Interactive Physically-Based Manipulation of Discrete/Continuous Models

Mikako Harada
Department of Architecture

Andrew Witkin
Department of Computer Science

David Baraff
Robotics Institute

Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Physically-based modeling has been used in the past to support a variety of interactive modeling tasks including free-form surface design, mechanism design, constrained drawing, and interactive camera control. In these systems, the user interacts with the model by exerting virtual forces, to which the system responds subject to the active constraints. In the past, this kind of interaction has been applicable only to models that are governed by continuous parameters. In this paper we present an extension to mixed continuous/discrete models, emphasizing constrained layout problems that arise in architecture and other domains. When the object being dragged is blocked from further motion by geometric constraints, a local discrete search is triggered, during which transformations such as swapping of adjacent objects may be performed. The result of the search is a "nearby" state in which the target object has been moved in the indicated direction and in which all constraints are satisfied. The transition to this state is portrayed using simple but effective animated visual effects. Following the transition, continuous dragging is resumed. The resulting seamless transitions between discrete and continuous manipulation allow the user to easily explore the mixed design space just by dragging objects. We demonstrate the method in application to architectural floor plan design, circuit board layout, art analysis, and page layout.

Keywords—Interactive techniques, physically-based modeling, grammars.

1 Introduction

Physically-based modeling has been a topic of interest in computer graphics for some time. A particular area of interest has been the use of physical simulation or related optimization techniques as a means of geometric interaction, allowing users to directly manipulate simulated objects subject to constraints. This approach has been applied to animation, image analysis, drawing, free-form surface modeling, mechanical design, and interactive molecular simulation [29, 28, 30, 5, 31, 33, 27].

Despite its successes, this approach has been subject to the severe limitation that it only supports manipulation of systems governed by continuous variables. Neither discrete operations such as the addition, deletion, or replacement of parts nor discrete changes in object shapes or spatial relationships can be accommodated at all within the physically-based manipulation framework. This is an unfortunate limitation because a great many problems of interest involve both continuous and discrete parameters. An important class of such problems are layout or arrangement problems in which a collection of objects are to be placed and reshaped subject to constraints that prevent interpenetration, require adjacency, bound dimensions, etc. Such problems arise in architecture for floor planning and furniture layout, in mechanical and electromechanical design, and even in fine art.

We would like to extend the physically-based interaction paradigm to encompass the exploration of this kind of continuous/discrete problem space. We envision systems in which objects respond continuously to the user's pushing and dragging actions in a physical or quasi-physical manner, until they "hit a wall" imposed by some constraint. They then spontaneously undergo some discrete rearrangement or other transformation that does what the user asked (essentially, moving the dragged or pushed object point in the indicated direction) with the least perceived disruption in the overall arrangement. The change should occur rapidly enough to maintain interactivity, and should be visually portrayed in a fashion that the user can clearly understand.

For example, in laying out a floor plan there are many degrees of freedom as well as many constraints. Rooms may not overlap. A residential plan must include a living room, some number of bedrooms, etc. The dining room should adjoin the kitchen. Rooms have minimum dimensions or areas. Typically, all but bathrooms must adjoin an exterior wall or courtyard wall. The whole house must fit within some pre-specified envelope. Within these constraints, the size, shape, and arrangement of rooms is free to vary. Some rooms (e.g. a library or den) might be optional. Although plans may be judged objectively by criteria such as cost, subjective aesthetic criteria also play a large role, so the problem cannot be reduced to one of "black box" mixed-integer combinatorial optimization. (And such problems are, as a general rule, *NP*-hard.)

If the basic arrangement is fixed, it is a straightforward matter using existing physical manipulation techniques to push and pull on walls, adjusting dimensions within the size or area constraints. The new ingredient we want to add is this: when a wall is pushed to one of its constraint limits, the system should find a re-arrangement, (e.g. swapping the living room to the other side of the dining room) that allows the wall on which the user pushes to continue to move in the indicated direction. The disruption to the rest of the plan should be minimal, and it should be immediately clear to the user what happened.

