CIFE CENTER FOR INTEGRATED FACILITY ENGINEERING

Perspectors: Composable, Reusable Reasoning Modules to Automatically Construct a Geometric Engineering View from Other Geometric Engineering Views

By

John Haymaker, John Kunz,
Benjamin Suter, and Martin Fischer

CIFE Working Paper #WP082
November 2003

STANFORD UNIVERSITY

If you would like to contact the authors, please write to:

c/o CIFE, Civil and Environmental Engineering Dept.,
Stanford University
Terman Engineering Center
Mail Code: 4020
Stanford, CA 94305-4020

PERSPECTORS: COMPOSABLE, REUSABLE REASONING MODULES TO CONSTRUCT A GEOMETRIC ENGINEERING VIEW FROM OTHER GEOMETRIC ENGINEERING VIEWS

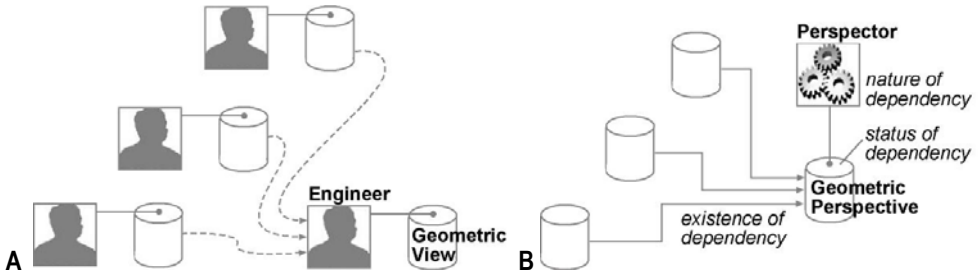
JOHN HAYMAKER, JOHN KUNZ, BEN SUTER, AND MARTIN FISCHER
Stanford University, Department of Civil and Environmental Engineering
Center for Integrated Facility Engineering, Building 550, Room 553H,
Stanford, CA 94305
haymaker@stanford.edu, kunz@stanford.edu, bsuter@stanford.edu, fischer@stanford.edu

Abstract

As they design, plan, and execute AEC projects, engineers today construct task-specific geometric views based on information contained in other engineers' geometric views. Traditionally, engineers have constructed these views manually, using pencils and more recently CAD. Manually constructing views is often difficult, time-consuming, and error-prone. Newer approaches develop central project models that predetermine all potential views for a project or for an entire industry; but these approaches are proving difficult to implement due to the multi-disciplinary, constructive, iterative, and unique nature of AEC projects. Current project modeling approaches lack simple formal methods that engineers can use to specify the automatic construction of a new dependent geometric view from information in one or many source geometric views. This research formalizes reusable reasoning modules, called geometric Perspectors, which engineers can use to automatically construct a task-specific geometric engineering view, called a geometric Perspective, from other Perspectives. This paper presents engineering test cases from the design and construction of the Walt Disney Concert Hall to motivate and retrospectively validate this approach. Through implementation of a prototype, the paper gives empirical evidence that engineers can select from a potentially small number of predefined, reusable Perspectors and easily compose them into a directed acyclic graph to construct useful dependent geometric views more quickly and accurately than current practice and theory allows. Perspectors may enable engineers from multiple disciplines to engage in novel automated yet integrated design and analysis by easily yet formally constructing and integrating Perspectives from other Perspectives.

1 Introduction

Engineers construct geometric engineering representations, or views, as they perform specific design, planning, and project execution tasks. They often construct these views from information in other engineering views created in earlier design, planning, and project execution stages (See Figure 1A). This research formalizes modular, reusable reasoning that enables engineers to specify to the computer how to formally construct (automatically or manually) a dependent geometric engineering view from source engineering views: This involves formalizing the existence, nature, and status of the dependency between views (see Figure 1B).



***A**. Engineers construct task-specific geometric views from other geometric views. The lines are dashed because the dependency between views on AEC projects is currently implicit. **B**. This paper discuses how to formalize the existence, nature, and status of the dependency between geometric views to enable the automatic construction of a dependent view from one or more source views.*

Figure 1: The dependency between views.

Traditionally, engineers construct and integrate views manually; for example, overlaying transparent drawings to assist in integrating the project geometry in two dimensions. Over the past thirty years, computer-aided three-dimensional drafting (CAD) and project modeling approaches have emerged, significantly improving the speed and accuracy with which engineers construct and integrate task-specific geometric views. CAD allows engineers to overlay geometric views and manually (or semi-manually, by using "design by feature" creation tools) select, reformulate, and generate geometric and semantic information into a dependent view in response to information in geometric source views. The problem with relying solely on this level of computational support is that manually constructing new views is often difficult, error-prone, and time-consuming. Additionally, in a manual process, the existence, status, and nature of the dependencies between views are not explicit or formal. As a result, changes in one view cause inconsistencies in other project views; finding and integrating these inconsistencies is time-consuming, error-prone, and difficult; manual processes do not guarantee repeatable results.

Project modeling representation approaches have extended these CAD approaches by formalizing a schema of objects, attributes, and relationships that engineers can instantiate to define project-specific information. Engineers construct task-specific views by selecting from a subset of these models. These relational models can include constraints to monitor the integrity of information in a model. However, constraints in these representation approaches only formalize the existence and status of a dependency; they rarely construct new geometric information.

Therefore researchers and software programmers continue to develop reasoning and management techniques to construct and control the evolution of an iteratively modified model. Research over the last ten years has investigated domain-specific transformations that select, reformulate, and generate from information in source views based on a predetermined schema, to construct useful task-specific dependent views.

Predefining a priori all representations and reasoning required for a project or for an industry is exceedingly difficult as the coverage increases. In addition, design and planning are creative processes in a dynamic world; the need for new task-specific views emerges during the lifecycle of a project, and from one project to the next. Therefore other research has developed generic reasoning; for example, query languages construct dependent views by selecting, reformulating, or generating from information in a model. Yet these query languages do not contain many of the generic geometric operators that engineers need to specify domain-specific transformations, and they are not intuitive for engineers to use. Parametric modeling approaches continue to be developed to enable engineers to formalize the nature of the dependence between concepts in a project model. These approaches are being broadly adopted in many mechanical engineering fields, and some parametric technologies have been commercially introduced for the Architecture, Engineering and Construction (AEC) industry, but as yet they have been slow to have a significant impact on integrating the work of multiple disciplines. As currently formalized, such techniques have not mapped well to the multi-disciplinary, constructive, iterative, and unique nature of AEC projects. These tools do not explicitly enable engineers to formally construct new geometric views from other geometric views, and achieve integration amongst these views as the project progresses.

This paper reviews test cases from the Walt Disney Concert Hall (WDCH) that were detailed in Haymaker et al (2003a). The test cases illustrate: (1) the multi-disciplinary, constructive, iterative, and unique nature of AEC projects; (2) that engineers on these projects need to construct and integrate task-specific geometric engineering views, and (3) that as practiced today, even with state-of-the-art tools, this process is time-consuming, error-prone, and difficult. From these observations we set the goal to create a simple way for engineers to quickly yet formally construct new task-specific geometric views from information in other geometric engineering views. After a discussion of related work in model-based reasoning and parametric design, we introduce the concept of Perspectors, which enable engineers to specify how to construct task-specific geometric views, called geometric Perspectives, from other geometric Perspectives. The paper shows how engineers can select from a relatively small number of Perspectors and compose them into acyclic graph structures, called Perspector graphs, to specify automatic (or manual) construction of dependent geometric views from source geometric views. This research is an initial step toward defining a language of mechanisms to improve the construction and sharing of project information among multidisciplinary engineering project teams, and enable engineers to engage in novel automated design and analysis.

2 Test Cases: Illustrating the Dependencies between Geometric Views

After several years on the drawing boards of architecture firm Gehry Partners, an aborted start by a first general contractor, and a two-year pre-construction phase, general contracting and construction management firm Mortenson

received a lump-sum, at-risk contract with an aggressive required completion date enforced by liquidated damages (Post 2002). Mortenson's job was to manage the detailed design, planning, and execution of the WDCH; as the project progressed, they subcontracted work to various engineering firms and subcontractors that specialize in specific tasks of the building lifecycle. In this section we describe two test cases from the design and construction of the Walt Disney Concert Hall (WDCH) in order to make the following observations about AEC practice. AEC practice is:

- Multi-disciplinary: Engineers from different organizations, representing different engineering criteria, form project-specific teams to design, plan, and construct one-of-a-kind projects in site-specific conditions.
- Constructive: Engineers construct and use task-specific geometric views containing features that contain geometric data types to describe their specific-tasks. Engineers construct these "dependent" views from information in other engineers' "source" views. A dependent view often serves as a source view for other dependent views. An implicit graph of dependencies between task-specific views forms as the design process progresses.
- Iterative: Engineers responsible for dependent views must become aware of modifications to source views through coordination meetings and amended documents[1], and must manually represent any implications of these modifications by integrating the dependent views with their source views.
- Unique: While the WDCH is an extreme example in terms of its shape, no two projects are alike because they are built with the aforementioned multi-disciplinary and distributed organization in an industry with changing building technologies, on a unique site, with a project-specific program. Design concepts and approaches emerge within and across projects. New kinds of dependent views of changing source views are often required.
- Error-prone, Time-consuming, and Difficult: Today engineers often manually construct and integrate these views. Manually constructing and integrating dependent views from source views causes many problems on AEC projects today. Formalizing and automating the dependencies between views would address these difficulties.

