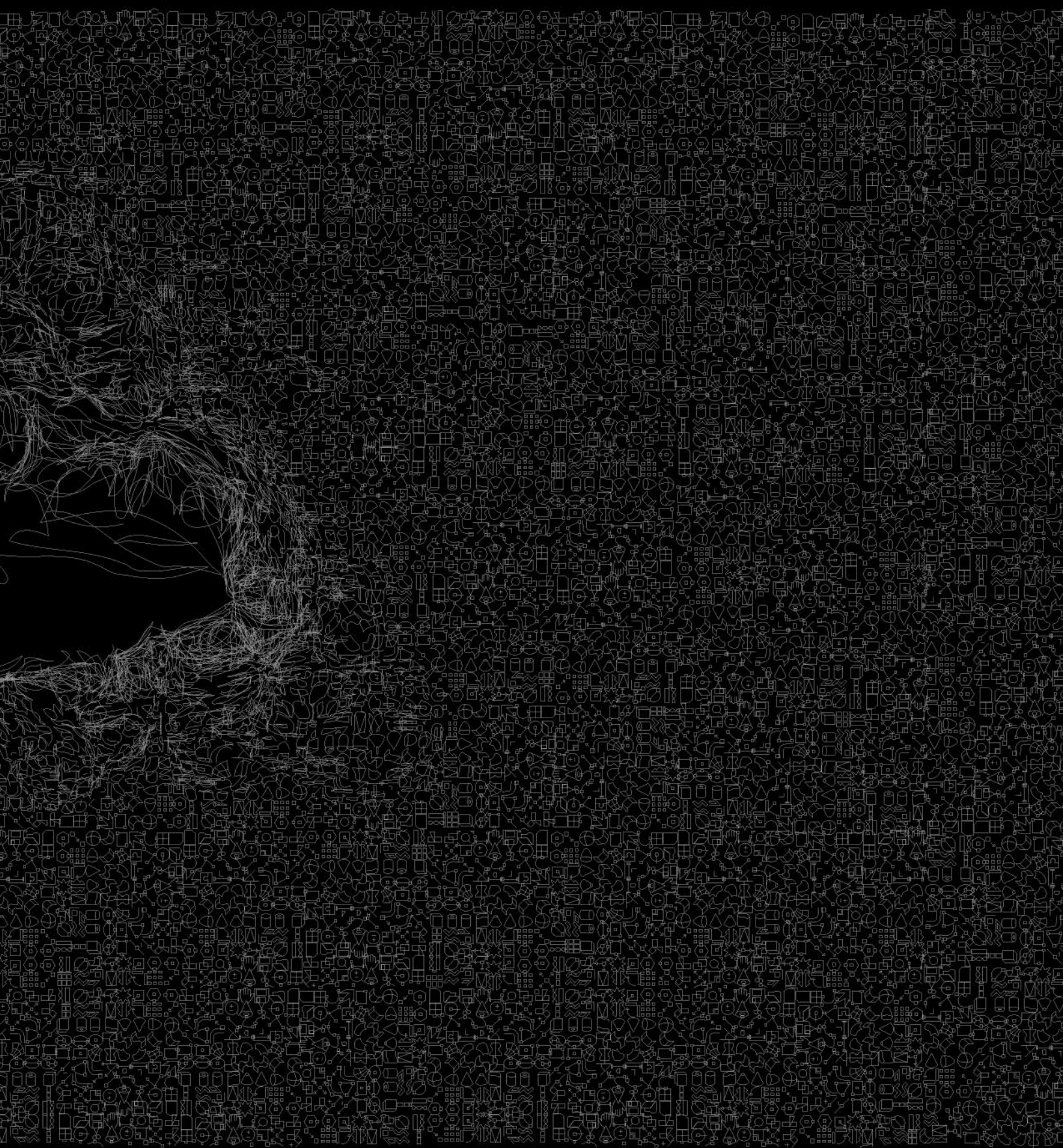


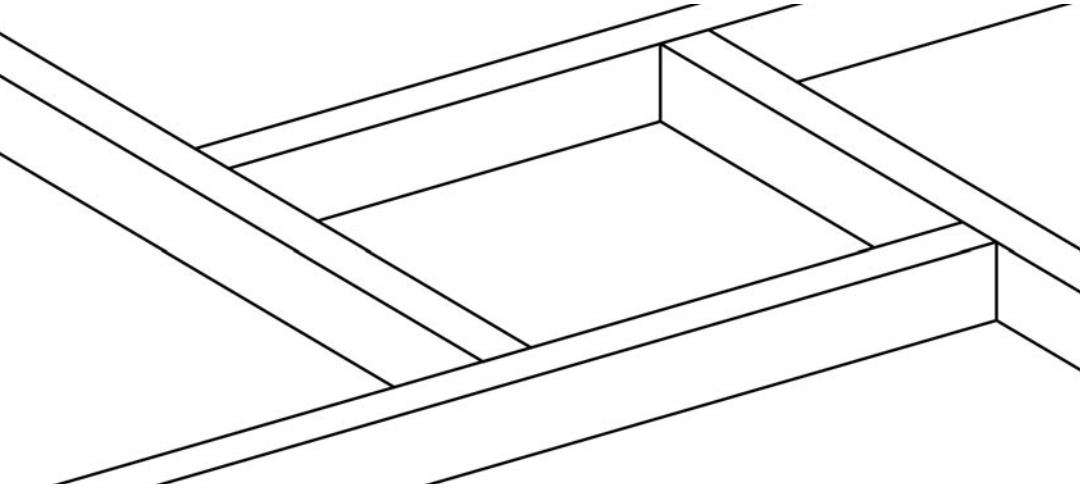
FABRICATION

EXAMINING THE DIGITAL PRACTICE OF ARCHITECTURE

BEESLEY CHENG WILLIAMSON

Linking Analysis and Architectural Data
why it's harder than we thought

Dr. Scott Johnson University of Michigan
Peter von Buelow University of Michigan
Patrick Tripeny University of Utah



Abstract

This paper considers high-level, architecturally oriented representations, like Building Information Models (BIMs), and examines the difficulty of integrating analyses with such representations. Structural analysis is selected as a sample analysis domain, and is examined by integrating a structural analysis into the test implementation of a program that utilizes architecturally oriented elements. A fundamental problem is found to be that architecturally oriented elements are inappropriate for structural analysis. Methods for sequentially analyzing architectural elements are discussed, but are found to be inadequate. Accurate analysis requires analyzing the entire structure at once using a representation specific to structural analysis. A method for generating a structural representation based on the architectural representation is discussed, but the process is not simple. The process is complicated by the fact that architectural elements and structural elements do not correspond in a one-to-one or even a one-to-many manner. An accurate structural representation may even require *semi-fictitious* elements not corresponding to actual physical components. These findings are believed to be true for other analysis domains, as well.