For any particular instance of this class of problem, we are given a model whose state is given by a set of continuous and discrete (integer or boolean) variables, with a graphical representation that depends on state, and a set of equality or inequality constraint func-

Author's affiliations: Mikako Harada, (mh3i@andrew.cmu.edu), Department of Architecture; Andrew Witkin, (aw@cs.cmu.edu), Department of Computer Science; David Baraff, (baraff@cs.cmu.edu), Robotics Institute. Carnegie Mellon University, Pittsburgh PA 15213.

This is an electronic reprint. Permission is granted to copy part or all of this paper for noncommercial use provided that the title and this copyright notice appear. This electronic reprint is ©1995 by CMU. The original printed paper is ©1995 by the ACM.

tions that likewise depend on state. Our approach to the problem handles continuous and discrete changes separately, although transitions between the two modes are seamless and transparent to the user. The user's dragging motions are treated as a "force" to be applied to the model's continuous parameters subject to the constraints. As long as this applied force is not entirely cancelled by constraint forces, the discrete variables remain frozen, and our method is just like that used in previous physical manipulation systems.

When the constraint forces cancel the user's applied force, the model is "stuck." We then search for a new state of the discrete variables. The main contribution of this paper lies in this area. We begin with a problem-specific search space that is defined by a set of transformations on the discrete variables. The form of these transformations depends on the structure of the models. The idea is to define this set of transformations to do a reasonable job of capturing perceptual similarity, so that each single transformation makes a perceptually small change to the arrangement of objects. In the domains on which we have experimented, constructing the transformations has not been difficult; we have done very well with simple operations such as swaps and moves. The set of transformations gives us a new space in which we may perform shallow searches. The search is limited to transformations that directly influence the element being dragged by the user, and is also subject to a depth limit. Subject to these restrictions, we look for a state that does the best job of moving whatever was grabbed by the user in the desired direction. To portray the transition to this new state we use simple animation tricks that provide smooth motion cues for the user.

This combination of discrete and continuous mechanisms provides seamless, pleasing interactions. The discrete search is hidden from the user. In the case of swapping rooms in a floor plan, the user's experience is something like this: we grab one wall of the living room, pulling toward the opposing wall, which is shared with the dining room. The living room shrinks until it reaches its minimum width. If we continue to pull strongly on the wall, the living room "pops" through to the other side of the dining room, with a brief interpenetration. Most of the rest of the plan is undisturbed, but the kitchen, which is constrained to be adjacent to the dining room, has swapped places as well.

The remainder of the paper is organized as follows: in the next section, we discuss the background of physical manipulation methods, as well as the relevant architecture background of shape grammars. In the following section we review the math and algorithms of continuous constrained manipulation. We then describe the discrete portion in detail. Next, we present results in architecture, board layout, page layout, and grammar-based analysis exploration of paintings. We conclude with discussion and directions for future work.

2 Background

An active research area in architecture involves the use of *Shape Grammars* as a means of formally capturing architectural style. Despite considerable evidence that such grammars are capable of representing styles, it has not previously been clear how they could actually be used to interactively explore design spaces or create designs. A main motivation of our work has been to develop means of interacting with shape grammars through direct geometric manipulation. The first portion of this section reviews prior work on shape grammars for architecture.

Embedded in the discrete problem of searching grammar-based design spaces is a problem of exploring a continuous design space by varying sizes, positions, etc. within the design constraints. Our approach to this sub-problem closely follows previous work in the use of physically-based methods for interactive manipulation. The second portion of this section reviews this prior work.

2.1 Shape Grammars in Architecture

Shape grammars were first introduced to architecture by Stiny and Gips in the 1970's [25, 24]. Since then, shape grammars have been used theoretically to describe the historical style of architectural designs [26, 7, 17] and to specify new forms [15]. Shape grammars are appealing to architecture researchers because they are visual, and because they offer a formal mechanism for capturing the previously vague and ambiguous notion of "style."

