Session F Designing and Systems:
Elements of a hierarchical representation for architectural design and decision support

Georg Suter[1] (georg_suter@yahoo.com), Ardeshir Mahdavi[1], Ramesh Krishnamurti[2]

[1]Institut für Hochbau, Abteilung für Bauphysik und Humanökologie, Vienna University of Technology, Vienna, Austria

[2]School of Architecture, College of Fine Arts, Carnegie Mellon University, Pittsburgh, PA, USA

1 Abstract

In computer-aided architectural design, one important research objective has been the development of rich representations for applications that support the early phases of building design. Efforts to address this problem have proven difficult. For one, designers tend to frequently and at times drastically adjust their designs, particularly at the beginning of the design process. In this paper, we review selected features of a representation which was developed for the building performance domain and caters to certain requirements of early design such as rapid manipulation of building models and automatic maintenance of geometric integrity. We discuss in more detail a recently added concept called procedural grouping. It takes advantage of the hierarchical structure of the original representation and enhances the ease of manipulation of building models. Examples from a system prototype illustrate how designers might benefit from such procedural grouping mechanisms.

Keywords: computer-aided architectural design, geometric modeling, procedural models

2 Introduction

As in many other industries, the use of computers has become common in the construction industry. Today, architects and engineers rely on sophisticated drafting, visualization, animation, document processing, and database systems to design, build, and maintain the built environment. Yet, despite considerable advances, building professionals use computers mainly as productivity tools, and information technology is only rarely employed in the early phases of building design, where the more crucial decisions with regard to a facility's shape, materials, and behavior are made.

A domain in which computational support appears particularly promising is building performance analysis. However, building performance simulation tools for energy, lighting, or acoustics, which have been available for a while, typically require advanced computational skills, a high level of accuracy, and considerable domain expertise for proper operation (Hand and Hensen 1995). As a result, simulation tools are often used by specialized professionals rather than architects. In order to make specialized design tools more accessible to designers, researchers have investigated the integration with commercial drafting systems (see, for instance, Mahdavi 1996, Pelletier and Keilholz 1999). There appears to be a consensus that the representations that currently underlie commercial drafting systems have only a limited potential to serve as a front-end for specialized design tools. There are several reasons for this situation, some of which are briefly summarized below (for more details, see Suter and Mahdavi 1998). In the building performance domain, informational completeness is critical, which, for instance, requires the definition of three-dimensional space cells and the designation of building elements such as walls, windows and doors, together with their material properties.

However, conventional drafting systems currently do not support the definition of three-dimensional spaces as required by certain simulation routines. Furthermore, many operations in conventional drafting systems are not scalable. We are not aware of any system that permits the generation of more than one floor, space or opening at a time while ensuring full integrity of the resulting model. The functionality provided by conventional drafting systems only partially exploits the computational redundancy that can be observed in many buildings.

In computer graphics and related fields, various techniques and representation schemes have been employed to represent "a large class of objects by a single model with easily adjusted parameters" (Foley et al. 1992). Among those representations are procedural models, grammar-based models, and particle systems. The representation which was developed for the building performance simulation domain and of which selected features are briefly described in the following, may be best classified as a procedural model (Newell 1975, Suter 1999). The representation incorporates aspects of space-based representations and subregion representations (Kundu 1988, Mahdavi 1996, Harada 1997).

3 Hierarchical representation

3.1 Partitioning

The dominant feature of the proposed representation is the hierarchical organization of spatial information. Designers manipulate building geometry by recursively applying *partitioning rules* to volumes, surfaces, and lines, which are the basic geometrical types. The concept of partitioning is analogous to partitioning regions in a subregion representation (Steadman 1983). An important feature of partitioning is the preservation of information with regard to parent entities, which remain accessible for further manipulation.