On these multi-disciplinary, constructive, iterative and unique projects, engineers do not have the time, budget, motivation, predictive foresight, or interdisciplinary knowledge to define project model schemas and dependencies a priori. The test cases suggest that engineers need to be able to easily yet formally define new concepts and dependencies between concepts at the beginning of a project, and as the project progresses. To enable engineers to define these dependencies, we propose that they could benefit from a simple, formal approach to specify to the computer how to construct and integrate dependent views from evolving source views. Specifically, these test cases illustrate that engineers could benefit from tools that enable them to easily construct geometric views from other geometric views.

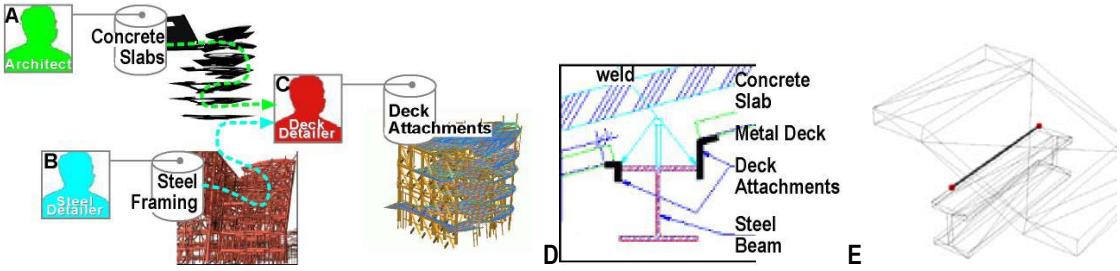
2.1 Deck attachment test case

Gehry Partners, who was under contract directly to the owner of the WDCH, constructed and maintained a Concrete Slabs view containing features that used a surface to describe the boundary of each concrete slab of the project (see Figure 2A). Gehry Partners constructed this view using information in several other project views (not shown). The steel detailer, who was under contract to Mortenson, constructed and maintained a Steel Framing view containing features that used a surface to describe the boundary of each steel member and other features to describe connections amongst steel members (see Figure 2B). The steel detailer constructed this view using information in several other project views (not shown). The metal decking detailer constructed a Deck Attachments view (see Figure 2C) containing features describing where to install metal angle attachments that connect the metal decking for

[1]On various projects, these design versions are referred to as addenda, supplemental instructions, etc.

concrete floor slabs to the structural beams (see Figure 2D). The metal decking detailer constructed this view using information in the Concrete Slabs and Structural Members views, by using CAD tools to manually measure the distances between each beam and slab and drawing a line along the edge of each beam where an attachment was required (see Figure 2E). This was difficult, time consuming, and error-prone work, costing the engineer over 120 hours to complete. When the metal decking detailer was finished, cost estimators, fabricators, and field installers used this Deck Attachments view to produce other views. These other views included cost estimates, numeric information that a CNC machine used to fabricate the attachments, and information detailing which beam should be welded to which attachment in the shop before the beam was delivered to the field.

As the project engineers performed their individual design and planning tasks, they iteratively modified the slab and beam views, generating some new metal decking attachment conditions, while modifying and eliminating others. The metal decking contractor needed to notice and annotate these new conditions in the Deck Attachments view. Views that were dependent on the Deck Attachments view, such as those generated by the cost estimators and fabricators, also needed to be updated. Missed or erroneously detailed deck attachment conditions and slow propagation of modifications to the steel framing or concrete slab views resulted in a deck attachment view that was not fully integrated with the steel framing and concrete slabs. This lack of integration resulted in field welding of about five percent of all the deck attachments on the job, wasting time and money. These difficulties resulted in over $160,000 worth of field welding for deck attachments.



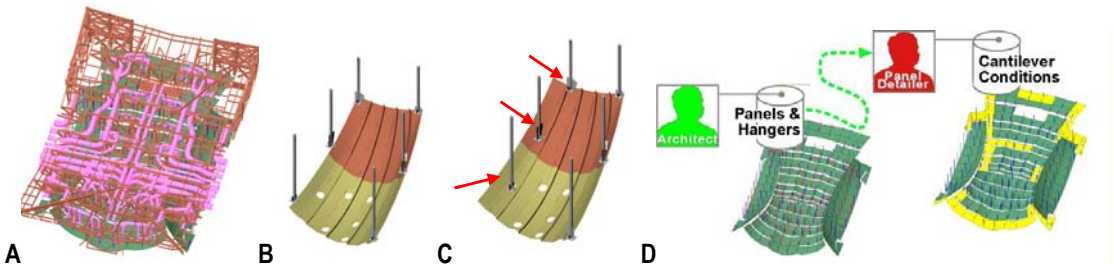
A. *The architect constructed a concrete slabs view.* ***B.*** *The steel detailer constructed a Steel Framing View.* ***C.*** *From these views, the Deck Detailer constructed the Deck Attachments view.* ***D.*** *Deck Attachments are metal angles that are welded to the Metal Deck and the Steel Beam.* ***E****. On the WDCH, the deck detailer modeled deck attachments using a single line along the edge of a beam, wherever an attachment was required.*

Figure 2: Images for the Deck Attachment test case.

2.2 Cantilevered ceiling panel test case

Gehry Partners, Mortenson, subcontractors, owner representatives, and vendors all collaboratively designed the ceiling system (Post 2003) of the WDCH (See Figure 3A). Ducts, catwalks, fire sprinklers, theater lighting, and several other systems vied for a tight space above 200 3m x 4m ceiling panels (see Figure 3B) that weigh in excess of 1 ton each and hang from roof trusses. “Cantilever” conditions occur where the edge of a panel extends significantly beyond a vertical steel tube hanger support (see Figure 3C). The engineer responsible for framing the panels needed to identify and keep track of the location, number, and severity of these conditions, as he designed the framing of the panels. Keeping the number and severity of these cantilever conditions to a minimum was desirable.

The WDCH engineers never constructed and maintained an explicit view of these cantilever conditions. Rather, these conditions were managed in an ad hoc fashion, based on the considerable engineering experience of the design team. This test case is therefore more speculative than the deck attachment test case described above. Perhaps the engineers did not construct and integrate this view because they lacked tools to enable them to easily construct a new view by specifying its dependencies on other views. If such a view could have been constructed quickly and accurately, it could have provided useful information for many tasks. Figure 3D mocks up a simple scenario where a formal view of cantilever conditions could be used as a design aid for an engineering team working with three systems: ducts, hangers, and ceiling panels. As the team moves hangers to make room for certain ducts, the panels that currently have cantilever conditions could be highlighted. Wishing to minimize the number of panels with cantilever conditions, the team could then use this information to choose which hanger to move when routing ducts.



A. *The concert hall ceiling from above with ductwork and with steel roof trusses overlaid.* ***B.*** *Two ceiling panels with no cantilever conditions.* ***C.*** *The same two panels after moving three supports causes four cantilever conditions (the middle hanger creates two cantilever conditions).* ***D.*** *A formal representation of the cantilever conditions would enable a visualization that highlights panels with cantilever conditions to assist in routing ducts.*

Figure 3: Images of the WDCH for the Cantilevered Ceiling Panel test case.

2.3 Conclusions from test cases: Engineers could benefit from a simple, formal method to construct a dependent geometric view from source geometric views

AEC projects today involve the design, planning, and construction of unique projects, in site-specific locations, involving multiple engineers, each of whom specializes in particular tasks of the building lifecycle. In such an environment, engineers construct geometric views that assist them in performing their tasks, and they construct these views based on information in other engineers' views. That is, using their domain knowledge, they perform complex transformations of the information in the source views to construct the information in the dependent view. They often have difficulty doing so today. While it may be possible to predetermine all possible views a priori, and to develop central or federated models using a predefined schema and dependencies, this has not occurred with any great rate of success in the AEC industry. The test cases suggest that this is due partly to the unique, multi-disciplinary, constructive and iterative nature of these AEC projects. Engineers could benefit from a simple formal method to construct a dependent geometric view from source geometric views. This research investigates such an approach that gives engineers a simple set of geometric representation and reasoning tools, or a language, to specify the transformation of geometric source views into useful task-specific geometric dependent views. Such tools could be used to augment existing project model approaches, or they could be used to generate a project model dynamically, as discussed in

Haymaker et al (2003b). In the next section, we review related research relevant to the automatic construction of task-specific project views.

3 Related Work in Model Based Reasoning

Enabling task-specific views of evolving multidisciplinary project information remains an active area of research. We categorize the relevant literature into representation approaches and reasoning and management approaches.

Representation approaches define model schemas that contain concepts required by engineers to formally describe project information in a particular domain. The Industry Foundation Classes (IFCs) are emerging as a standard representation schema for the AEC industry (IAI 2003). Using the IFCs, Engineers construct building models by constructing instances of the concepts defined in these schemas. Using knowledge of the schema, other engineers enter and retrieve information from the IFCs model that is relevant to their particular tasks. Some of the difficulties associated with relying solely on pre-defined representational approaches are:

- Pre-defined schemas grow large and difficult to manage as the coverage increases. For example, the IFCs version 2X currently defines over nine hundred concepts. However, it does not yet contain any formalization of deck attachments, or cantilever conditions.
- No one schema satisfies all engineers' tasks, as engineers conceptualize AEC projects in different ways (Turk 2001). For example, the deck detailer needs the concept of "top edge of beam" which the steel detailer does not explicitly need or provide.
- Dependencies in the project information must be noticed and manually addressed (Eastman and Jeng 1999). For example, changes to a beam or slab result in changes to deck attachment conditions.
- New concepts emerge as the project progresses that must be integrated with the existing project information. Deck attachments and cantilever conditions became an important concern on this project only as the design progressed.