Introduction

Before architecture is made, it has to be represented. It has to be represented by architects, engineers, and consultants, for their own work, for each other, and for contractors, clients, and others. It should be noted, however, that all representations are not created equal. Some representations are inherently better suited for certain tasks than other representations are (Day 1988).

Experts in different domains, therefore, often use different representations. Different computational tasks likewise require different representations. Structural analysis requires representations describing structural members, loads, connections, material properties, and so forth. Other analyses use other representations.

Yet, working with multiple representations is difficult. The representations contain much duplicate data, and coordinating the different representations has become a significant issue in Computer-Supported Collaborative Work.

Providing a single representation with all the data needed to support multiple types of analyses is one of the motivating factors behind architecturally oriented representations, like the Building Information Models (BIMs) developed by CAD software companies. Various research projects have also approached the issue of linking various sorts of analyses to architectural models. These projects include SEMPER (Mahdavi et al. 1996), BDA (Papamichael et al. 1996), P3 (Kalay 1997; Khemlani et al. 1997), and other projects (Mitchell et al. 1992; Mahalingam 1999). However, commercial approaches are only just beginning to integrate analyses with architectural modeling software, and research approaches have a tendency to be speculative, to rely on simplified analyses, and/or to organize the architectural model around particular analyses. True integration of analyses with such representations has been difficult to achieve.

This paper documents attempts to link structural analysis to architectural data. The undertaking has convinced the authors that analyses are not easy to perform on data associated with architectural elements like those found in BIMs. It has shed light on why successful linking is so difficult to achieve, and what the likely solution may be.

The Test Implementation: Proteus

The undertaking described in this paper is part of a larger investigation of architecturally oriented representations. This larger investigation involves the development of elements called *protean elements* (Johnson 1998), intended to allow architects to create digital models suitable for design exploration, visualization, and various analyses (structural, thermal, etc.). A test implementation called *Proteus* (Fig. 1) has been developed at the University of Michigan to examine these elements. Although protean elements approach several issues (notably model correctness and joint elements, described below) differently from BIMs and other previous approaches, results from Proteus were very enlightening, and lessons learned from Proteus are believed to also be applicable to other architecturally oriented representations.

1. The Architectural Representation

Like elements in many BIM systems, protean elements are objects in the object-oriented programming sense. They are intended to correspond to architectural elements like walls, rooms, floors, beams, doors, and so forth, with values for whatever parameters (height, radius, direction of swing, etc.) are necessary in order to describe each particular class of element. The intent of the Proteus project was to make the elements easy for architects to use, allowing maximum design flexibility while still supporting technical analyses.

Protean elements differ from other architectural representations in that protean elements maximize flexibility by de-emphasizing *model correctness*. This means that the architect is able to create model elements freely, with almost no restrictions on relationships between elements: *rooms* are not required to be bounded by *walls; beams* are not required to stop when they meet a column or another *beam*, etc.

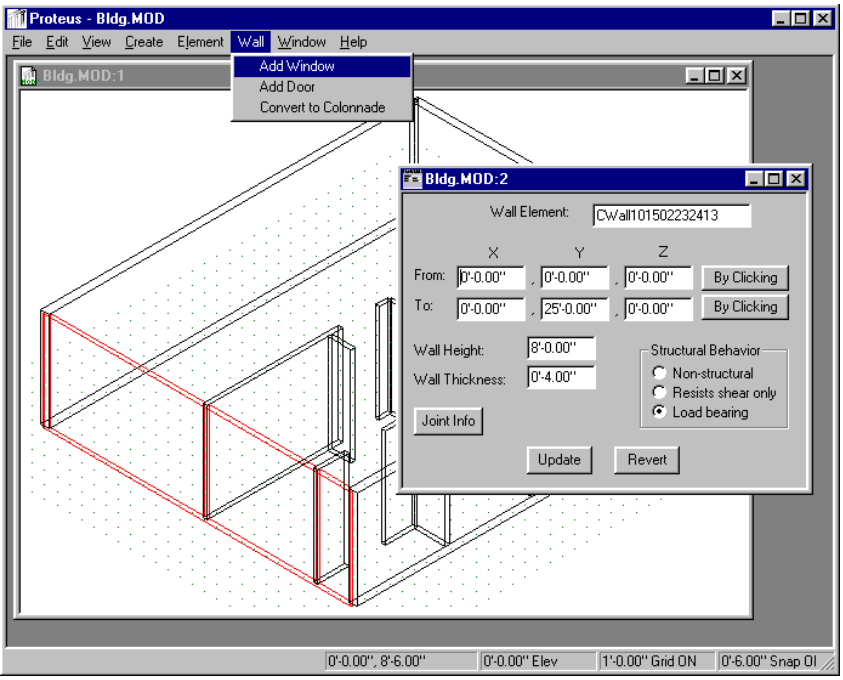


Figure 1. Screen shot from Proteus

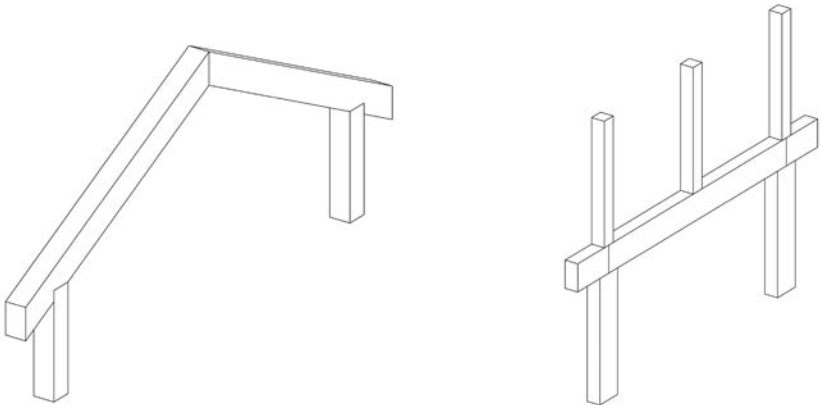


Figure 2. Situations where structural elements cannot be ordered by element class

Another notable difference between most BIMs and protean elements is the inclusion of joint elements, generated automatically where other elements come together. These elements are included for two reasons. The first is to facilitate the generation and manipulation of product-level constituent elements, like studs and base plates. The second is to facilitate structural analyses, which rely not only on information about elements, but also on information about how elements are connected.

