"Hyperwalls" or an Application of a non Deterministic Rule Based System in Interactive Architectural Modelling

Vasileios Gougoulidis

The Ohio State University
USA

This paper presents the architectural modeling as a process of augmenting spatial information; a chain of actions that leads from a sketched idea to the elaborated model. A symbolic constraint solver tool is connected to traditional CAD techniques, as well as to a data representation scheme efficient for architectural elements. The orchestration of the available and added tools allows the designer to 'edit ideas' fast, keeping in mind that different design profiles require adaptive tools to support the varying methodologies. Until the moment that automated design will be both possible and desirable, machines can really shorten the time needed to visualize design ideas in the sense of a handy but non-decisive 'calculator'. The discussion is built around illustrated examples from the implemented constraint based modeler.

Keywords: non-deterministic, rule-based system, architectural modelling,

1 Introduction

Although fascinating ideas about how a design system should be formatted appeared as early as in seventies [1], the critical barrier that prohibited the envisions of design automation from becoming reality has been the "ill defined" nature of the design decision making process [2]. By focusing on the machine perception rather than the decision making factor, it is still possible to establish the designer-machine interaction at a higher level without seizing any of the decision making privileges from the designer.

This paper discusses through examples an architectural modeling system, which integrates traditional geometric modeling techniques, inherent architectural representations [3], and rule based geometric reasoning [4], to provide the designer with such an interactive environment. The hypothesis tested in this study has been that symbolic computation techniques can be effectively applied in an architectural modeling system to reduce the bulk of editing by using a reasoning mechanism capable of capturing the user's intention and compiling it into sequences of well defined modeling system actions.

This mechanism cooperates with graphical interfaces and relies on efficient representation schemes of architectural elements to increase efficiency in the modeling process. The prototypical system used for the examples has four conceptual layers of work. At the very inner system level a lisp interpreter has been implemented [5] which was chosen as the medium to represent and manipulate in symbolic form the modeling structures, the inference mechanism, the rules controlling the system's behavior and the constraints database. At a second level a solid modeler has been built that relies internally on the lisp forms to store boundary representations of solids, along with viewing information and non-geometric attributes. Controlling the modeler is the output of a reasoning machine[6] that takes as input a set of rules and a set of constraints in symbolic form. Finally the constraint editing of this mechanism has been interfaced to the designer using the visual semantics of drafting dimensioning.

First the major components that comprise the architectural modeler are individually presented and then an example of integrated usage of the above components is given.

2 Sketches and offsets

One important component of the three dimensional sketching process, as described here, is the virtual space cursor. This locator driver was developed to support three dimensional input and editing of joints inside a two dimensional viewport (i.e. window). This is accomplished by inverting the standard viewing models [7] mapping, under certain assumptions, and attaching the result to the mouse tracking mechanism, allowing virtual 3d interaction. The joints are defined as sets of 2d or 3d vectors and most of the time they are the result of some action with the locator device. Joints are the elements used to define shapes, and they carry at least geometric information. Shapes compose sketches and they are collections of curves. In our case these curves are the axial curves of walls and they are not necessarily linear. A skeletal model built after a few clicks is shown in figure 1.

Figure 1. Three dimensional joints

Using the axial curves and other information saved at the joints, like local offsets, we can derive offset curves, or more precisely speaking:

Let $e1$, $e2$, and $e3$ be the standard orthonormal basis of R3 and $\gamma:(0,1) \to P$, where P is the plane

generated by $e1$, $e2$ for any t in $(0,1)$

Define $T(t) = \lim_{h \to 0} \frac{\gamma(t+h) - \gamma(t)}{h}$ the tangent vector at $\gamma(t)$

Let $N(t)$ be the unit normal vector at the point $\gamma(t)$ such that $\{N(t) \times T(t)\} \cdot e3 \geq 0$ for every t.

For every λ in R we define the offset curve $\gamma_\lambda(t) = \gamma\{t + \lambda \cdot N(t)\}$

Figure 2. Offset curves

Extruding the offset curves generates ruled surfaces which are the walls. A quite extreme case of such an approach is shown in figure 3, and demonstrates variation of all the available parameters, even of the height as input from the virtual space cursor. A solid wall which is derived from a cubic degree curve in the floor plan, starts as tall and thin, and ends up as low and wide. The solid is constructed based on the skeletal information of the 3d joints as shown in figure 1.