To address these difficulties, some researchers investigate reasoning and management approaches to formalize dependencies between project information.

Considerable research involving reasoning about project information has been performed in the context of single AEC tasks. This body of work formalizes representation, reasoning, and management approaches that construct task-specific dependent views from information in source views. For example, some projects at the Stanford Center for Integrated Facility Engineering (CIFE) include: Darwiche et al (1988) perform model-based reasoning to produce a construction schedule; Akinci et al (2000) analyze a 4D model to infer time-space conflicts for workspaces; Akbas at al (2001) analyze project geometry with productivity constraints to determine daily work zones; Fischer (1993) analyzes product models for constructability; Han et al (2000) analyze an IFCs-based project model for handicapped accessibility; Korman and Tatum (2001) perform MEP coordination; and Staub-French et al (2002) formalize the automation of cost analysis. Outside CIFE many others have created similar model-based reasoning systems: For example, Dym (1988) performs automated architectural code checking; and Shea and Cagan (1999) design novel roof trusses using shape-annealing techniques. Others (Flemming and Woodbury 1995, Aouad et al 1997, Haymaker et al

2000) perform a series of design tasks around a central model. Generally, in all these systems, a computer programmer with engineering knowledge programs task-specific reasoning that transforms information in a source view into task-specific dependent views that are limited to the concepts formalized by the programmers.

Other approaches to constructing task-specific views of project information are more generic. Query languages and approaches (Date and Darwen 1993, Hakim and Garrett 1997) enable the automatic transformation of source information into dependent information. However, existing query languages are not used broadly in AEC practice today to construct dependent geometric views from source geometric views. This is in part because these languages rarely define geometric transformations engineers find useful, and in part because there is not a framework that enables engineers to define and manage these dependencies.

Other research formalizes generic geometric reasoning that constructs geometric views of, or adds geometry directly to, a model. Shape Grammars (Stiny 1980) define rules that match a design to the left-hand side of a rule and add geometry to the design according to the right-hand side of the rule. Others (Argarwal and Cagan 1998, Duarte 1999) have used these rules to automate the construction of various complex products. Feature Recognition (Dixon and Poli 1995) identifies and formally represents instances of feature classes in a geometry model. Parametric techniques (Shah and Mäntyla 1995) define sets of related numeric or symbolic equations that can be solved to realize feasible designs. Commercially available parametric modelers, such as CATIA, provide tools to assist engineers to generate 2D sketches from which 3D shapes are parametrically generated and to specify the assembly of physical components parametrically with respect to the positions of other components. Some systems employing parametric techniques are being commercially introduced specifically for the AEC industry, such as Xsteel (Tekla 2003), Revit (Autodesk 2003), and TriForma (Bentley 2003). While some successes are being reported within the context of single domains, parametric techniques are not being widely used in the AEC industry to integrate the work of multiple disciplines. This is because, as currently formalized, these techniques have not mapped well to the multi-disciplinary, constructive, iterative, and unique nature of AEC projects, i.e., they do not enable engineers to easily and formally construct new views from information in other engineers' views.

Some recent parametric approaches in the mechanical engineering domain develop tools to enable the rapid development of new dependencies between information. For example, A-Teams (Talukdar et al 1996) is a problem solving architecture in which agents are autonomous and modify each other's trial solutions. Exemplars (Bettig et al 2000) describe complex situational patterns and extract information of interest. These approaches use reasoning to construct information in one view of a model from information in other views of the model. The research presented in this paper shares a similar goal, but specifically develops tools to assist engineers in constructing a geometric view by formalizing the existence and nature of its dependency on other geometric views; it formalizes how to apply this method iteratively and at multiple levels of detail to compose formal transformations of source views in order to construct dependent views.

4 Perspectors: Formal, Reusable Reasoning to Automate the Construction of a Dependent Perspective from Source Perspectives

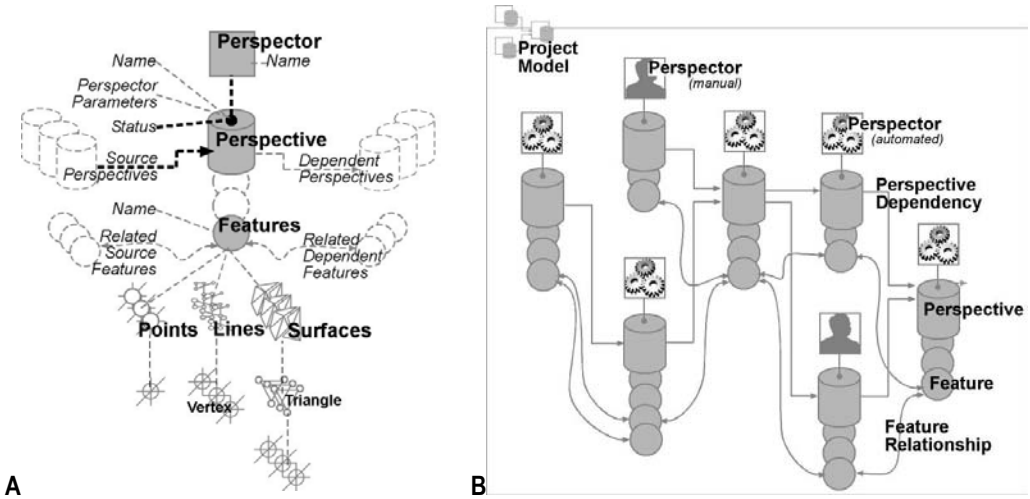
A geometric Perspector is a reusable geometric reasoning mechanism that engineers can modify, compose, and subsume into higher level geometric Perspectors in order to construct a view, called a geometric Perspective, from other geometric Perspectives. Haymaker et al (2003b) formalize the concept of geometric Perspective. After a brief review of this formalization, this paper focuses on the formalization of geometric Perspectors.

> *A* geometric ***Perspective*** *is a task-specific geometric engineering view that formalizes its dependency on other Perspectives.*

We diagram the formalization of a geometric Perspective in Figure 4A. A Perspective is like many CAD layers today: It has a name, and contains any number of named geometric ***Features*** that use Surfaces, Lines, Points, and relationships to Features in other geometric Perspectives to describe engineering concepts. The definition of Feature used in this research is consistent with that of Dixon and Poli (1995). The representation is simple and adequately expressive for the test cases; however, there is a trade-off between conceptual simplicity and the coverage of the representation. More complex data types, such as NURBS or Solid Models, or deeper or more complex hierarchies or relationships of Features could be formalized as part of a geometric Perspective. This would enable an engineer to describe concepts such as curvilinear beams, or the fact that these beams are solid. Instead we chose to develop reasoning around the simple, feature-based geometric representation we describe to scope the research, to lessen the learning curve for engineers who are neither computer programmers nor geometry experts, and because the test cases require only these data types.

A Perspective is different from CAD layers because it formalizes its dependency on other Perspectives. We formalize this dependency into three parts:

- *Existence*: An ordered list of references to the source Perspectives on which this Perspective depends. For example, in the test case a Deck Attachments Perspective depends on the Concrete Slabs and Steel Framing Perspectives. A Perspective also maintains relationships to dependent Perspectives, which a Perspector uses when modifying a Perspective to notify its dependent Perspectives (iteratively down the graph) that they are “Not_Integrated”.
- *Status*: A single integer to represent the integration status of the Perspective with respect to its source Perspectives as the design evolves (0 = Integrated, 1 = Not_Integrated, 2 = Being_Integrated). For example, after a steel beam in the Steel Framing Perspective is modified, the Deck Attachments Perspective is “Not_Integrated.”
- *Nature*: A relationship to the Perspector that formalizes the reasoning to construct Features in this Perspective from Features in the source Perspective(s). For example, reasoning that constructs deck attachments in the Deck Attachments Perspective from slabs and beams in the Concrete Slabs and Steel Framing Perspectives.