Partitioning terminology is introduced with a volume partitioning example. A parent volume, or *super-entity*, is defined as a polyhedron, and its surfaces are referred to as *bounding elements* (Figure 1a). New or reevaluated planes are intersected with those surfaces, thereby creating a volume *partition*. The intersection thus fits properly inside the parent volume (Figure 1b). Each new partition defines a boundary for a new sub-volume, or *sub-entity*. Individual partitions and sub-entities have *attributes* that allow for the attachment of semantic information as required by simulation.

The *partitioning direction* is a vector stored at a parent level. It indicates the orientation for the insertion of new partitions. Partitions are always inserted orthogonal to the partitioning direction and at a specific distance from other partitions. A partitioning direction may be modified by designers anytime, even if partitions already exist. The modification of a partitioning direction triggers a reevaluation of the entities affected by that operation. Making partitioning directions available for manipulation facilitates operations such as mirroring and rotation.

Designers can further modify the *dimensions* of sub-entities. Relevant for determining dimensions are minimum and maximum points at the parent volume level. These points are determined through projection of entity vertices on the partitioning direction line. The first rule describes the creation of child entities at a new level in the hierarchy tree (Figure 2a). It is only applicable to unpartitioned entities. Once a new hierarchy level has been created, existing entities may be modified, or new ones may be added. Figure 2b illustrates an insertion rule. New partitions are always inserted orthogonal to the partitioning direction. This ensures that all partitions are parallel with respect to each other. Existing sibling partitions are forced to adjust

their position to make room for new entities. This is also the case when the dimension of an existing entity is modified. Designers can control the behavior of dimensions and partitioning directions through simple constraints. Rules for partitioning surfaces and lines are analogous. More elaborate rules for generating non-orthogonal geometries also exist but are not covered here (for details, see Suter 1999).

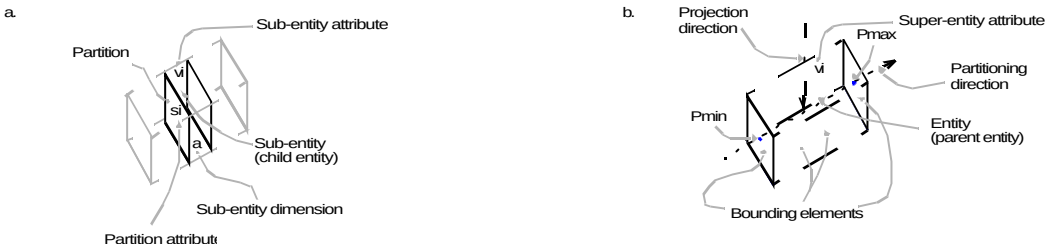


Figure 1. Partitioning terminology

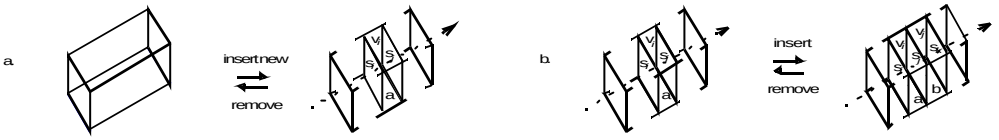


Figure 2. Rules for orthogonal partitioning (shown for volumes)

3.2 Primary hierarchy

As mentioned, partitioning rules can be applied recursively; this lets designers organize spatial information in a nested, ordered hierarchy. Figure 3 illustrates the decomposition of a simple, mostly orthogonal building into volumes, surfaces, and lines. The volume hierarchy is the main hierarchy to which additional surface or line hierarchies may be attached. Together, these hierarchies constitute the *primary hierarchy*. The root volume serves as the initial container for recursive partitioning and refinement, which is performed in the x, y, or z direction. For simplicity, information about bounding elements, partitioning directions, and volume labeling is not included in this illustration. Grayed surfaces indicate labels representing materials and construction types, including air surfaces. All surfaces at the leaves are required to have unique labels. Detailed labeling rules are introduced later.

Surfaces are accessed by traversing the volume decomposition tree. Whenever a surface is further partitioned or refined independent of volume partitioning, a separate surface hierarchy is attached to the original surface, which is included in the volume hierarchy. Similarly, line hierarchies may be attached to an original line in a surface hierarchy.