3 Constraints

As already mentioned a sketch is a set of joints and a set of curves connecting the joints. The joints, among other information, carry the primary geometric description of the model. Axial curves that connect the joints and offset curves used to derive the wall boundaries can be placed as soon as the location (and orientation) of the Joints is established. The geometry of the joints is calculated by the constraint solver, which is a vital part of the modelIs construction line because it makes possible to edit graphic relationships and parameters of the sketched idea a posteriori[8]. The sketched idea can be modified by altering the geometric constraints imposed on the joints. In the case that the sketched shape carries additional attributes, everything including curve intersections and offsets is going to be re-evaluated up to the final model. Geometric constraints have a visual representation for the user interface and an internal syntactic representation used by the

constraint solver. The table in figure 4 shows an entry for every used constraint along with its visual representation and internal syntax. Internally constraints are represented as lists of expressions as shown under the syntax column of the table. The first item of the list is a label identifying the type of the constraint, followed by a sublist of the indices to the joints involved, and if applicable by a lisp form, that evaluates to a quantity relevant to the constraint. In the process of constraint solving these lists of expressions are considered to be constant facts, even though they may contain lisp variables.

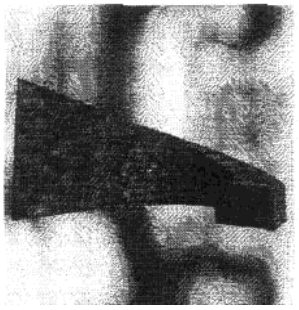

Figure 3. Example wall

There are three levels of constraints classified by the type of information they carry:

- positioning constraints, like point, slope. These are the basic constraints. The constraint solver will attempt to substitute any other category of constraints with this one.

- dimensional constraints, like distance, angle, and collinear. These constraints represent the actual geometric information as entered and recognized by the user, however they have to be eventually rewritten in terms of positioning constraints.

- relational constraints, like perpendicular lines, parallel lines, equal points, equal distances, equal slopes, and equal angles. This type of constraints associates two dimensional or positioning constraints to each other, without carrying any specific value. They are usually rewritten in terms of an existing and a new dimensional or positioning constraint. The constraints reflect design decisions, restrictions, or just style preferences and they accumulate while sketching or revising a project in an unordered fashion.

Name	Description	Representation	Syntax
point	fixed point		(point A vector)
slope	fixed slope		(slope A B value)
distance	distance between points		(distance A B value)
angle	minimum angle formed by three points		(angle A B C value)
collinear	point lying on a line segment	N/A	(collinear A B C offset)
door	door opening		(door A B C offset)
window	window opening		(window A B C offset)
equal-point	equal points	N/A	(equal-point A B)
equal-slope	equal slopes	N/A	(equal-slope A1 B1 A2 B2)
equal-distance	equal distances	N/A	(equal-distance A1 B1 A2 B2)
equal-angle	equal angles	N/A	(equal-angle A1 B1 C1 A2 B2 C2)
parallel	parallel line segments		(parallel A1 B1 A2 B2)
perpendicular	perpendicular line segments		(perpendicular A B C D)

Figure s. me constraints, their visual representation, and their internal syntax

It is the constraint solver that is responsible to build the structure out of the information mass available, to check for consistency, and to identify redundant data. To do so, the constraint solver uses some structured patterns: the rules. Rules comprise of two parts: a condition and a consequence. The condition contains a pattern of goals. The only syntactic difference between a goal and a constraint is that the goal may contain matching variables, while the constraint contains only constants. A rule may contain variables in its consequence part too, but the same variables always have consistent values within the scope of a rule. Rules are applied in the order they were defined, therefore if some rules should be tested earlier than others then they should be defined first. Following is a sample

of implemented rules. The part on the left of the arrow is the condition of the rule, while the part on the right is the consequence. The rules appear in a simplified form which does not include the values involved in the geometric calculations of each substitution. Capitalized letters represent the matching variables.

```
Point(a), equal-point(a b) -->point(a), point(b)
Distance(a1 b1), equal-distance(a1 b1 a2 b2) -->distance(a1 b1), distance(a2 b2)
Angle(a1 b1 c1), equal-angle(a1 b1 c1 a2 b2 c2) --> angle(a1 b1 c1), angle(a2 b2
c2)
Slope(a1 b1), equal-slope(a1 b1 a2 b2) -->slope(a1 b1), slope(a2 b2)
Point(a), point(c), collinear(a b c) -->point(a), point(b), point(c)
Point(a), point(b), distance(a c), distance(b c) -->point(a), point(b), point(c)
Point(a), point(b), distance(a c), angle(c a b) -->point(a), point(b), point(c)
Point(a), point(b), angle(c a b), angle(a b c) -->point(a), point(b), point(c)
Point(a), point(b), angle(c a b) -->point(a), point(b), slope(a c)
Point(a), point(b), slope(a c), slope(b c) -->point(a), point(b), point(c)
Point(a), slope(a b), distance(a b) -->point(a), point(b)
```