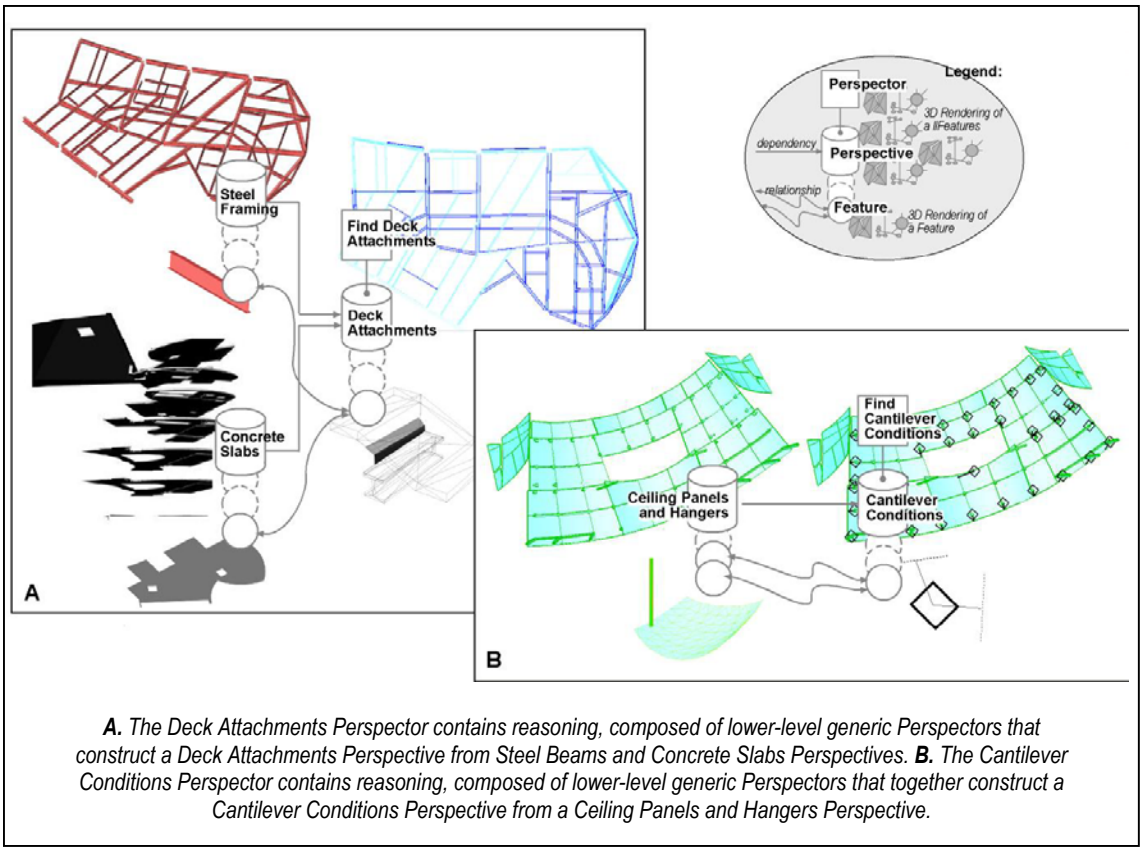


A. *The formalization of a geometric Perspective: a generic geometric view that formalizes the existence (source Perspectives), status and nature (Perspector) of its dependency on other Perspectives.* ***B.*** *Perspectives (and therefore Perspectors) are composable. A Perspector can be either automated or manual; the focus of this paper is on automated Perspectors. Because a Perspector's input and output are both Perspectives, a graph of Perspectors can be subsumed into one Perspector.*

Figure 4: Perspectives: Formalizing the dependency between views.

A geometric ***Perspector*** *is a reasoning mechanism that analyzes the geometric Features in source Perspective(s), to construct geometric Features in a dependent Perspective, and relates these Features to Features in the source Perspective(s).*

Because every Perspective has one associated Perspector, together they can be composed into Perspector Graphs to specify complex transformations of source Perspectives into a dependent Perspective (Figure 4B). Each Perspector can formalize an automated transformation, or the Perspector algorithm can simply provide CAD tools to an engineer to perform the needed transformation. The focus of this research is on automated Perspectors, but manual Perspectors can also be easily incorporated into a Perspector Graph. Because a Perspector's input and output are both Perspectives, a graph of Perspectors (and their associated Perspectives) can be subsumed into one Perspector to represent higher-level transformations. To enable greater reuse of Perspectors, a Perspective can also specify Perspector Parameters that its Perspector uses when constructing its Features. For example, an Extrude Perspector (shown and described in Figure 6C) uses a Parameter to specify the distance to extrude the line along the normal vector that is contained in each source Feature. Therefore the Extrude Perspector can be reused in other contexts, such as in Figure 6G, using a different extrusion distance. Once composed, an engineer can iteratively modify a Perspector Graph by adjusting parameters, or by reconstructing portions of the graph.



A. *The Deck Attachments Perspector contains reasoning, composed of lower-level generic Perspectors that construct a Deck Attachments Perspective from Steel Beams and Concrete Slabs Perspectives.* ***B.*** *The Cantilever Conditions Perspector contains reasoning, composed of lower-level generic Perspectors that together construct a Cantilever Conditions Perspective from a Ceiling Panels and Hangers Perspective.*

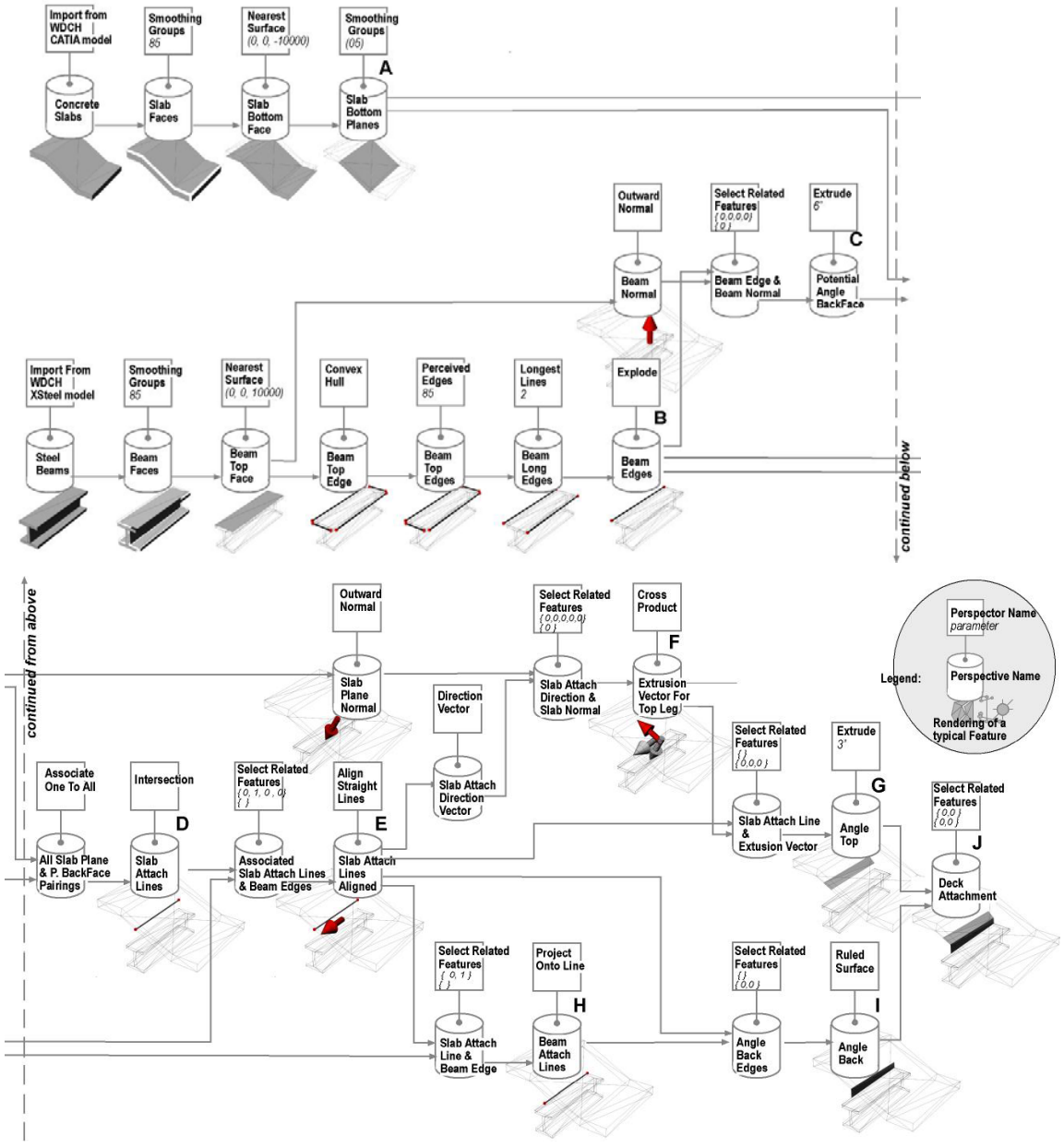
Figure 5: Applying Perspectors to the test cases.

In Section 4.1, we compose reusable geometric Perspectors to construct a Deck Attachments Perspective from Steel Beams and Concrete Slabs Perspectives. Figure 5A shows these reusable Perspectors subsumed into the "Find Deck Attachment Perspector". In Section 4.2, we compose reusable Perspectors to construct the Cantilever Conditions Perspective from the Ceiling Panels and Hangers Perspective. Figure 5B shows these reusable Perspectors subsumed into the "Find Cantilever Conditions Perspector". These graphs become a formal specification that produces repeatable results.

This research developed the Perspectors needed to solve the two test cases described, and investigated whether Perspectors can be reused for different engineering tasks: For example, automating the design of deck attachments between slabs and beams, and automating the analysis of an architectural ceiling system for cantilever conditions between architectural ceiling panels and their hangers. Each of the reusable Perspectors we have formalized to date has involved selecting, reformulating, or generating from the Features in the source Perspectives to construct Features in the dependent Perspective. We composed these Perspectors into Perspector Graphs and then subsumed these graphs into higher-level Perspectors that involve a combination of these selection, reformulation, and generation Perspectors.

4.1 Deck Attachment Perspector

In this section, we compose a collection of reusable Perspectors in a graph that constructs a Deck Attachments Perspective from Concrete Slabs and Steel Beam Perspectives. The Graph compares the locations and orientations of the bottom faces of the slabs to the top faces of the beams and constructs a deck attachment where these faces are near to each other, but not flush. In Figure 6, we show the graph and describe in the caption how these Perspectors incrementally construct the Deck Attachments Perspective. This graph is subsumed into the Find Deck Attachments Perspector in Figure 5A above. In Section 4.3, we describe each individual generic Perspector.