3.3 Labels

Various types of labels can be attached to geometric entities to identify these as buildings, zones, spaces, openings, construction types, and so forth. Labels are important for integration with simulation, and permit simulation routines to perform selective queries on volume and surface hierarchies for specific semantic information.

Inheritance mechanisms similar to those used in the SPHIGS graphics library facilitate the manipulation of labels (Van Dam and Sklar 1990). These enable designers, for example, to rapidly explore the impact of various enclosure materials on energy use. Designers can assign

default values for volume, surface, and line labels at the root volume. These values are passed down to child entities, which may override them. Overriding labels are not affected by modifications of labels at parent levels.

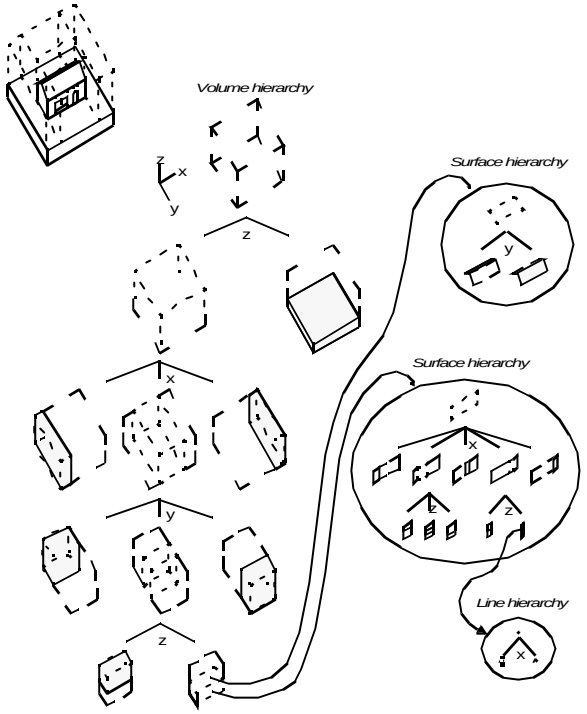


Figure 3. Example for hierarchical decomposition of a house

4 Procedural grouping

4.1 Concepts

The operations described so far take advantage of the hierarchical organization of spatial information. However, this may not include certain relations among building components that are relevant to a designer, or a building may be decomposable in more than one way. In the latter case, designers need to commit to a particular decomposition. This preference for certain relations comes at the expense of weakening or eliminating others which may be important as well. In buildings that are decomposed in a hierarchical manner, cross-hierarchy relations among elements may co-exist within the main hierarchy. Massing of buildings, alignment and proportion of openings, or construction types for enclosures may be of interest to designers but the relations with regard to each other may not be sufficiently captured by a main hierarchy. For example, attribute inheritance is useful for the propagation of modifications in a top-down manner, but only facilitates the modification of entities that inherit parent properties. It does not allow for the manipulation of overriding entities with similar ease. Windows, for instance, typically override the parent wall's attributes. In order to modify the glazing material of window entities, users would need to visit each affected window entity. *Procedural grouping* mechanisms are introduced for the efficient maintenance of such relations.

Before describing the details of procedural grouping, it is useful to establish key terms and concepts. A *dynamic group label* can be attached to an entity similar to labels mentioned earlier that signify an entity as a space or assign material properties. Supported by a set of operations

called *labeling styles*, designers can generate and maintain *dynamic groups*. The collection of dynamic groups is called a *dynamic group hierarchy.* A dynamic group hierarchy, or secondary hierarchy, relies on a primary hierarchy as shown in Figure 3. Invoked by a user operation at run time, this primary hierarchy is traversed to compute the elements that make up a dynamic group. Procedural grouping refers to the operations and mechanisms to create, maintain, and derive dynamic groups. In analogy to procedural models, procedural grouping mechanisms amplify user effort, in other words, a small user change can lead to drastic changes in a dynamic group hierarchy, and hence greatly influence model behavior.