The basic idea of shape grammars is to define a set of rewrite rules that operate on geometric models by replacing, re-arranging or modifying elements. The grammar serves to implicitly define a set of geometric models, i.e. all those that can be derived by some legal sequence of rule applications. The hope is that relatively simple rule sets can be devised that capture some aspect of architectural style, for example, generating designs for all and only Queen Anne Houses [13].

Several computer programs have been developed based on the theoretical foundation of shape grammars.[1] These systems have been used to demonstrate grammars representing floor plan layouts [8], framing structures [21], and a class of building designs [13].

Interacting with grammar-based systems has been problematic. The only means of exploring the discrete space of designs defined by a grammar has been sequential manual rule selection by the user, with, in some systems, the ability to edit an existing derivation. This mode of editing can be frustrating and extremely difficult, because the relation between a rule and its geometric effect can be obscure, particularly for rules that occur early in the sequence. We believe that the formalism of shape grammars would be far more useful if the architect could explore the space by directly manipulating the grammar's geometric output rather than the sequence of rules that produced it.

A further issue has been that these systems do not handle continuous dimensional variations (the flip side of the problem we have faced in physically-based manipulation.) A few applications allow grammars to include continuous parameters, with some user control over parameter values [21]. However, the problem of solving simultaneously for parameter values that satisfy constraints has been finessed. No existing grammar-based system allows the user to directly manipulate designs to produce geometric variations. Existing physically-based techniques, on the other hand, appear to be ideally suited to supporting such direct manipulation.

2.2 Physically-Based Modeling for Graphical Interaction

Interactive physical simulations have been used as a technique for interaction with a variety of graphical models. In computer vision and image analysis, quasi-physical active contour models known as *snakes* are widely used for interactive edge finding and tracking [14]. Snakes are essentially simulated springy "wires" that are attracted to edge features in images, and simultaneously subjected to manipulation forces imposed by the user.

The use of constrained dynamics simulations for interactive geometric modeling was described by Witkin *et al*. [32]. Flexible-surface simulations for interactive computer vision and free-form surface modeling are areas that have also been extensively investigated [29, 28, 30, 5, 31, 33]. Constrained dynamics simulations have also been used to support drawing applications [11] and for interactive camera control for animation [10]. Surles [27] describes

[1] Shape grammars in their original form could not be implemented directly, due to problems of ambiguity. Several additional formalisms were introduced on which system implementations were based [4, 13]. Henceforth we use "shape grammar" in the broader sense of these extended geometric grammars.

a system for interactive molecular simulations. Interactive simulations that include contact constraints were described by Baraff [3]; the continuous simulation techniques in this paper most closely follow that work.

3 Continuous Manipulation

During any interval of time in which the discrete variables of the model stay fixed, the model is represented simply as a continuous function of state $\mathbf{q}(t)$, where $\mathbf{q}(t) \in \mathbb{R}^n$ is a vector of the model's n continuous variables. Over such an interval, the model may be subject to some constraints: that is, we may require the model to satisfy the conditions

$$\begin{array}{rcl} C_1(\mathbf{q}(t)) & \geq & 0 \\ C_2(\mathbf{q}(t)) & \geq & 0 \\ & \vdots & \\ C_m(\mathbf{q}(t)) & \geq & 0 \end{array} \tag{1}$$

where each function C_i is a scalar function of state, $\mathbf{q}$. The evolution of this type of constrained system over time exactly parallels work on physical simulation of impenetrable objects [2]; accordingly, we cast the continuous manipulation of our models as a problem in constrained physical simulation.

