the size of box B1 along the x-axis (see Figure 1):

$$B_1x.e > B_1x.b$$
$$B_1x.e - B_1x.b = 3$$

Similarly the size is specified along the y- and z-axis. Here, $B_1x.b$ denotes the beginning of the interval of box B1 along the x-axis. The length of the box is fixed at 3 units.

Specifying that box B1 meets box B2:

$$B_1y.e > B_1y.b$$
$$B_2y.e > B_2y.b$$
$$B_1y.e = B_2y.b$$

In general, the geometric constraints can be depicted in the equations:

$$\mathbf{A_1\, v = c_1}$$
$$\mathbf{A_2\, v > c_2}$$
$$\mathbf{v_{xi} > 0,\ v_{yi} > 0,\ v_{zi} > 0}$$

$\mathbf{A_1}$ and $\mathbf{A_2}$ are matrices of size $m_1 \times n$, and $m_2 \times n$, where m_1 equals the number of equalities and m_2 equals the number of inequalities. Vector $\mathbf{v}$ contains all x-, y-, z-coordinates of the intervals of box i, $i = 1..n$. $\mathbf{c_1}$ and $\mathbf{c_2}$ are the right hand sides written as a column vector having size m_1 respectively m_2. As precondition all coordinates are assumed to be non-negative.

2.2 Constraint types

The following geometric constraint types can be deduced from the spatial constraint and the access

constraint of the case study:

- Unary constraints to limit the shape of a box, in particular, the position and the dimensions of a box: FixPosition, Dimension.
- Binary constraints to specify relations between two boxes, in particular, connections, distances and intersection relations: Touch, Align, Distance, LatDistance, Contains, NonIntersect.

A summary adapted from [Kelleners99] of the constraints types is listed below:

`FixPosition(BB b, Point p)`

The constraint dictates that the center point of bounding box b must be equal to the specified point p.

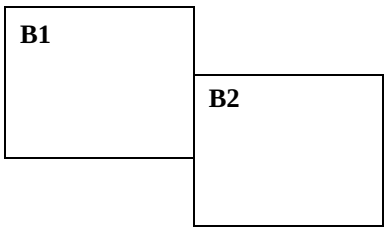
`{Min/Max/Fix}Dimension(BB b, LocAxis ax, float x)`

The constraint MinDimension dictates that dimension ax of bounding box b must be at least value x. The constraint MaxDimension dictates that dimension ax must be at most value x. The constraint FixDimension dictates that dimension ax must be equal to value x.

`NonIntersect`

This constraint dictates that the two boxes b1 and b2 may not intersect

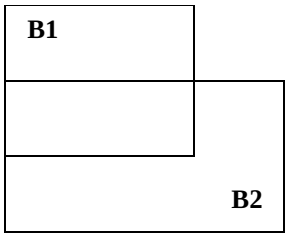
`Touch(BB b1, BB b2, Axis a)`



The constraint dictates that the farther side of the box b1 (seen from the origin) must touch the closer side of box b2 on axis a.

Figure 2: Touch constraint

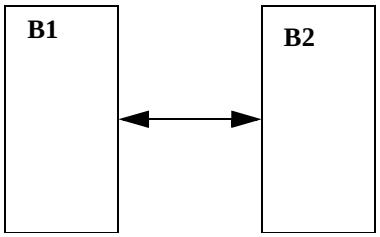
`Align(BB b1, BB b2, Axis a, int n)`



The constraint dictates the operand b1 should be aligned with operand b2 on axis a. The integer n indicates that the closer sides of the boxes (seen from the origin) must be aligned or the farther sides must be aligned.

Figure 3: Align constraint

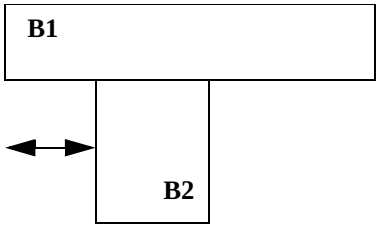
`{Min/Max}OppDistance(BB b1, BB b2, Axis a, float d)`



The constraint dictates the operand b1 should be aligned with operand b2 on axis a. The integer n indicates that the closer sides of the boxes (seen from the origin) must be aligned or the farther sides must be aligned.