Multiple dynamic groups can be maintained simultaneously through labeling styles and based on *propagation settings* and the position of a selected element within the primary hierarchy. Labeling style operations are usually executed on the level of the current selection. According the pattern of the labeling style selected by the user, the labeling style procedure iterates over each element on a level and assigns dynamic group labels. An important property of labeling style operations is that these can be propagated within a dynamic group hierarchy like other operations, that is, a dynamic group structure can be efficiently created in top-down fashion using a portion of the final dynamic group structure which already exists.

Users can control whether grouping on a level is active and/or if the scope should be expanded to the parent element and its siblings. These settings may be adjusted for both sending and receiving operations. This permits the interaction between dynamic group elements to vary from one-to-many to many-to-many. The scope of an operation is local if grouping is turned off or restricted to the current level. It is one-to-many if a group member sends operations, while all other grouped elements in that group only receive operations. Finally, the scope is many-to-many if all elements are capable of sending as well as receiving operations. The last setting is active by default. This kind of functionality gives users further control over how operations are propagated within a particular dynamic group.

Within dynamic groups, a distinction is made between *relative* and *absolute* groups. In most situations, relative groups should be sufficient. The scope of these groups is dependent on each element's position within the primary hierarchy, propagation setting, and label. Matching for absolute groups, on the other hand, is done globally, with identical group labels being the only condition. Elements with *undefined* group labels, which is the default setting for all elements, are not considered for dynamic grouping. In other words, users need to explicitly assign group labels to elements. This is in order to avoid unexpected side effects of applying operations to dynamic groups of which users may not be aware.

4.2 Labeling styles

Setting up and maintaining dynamic group labels on a level in the primary hierarchy can be a time-consuming activity without some form of automation. Labeling styles are scalable operations that support these tasks. They incorporate simple compositional principles such as repetitions, rhythms and symmetries, which are ubiquitous in architectural design, graphic design, and other art forms. Each labeling style defines a pattern that is applied to the ordered list of siblings on a level. Instead of users performing repetitive tasks of individually assigning labels to a potentially large number of sibling elements, this task can be delegated to labeling

styles. Below is a collection of common labeling style operations (the instructions refer to a set of child elements):

No links	_ - _ - _ -... Remove all group labels on all elements
Monotone	*A* - *A* - *A* -... Assign *A* to each element
Alternate	*A* - *B* - *A* - *B* -... Assign A to each odd element and B to each even element
Symmetric	...- *D* - *C* - *B* - ***A*** - *B* - *C* - *D* - ... Determine center element (choose last element in first half if number of elements is even), assign *A*, move towards first and last element and assign identical, otherwise unique labels to elements with same distance from the center element.
Symmetric Interval	...- ***B*** - *D* - ***B*** - *C* - ***B*** - *A* - ***B*** - *C* - ***B*** - *D* - ***B*** -... Same as symmetric, but assign *B* when distance to center element is odd
First / Last	*E* -...- *E* Assign identical, otherwise unique label to first and last element
Reverse	*A* - *B* - *C* - *D* → *D* - *C* - *B* - *A* Reverse labeling order (while preserving the element order)

Table 1. Examples for link styles

Note that labeling style operations may overwrite existing labels, that is, labeling styles may be combined with each other. For instance, an empty sequence '_-_-_-_-_-_' may be transformed into *'A - B - A - B - A - B - A'* and *'E - B - A - B - A - B - E'* by 'alternate' and 'first/last' labeling styles. Moreover, users may customize labels on an individual basis, or they may define their own labeling styles.

4.3 Relative grouping

The creation of a relative group structure is introduced with a facade example. This example takes advantage of the property mentioned above that labeling style operations can be propagated through already existing dynamic groups.