2. Facilitating Analyses

Proteus explores several aspects of protean elements. One of the most important is the ability to facilitate analyses (structural, thermal, cost, etc.). The implementation explored one analysis domain in depth, in order to identify issues concerning analyses in general. Structural analysis was selected as the analysis domain to examine in detail. The plan was to let the architect build an architectural model using protean elements, then extract structural information from the elements and use it for a structural analysis.

Element Ordering

One method for performing a structural analysis is to analyze each structural element independently, calculating total forces and moments using the laws of statics. Each element is considered in turn, its loads considered, reactions calculated, and internal shear and moment computed. Once reactions for an element are calculated, they can be applied as loads to supporting elements. Processing elements independently or sequentially in this manner has been proposed previously as a method for analyzing structural behavior of architectural models (Mitchell et al. 1992; Mahalingam 1999).

However, accurately analyzing a structure in such a way requires ordering the elements in some manner, so that loads from elements can be applied to elements that support them, and no element is analyzed before an element that bears upon it. The authors considered several ordering schemes, described in the following subsections.

1. By Element Type

As a general rule, floors and roof surfaces bear on joists, which bear on beams, which bear on girders, which bear on columns and walls, which bear on foundations. Since class information indicates whether an element is a beam, column, wall, etc., it might at first seem obvious and reasonable to order elements for a structural analysis according to class.

However, violations of this rule are frequent enough that it cannot be considered as a workable, universal algorithm. There are too many designs for which the algorithm would not work. Examples are shown in Figure 2. In the example on the left, two beams bear on each other; in the example on the right, a column bears on a beam.

2. By Elevation

Another scheme for ordering structural elements is to base the ordering on the principle that higher elements generally bear upon lower elements. However, while this relationship is true more often that not, exceptions to the rule are numerous. A fairly common example is the clerestory shown in Figure 3, where rafter (A) bears upon beam (B), even though most of A is lower than B. Examples like this demonstrate the problems of trying to order elements by elevation.

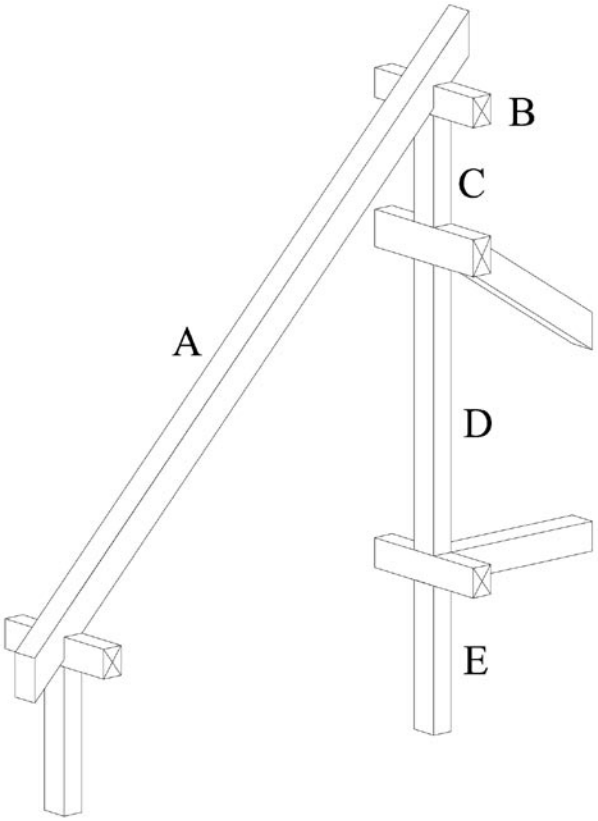


Figure 3. A situation where the average elevation of elements does not reflect which elements bear on which others

3. By Joint Configuration

A slightly more sophisticated variation of ordering elements by elevation would be to examine the relative positions of elements in the vicinity of a joint. If one element lies on top of another element at a joint, the element on top generally bears on the lower one. This algorithm would work correctly in situations like the one shown in Figure 3.

However, this algorithm also has problems. It does not work well at joints where one element is welded or bolted to the side of another. Even in situations where one element is directly above the other at a joint, it may be the case that the lower element is suspended from the upper one, as seen in Figure 4. Thus, ordering structural elements by joint configuration is not an algorithm that would work consistently.

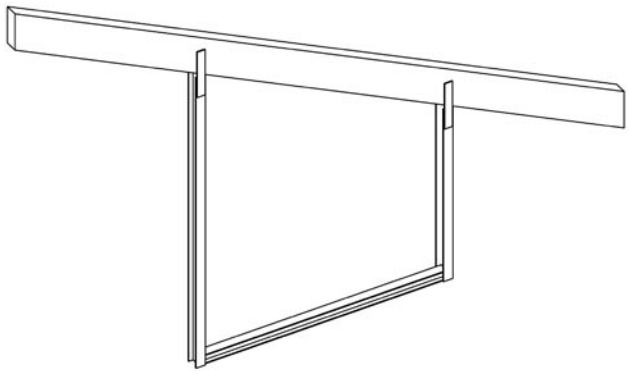


Figure 4. A situation where joint configuration does not reflect which elements bear upon which others

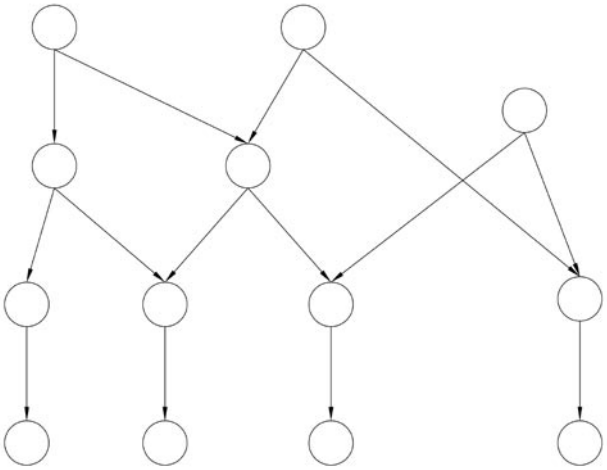


Figure 5. A directed net

4. By Tracing from Foundations

A more sophisticated scheme for ordering elements is to generate a directed net based on the elements (Fig. 5). In this scheme, structural elements are represented by nodes, connected by directional *bears upon* links, giving the net a directional character (a top and a bottom). No element in the directed net can be analyzed structurally until all the elements linked to it from above in the directed net are analyzed.