Figure 4: Align constraint

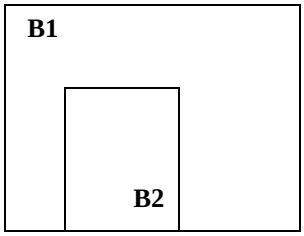
`{Min/Max}LatDistance(BB b1, BB b2, Axis a, float d)`



The constraint dictates that operand b1 should have lateral distance d to operand b2. The sides of the boxes that are the closest to the origin on axis a have distance d, moreover b1 is closer to the origin than b2. Min and Max variants can be used to specify minimum and maximum distances.

Figure 5: LatDistance constraint

```
Contains(BB b1, BB b2)
```



This constraint dictates that box b1 contains box b2.

Figure 6: Contain constraint

3 CONSTRAINT SATISFACTION

First we tried to solve the set of equations using Linear Programming [7] techniques. Looking at the example of section Interval constraints first a set of so-called auxiliary variables was introduced such that the example could be rewritten into a form containing only equalities:

$$B_1x.b - B_1x.e + aux_1 = 0$$
$$B_1x.e - B_1x.b = 3$$
$$B_1y.b - B_1y.e + aux_2 = 0$$
$$B_2y.b - B_2y.e + aux_3 = 0$$
$$B_1y.e = B_2y.b$$

The objective function in this case was:

$$minimize\ f(x) = B_1x.b + B_1x.e + B_1y.b + B_1y.e + B_1z.b + B_1z.e + B_2x.b + B_2x.e + B_2y.b + B_2y.e + B_2z.b + B_2z.e$$

Obviously to find a solution for the example the intervals must be specified in x-, y- and z-direction ($m_1 = 3$) for the two boxes ($m_2 = 2$), the size ($m_3 = 2$), and the meeting conditions ($m_4 = 3$), resulting in $3 \times 2 \times 2 + 3$ equations. Likewise, the objective function consists of $3 \times 2 \times 3$ variables.

In general Linear Programming is concerned with the problem of optimizing an objective function that is subject to the primary constraints. A well-known method that suits this purpose is the Simplex [Press92] method. The Simplex method may result in the following states:

- No solution is found.
  In this case the model is *over-constrained*. This means that all possible solutions exclude each other.
- Exactly one solution is found.
  In this case the objective function reaches a minimum at exactly one set of values that satisfies the constraints, the so-called *optimal feasible vector.*
- An infinite set of solutions is found.
  In this case the model is *under-constrained*. This means that the objective function reaches its minimum for an infinite set of feasible vectors.

When no solution is found, it is rather difficult to determine which constraint(s) should be relaxed to find at least one solution. A set of solutions could be interesting if the number in the set is limited and if the designer is allowed to browse through the set. Limiting the set could be accomplished by searching only for solutions that are on the intersections of the contours of the solution space. Still it is hard to explain the meaning of these feasible solutions in an architectural design context. By experimenting with the Simplex method we found that the solutions did not relate at all to the designers intentions and expectations. The solution largely depended on the initial situation. Moreover, a minor change may lead to a completely different solution. This new solution, though correct with respect to the constraints, was hard for the designer to understand since it deviates greatly from the previous situation.

Our conclusion was that we had to look for a solution method that searches for a solution that is as close as possible to the former situation (see also [Gleicher94]).
If the system of interval constraints that is rewritten to a set of linear equations and inequalities is represented as follows (compare with section Interval constraints):

$$\mathbf{A_1\ x = b_1}$$
$$\mathbf{A_2\ x > b_2}$$

then this can be achieved by minimizing the change of each variable of vector x. It leads the following objective function:

$$\mathbf{minimize\ f(x) = \tfrac{1}{2} \| x - x_c \|^2}$$

dictating that the 'distance' between the new solution x and the old solution x_c should be as little as possible.