The rows in the matrix in Figure 4 indicate the primary hierarchy levels, the columns the states after each labeling style operation. Dark-shaded rectangles represent the selected elements, and dotted arrows the sibling order. Elements that have been assigned to a dynamic group are labeled. Note that there are no dynamic groups initially. In this example, a relative group structure is created as follows:

- On Level 1, the user selects an 'alternate' labeling style. As a result, each element on level 1 is assigned a label A or B depending on its position. The user navigates to a virtual, further decomposed window on Level 2, where the 'alternate' labeling style is selected. Since linking to the parent level is activated by default, the style procedure is not only applied to the children of the selected window, but also to related elements on Level 2 where the parent element on Level 1 is labeled as 'B'. In other words, the style procedure is only performed on branches that have windows.

- The user navigates to a center window pane on Level 3. Again, the 'alternate' labeling style is selected. The grouping procedure identifies all Level 2 windows as candidates which may contain children that need to be labelled. The selected labeling style is subsequently applied to all Level 3 window elements.

Three operations are sufficient to create a dynamic group structure with 37 elements and 6 dynamic groups (one for each label type). With this structure in place, the user could modify attributes such as glazing material of the two peripheral window panes for each window, or the dimension of each center window pane in one step because these operations would be applied to the proper dynamic group elements. Applying a 'uniform' labeling style to the window panes, that is, mapping a *'F-G-F'* to a *'F-F-F'* sequence, affects the model insofar as a new color assigned to a center window pane, for instance, would result in all window panes changing their color rather than just the center panes. Note that relative group labels need not be unique, that is, labeling Level 3 elements *A-B-A* instead of *D-E-D* would not cause a conflict with Level 1 labels. Different labeling is used for clarity only.