Assuming that the model's initial state $\mathbf{q}(t_0)$ satisfies $C_i(\mathbf{q}(t_0)) \geq 0$ for $1 \leq i \leq m$, the model evolves from time t_0 according to the first-order differential equation

$$\dot{\mathbf{q}}(t) = \frac{d}{dt}\mathbf{q}(t) = \mathbf{M}^{-1}\mathbf{F}(t) \tag{2}$$

where $\mathbf{F}(t) \in \mathbb{R}^n$ is the "force" acting on the model at time t and $\mathbf{M}$ is an $n \times n$ diagonal "mass" matrix. By adjusting entries in $\mathbf{M}$, state variables can be "heavier" or "lighter" (that is, harder or easier to change). Aside from user preferences, the matrix $\mathbf{M}$ is needed to properly scale the problem in the case when two or more of the state variables have vastly different scale/ranges from each other. The force $\mathbf{F}(t)$ has the form

$$\mathbf{F}(t) = \mathbf{F}_a(t) + \mathbf{F}_c(t)$$

where $\mathbf{F}_a(t)$ represents the force applied by the user to alter the model, and $\mathbf{F}_c(t)$ represents a constraint force that is introduced so that the conditions of equation (1) can be maintained.

To allow the model to be manipulated with as little interference as possible, we seek a constraint force $\mathbf{F}_c(t)$ that interferes with the user's applied force as little as possible. Treating our manipulation problem as a physical problem, we choose $\mathbf{F}_c(t)$ to be a workless, compressive constraint force [19]. At any particular time t, such a constraint force depends on the applied force $\mathbf{F}_a(t)$ and the set of *active* constraints. A constraint C_i is said to be active at time t if $C_i(\mathbf{q}(t)) = 0$. The workless, compressive constraint forces we seek have the form

$$\mathbf{F}_c(t) = \frac{\partial C_1(\mathbf{q}(t))}{\partial \mathbf{q}} f_1 + \cdots + \frac{\partial C_m(\mathbf{q}(t))}{\partial \mathbf{q}} f_m$$

where the scalar variables f_i are all nonnegative and f_i is zero if the ith constraint is not active.

Suppose at time t only k of the m constraints on the model are active; without loss of generality, let $C_1(\mathbf{q}(t)) = C_2(\mathbf{q}(t)) = \cdots = C_k(\mathbf{q}(t)) = 0$. Only the k active constraints have an effect on the model's motion at time t; in particular, if k is zero the system is (momentarily) completely unconstrained in its motion. Let $\mathbf{C}(\mathbf{q}(t)) \in \mathbb{R}^k$ be defined as the vector-collection of these k active constraints,

$$\mathbf{C}(\mathbf{q}(t)) = \begin{pmatrix} C_1(\mathbf{q}(t)) \\ C_2(\mathbf{q}(t)) \\ \vdots \\ C_k(\mathbf{q}(t)) \end{pmatrix},$$

and let the $n \times k$ matrix $\mathbf{J}$ be defined by

$$\mathbf{J}^T = \frac{\partial \mathbf{C}(\mathbf{q}(t))}{\partial \mathbf{q}}.$$

Using this notation, $\mathbf{F}_c(t)$ has the form $\mathbf{F}_c(t) = \mathbf{J}^T\mathbf{f}$ where the vector $\mathbf{f} = (f_1, f_2, \ldots, f_k)$ satisfies $\mathbf{f} \geq \mathbf{0}$ (with $\mathbf{0}$ a k-vector of zeros). To prevent the active constraints from decreasing past zero, we require that $\dot{\mathbf{C}}(\mathbf{q}(t)) \geq \mathbf{0}$. Using the chain rule, this yields

$$\dot{\mathbf{C}}(\mathbf{q}(t)) = \frac{\partial \mathbf{C}(\mathbf{q}(t))}{\partial \mathbf{q}}^T \dot{\mathbf{q}}(t) = \mathbf{J}\dot{\mathbf{q}}(t) \geq \mathbf{0}. \tag{3}$$