The algorithm for generating the directed net recognizes that the elements at the bottom of the directed net—those elements upon which all the others ultimately bear—are the foundations of the building. Any structural element which is connected to a foundation is assumed to bear on it. Structural elements connected to these are assumed to bear on them, and so forth. The network of structural elements is therefore generated by tracing backwards (i.e., generally upwards) from the foundations, until structural elements that have no additional elements bearing upon them are eventually reached. The process is analogous to tracing a tree upwards from the roots to the leaves. Once the directed net is generated, however, actual analysis of the structural members proceeds in the opposite order, from *leaves* to *roots*.

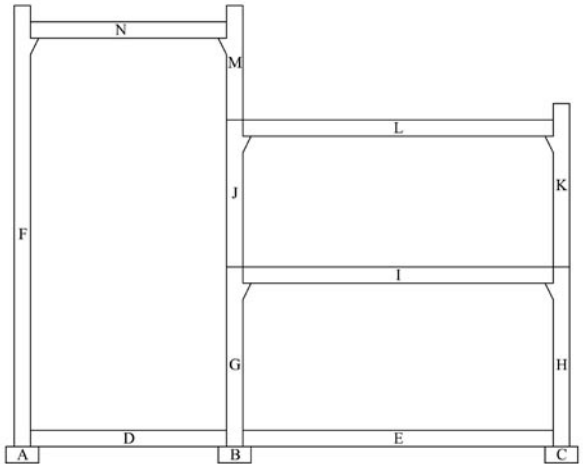


Figure 6. An example of a building for which a directed net cannot be determined by tracing upwards from the foundations

However, this algorithm too is flawed. Consider the building shown in Figure 6. Tracing from the foundations, we would build the partial directed net seen on the left side of Figure 7. This net is flawed, however. The correct version would be the net depicted on the right side of in Figure 7. The problem is that the algorithm treats all elements equally, whether they are several stories tall, or only a partial story. As a result, the second story column (J) is considered just as far removed from the foundations as the roof beam (N). This is simply incorrect.

5. Other Problems with Element Ordering

Ultimately, the authors were unable to identify an algorithm for ordering structural elements that worked correctly in a consistent, generalized manner. Even worse, configurations of elements were identified for which there was no possible correct ordering. One such configuration, called a "*lever-beam structure*" (Bertin 2001), or "*cyclo-symmetric grid*" (Rosman 1996), is shown in Figure 8.

Attempting to order structural elements becomes even more difficult when horizontal loads are considered. Furthermore, even if a reliable ordering algorithm could be found, and a means devised for handling cyclic *bears upon* relationships, ordered analyses would still be unable to handle statically indeterminate structures. A simple (one-element) example is shown in Figure 9: reactions for the beam cannot be determined using the equations of statics, because the number of unknowns (four) exceeds the number of usable static equations (three).

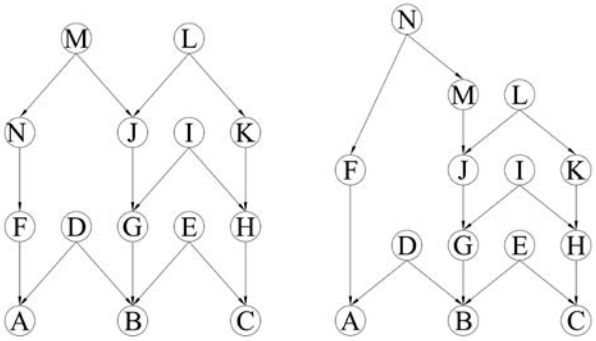


Figure 7. Incorrect (left) and correct (right) directed nets depicting the structure of the building shown in Figure 6

It can therefore be seen that all algorithms for analyzing structural members one at a time in this manner are flawed, or limited to special cases. Performing an accurate structural analysis in a consistent, generalized manner is only possible by performing the analysis all at once on the entire structure. A global stiffness analysis, using finite elements, is the typical method for doing this.

Finite Element Analysis

Finite Element Analysis (FEA) involves modeling something (a building structure, machine part, volume of air, etc.) as a *finite* set of smaller *elements*. Each finite element typically has the form of a line segment, triangle, quadrilateral, tetrahedron, or hexahedron, defined by two or more nodes. In a stiffness analysis, attributes associated with the finite elements describe characteristics of a structure related to stiffness: material and geometric properties. Matrix algebra is used to identify displacements at the nodes. Once these displacements are determined, the related forces at each node can be calculated. Other analyses based on finite elements can be used to study heat transfer or air movement in a somewhat similar manner.

1. The Finite Element Model

In architecture, a beam, column, truss member, cable, or similar member, is generally modeled as a linear, finite element connecting two nodes. A wall, floor, etc., is usually modeled as a mesh of triangular or quadrilateral finite elements. These elements must be created and organized carefully in order to yield accurate analysis results.

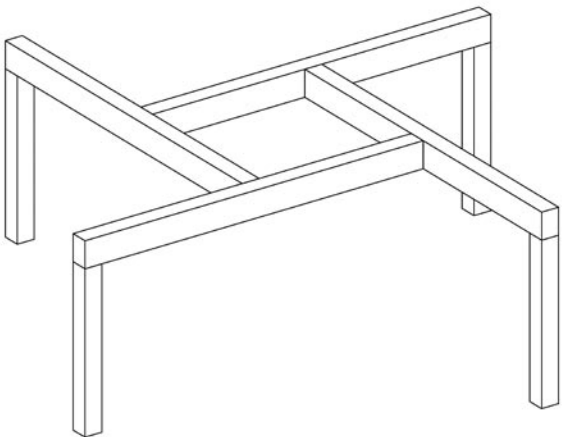


Figure 8. An arrangement of beams that cannot be ordered.