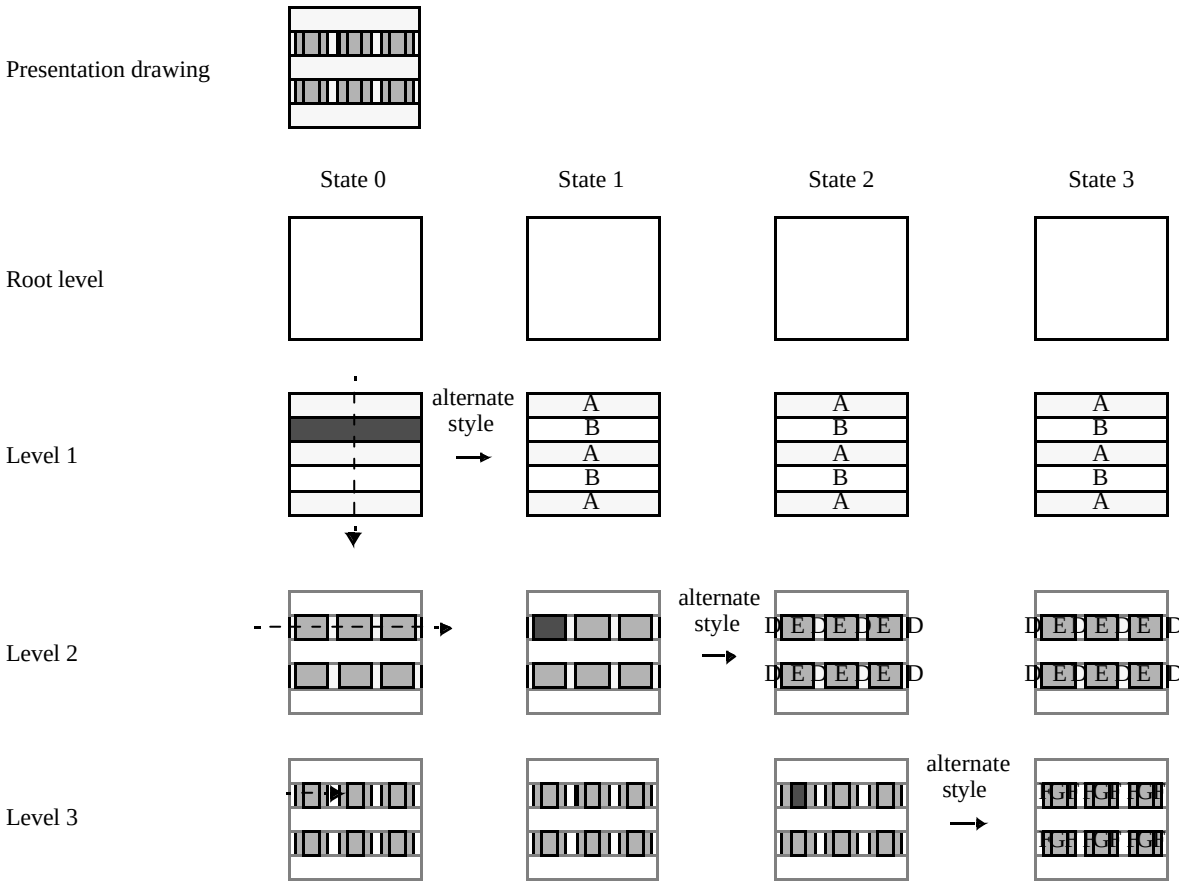


Figure 4. Example for the creation of a relative group structure

4.4 Absolute grouping

Relative groups are based on the position of elements in the primary hierarchy and their labeling. However, users may encounter situations where they would like to link portions of a group structure that are similar but happen to be on different hierarchy levels. This is where absolute grouping functionality is convenient, because it lets users synchronize the behavior of elements irrespective of their position in a primary hierarchy.

Users explicitly signify absolute groups as such by assigning labels to group elements. The main differences between absolute and relative labeling are that absolute labels have global

scope, and that there are no absolute linking styles. This is because absolute group elements are assumed to be fewer and distributed across a primary hierarchy. An element can only have one dynamic group label at a time, either absolute or relative. When the selected element has an absolute label, a global search for identical absolute labels is triggered and, similar to relative grouping, the desired operation is applied to all elements with the corresponding label. Absolute grouping is particularly useful when it is combined with relative grouping. If the selected element has a relative label and an absolute label is found during the screening process, then again a global search is conducted to find matching elements. In such situations, absolute labels can be viewed as serving as an anchor for linking relative groups.