A. *Reformulate each Slab Feature into individual faces, and select the Bottom Face of each slab.*

B. *Reformulate each Beam Feature into faces, select the Top Face of each beam, reformulate this Top Face into the Two Longest Edges, and Reformulate (explode) these into individual Beam Edge Features.*

C. *Select each Beam Edge and the Top Face Normal of the respective Beam Top Face, and generate each Potential Angle Back Face by extruding this Beam Edge along this Normal by six inches.*

D. *Select each Potential Angle Back Face and each Slab Bottom Face. Wherever an intersection occurs between pairs, generate a Slab Attach Line Feature representing where the Deck Attachment should connect to the Slab.*

E. *Reformulate each Slab Attach Line so that it is aligned with its associated Beam Edge. This is necessary because the Beam Edges (due the way the Convex Hull Perspector constructs its Features) have an implicit orientation in their vertices that is counterclockwise around the Beam Face (looking from above). Assuring the Slab Attach Edge also contains this orientation (the lines may be pointing in opposite directions, because the Intersection Perspector is working on surfaces, which have no direction) enables a Perspector to take a cross product of this direction (in Step F) and the Bottom Face of the Slab to find the appropriate direction to extrude the Slab Attach Line, so that it extrudes away from the interior of the Beam Face.*

F. *Select each Slab Attach Direction Vector and the corresponding Slab Bottom Face and reformulate these into the Extrusion Vector for Each top leg, by taking the cross product.*

G. *Select this Extrusion Vector and the corresponding Slab Attach Line and extrude this Line along the Vector by 3 inches to generate the Angle Top.*

H. *Select each Slab Attach Line and the corresponding Beam Edge and project the former onto the latter, to generate each Beam Attach Line.*

I. *Select each Slab Attach Line and the corresponding Beam Attach Line and rule a surface between these lines to generate each Angle Back.*

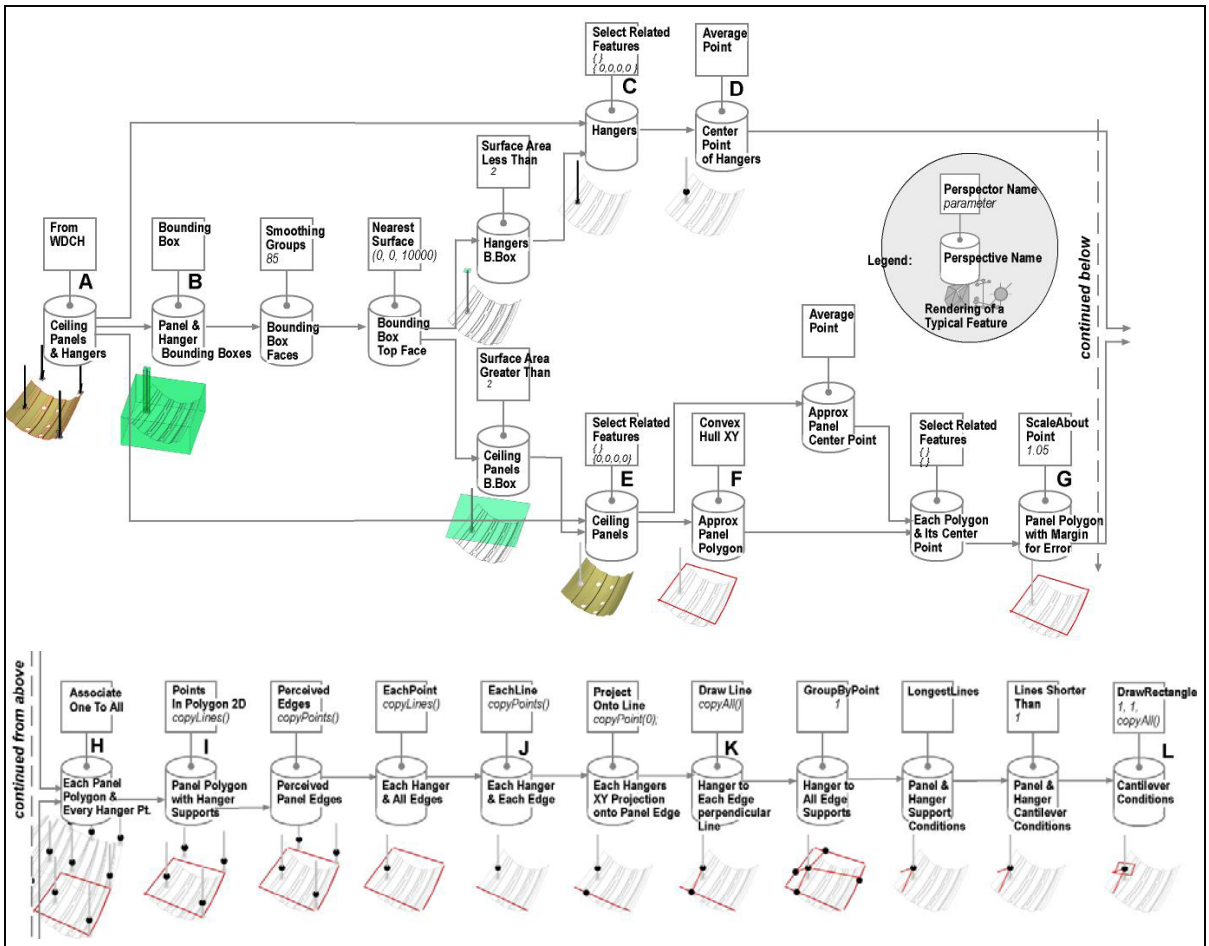
J. *Select each Angle Back and the corresponding Angle Top to generate each Deck Attachment.*

Figure 6: A composition of generic Perspectors constructs the Deck Attachments Perspective from Steel Beams and Concrete Slabs Perspectives.

To summarize, the engineers on the WDCH needed to construct a new task-specific view (Deck Attachments) that was not predefined in the schema of the architect or steel detailer. Lacking simple, formal methods to specify to the computer how to construct a dependent engineering view from source engineering views, these engineers constructed the Deck Attachments view manually: an error-prone, time-consuming and difficult process. To address this difficulty, we show how engineers composed and modified reusable geometric Perspectors, and subsumed this graph into a higher-level Find Deck Attachments Perspector that automatically constructs a Deck Attachments Perspective on the current and then on subsequent projects

4.2 Cantilevered Ceiling Panel Perspector

In this section, we compose a Perspector Graph (Figure 7) that constructs the Cantilever Conditions Perspective from the Ceiling Panels and Hangers Perspective. We reused several Perspectors from the Deck Attachment Perspector Graph. Because of a technical difficulty in transferring data, the formal knowledge about which features described a ceiling panel or a hanger was lost. Rather than address the technical difficulty, we chose to demonstrate the power of Perspectors by re-constructing the lost knowledge using simple feature recognition. The Perspectors in the Graph first determines which of the Features are Ceiling Panels, and which are Hangers. The Perspector Graph then reformulates each Hanger into a center point, and each Ceiling Panel into a polygon describing the Panel's boundary. Finally the Perspector Graph determines which Hanger points are inside (or very close to inside) which Panel's polygon, establishing a support relationship, and then measures the distance of the point to the edges of the Panel, determining which of these support conditions are cantilevered. We subsumed this Perspector Graph into the Cantilever Conditions Perspector in Figure 5B. In Section 4.3 below, we describe each individual Perspector.



- ***A.*** *The original Perspective contains both Ceiling Panels and Hangers (although nothing else), with no semantic information distinguishing between the two.*
- ***B.*** *Generate a Bounding Box around each Feature.*
- ***C.*** *Reformulate these Bounding Boxes into Faces, generate the area of the Top Face of the Bounding Box, and select the original associated Feature (at A), copy the geometry, and label this Feature a Hanger if this area is less than 2 square feet.*
- ***D.*** *Reformulate each Hanger into the Center Point of the Hanger.*
- ***E.*** *Similar to C. Select the original related Feature (at A), copy the geometry, and call this Feature a Panel, if the area of the Top Face of the Bounding Box is greater than 2 square feet.*
- ***F.*** *Reformulate this Panel into an approximate representation of the edges of each Panel using a convex hull algorithm that generates a polygon (formalized as a closed Line) around these Surfaces.*
- ***G.*** *Reformulate the Edge Polygon of each Panel by scaling it around its respective Center Point. This will provide a margin of error for subsequent calculations.*
- ***H.*** *Select each Panel Edge, and every Hanger Center Point.*
- ***I.*** *Select the Points with an XY projection that is contained within the XY projection of the Panel Edges. This represents which Hangers support a Panel. An individual Hanger can support more than one Panel.*
- ***J.*** *Reformulate the Panel Polygon into distinct edges, and select each Panel Edge and Hanger Center Point.*
- ***K.*** *Generate the projection of the Hanger Center Point onto the Panel Edge, and generate a Line from the Center Point to this Projection Point. This Line represents the perpendicular distance from the Center Point to each respective edge.*
- ***L.*** *Select the four perpendicular distance Lines corresponding with each Hanger Center Point and select the two shortest of these; select the Lines that are longer than the cantilever distance limit (provided as a Perspector Parameter by the engineer). Generate a square (formalized as a closed Line) around the corresponding Point in the Feature. These are the Cantilever Conditions.*

Figure 7: The Perspector Graph to construct a Cantilever Conditions Perspective from a Ceiling Panels and Hangers Perspective.