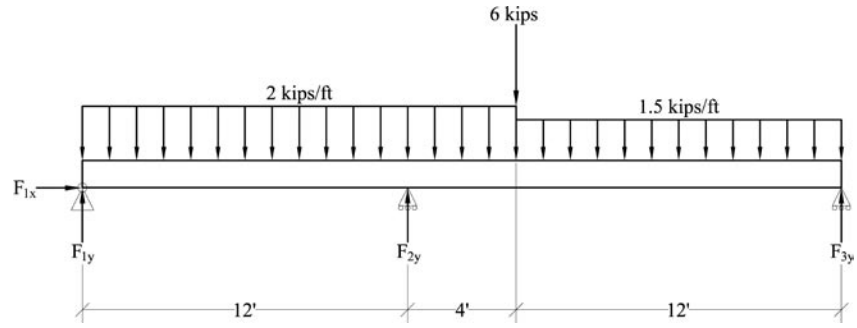


Figure 9. A statically indeterminate beam

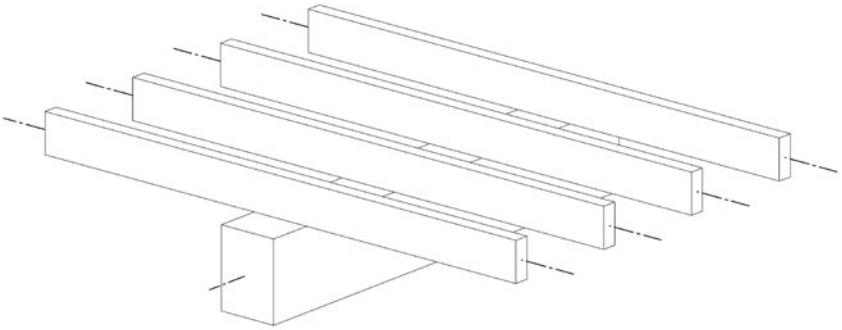


Figure 10. Girders and beams to be abstracted

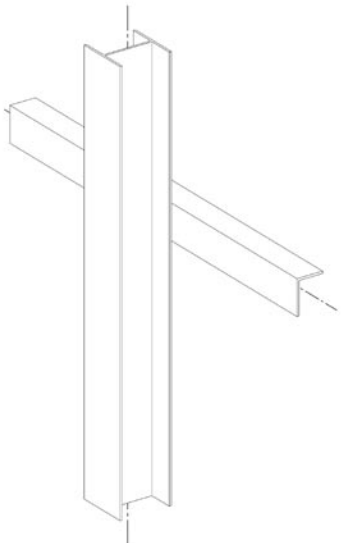


Figure 11. Joint between a beam and column with skew centerlines

Elements have to be properly *buttoned together* at the nodes in order to provide shared deformation conditions. If two finite elements meet at a point in space but do not share a node in the model, they will not deform together and will therefore not directly interact. If the two finite elements are intended to directly interact, then they must share nodes.

In addition, planar elements in a Finite Element Model must be properly scaled and proportioned. Elements that are too large can *gloss over* local peak stresses, allowing potential structural failures to go undetected. However, elements that are too small can exaggerate stresses in certain conditions, particularly where point loads are involved. Furthermore, planar elements with high aspect ratios (e.g., triangles or quadrilaterals that are too elongated) can also distort analysis results.

2. Differences between the Architectural and Finite Element Models

These restrictions on a Finite Element Model result in considerable differences between a structural Finite Element Model, and BIMs, or other architecturally oriented models. Consider, for instance, a girder and several beams arranged in the configuration shown in Figure 10.

Note that an actual physical girder is generally a single, uninterrupted member spanning from one column to the next, or sometimes farther. When building a computer model, an architect would likely want to specify the girder in a similar manner. This is the way that a Building Information Model or other architecturally oriented representation would probably represent the girder.

However, a finite element representation for structural analysis needs to treat the girder differently. The girder requires connections to the beams bearing on it at intermediate points between columns, therefore, the girder must be 'broken' into many short segments, each stretching only from one beam to the next.

3. The Need for Offsets or 'Semi- Fictitious' Elements

Complicating matters is the fact that linear members like beams and columns must be represented by linear finite elements corresponding to the centerlines of the members. In the example shown in Figure 10, finite elements representing the girder and the beams need to be connected *even though the centerlines of the girder and beams do not insect.*

A related situation can be seen in Figure 11, where a beam is bolted to the side of a column. Neither the column nor the beam should be moved horizontally in the structural abstraction to align it with the other member. Doing so might mask the effect of eccentricity and result in an incorrect solution, or alter the lengths of other structural members connected to the beam or the column. Furthermore, the structural representation must leave the column vertical.

The safest way to model such a situation is using *semi-fictitious* members, like the one shown in Figure 12. A finite element is created to span from the column node to the beam node, and is treated as being infinitely strong with no weight or chance of failing. Such an element is 'semi-fictitious' in that it does not correspond to an actual physical structural member, but it does embody structural behavior that needs to be modeled.

In some FEA packages (including the STAAD-Pro®, the structural Finite Element Analysis program used with the test implementation of Proteus), these semi-fictitious elements are subsumed into the concept of *offsets*. They are not described literally as elements connecting two nodes; instead, they are described as an *offset* of a structural member from a node. Thus, each span of the beam is described as terminating at the column node, but being offset from this location by a certain amount.

Abstraction Rather than Extraction of Data

It can thus be seen that if an architectural model is comprised of elements meant to correspond to actual building components (or to assemblies like *walls* that architects tend to treat conceptually as elements), then the architectural model itself is not going to be appropriate for structural analyses.