The constraint solver uses rules, like the above, to structure the unordered constraint data and to instantiate the joints geometry. Matching is the basic mechanism to detect if a pattern of goals which appears in a rules condition has corresponding facts in the constraint database. In such case the rule applies to the database, which means that the matched constraints are substituted by the consequences of the applied rule. Matching can be thought as a slot filling process as shown in figure 5. The condition part of a rule is represented as a pattern of hollow shapes of different type and different shade. Constraints are represented as solids of different type and shade. The shape type represents the constraint type, like point, dimension, and angle, while the shade represents the particular constraints arguments. To fill a slot both the shape and shade should match. To test the pattern of a rules condition against the constraints database, in a first stage rows of candidates are formed for each slot based purely on the constraint type, then all permutations are checked to find the one with a consistent combination of variable arguments. If a match was detected a substitution follows with the consequences of the applied rule and matching starts again from the first rule. Substitution implicitly builds the mathematical function that is going to evaluate a joints geometry after solving has successfully finished. If there was no match then the next rule is tested (if any is left).

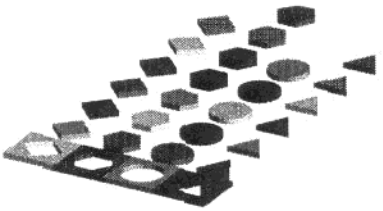

Figure 5. Matching constraints against a rule

4 Examples

Following is the first example case of generating and editing a constrained parameterized model. The model, that has been kept minimal for the clarity of presentation, introduces the benefits of using constrained, parameterized skeletal shapes, to control void representations [9] of architectural structures.

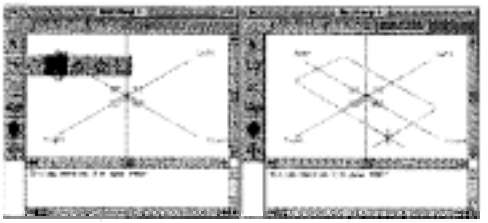

Figure 6. Generating a sketch

In the first place the user has to define a shape (figure 6) using a sketching tool, which in our particular example is a rectangle. This sketch sets a starting topology, and an initial geometry. The topology is roughly the knowledge we originally have when we start our design project, while the geometry is what we try to instantiate and it may change in the process many times.

As a next step we want to constrain the rough design idea to adhere to certain specifications (figure 7-left), in our case dimensional decisions about the design project. We may even decide to attach attributes to the rough sketch at any point of the process, i.E. Before or after we constrain the sketch (figure 7 right). The initial values set in the constraint fields are automatically assigned pre-calculated values from the sketchs geometry. This approach reduces the overhead needed to set constraint values for parts of the project that are of less significance. In our example these are the longer sides of the rectangle. Whenever we assign a dimensional constraint the system offers two options: either we set the constraint to a fixed value (i.e. Distance = 200), or we set the constraint to an expression which may include lisp variables (i.e. Distance = 2 x side, where side is an LISP variable). In the former case after resolving the constraints we end up with a single shape, provided that there is a solution. In the latter case, which is of more interest, after resolving the constraints we end up with a family of constraintwise similar, parameterized shapes.

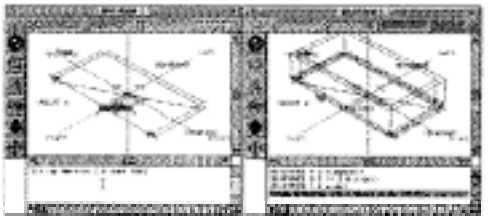

Figure 7. Constraining, and setting offset and height attributes

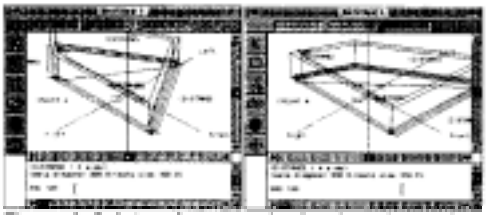

Figure 8. Solving the example sketch and reevaluating it

The last two figures (figures 8 left, 8 right) are two different evaluations of the same shape as in figure 7. The shape has been constrained to have the main diagonal equal to the value of the lisp variable diagonal', one of the smaller sides equal to the e-lisp variable 'side', and its opposite side equal to 'two times side'. The first evaluation is for 'diagonal' equal to 200 and the second is for 'diagonal' equal to 300. A variable can be used to control more than one constraint. In the presented example this is the case with the two opposite sides that equal to 'side' and '2 x side.' a complicated project which has many

dimensional constraints can be controlled sometimes by a few dimensional parameters that appear in multiple constraints, standalone or as components of expressions.