The equality constraints of A_1 were solved using the conjugate gradient method (see [Press92]), which minimizes the function:

$$\mathbf{f(x) = \tfrac{1}{2}\ x\ A_1\ x - b\ x}$$

where A_1 is symmetric and positive definite.
This function is minimized with its gradient:

$$\mathbf{\nabla f = A_1\ x - b}$$

The following heuristic was applied for the inequality constraints of A_2. A set of inequalities constraints

is treated as active and there is a set that is treated as inactive. Active constraints of A_2 are added to the matrix A_1. The inactive constraints are ignored as far as solving is concerned. The set of active and inactive constraints is updated as the conjugate gradient method is iterating towards a minimum. Which inequalities should be active and which inactive is determined while checking their validity.

Again, the three solution situations may occur, starting from some constellation of the boxes in space:

- No solution is found.
  In this case the 'average' position is reached since minimizing a quadratic objective function will always iterate to some value. Thus, despite the fact that theoretically there is no solution, the 'average' solution is displayed. The 'average' solution appears to be very close the users expectations.
  This situation is illustrated by two boxes B1 and B2 that are placed on a plane within a certain distance of each other and that have two parallel sides S1 and S2. If the position and all dimensions of both B1 and B2 are constrained and moreover a Touch constraint is added to S1 and S2, then there is no solution. In this situation the Touch constraint will be violated, consequently B1 and B2 stay in place. The explanation for this outcome is that the constraint solving mechanism will stop at that point where constraint violations and changes of positions and dimensions are minimal.
- Exactly one solution is found.
  This is rarely the case since it means that the designer has added the number of constraints that is exactly equal to the number required to find one solution.
- Infinite set of solutions is found.
  In this situation the closest solution on the edge of the solution space starting from the former situation is reached. This system behavior appears to be very intuitive.
  This situation is illustrated by two boxes B1 and B2 that are placed on a plane within a certain distance of each other and that have two parallel sides S1 and S2. If only a Touch constraint is added to S1 and S2, then the post condition is that S1 and S2 should meet and that B1 and B2 can be located anywhere on the plane. This situation is resolved by moving B1 and B2 over an equal minimal distance to achieve that S1 and S2 meet.

4 APPLICATION IN ARCHITECTURAL DESIGN

4.1 Design constraints

We learned from the case study [Achten98] that designers and users express their intent on a higher level of abstraction than geometrical constraints. That is, they use explicit and implicit relationships between building elements. Therefore, we defined a set of so-called design constraints at a semantically higher level using the geometrical constraints:

1 Building_Element_A *adjacent* to Building_Element_B
  This relation will be implemented using the Touch constraint. The axis along which the building elements must touch is determined from their current position.
2 Building_Element_A *in* Building_Element_B
  This relation is implemented using the Contain constraint.
3 Building_Element_A *above* Building_Element_B
  This relation is implemented using the Touch constraint. The axis along which the building elements touch is always the Z axis.
4 Building_Element_A *aligns* Building Element_B
  This relation is implemented using the Align constraint and the LatDistance constraint. The sides of the building elements that need to be aligned must be specified respectively.

Two prototype applications have been developed using these design constraints, namely one to support space studies in 3D and one to investigate the behavior of geometrical constrained building elements. In both prototypes, constraints are imposed and revoked by adding or deleting the appropriate commands using file input or by user input.

4.2 Space study prototype

The space study prototype was developed to examine whether spatial layout and mass study in 3D can be supported by design constraints. Note that the prototype system does not contain an optimization algorithm for spatial layout as in [Medjdoub98]. The system serves as a method for positioning and

dimensioning spaces in a 3D environment.