Since $\dot{\mathbf{q}}(t) = \mathbf{M}^{-1}\mathbf{F}(t)$ and

$$\mathbf{F}(t) = \mathbf{F}_a(t) + \mathbf{F}_c(t) = \mathbf{F}_a(t) + \mathbf{J}^T\mathbf{f},$$

we can rewrite equation (3) as

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{f} + \mathbf{J}\mathbf{F}_a(t) \geq \mathbf{0}. \tag{4}$$

For the constraint force to be workless, we require not only that $\mathbf{f}$ be nonnegative and satisfy equation (4), but also that the ith component of $\mathbf{f}$ be zero if $\dot{C}_i(\mathbf{q}(t)) > 0$. This last condition prevents the constraint force from gratuitously changing $\mathbf{q}(t)$ so that an active constraint becomes inactive. Baraff [2] discusses the implementation of a fast, efficient algorithm for computing a vector $\mathbf{f}$ which satisfies all of these conditions; our system computes $\mathbf{f}$ using the described implementation.

Once $\mathbf{f}$ is computed for a given t, the constraint force and thus the total force $\mathbf{F}(t)$ acting on the system is known. Standard numerical methods are then used to integrate the system $\dot{\mathbf{q}}(t) = \mathbf{M}^{-1}\mathbf{F}(t)$ forward in time. Since constraint forces only prevent currently active constraints from being violated, and since numerical integration techniques take discrete steps from time t to time $t + \Delta t$, a constraint j that is inactive at time t may occasionally be violated at time $t + \Delta t$ (i.e. $C_j(\mathbf{q}(t + \Delta t))$ may be negative). In this case, we employ standard root finding techniques to evolve the system forward to the time t_c between t and $t + \Delta t$ at which $C_j(\mathbf{q}(t_c))$ becomes zero. The jth constraint then becomes an active constraint at this time, and a more accurate projection of $\mathbf{q}(t + \Delta t)$ can be computed. The process is the same as the collision-resolution processes employed in physical simulation systems [1].

4 Discrete Manipulation

We wish to extend the paradigm of continuous physically-based interaction to allow the state of the system to undergo discrete structural change in response to the user's actions. Although such discontinuous behavior is inherently non-physical we want to retain the spirit of continuous physical interaction, letting the user operate on the design simply by dragging objects subject to whatever constraints are in force.

In a continuous system, when the user drags an object into a wall or other constraint-imposed "dead end," the object simply halts. Instead, if the user continues to pull, we want the system to seek some discrete change that permits the object to move further in the indicated direction.

Finding such a change is a search problem. Of the discrete space of alternative states, we must find a new state that satisfies several criteria:

- The new state must match our interpretation of the user's intent. In the case of simple dragging, this means that the element being dragged must be displaced in the direction indicated by the user.
- The new state should be consistent with all problem constraints, although it turns out we will want to briefly portray illegal intermediate states as a visual cue.
- The change should not surprise the user. Rather than an involved global rearrangement, we want the change to be simple, local, and easily grasped. Ideally, we want the user to perceive some sort of direct mechanical connection between his or her action and the resulting change, as if, for example, normally impenetrable rigid objects were allowed momentarily to squeeze past or pop through each other.
- If several otherwise equivalent alternatives are found, we may wish to choose the one that rates best according to a problem-specific objective function.
- We must find the new state quickly enough to maintain interactivity, which rules out a large-scale combinatorial search.

We approach the problem by creating a set of domain-specific transformations that map states in the discrete space into other states. We use these to structure a local search and to define a metric intended to capture perceptual similarity, with distance to a new state defined by the number of transformations required to reach it. In practice, this means that we want transformations that make small, local changes, such as swapping two adjacent elements.