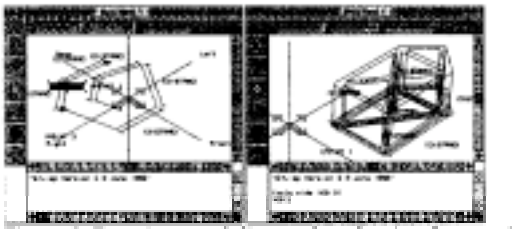

Figure 9. Drawing a polyline and evaluating for certain constraints

In the second example (figure 9) we have a single polyline of eight segments sketched with no particular structure. Then we add height, wall thickness, and the following constraints:

Point (1), point (3)
Distance (1 2), distance (2 3), distance (4 5),
Distance (5 6), distance (6 7), distance (7 8)

It should be repeated at this point that the order the user attaches attributes and constraints to the initial skeletal shape does not matter. As soon as the constraints are resolved there is a dramatic change to the original almost randomly put polyline. The result is an extruded solid from the carefully dimensioned and placed solution to a line puzzle. As it can be seen on the right-hand side of figure 9, all the structure that the initial polyline was carrying has been reevaluated including the intersections of the offset curves that generate the wall surfaces, and the final solid. The user can continue altering the shape from this point either deleting and adding constraints to form a different structure or just editing the parameters to modify structural dimensions. In all cases the full model is evaluated immediately, a result of the efficiency of the internal floor plan representation. The above examples demonstrated that constraints, parameterized shapes, and efficient representations can contribute to facilitate the process of transforming a design idea into a traditional solid model. The ability offered to alter the object at any level of the process (i.E. The sketch, the constraints, the parameters, or the produced solid) increases the overall flexibility of the interaction with the user.

5 Conclusion

In conclusion the use of symbolic expressions for resolving geometric constraints applied to architectural modeling was investigated. A prototypical design program has been implemented, based on a lisp interpreter, where the effectiveness and efficiency of symbolic computation in constrained modeling has been tested. The results have shown that constraint modeling in architectural context can be of significant importance in at least two non exclusive design situations: when specification driven design is required, or when a few parameters which are repeatedly used control a large amount of geometric data. Additionally the uniformity offered by symbolic data representation at the implementation level, combined with the interactive graphic interface provided to edit these structures, has been proven to be extremely effective.

6 Endnotes

1 Negroponte N., The Architecture Machine
2 Mitchell W., Computer-Aided Architectural Design
3 Yessios, C., The Computability of Void Architectural Modeling
4 Sohrt, W., & Beat D. B.,, Interaction with Constraints in 3D Modeling"
5 A subset of common lisp as defined in Steele, L. G., Common LISP the Language. The initial implementation was based on the tiny interpreter presented in Winston, Patrick H., & Berthold K. P. Horn, LISP, pp. 321-332

6 Schalkoff, .G. J.,., Artificial Intelligence: An Engineering Approach

7 As defined in Foley J, van Dam A., Feiner K.,, Hughes I., Computer Graphics Principles And Practice, pp.261-275.

8 When we know the parameters beforehand we set them while sketching the entity as in most drafting systems.

9 Spreacher, T., Design Of Architectural Space Through Void Modeling Representations

7 Bibliography

Foley J., van Dam A., Feiner K., Hughes J., Computer Graphics Principles And Practice, Addison Wesley, 1990, 261-275.

Graham, P., On LISP: Advanced Techniques For Common LISP, Englewood Cliffs, New Jersey: Prentice-Hall, 1994, 321-346.

Horowitz, E., & Aartaj S., Fundamentals Of Data Structures, Rockville, Maryland: Computer Science Press, 1976, 218-344.

Light, R., & Gossard, D., "Modification of Geometric Models through Variational Geometry, Computer Aided Design, 1982, vol.14, 209-214.

Mitchell W., Computer-Aided Architectural Design, New York: Van Nostrand Reinhold Co., 1977.

Negroponte N., The Architecture Machine, MIT Press, 1970.

Roller, D., 'An Approach to Computer Aided Parametric Design, Computer Aided Design, 1991, vol.23, 385-391.

Schalkoff, R. J, Artificial Intelligence: An Engineering Approach, New York: Mcgraw-Hill, 1990, 248-324.

Sohrt, W., & B. D. Bruderlin, 'Interaction with Constraints in 3D Modeling", in Proceedings of ACM/SIGGRAPH Symposium on Solid Modeling and CADCAM Applications, Austin, TX, 1991.

Spreacher, T. B., Design Of Architectural Space Through Void Modeling Representations, Master's Thesis, The Ohio State University, 1989.

Steele, L.G., Common LISP the Language, Bedford, Massachusetts: Digital Press, 1990.

Winston, P. H., Berthold K., & P. Horn, LISP, Reading, Massachusetts: AddisonWelsey, 1984, 321-332.

Witkin, A., Fleischer, K., & A. Barr, "Energy Constraints On Parameterized Models", Computer Graphics, 1987, vol.21, 225-232.

Yessios, C. I., "The Computability of Void Architectural Modeling", The Computability of Design, Proceedings of Symposium on Computer-Aided Design at SUNY, Buffalo, 1986.