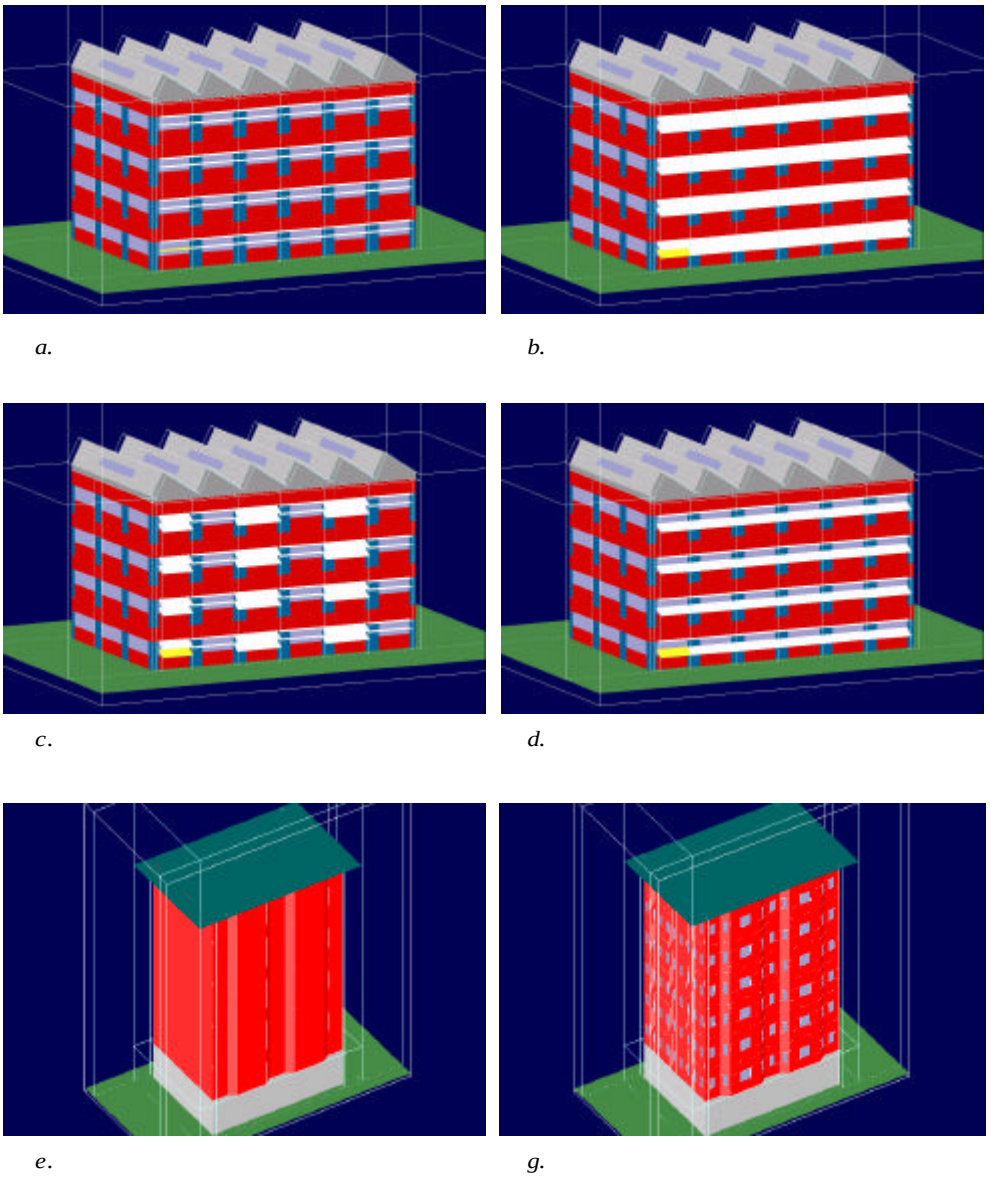
5 Two examples

The representation described above has been implemented in a building editor which can serve as a front-end application for detailed building performance simulation (Suter 1999). The examples demonstrate how automated integrity maintenance by the representation combined with procedural grouping lets users efficiently create and maintain large, detailed models.

The first example of an office building model with movable external shades demonstrates how a local change in the dynamic group structure can affect model behavior. In this scenario, a facility manager explores alternative operation schemes for the shades in order to minimize solar gain and maximize the daylight inside the building. For simplicity, feedback provided by simulation routines is not shown. Initially, all shades are in horizontal position (Figure 5a). The operations are propagated within the model from a shade which is highlighted in yellow. Figure 5b - d illustrate the consequences of tilting the selected shade based on three different, user-defined dynamic group hierarchies. There are 6 bays, each containing 4 pairs of shades. These bays are initially linked with the 'monotone' labeling style, that is, all bay elements have the same label. Furthermore, the shades within each bay are also linked by the 'monotone' labeling style. When the selected shade is tilted, that operation is not only applied to the entity itself, but to all louvers that are determined to belong to the same group by the grouping procedure. Thus, when 'monotone' labeling style is selected both at the louver and the parent bay level, all shades on the facade are tilted (Figure 5b). However, when the designer switches from 'monotone' to 'alternate' labeling style at the bay level, only every second bay is scanned for group members (Figure 5c). Rotating the same shade now affects only half the shades, because their parent bay labels are identical with the selected louver's parent label. Similarly, selecting 'alternate' labeling style at the shade level with 'monotone' style at the parent level affects yet another set of louvers (Figure 5d). Note that these different configurations of shades are achieved with minimal user intervention. Rather than manually selecting and modifying individual entities, which can easily become unmanageable in larger models, user involvement is limited to applying labeling styles and manipulating a single entity. The manipulations shown in this example are not as readily replicated with the traditional master-instance concept, a grouping mechanism common in drafting applications where changes in a blue print entity are applied to all copies or instances of that blue print (Sutherland 1963). Master-instance is most effective for a stable fixed set of entities.