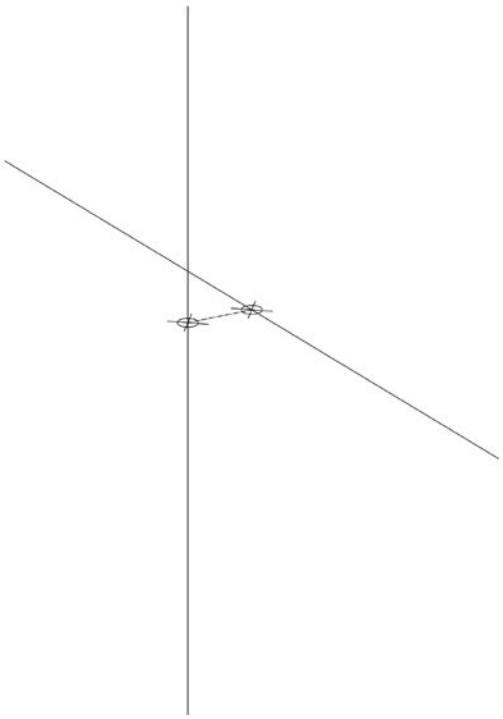


Figure 12. A 'semi-fictitious' member spanning from column to beam

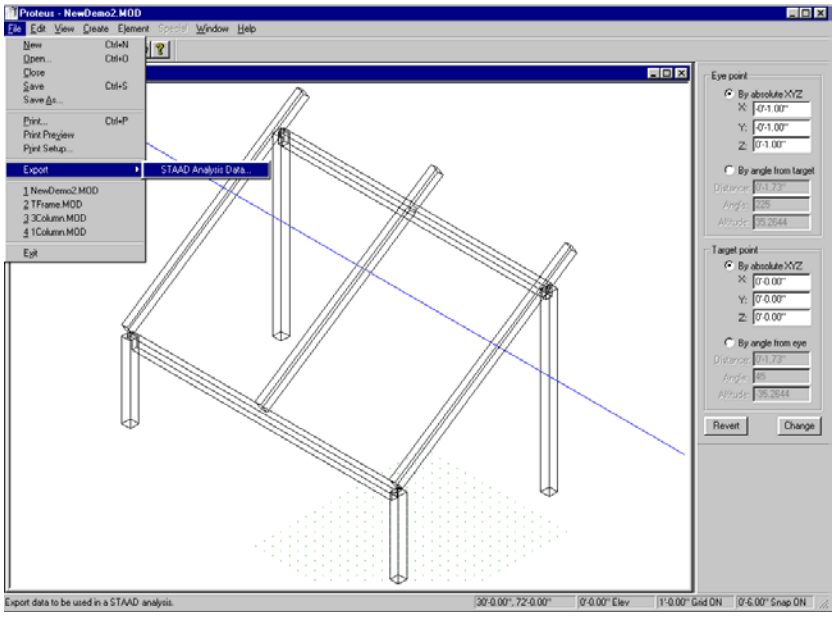


Figure 13. A structure to be abstracted by Proteus

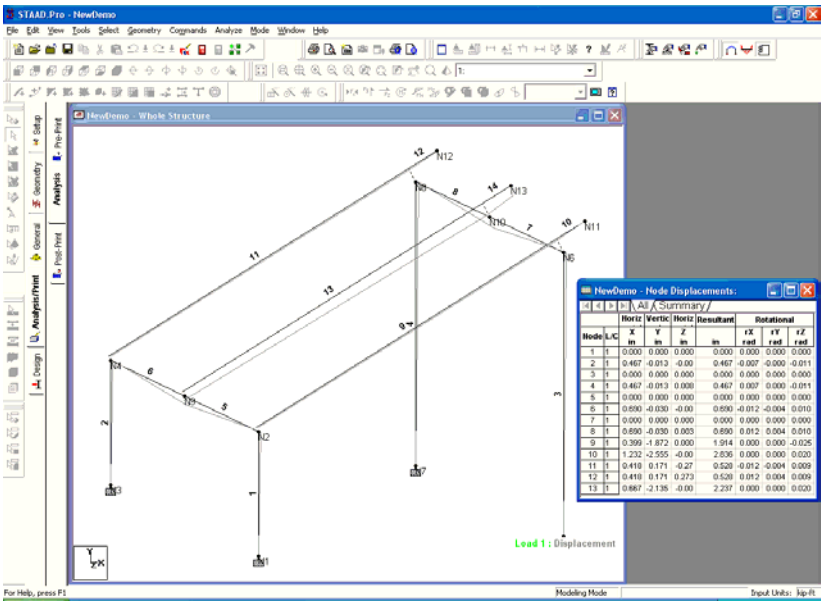


Figure 14. The finite element representation of a structure is abstracted from the architectural model, imported in STAAD-Pro, and analyzed. STAAD-Pro is copyright 1997-2003 by Research Engineers, Intl., a division of netGuru, Inc.

Instead, an algorithmically complex abstraction must be performed, to generate a separate Finite Element Model. Elements like girders and beams have to be subdivided where they connect to other elements. Planar elements like walls and floors have to be subdivided where they connect to other elements, and also have to be meshed with appropriately sized, well-proportioned triangles or quadrilaterals. In addition, connected elements need to share nodes, even when those elements have skew centerlines. This means that no architectural element can be abstracted based solely on the element's own data. The element's relationships to other elements in the model must be considered in order to generate an accurate Finite Element Model.

Abstraction in Proteus

Deriving a Finite Element Model from information about architectural elements involves complicated abstraction, but it is abstraction that has been routinely performed by structural engineers for decades. This suggests that an algorithm (albeit possibly a complex one) for performing the abstraction might be able to be developed. Such an abstraction algorithm has been developed for Proteus.

1. Factors Affecting Abstraction

In the test implementation of Proteus, a Finite Element Model is generated by processing the architectural model's elements one at a time. As the elements are processed, data needed for finite elements is computed, and when all the elements have been processed, the finite element data is written in a text file that can be read by STAAD-Pro (see Fig. 13 and 14).

This processing is affected by several factors. Firstly, it is affected by the type of architectural element being processed. Linear elements like beams must be processed differently from planar elements like floors, since they are represented with different types of finite elements. Even a dome and a floor would most likely need different routines to generate their finite elements, because even though the same finite element type (planar shells) would be generated in either case, the shells would still be generated in different ways, with the different geometries generating different configurations of elements using different sets of rules. Each class of architectural element needs a separate abstraction function to handle the particular characteristics of that class, although in some cases (e.g., beams and columns), similar element classes can perform abstraction using algorithmically similar functions.

Secondly, abstraction is affected by the parameters of the element being processed. This affects which coordinates are used for nodes and offsets, what connections are made to other finite elements, etc.

Thirdly, the abstraction process is affected by other architectural elements connected to the element being processed. The locations of these other elements, together with the characteristics of element being processed, affect how an element should be subdivided and whether offsets/semi-fictitious elements are needed in order to get finite elements to connect.

2. Abstraction of Linear Elements

When a linear architectural element is processed (only beams and columns are processed in the test implementation), its endpoints and joints with other elements are ordered according to distance from one end of the element (see Fig. 15). Each end/joint is considered in turn. If the other structural member coming into a joint was processed previously, then an existing finite element node for the joint will exist and can be referenced; otherwise, a new node is generated for the joint. If an existing node is found, but that node does not lie on the centerline of the element being processed, an offset from the node to the centerline is computed. A finite element is then created, spanning from the previous node to the current one, or the nodes are consolidated, if the finite element would be zero-length.