To summarize, the panel detailer on the WDCH needed a task-specific view (Cantilever Conditions) that was not predefined in the schema of the architect or other engineers designing the ceiling system. Because the engineer had no tools to specify to the computer how to construct a dependent geometric engineering view from source engineering views, the engineers did not construct and maintain an explicit view of these cantilever conditions. To address this difficulty, we show how engineers could modify and compose reusable geometric Perspectors, and subsume this graph into a higher-level Perspector, in order to construct a Cantilever Conditions Perspective automatically.

4.3 Description of reusable Perspectors implemented in the research

In this section we describe the reusable Perspectors that we composed in the previous sections to construct the Deck Attachment and Cantilevered Conditions Perspectives. We first describe the generic characteristics and behavior common to all Perspectors.

Generic Perspector: Every Perspector analyzes the Features in source Perspectives to construct Features in the dependent Perspective. To do this, every Perspector does the following steps every time it runs:

1. Check that each source Perspective's Integration Status is INTEGRATED. Request that Perspective to run its Perspector if it is not. (This check back-propagates upstream through the Perspector Graph.)
2. Set the dependent Perspective's Status to BEING_INTEGRATED.
3. Construct dependent Features in the dependent Perspective from source Features in the source Perspective(s) and relate the dependent Features and the source Features. Use the dependent Perspective's Perspector Parameter (if provided by the engineer).
4. Set the dependent Perspective's Status to INTEGRATED.
5. Set the Status of all subsequent dependent Perspective's to NOT_INTEGRATED. (This step forward-propagates downstream through the Perspector Graph.)

In addition to any Perspector Parameter that is specific to a particular Perspector, all Perspectors include several generic parameters that an engineer can specify to modify a Perspector's construction. These parameters call methods that copy one or several Surfaces, Lines, and/or Points from the source Feature(s) to the dependent Feature(s) during construction of the dependent Perspective. For example, in Figure 7K, the Draw Line Perspector takes a source Feature containing two or more ordered Points and constructs a dependent Feature containing a Line. However, for subsequent Perspectors, it is desirable to keep the Points and the other Line in the dependent Feature. We therefore called copyAll(), which copies the entire source Feature's geometry into the dependent Feature's geometry.

The next sections present the reusable Perspectors that we implemented to solve the test cases. The Perspectors are categorized in terms of whether they perform Selection, Reformulation, or Generation from the source Features when constructing the dependent Features. The name of the Perspector is in **bold**; Perspector Parameters are in parentheses if the Perspector uses them. The algorithms contained in the Perspectors described below are not new contributions; similar algorithms are implemented in many CAD programs, such as CATIA (Dassault 2003). The contribution to AEC project modeling is the generic formalism of Perspectors that make these algorithms composable

and accessible to engineers from multiple disciplines on unique AEC projects to construct a dependent Perspective from source Perspectives.

4.3.1 Selection Perspectors

Associate One To All: For each source Feature in the first source Perspective, construct a dependent Feature in the dependent Perspective that contains the geometry of this source Feature and every source Feature in the second source Perspective. Relate the dependent Feature to all related Features.

Each Line: For each Line in each source Feature, construct a dependent Feature containing that Line in the dependent Perspective. Relate each dependent Feature to the source Feature.

Each Point: For each Point in each source Feature construct a dependent Feature containing that Point in the dependent Perspective. Relate each dependent Feature to the source Feature.

Explode: For each Feature, for each Surface, Line, and Point in the Feature, construct a dependent Feature in the dependent Perspective containing just that Surface, Line, or Point. Relate each dependent Feature to the source Feature.

Group By Point (Int PointNumber)**:** Find all source Features that have the same Point (the same X,Y,Z position) in the PointNumber position of the source Feature's Points. Group the geometry of these Features into one dependent Feature. Relate the dependent Feature to each related source Feature.

Lines Shorter Than (Float distance)**:** For each Feature containing a collection of Lines, construct a dependent Feature that contains the Lines that are shorter than *distance*, and relate the dependent Feature to the source Feature.

Longest Lines (Int number)**:** For each source Feature containing a collection of Lines, order the Lines by length, then construct a dependent Feature that contains the longest *number* of Lines, and relate the dependent Feature to the source Feature.

Nearest Surface (Vertex v1, Vertex v2, Vertex v3)**:** For each Feature in the source Perspective construct a dependent Feature in the dependent Perspective that contains the Surface in the source Feature that is closest to the plane defined by *v1, v2, v3*, and relate the dependent Feature to the source Feature.

Points In Polygon: For each source Feature that contains a closed Line and a collection of Points, construct a dependent Feature that contains the same Line, and the Points that lie within this polygon, and relate the dependent Feature to the source Feature.

Related Feature (Int [ ] path1, Int [ ] path2)**:** For each Feature in the first source Perspective, construct a dependent Feature that associates it with the source Feature in the second source Perspective that shares a related Feature. *Path1* and *Path2* define the paths through the source Feature relationships to check for this relationship. For example, in Figure 6C, just prior to extruding the potential back faces, this Perspector is used to associate each Beam Edge and Outward Facing Normal that are Features of the Beam Top Face and relate the dependent Feature to the source Features.

Shortest Lines (Int number)**:** For each source Feature containing a collection of Lines, order the Lines by length, then construct a dependent Feature that contains the shortest *number* of Lines, and relate the dependent Feature to the source Feature.

Surface Area Greater Than (Float size)**:** For each Feature that contains a Surface with an area that is greater than *size* construct a dependent Feature that contains that Surface and relates the dependent Feature to the source Feature.

Surface Area Less Than (Float size)**:** For each Feature that contains a Surface with an area that is less than *size* Construct a Feature that contains that Surface, and relate the dependent Feature to the source Feature.

4.3.2 Reformulation Perspectors

Align Straight Lines: For each source Feature align the first Line with the second Line, and construct a dependent Feature that contains the aligned second Line, and relate the dependent Feature to the source Feature.

Bounding Box: For each Feature, construct a dependent Feature containing a bounding box surrounding all the geometry in the source Feature, and relate the dependent Feature to the source Feature.

Convex Hull XY: For each Feature in the source Perspective that contains at least one instance of geometry, construct one closed Line in the dependent Feature, representing the convex hull of all the geometry in the source Feature, and relate the dependent Feature to the source Feature. Although the algorithm does not consider the z value of any vertex while calculating the hull, it maintains the z values of all Points selected in the hull.

Cross Product: For each Feature, construct a dependent Feature containing a point that is to be interpreted as the cross product of the vectors stored in the first and second Points of the source Feature.

Direction Vector: For each Feature, for each Line in the Feature, create a dependent Feature containing a Point that is to be interpreted as a Direction Vector or average orientation of the Line, and relate the dependent Feature(s) to the source Feature.

Smoothing Groups (Float creaseAngle)**:** For each source Feature that contains at least one Surface, construct a dependent Feature in the dependent Perspective breaking the source Surface into individual Surfaces wherever the crease angle between the normals of adjacent triangles is greater than the value of *creaseAngle*, and relate the dependent Feature to the source Feature.

Outward Normal: For each source Feature, construct a dependent Feature containing a Point that is to be interpreted as the average outward normal of the Surface(s), and relate the dependent Feature to the source Feature.

Perceived Edges (Float creaseAngle)**:** For each source Feature, for each Line in that Feature, construct a dependent Feature that contains a Line for each perceived Line segment. Perceived Line segments are created where the angle between segments in the source Line exceeds the value of creaseAngle. Relate the dependent Feature(s) to the source Feature.

4.3.1 Generation Perspectors

Draw Line: For each Feature, construct a dependent Feature containing a Line that connects each Point in the source Feature and relate the dependent Feature to the source Feature.

Draw Rectangle (Float X, Float Y)**:** For each source Feature construct a dependent Feature containing a Line describing a rectangle around the first Point, and relate the dependent Feature to the source Feature.

Extrude (Float distance)**:** For each source Feature, extrude all the Lines in the Feature along the direction of the normal that is stored in the first Point of the Feature; construct a dependent Feature containing the resulting Surface(s), and relate the dependent Feature to the source Feature.
Intersection: For each source Feature, find the intersection between the first Surface and the second Surface, construct a dependent Feature containing the resulting Line.
Project Onto Line: For each source Feature, project all the Feature's geometry onto the first Line in the source Feature. Construct a dependent Feature containing the resulting Lines (If Lines or Surfaces were projected) and/or Points (if Points were projected), and relate the dependent Feature to the source Feature.
Ruled Surface: For each source Feature, rule a Surface between the first and second Lines in the source Feature, construct a dependent Feature containing this Surface, and relate the dependent Feature to the source Feature.
Scale About Point (Float scaleFactor)**:** For each source Feature, construct a dependent Feature containing the source Feature's geometry scaled about the first Point in the source Feature by scaleFactor, and relate the dependent Feature to the source Feature.