The second example illustrates a discrete operation propagated within a dynamic group structure. A seven storey building is shown in Figure 5e with a plain, windowless external enclosure. One facade features bay windows. Each side above the grey pedestal is divided horizontally into nine fields, with each field having identical labels and spanning 6 floors. The parent facade zones in turn are linked by absolute labels. When the user selects a vertical field and inserts six windows, these are first inserted on a new hierarchy level below the selected field. The original field is preserved as an entity with a geometry, but it is not drawn because it

is decomposed into child elements. Due to dynamic grouping, the same operation is then performed for all other fields (Figure 5f). As a consequence, 54 windows are created and placed in the building envelope for each facade in a single operation. Discrete, high-level operations such as inserting a window are automated by the underlying geometry representation, that is, the user can delegate to the application the task of defining default values for window attributes such as location within the wall geometry, size, and proportion.



a. *b.*

c. *d.*

e. *g.*

Figure 5. Manipulation examples

6 Discussion

This paper elaborates on a hierarchical representation for the building performance simulation domain. It introduces concepts for procedural grouping, a dynamic grouping technique, which can enhance ease of manipulation for large, complex models. Its feasibility and benefits have been demonstrated with examples from an application prototype. In light of the capabilities of current, commercial computer-aided design systems, we believe that significant improvements with regard to usability could be achieved by investigating representations that

rely on a hierarchical organization of building information. Hierarchical representations appear to be particularly useful when users need to deal with massive and diverse information, as in the building performance domain. These claims, however, are speculative at this point and need to be validated by user studies.

More work is necessary to address user interaction issues, particularly in how dynamic grouping information can be conveyed to users. One idea is to have a preview functionality that would highlight affected elements prior to executing an operation. The challenge is to develop a user interface functionality for procedural grouping that would sufficiently expose the complexity of the underlying model in a transparent, easily understandable way. As mentioned, the representation automatically maintains geometric integrity, that is, entities do not intersect or overlap in an illegal manner. However, there are currently no mechanism to ensure semantic integrity, an issue which is relevant particularly with regard to integration with simulation.

7 References

Amburn, P., Grant, E., and Whitted, T. (1986), Managing the complexity with enhanced procedural models, Proceedings of the 13th annual conference on Computer graphics.

Foley, J., A. van Dam, S. Feiner, and J. Hughes (1992) Computer graphics: principles and practice, Addison-Wesley, Reading, MA.

Hand, J.W., and J. L. M. Hensen (1995) Recent experiences and developments in the training of simulationists, *Fourth international conferenc*e, International Building Performance Simulation Association, Madison WI, p. 346 - 353.

Harada, M. (1997) Discrete and continuous design exploration by direct manipulation, Ph.D. thesis, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA.

Kundu, S. (1988) The equivalence of the subregion representation and the wall representation for a certain class of rectangular dissections, Communications of the ACM, 31 (6), 1988, 752 - 63.

Mahdavi, A. (1996) SEMPER: a new computational environment for simulation-based building design assistance, *1996 International Symposium of CIB W67 (Energy and mass flows in the life cycle of buildings*), Vienna, Austria.

Pelletier, R., and W. Keilholz (1999) Coupling CAD tools and building simulation evaluations, *Sixth international conferenc*e, International Building Performance Simulation Association, Kyoto, Japan, p. 1197 - 1202.

Steadman, J.P. (1983) Architectural morphology : an introduction to the geometry of building plans, Pion, London, England.

Suter, G., and A. Mahdavi (1998) Generation and communication of design information: a building performance simulation perspective, *Fourth Conference on Design and Decision Support Systems in Architecture and Urban Planning*, Maastricht, NL.

Suter, G. (1999) A representation for design manipulation and performance simulation, Ph.D. thesis, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA.

Sutherland, I.E. (1963) Sketchpad: a man-machine graphical communication system, Technical Report 296, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA.

Van Dam, A., D. Sklar (1990) Object Hierarchy and Simple PHIGS (SPHIGS), in Foley, J., A. van Dam, S. Feiner, and J. Hughes (1990) Computer graphics: principles and practice, Addison-Wesley, Reading, MA, p. 285 - 346.