The user actions that triggered the search provide information that can be used to limit it, for instance, considering only those transformations that directly influence the element being dragged. In addition, to maintain interactivity, we limit the depth of the search to at most a few levels. Subject to these restrictions we seek the closest state that satisfies the user's intent subject to the design constraints, possibly weighting the similarity of the old and new states against the absolute merit of the new one as defined by the design objective function. In the subsections that follows, we describe the method in detail.

4.1 Triggering a Local Search

In our system, continuous manipulation is the main mode of interaction. The search for a discrete change is triggered automatically when continuous manipulation "hits a wall" in the sense that no further continuous change can be made in response to the force $\mathbf{F}_a$ applied by the user. Specifically, the discrete search is activated when the following conditions apply:

- there are user-applied forces ($\mathbf{F}_a \neq \mathbf{0}$),
- the applied forces do no work ($\mathbf{F}_a^T \dot{\mathbf{q}} \approx 0$),
- the applied forces are large enough to signal the direction of user action ($\|\mathbf{F}_a\| \geq k$, where k is the smallest force needed to activate discrete manipulation), and
- the configuration is not identical to that preceding the most recently performed search.

The last condition is added to avoid "thrashing" after a local discrete search fails. If these conditions are met, we initiate a local search, using the object being pulled by the user and the direction in which it is pulled to limit the search.

Although the choice of transformations is problem-specific, we generally want transformation rules that make minimal, local changes so that nearby states will appear globally similar to the original. Additional criteria are that the transformations should tend to yield nearby states that are consistent with the topological and geometric constraints, and steerable in the sense that we can directly limit the search to states that are consistent with the user's intent.

We limit the search to operations that move the target element in the desired direction, and also place an absolute limit on the depth of the search to maintain interactivity. Each candidate state is tested against the constraints and objective functions. An example of a set of transformation rules is described in the next section.

4.2 Constraints and Objectives

In addition to the "user's intent" constraint which has already been built into the search, we must consider topological and geometric constraints that are part of the problem specification. We also have two sorts of objective function: a relative one, measuring distance from the old state to a candidate new state, and an absolute one measuring the goodness of the overall design. We take a weighted sum of the two, choosing the best solution that satisfies the constraints. If no legal state is found within the depth limit of the search we remain at the old state. In this last case, the user's experience is that the object being pulled "can't go that away," in which case he or she is free to try something else.

To satisfy the constraints, we consider topological constraints first, such as adjacency requirements. If a candidate state meets all the topological constraints, we then perform a numerical constraint solution in an effort to meet the applicable continuous geometric constraints, using a general purpose continuous constrained optimization program [9]. Since we are making incremental changes, the previous geometric parameter values generally give good initial estimates for finding a new state that satisfies the constraints.

Of the states that satisfy the topological and geometric constraints, we choose the best one as measured by a compound objective function. This objective function has discrete and continuous parts as well. We let the "closeness" of the new state to the old depend on continuous geometric measures of the change, as well as on the integer-valued transformation distance between them. We could use an additional objective function to modify the search, either a persistent objective function that measures the "goodness" of the design, or a temporary one that measures the degree to which the change matches what the user asked for, or both.

After a new state is chosen, the numerical values obtained by optimization become the initial continuous parameter values for the new state. We then return to the continuous manipulation phase.

4.3 Visualizing Discrete Changes

Although we are trying to make the smallest possible discrete changes, any abrupt change to the displayed state of the model tends to be confusing. We have found that the addition of some simple visual cues is of great help to the user in understanding the discrete changes. Instead of switching directly to the new state we smoothly animate the transition, using highlighting and icons to help direct the viewer's attention. With some tuning of the appearance and timing of these visual effects we have been able to produce seamless transitions between the continuous and discrete modes of interaction, with the subjective impression that the discrete change, like the continuous one, takes place in direct mechanical response to the user's action.

5 Implementation Examples and Results

We have built an interactive system that implements the discrete and continuous manipulation techniques described in this paper.

Our initial experiments have centered on a class of layout problems known as *floor planning* problems. The floor planning problem is also known as the *rectangular packing* or *rectangular dissection* problem.