5 Results: Faster, More Accurate Dependent Geometric Views on the WDCH

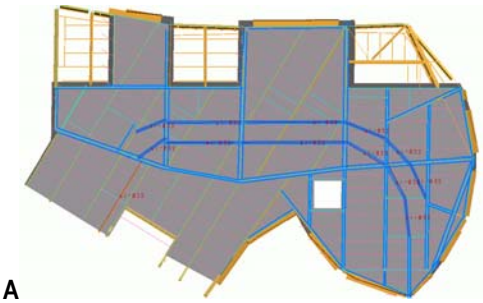
The test cases from the WDCH establish that engineers could use simple formal methods to specify the automatic construction of a new dependent geometric view from information in many source geometric views. Section 4 formalizes: (1) the concepts of Perspector and Perspective; (2) several reusable Perspectors; and (3) composed graphs of these reusable Perspectors that automatically construct a useful dependent Perspective for the test cases. This section reports the results of implementing these methods in a prototype called PerspectorApp, described in Haymaker et al (2003b), in which engineers select from a predefined collection of reusable geometric Perspectors, compose them into graphs, and construct and control the integration of dependent Perspectives as source Perspectives change. We implemented the reusable Perspectors described in Section 4.3, and composed the Perspector Graphs described in Section 4.1 and 4.2 to automatically construct the Deck Attachments and Cantilever Conditions Perspectives from Perspectives containing Features that were imported from the WDCH project.

Figure 8A shows an as-built model of the deck attachments installed by the WDCH engineers on the roof of a portion of the project called Element 2. All of the deck attachments on this roof required field welding. Figure 8B shows the deck attachment Features constructed by the Find Deck Attachments Perspector in PerspectorApp. Figure 8C shows a table that compares the results of the WDCH engineers and Perspectors on the Element 2 roof. Perspectors found 114 deck attachment conditions whereas the WDCH engineers installed (using field welding) 86. The 28 false positives are due to two factors: First, several of the deck attachments constructed by Perspectors were constructed on stiffener beams, which do not require deck attachments (these beams resist buckling of the longer beams, and they are not needed to support the metal deck). Second, due to the iterative nature of AEC processes, the position of a few of the beams had to be modified by the engineers very late in the design process. We did not receive an updated geometry model from the design team, so those results are on an older version of the model with slightly different beam conditions than the as- built conditions. This modified geometry was also the reason for the three false negatives. A

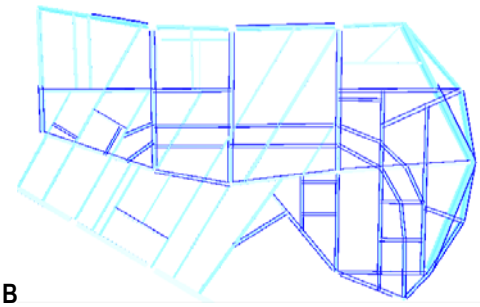
visual inspection shows that Perspectors found all the deck attachments required by the detail shown in Figure 2D and are therefore functionally very accurate.

Figure 8C also shows the amount of detail the WDCH engineers manually constructed to describe the deck attachments, versus the amount of detail constructed using automated Perspectors. The WDCH designers modeled deck attachment as a line, from which a fabricating engineer determined the length and location of each deck attachment; however, further work was required to determine the size of the deck attachment. The Find Deck Attachments Perspector constructs two surfaces of the deck attachments. This added detail contains additional information to establish the length, location, and size of each deck attachment.

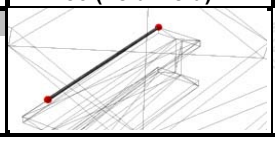
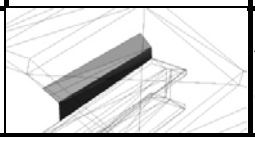
A



B



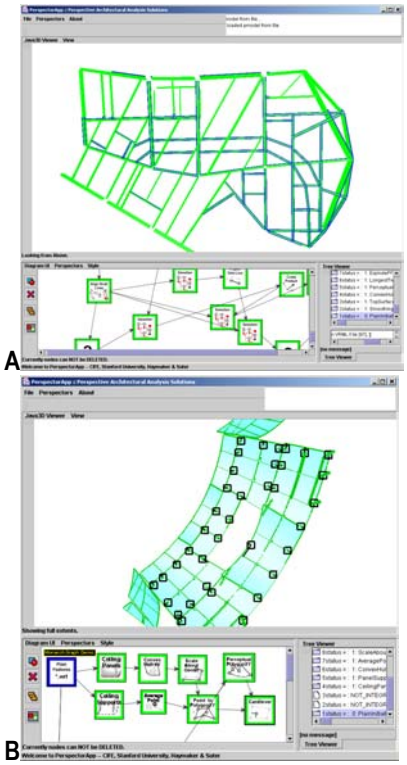
C

	WDCH	Perspectors	Comments
Accuracy			Significant improvement over current practice. Further improvement possible.
- Found in test case	**0 (shop weld)**	**114**	
- Missed in test case	**86 (field weld)**	**2**	
Completeness			
- Amount of detail			Automation with Perspectors makes creating more detail cost-effective.

***A**. The as-built condition of the deck attachments on the Element 2 roof (overlaid on concrete and steel) provided by Mortenson. All deck attachments on the Element 2 roof required field welding. **B.** The deck attachments constructed by Perspectors (overlaid on steel). **C**. The table compares the Accuracy and Completeness of the WDCH design team and Perspectors.*

Figure 8: Comparing current practice (WDCH) to Perspectors on the deck attachment test case.

Figure 9A shows PerspectorApp used to implement the deck attachment test case. Slab and beam features can be iteratively modified, causing the Deck Attachment Perspective's integration status to be automatically set to "Not_Integrated". The Find Deck Attachments Perspector can then be run to reconstruct the Deck Attachment Perspective. Figure 9B shows PerspectorApp used to implement the cantilever condition test case. Ceiling Panels and Hangers can be iteratively modified, causing the Cantilever Conditions Perspective's Integration Status to be automatically set to "Not_Integrated". The Find Cantilever Conditions Perspector can then be run to reconstruct the Cantilever Conditions Perspective. Figure 9C shows that seven of the reusable Perspectors are used in both Perspector Graphs, and six of the geometric Perspectors are re-used more than once in the same Perspector Graph. The high performance on these complex, industrial test cases provides evidence for the power of Perspectors.



A

B

C

Selection	Find Deck Attachments	Find Cantilever Conditions
Associate One To All	1	1
Each Line	0	1
Each Point	0	1
Explode	1	0
Group By Point	0	1
Lines Shorter Than	0	1
Longest Lines	1	0
Nearest Surface	2	1
Points In Polygon XY	0	1
Select Related Feature	7	3
Shortest Lines	0	1
Surface Area Greater Than	0	1
Surface Area Less Than	0	1
Reformulation		
Align Straight Lines	1	0
Average Point	0	2
Bounding Box	0	1
Convex Hull XY	1	1
Cross Product	1	0
Direction Vector	1	0
Smoothing Groups	3	1
Outward Normal	2	0
Perceived Edges	1	1
Generation		
Draw Line	0	1
Draw Rectangle	0	1
Extrude	2	0
Intersection	1	0
Project Onto Line	1	1
Ruled Surface	1	0
Scale About Point	0	1

A. *The deck attachment Perspectors implemented in PerspectorApp* ***B****. The Cantilever Condition Perspectors implemented in PerspectorApp.* ***C****. Many Perspectors were reusable in both test cases, or multiple times in one test case (shown in gray).*

Figure 9: The same method is applied to both test cases.

6 Discussion: Better, Faster, Integrated Project Views for AEC

This research has shown that engineers can use simple formal approach to automatically construct a dependent view from information in source views to support their tasks with timely, accurate geometric information that is integrated with geometric information that is constructed by other engineers. This approach could be used to augment predefined central or federated project model approaches, or as demonstrated in this paper, the approach can be used to construct and control an emerging, integrated, multidisciplinary project model. We: (1) formalized a generic task-specific engineering view, called a geometric Perspective, and a generic reusable reasoning module, called a Perspector; (2) showed that engineers can modify, compose into graphs, and subsume geometric Perspectors; (3) implemented these test cases in a prototype called PerspectorApp to show that Perspector Graphs can automatically construct useful Perspectives; and (4) collected empirical test data that show the performance of this system on two complex industrial test cases. Such integrated views could have enabled the WDCH engineers to more efficiently and cost-effectively plan, design, and execute their project.

The power of Perspectors is demonstrated by the deck attachment test case. Engineers select from predefined geometric Perspectors and compose them into a Perspector Graph, which automatically constructs a Deck Attachments Perspective more accurately and with more useful detail than current, manual practice allows. The Perspector Graph did yield some false positives in the results: At a minimum, an engineer using manual Perspectors could remove these false positives more easily than manually constructing all the deck attachments, or he could assemble more Perspectors to remove these false positives automatically. Once composed, the Find Deck Attachments Perspector becomes a formal procedural specification for the deck attachment conditions, which can be reused on subsequent projects. The modularity may prove powerful on multi-disciplinary AEC projects, where organizational boundaries create liability issues. Enabling engineers to flexibly determine who produces what information, and in what format is an important issue in designing integrated project models. For example, a design engineer may only want to be responsible for representing the attachment as a line along the beam. Using geometric Perspectors, the fabricating engineer can then generate the remaining information.