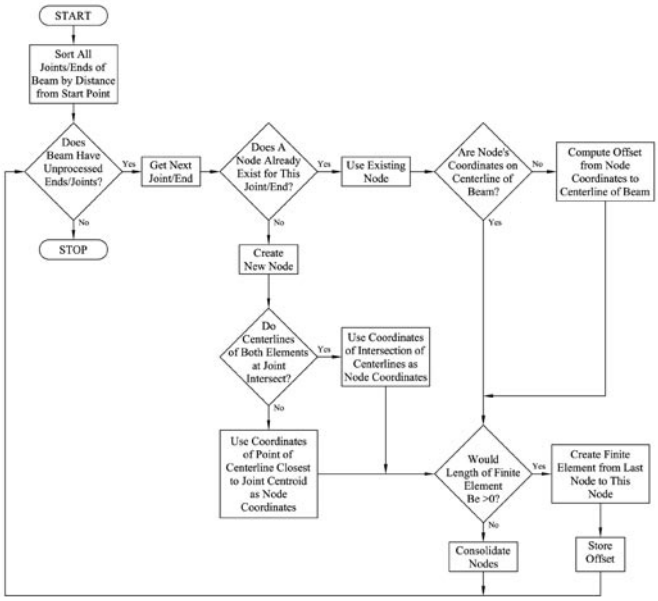


Figure 15. Algorithm for producing an abstraction of a beam (or column)

3. Abstraction of Planar Elements

For planar architectural elements like floors and walls, the abstraction has several stages, as diagrammed in Figure 16. Like linear elements, planar elements must be *cut* wherever they connect to other elements. This includes cutting a planar element where it is divided by a joint with another element (e.g., where a floor beam or wall extends across the width of a floor). It can also include adding nodes to the interior of a planar element (e.g., where a beam *punches into* a wall) or cutting partway across a planar element (e.g., where a strip buttress helps to support a wall but does not extend to the wall's top).

Floor elements may need to be further subdivided, based on floor loads. For instance, if part of a floor is located under *book storage* while the remainder is located under *reading*, the floor loads may be different, and the finite element software may allow only a single uniform load to be applied to any single finite element.

After these subdivisions, the areas need to be meshed to produce finite elements of the proper sizes and proportions. Since meshing algorithms are well established in the field of computational geometry, but are time consuming to implement, planar architectural elements are not converted into finite elements in the test implementation.

Note that because connected elements need to be *buttoned together* in order to create integrated meshes, adjoining elements may also need to be subdivided. In a sense, it isn't just planar elements that need to be meshed; linear elements connected to meshed planar elements must also be *meshed*.

4. Abstraction of Idiosyncratic Elements

Architecture, as a profession, tends to place a high value on innovative design. Thus, with any architectural modeling system, there is a significant possibility that an architect may want to model idiosyncratic elements, the existence, appearance, and function of which cannot be anticipated and specifically supported by the programmers who write the architectural modeling system.

Such idiosyncratic elements tax the capabilities of the architectural modeling system, but a good architectural modeling system will not let such elements *fall through the cracks*. A good modeling system will

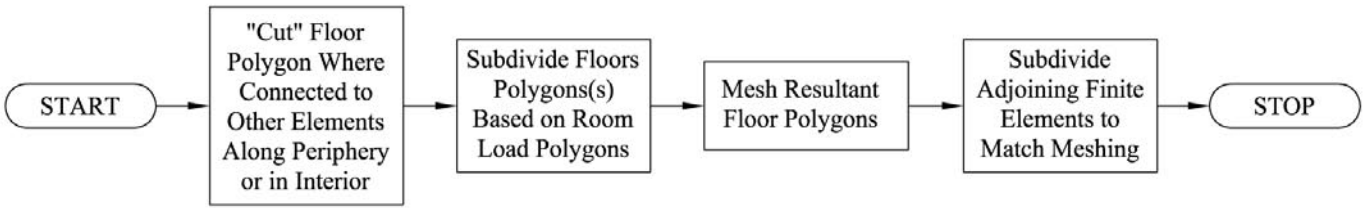


Figure 16. Stages of algorithm for producing an abstraction of a floor (walls would be handled in a similar manner)

leave the user with some means for modeling such elements, even if it does not support them with specific, tailor-made element classes. For example, the user may be able to model idiosyncratic elements by generating their forms, using set operations performed on primitive solids.

However, such idiosyncratic elements present a special problem when trying to produce finite element abstractions for structural analysis. Is it possible to automatically produce finite element representations for the most bizarre elements? Perhaps this can be done based on element geometry, but perhaps not. This topic was not fully explored in the test implementation of Proteus.

It may be the case that for the most unusual elements, user intervention is necessary in order to help generate a proper finite element model. Even if such action is unavoidable, however, more standard parts of the model should still be able to produce proper finite element representations. Thus, while completely automated generation of finite element models may not be possible in every case, a significant reduction of work should still be possible by producing most of the Finite Element Model based on an architectural representation.

Discussion

It should be stressed that Proteus is not just a *front end* for a structural analysis. Proteus is meant to represent architectural elements in a manner that allows special-purpose representations to be generated as needed for graphics, analyses (structural, thermal, lighting, acoustic, etc.), or other purposes. Proteus tests the degree to which a single base representation can be abstracted in various ways to facilitate such tasks.

The experiment with Proteus demonstrates that while data for analyses can be derived from a suitably rich general-purpose architectural model, doing so can require a fairly complicated process of abstraction. Deriving analysis data is not necessarily a matter of merely extracting data already present in the architectural model elements. The architectural model may need to be used as a basis for deriving an entirely separate representation, containing elements that don't necessarily correspond one-to-one with the architectural elements. In fact, as the test implementation shows, the representation created for analysis purposes may contain *semi-fictitious* elements not corresponding to elements in the architectural model, or *offsets* may need to be computed in order to reflect both behavior and actual geometry. In addition, the special-purpose representations for analyses cannot necessarily be created based on information about a single element; the configuration of elements may affect how an element in the configuration should be represented.

While the test implementation focused on structural analysis, the same results are believed to apply for other analyses, as well. Many analyses require their own distinct representations. Air flow is often analyzed with a Finite Element Model of spaces, for instance. Thermal analyses typically require information about spaces, enclosures, and equipment, which might not correspond in a one-to-one manner with architectural elements.