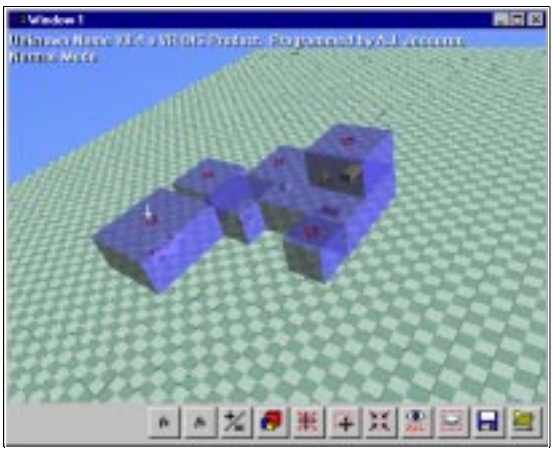


Figure 7: Space study prototype

Only the *adjacent* and the *above* constraint are used in this prototype. Spaces are presented in 3D by transparent boxes. To obtain visual input about the activity that will take place in a space, an icon representing that activity is located in the center of the transparent space (e.g. a sofa). In our experience, the best way to work with the system is to first specify the constraints that directly follow from the functional brief of the building (e.g. the garage must be next to the kitchen). The user of the system manually triggers the constraint solving process. Next, additional constraints are entered by the designer whenever he/she feels that a specific ordering of spaces should be kept intact during the design session. Spaces can be pinned to a specific position (i.e. activating the FixPosition constraint) and dimensions can be fixed (i.e. activating the Dimensions constraint) using a toggle switch. The designer can add and relax these constraints to obtain a situation he/she is satisfied with.

4.3 Building element behavior prototype

The building element behavior prototype was developed to examine whether a building concept consisting of building elements can behave intuitively when the positions an dimensions of each building element are modified individually.

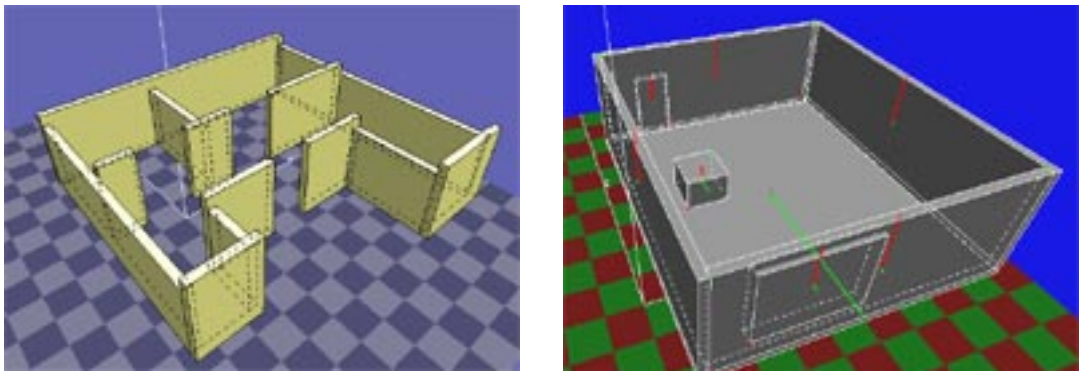


Figure 8: Building element behaviour prototype

In this prototype all types of design constraints are used. The designer can dynamically stretch the room while all walls stay connected. The floor will resize accordingly. Moreover, the size and location of the window in the wall can be changed. It will stay within a minimum distance from the end of the wall and it cannot be dragged outside of the wall. Inside the room a box representing furniture can be moved around the floor without getting detached from the floor. Thus, new solutions that obey the geometric constraints are calculated while changing the geometry by pushing and pulling. Since this all happens in a 3D environment the user can walk around the model or go inside the room to get a better view.

5 CONCLUSIONS AND VENUES FOR FUTURE RESEARCH

This paper was to present the results of a research project that aimed the use of geometric constraints for support of architectural design. A set of constraint types was defined to express explicit and implicit geometric relations between building elements. A constraint satisfaction technique based on Linear Programming was introduced to achieve that the design support system responds in accordance with the users' expectations. To examine and demonstrate the application of geometric constraints in a 3D environment two prototype systems were developed. Simple experiments with the prototype systems indicate that the proposed design constraints serve the design operations as expected.