The generality of Perspectors is demonstrated by the cantilever conditions test case. The Perspector method applies to two very different kinds of problems: automating the design of deck attachments between slabs and beams, and automating the analysis of an architectural ceiling system for cantilever conditions between the panels and the hangers. In addition, Figure 9C also shows that several Perspectors are used in both test cases. Depending on the programmer and the required construction of Features, a Perspector could be programmed in a number of minutes to a number of hours. However, their reuse suggests that a language of reusable Perspectors can emerge. The development of such a language is enabled by the modularity of Perspectors, as additional Perspectors can be programmed and added to PerspectorApp, and because they use geometric Perspectives that are based on feature-based geometric views in use today. Engineers may be able to learn to compose, modify, and subsume these Perspectors in order to construct many different kinds of useful dependent Perspectives of changing source Perspectives. Engineers could construct deck attachments or cantilever conditions Perspectives in different ways, thus composing varying, progressively better, Perspector Graphs. A Perspector Graph can serve as a template for similar problems; for example, the Find Deck Attachments Perspector would serve as a good starting point to define a Perspector that details the connection between beams and curtain wall systems that attach to the beams. Perspectors may enable engineers to generate the task-specific views they need when they need them and to contribute to an overall project model that emerges as a directed acyclic graph of views and dependencies as the design progresses.

The limitations of the Perspector formalism include: (1) Representation: Any representation is an abstraction. The data types implemented in Features are Surfaces, Lines, and Points. Other geometric data types, such as NURBS, Solid Models, and other non-geometric data types, can increase the expressive power of Features, however at the expense of greater complexity for engineers working with the Perspectives and the Perspectors that transform them. (2) Reasoning: While they perform well on the test cases, we have only provisionally tested the individual Perspectors and Perspector Graphs presented in this paper. The research to date is an initial investigation into the power and generality of the approach. It was beyond the scope of this initial research effort to develop an exhaustive catalog or classification

of reusable Perspectors, or to test them on hundreds of types of test cases. Rather, the research goal was to establish whether a formal, modular way to conceptualize constructing a dependent view from source views is feasible. (3) Management: The formalization does not currently support cycles in the dependencies that could support feedback for optimization searches. (4) Implementation: PerspectorApp is currently formalized to run on a single computer, issues of deployment of Perspectors and Perspectives over a network were scoped out of this research.

The future work for the Perspector formalism will focus on developing a language of Perspectors. By applying Perspectors to more AEC design automation and analysis test cases, a better understanding of aspects of this language is expected to emerge: (1) What are the reusable Perspectors that multidisciplinary engineering teams need to construct their task-specific Perspectives of other Perspectives? (2) How general is the current formalization of a Perspective and Perspector? Put another way, in what contexts are new data types such as NURBS, Solid Models, and non-geometric data types and reasoning to handle these data types required? (3) How can Perspectors and Perspectives be distributed over a network to enable multiple engineers to construct task-specific Perspectives from other engineers' Perspectives? The development of such a language is enabled by the modularity of Perspectors, as additional Perspectors can be programmed, or composed, modified, and subsumed, and added to PerspectorApp's catalogue of Perspectors.

In summary, to date it has been difficult to formalize integrated models for AEC projects because of their multi-disciplinary, constructive, iterative, and unique nature. Many researchers are recognizing the need for model evolution, or customization; however, who will do the customizing, and when, has been an open question. Perspectors are a powerful and general approach that can enable engineers to iteratively construct useful dependent geometric views of changing source views of other engineers, enabling multidisciplinary engineering teams to engage in multi-disciplinary but integrated and automated design and analysis. The theoretical contributions of the research are the formal framework to enable engineers to work in this way.

7 References

Akbas, R., Fischer, M., and Kunz, J. (2001). "Formalizing Domain Knowledge for Construction Zone Generation." *Proceedings of the CIB-W78 International Conference IT in Construction in Africa 2001: Implementing the Next Generation Technologies*, CSIR, Division of Building and Construction Technology, Pretoria, South Africa, 30-1 to 30-16.

Akinci, B., Fischer, M., Levitt, R., and Carlson, B. (2002). "Formalization and Automation of Time-Space Conflict Analysis." *Journal of Computing in Civil Engineering*, ASCE.

Aouad, G., Marir, F., Child, T., Brandon. P., and Kawooya. A. (1997). "Construction Integrated Databases: Linking Design, Planning and Estimating." *Proceedings of the International Conference on the Rehabilitation and Development of Civil Engineering Infrastructures*, American University of Beirut, June, 51-60.

Agarwal M. and Cagan J., (1998). "A Blend of Different Tastes: The Language of Coffee Makers." *Environment and Planning B: Planning and Design*, 25:2, 205-226.

Autodesk (2003). *Autodesk Revit*, www.autodesk.com.

Bentley (2003). Microstation Triforma www.microstation.com.

Bettig, B., Shah, J., and Summers, J. (2000). "Domain Independent Characterization of Parametric and Geometric Problems and Embodiment Design." *Proceedings of DETC2000: Design Engineering Technical Conference*, Sept., Baltimore, MD.

Dassault Inc. (2003). *CATIA R7*, www.catia.com.

Darwiche, A., Levitt, R.E., and Hayes-Roth, B. (1988). "Oarplan: Generating Project Plans in a Blackboard System by Reasoning about Objects, Actions, and Resources." *Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 2(3):169-181.

Date, C. J., and Darwen H. (1993). *A Guide to the SQL Standard*, Third Edition, Addison-Wesley Publishing Company, Inc.

Dixon J., and Poli, C. (1995). *Engineering Design and Design for Manufacturing*, Field Stone Publishers, MA.

Duarte, J. P. (1999). "Democratized Architecture: Grammars and Computers for Siza's Mass Housing." *Proceedings of the International Conference on Enhancement of Computational Methods in Engineering and Science, Macau, Elsevier Press.*

Dym, C. L., Henchey, R. P., and Gonick, S. (1988). "A Knowledge-based System for Automated Architectural Code Checking." *Computer Aided Design*, 20(3), 137-145.

Eastman, C., Jeng, T-S. (1999). "A Database Supporting Evolutionary Product Model Development for Design." *Automation in Construction, 8* (3), 305-323.

Fischer, M. (1993). "Automating Constructibility Reasoning with a Geometrical and Topological Project Model." *Computing Systems in Engineering*, 4(2-3), 179-192.

Flemming, U., and Woodbury, R. (1995). "Software Environment to Support Early Phases in Building Design (SEED): Overview." *Journal of Architectural Engineering*, Vol. 1, 147-152.

Granlund Olof Oy (2003). BSPro, www.bspro.net

Hakim, M. M. and Garrett Jr., J. H (1997). "An Object-Centered Approach for Modeling Engineering Design Products: Combining Description Logic and Object-Oriented Models." *Journal of AI in Engineering Design and Manufacturing* (AI EDAM), Vol. 11, 187-198.

Han, C. S., Law, K., and Kunz, J. (2000). "Computer Models and Methods for a Disabled Access Analysis Design Environment." CIFE Technical Report, No. 123, Stanford University.

Haymaker, J., Ackermann, E.; and Fischer, M. (2000). "Meaning Mediating Mechanism: A Prototype for Constructing and Negotiating Meaning in Collaborative Design." *Sixth International Conference on Artificial Intelligence in Design*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 691-715.

Haymaker J., Fischer, M., Kunz, J., and Suter, B., (2003a). "Engineering Test Cases to Motivate the Formalization of an AEC Project Model as a Directed Acyclic Graph of Views and Dependencies Between Views" Working Paper Nr 080, CIFE, Stanford University.

Haymaker J., Suter, B., Fischer, M., and Kunz, J. (2003b). "The Perspective Approach: Enabling Engineers to Construct and Integrate Geometric Views and Generate an Evolving Project Model" Working Paper Nr 081, *Center For Integrated Facility Engineering*, Stanford University.

IAI (2003). "Industry Foundation Classes, Version 2.X." *International Alliance for Interoperability*.

Korman, T. M., and Tatum C. B. (2001). "Development of a Knowledge-Based System to Improve Mechanical, Electrical, and Plumbing Coordination" Technical Report Nr 129, CIFE, Stanford University.

Post, N. (2002). "Movie of Job that Defies Description Is Worth More Than A Million Words", *Engineering News Record,* 248(13), 24.

Post, N. (2003). "Monster Wood Ceiling Crowns City of Angels' Music." *Engineering News Record,* 251(6), 30-33.

Staub-French, S., Fischer, M., Kunz, J., and Paulson, B. (2002). "A Formal Process to Create Resource-loaded and Cost-loaded Activities Related to Feature-based Product Models." Accepted for publication in *The Journal of Advanced Engineering Informatics*.

Stiny, G, (1980). "Introduction to Shape and Shape Grammars." *Environment and Planning B: Planning and Design* 7, 343-351.

Shea, K. and Cagan, J. (1999). "The Design of Novel Roof Trusses with Shape Annealing: Assessing the Ability of a Computational Method in Aiding Structural Designers with Varying Design Intent." *Design Studies*, Vol. 20, 3-23.

Shah, J. and Mäntyla, M. (1995). *Parametric and Feature-Based CAD/CAM*, Wiley & Sons Inc., New York, NY.

Talukdar, S., Baerentzen, L. Goce, A. and derSouza, P. (1998). "Asynchronous Teams: Cooperation Schemes for Autonomous Agents." *Journal of Heuristics*, Vol. 4, 295 – 321.

Turk, Z. (2001). "Phenomenological Foundations of Conceptual Product Modeling in Architecture, Engineering and Construction." *Artificial Intelligence in Engineering*,15 (2).

Tekla (2003). *XSteel*, www.xsteel.xom.

Wilson R. H. and Latombe J. C. (1994). "Geometric Reasoning about Mechanical Assembly." *Artificial Intelligence*, 7:1, 371-396.