Even when the same architectural element appears in representations for multiple types of analyses, the same attribute for the same element may be different in each different abstraction. For instance, a wall may have one height for acoustic purposes, another for thermal purposes, and another for structural purposes.

It is therefore not sufficient to organize architectural modeling elements around the needs of a single analysis. For instance, it would not be sufficient to organize BIM elements around the needs of structural analysis, by, say, forcing girders to be defined as spanning only from one beam to the next, or defining floors as meshes. This would eliminate the need to perform an abstraction prior to analyzing structural behavior, but it would likely make it even harder to produce the abstractions needed for other analyses. Furthermore, these *structural BIM*s would likely be very difficult to use for architectural design, due to their lack of direct correspondence to physical components or to assemblies treated as conceptual elements.

Deriving an architectural model from an analysis model would likewise cause problems. An abstraction process would be necessary in order to generate the architectural model, but this abstraction process would be complicated by the need for data completely absent from a structural model, concerning attributes relevant to other analyses.

Conclusions

Integrating data for visualization and analyses has long been a goal in Computer-Aided Architectural Design, but it has been difficult to achieve. Models utilizing architecturally oriented elements (e.g., BIMs), instead of 2D and 3D primitives, have recently gained enthusiastic support as a likely means of achieving this integration. However, smooth integration of analyses with architecturally oriented models has remained difficult to achieve. The study described in this paper helps to illuminate reasons why this has been the case, but also suggests that the problem can be solved with suitably sophisticated abstraction routines.

The main hurdle is that complex analyses require their own distinct representations, with their own elements and their own organization. Elements of these representations do not necessarily correspond to elements of representations used for other analyses, nor to elements of a general-purpose architectural representation. Furthermore, the elements in a representation often need to be considered together in order to yield accurate analysis results. For structural analyses and likely for other analyses, relationships between elements are complex, and the entire model needs to be considered in order to ensure accurate analysis.

It is therefore necessary to abstract an architectural model into a special-purpose representation for each complex analysis. This is difficult because elements in the various representations do not necessarily correspond to each other. However, the abstractions do appear to follow consistent rules, and therefore can be automated, except possibly for the most bizarre and idiosyncratic of elements.

References

Bertin, V. (2001). Variations of lever-beam structures. In *On growth and form: The engineering of nature,* ed. P. Beesley and S. Bonnemaison. N.P.: Association of Collegiate Schools of Architecture, East Central Region.

Day, R. S. (1988). Alternative representations. In *The psychology of learning and motivation,* vol. 22, ed. G.H. Bower, 261-305. New York: Academic Press.

Johnson, S. E. (1998). Making models architectural: Protean representations to fit architects' minds. In *Digital design studios: Do computers make a difference? ACADIA '98.* ed. T. Seebohm and S. Van Wyk, 354-365. Québec City, Canada: The Association for Computer-Aided Design in Architecture.

Kalay, Y. (1997). P3: An integrated environment to support design collaboration. In *ACADIA 97: Representation and design,* eds. J. P. Jordan, B. Mehnert, and A. Harfmann, 191-205. Cincinnati, Ohio: The Association for Computer-Aided Design in Architecture.

Khemlani, L., Timerman, A., Benne, B., and Kalay, Y. E.. (1997). Semantically rich building representation. In *ACADIA 97: Representation and design,* eds. J. P. Jordan, B. Mehnert, and A. Harfmann, 207-227. Cincinnati, Ohio: The Association for Computer-Aided Design in Architecture.

Mahalingam, G. (1999). A parallel processing model for the analysis and design of rectangular frame structures. In *Media and design process,* ed. O. Ataman and J. Bermudez, 346-347. Salt Lake City, Utah: Association for Computer-Aided Design in Architecture.

Mahdavi, A., Mathew, P., Lee, S., Brahme, R., Kumar, S., Liu, G., Ries, R., and Wong, N.H.. (1996). On the structure and elements of SEMPER. In *Design computation: Collaboration, reasoning, pedagogy,* ed. P. McIntosh and F. Ozel, 71-84. Tucson, Arizona: Association for Computer-Aided Design in Architecture.

Mitchell, W. J., Liggett, R. S., Pollalis, S. N., and Tan, M. (1992). Integrating shape grammars and design analysis. In *CAAD futures '91: Computer-aided architectural design futures: Education, research, applications,* ed. G. N. Schmitt, 17-32. Braunschweig/Wiesbaden: Friedr. Vieweg & Sohn Verlagsgesellshaft mbH.

Papamichael, K., La Porta, J., H. , Collins, D., Trzcinski, T., Thorpe, J., Selkowitz, S. (1996). The Building Design Advisor. In *Design computation: Collaboration, reasoning, pedagogy,* ed. P. McIntosh and F. Ozel, 85-97. Tucson, Arizona: Association for Computer-Aided Design in Architecture.

Rosman, R. (1996). Contribution to the conceptual design of building floor grids. In *Conceptual design of structures: Proceedings of the international symposium*, University of Stuttgart, October 7-11, 1996, Stuttgart/Germany. vol. II, Case studies, 919–926. Stuttgart, Germany: Institut für Konstruktion und Entwurf II, Universität Stuttgart.

Scott Johnson is a Lecturer in Architecture at the University of Michigan, teaching CAD Fundamentals. His doctoral dissertation involved development of protean elements, including their ability to support analyses, their de-emphasis of model correctness, and issues relating to their implementation. His other research interests include cognition and mental representation, the relationship between product-level elements and conceptual level elements/assemblies, and representations to support genetic algorithms in architecture.

Peter von Buelow currently teaches structural engineering courses in the Taubman College of Architecture and Urban Planning at the University of Michigan. He holds professional registration and degrees in both architecture and engineering. His research interests include the optimization and exploration of structural forms using Genetic Algorithms. He has had experience in both coding his own, and using commercially available, Finite Element Analysis programs.

Patrick Tripeny is an Associate Professor in the College of Architecture and Planning at the University of Utah, where he teaches structures and design. He has degrees from the University of Notre Dame, California Polytechnic State University, and the University of Michigan. He is a licensed architect in California. Tripeny's primary area of research is in structural morphology as it pertains to architecture. He is currently working on the development of a universal grammar for buildings based upon structural systems, and the 10th edition of Simplified Structures for Architects and Builders with James Ambrose.