Geometric constraints are a powerful concept for capturing specific design knowledge. Still several limitations of the current prototype systems have to be overcome and some fundamental issues need to be solved.

In the described geometric constraint solving mechanism the bounding box plays a central role. Although many building elements fit a bounding box quite well (e.g. wall, floor), it imposes limitations on architectural design, such as:

- Walls, floors, etc. can only meet perpendicular.
- Curved walls, roofs etc. cannot be represented adequately.
- Caved structures cause modeling problems (e.g. a chair cannot be moved under a table).

Thus, bounding boxes suit well for an important category of building elements, but for non-rectangular shapes, a different strategy must be developed.

The graphical user interface of the prototype system does not allow yet the direct creation and manipulation of design constraints. A graphical representation of constraints gives the user better control of the design and its behavior. Widgets should allow the user to interactively attach and detach constraints appropriate for the building element at hand.
The performance of the prototype systems decreases rapidly with an increasing number of constraints. The complete set of equations is currently solved when a system event that signals a change in the position of one of the bounding boxes is raised. The constraint solving algorithm could be optimized by recalculating only local changes.

A more fundamental question is which design knowledge should be represented by constraints. In general static and procedural knowledge can be described by any formalism such as Object Oriented Modeling, Rule Based Design, etc. In the VR-DIS program, for the time being, we have chosen to rely on the expressive power of Feature-Based Modeling [Leeuwen99]. If FBM does not suffice we will use constraints. This rather arbitrary choice was made because computing the Feature Model takes less computational power than dynamically calculating constraints. As with a natural language, there are many ways to express a design, but all with a (slightly) different meaning. In due time description styles will emerge supporting the designers' intent.

Implementing a design system prototype immediately raises questions about user access to geometry data. There seems to be a trade off between having a complete and consistent set of design data and allowing a designer to freely edit geometry. The underlying question however, is how user interaction with the design at the building element description level should be communicated with the drawing primitives level.

We hope to report on the solution to these challenges in future publications.

6 ACKNOWLEDGEMENTS

This research project is part of the VR-DIS research program of the Design Systems Group. We would like to thank the Computer Graphics Group of Computing Science Department of the Eindhoven University of Technology for their collaboration.

7 REFERENCES

[Achten98] Achten, H.H., and J.P. van Leeuwen, *A Feature-Based Technique for Design Processes: A Case Study,* Proceedings of the 4th Conference on Design and Decision Support Systems in Architecture and Urban Planning, Maastricht, The Netherlands, 1998.

[Allen93] Allen, F., *Maintaining knowledge about temporal intervals*, Commun. ACM, 26(11) pp. 832-843, 1993.

[Gleiger94] Gleicher, M., and A. Witkin, *Drawing with constraints*, The Visual Computer, 11(1), pp. 39-51, 1994.

[ISO/TC184] ISO/TC184, Description methods: the EXPRESS language reference manual, ISO TC184/SC4 10303 part 11, International Organization for Standardization, Geneva, Switzerland, 1992.

[Kelleners99] Kelleners, R.H.M.C., *Constraints in Object-Oriented Graphics*, Thesis Eindhoven University of Technology, Eindhoven, The Netherlands, 1999.

[Leeuwen99] Leeuwen, J.P., *Modelling Architectural*

Design Information by Features, Thesis Eindhoven University of Technology, Eindhoven, The Netherlands, 1999.

[Medjdoub98] Medjdoub, B., and B. Yannou, *Topological Enumeration Heuristics in Constraint-Based Space Layout Planning*, in: Proceedings of Artificial Intelligence in Design '98, Lisbon, Portugal, 1998.

[Odell98] Odell, J.J., *Advanced object-oriented analysis and design using UML*, Cambridge University Press, Cambridge, 1998.

[Press92] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, *Numerical recipes in C: the art of scientific computing*, 2nd ed., Cambridge University Press, Cambridge, 1992.

[Vries97] Vries, B., de, *VR-DIS Research program*, Internal report Faculty of Architecture and Building, Eindhoven University of Technology, Eindhoven, The Netherlands, 1997.