The floor planning problem, in general, is to arrange a given number of rectangles within a larger rectangular space so that the space is completely filled, without any overlap between boxes.[2] Each rectangle may have lower-bound and upper-bound constraints on its dimensions and area. There may also be structural constraints between rectangles, such as adjacency constraints. The goal of floor-planning problems varies with the application, but often involves an optimization subject to the constraints, and may also involve aesthetic or other subjective criteria. Floor planning problems have long been investigated [12] and are of particular interest in architecture [22, 6, 8] and VLSI design [34, 20, 35]. In this section, we describe the implementation of a system for attacking this class of floor layout problem interactively. After showing examples, we describe the results of our experiments.

5.1 Implementation of a Floor Planning Program

Our representation for floor planning problems is based on rectangular dissection. We begin with a single rectangular region, then recursively subdivide regions, either horizontally or vertically, into two or more sub-regions.[3] Each dissection structure can be defined by a *subregion tree* in which nonterminal nodes represent either horizontal or vertical subdivisions, and leaf nodes represent the final rectangles that comprise the dissection. Kundu [18] gives a detailed description of subregion trees. A rectangular dissection and its subregion tree are shown in figure 1a. The chief advantage of this representation is that it guarantees a non-overlapping arrangement of boxes, with no empty space. In addition, the subregion tree allows us to easily extract adjacency relationships between rectangles.

The structure of the subregion tree defines the topology of the dissection. Additional information is required to define the rectangles' dimensions. In the root node we store the absolute height and width of the bounding rectangle. In each other node we store a dimensionless value r that defines its height or width as a fraction of the enclosing rectangle's height or width. If V is a vertical subdivision node with dimensions $[x, y]$ and children $R_1, R_2 \ldots, R_n$, then the dimensions of rectangle R_i are $[r_i x, y]$, where r_i is the value stored in R_i, and similarly for horizontal subdivisions. The dimensions of any rectangle, and the derivatives of its dimensions, are easily evaluated by recursively descending the tree. We enforce the constraint that $r > 0$ for all nodes, and that $\sum_i r_i = 1$ for each set of sibling nodes.

Our current system implements constraints that place upper and lower bounds on width, height, area, and aspect ratio, as well as rectangle adjacency constraints. Additional geometric constraints are easily added, since we only need to add code to compute $C(\mathbf{q})$ and $\partial C(\mathbf{q})/\partial \mathbf{q}$ to handle a new type of constraint.

In addition to constraint forces, it is sometimes desirable to add default values or "preferences" to a model. For example, a given rectangle may have a desired size, area, or aspect ratio, which the system should try to achieve unless blocked by constraints. Such preferences are easily implemented by adding penalty forces to the model. Penalty forces are obtained by negatively scaling the gradient of the squared difference between the actual state and the desired one. Penalty forces act like springs that pull objects toward the desired configuration. They are subordinate to the hard constraints, but compete with each other. The strength of each penalty force is controlled by a user-assigned weight. During continuous manipulation, penalty forces are added to the manipulation force supplied by the user to form the total applied force, $\mathbf{F}_a(t)$. As described in section 3, a constraint force $\mathbf{F}_c(t)$ is then computed which maintains the dimensional constraints. As long as constraints do not prevent a particular preferred motion of the model, or there are not multiple contradictory preferences, the system will gradually enforce the desired preferences, while still strictly maintaining the dimensional constraints of the system.

[2]Non-overlapping, no-unused-space arrangements are sometimes called "tightly packed." If we allow unused spaces, the problem becomes a "loosely packed" problem. As we see in our examples, loosely packed problems are a subset of tightly packed problems.

[3]We define a vertical subdivision to be one that introduces a vertical boundary, and a horizontal subdivision, a horizontal boundary. Thus the rectangles created by a horizontal subdivision are one above the other.

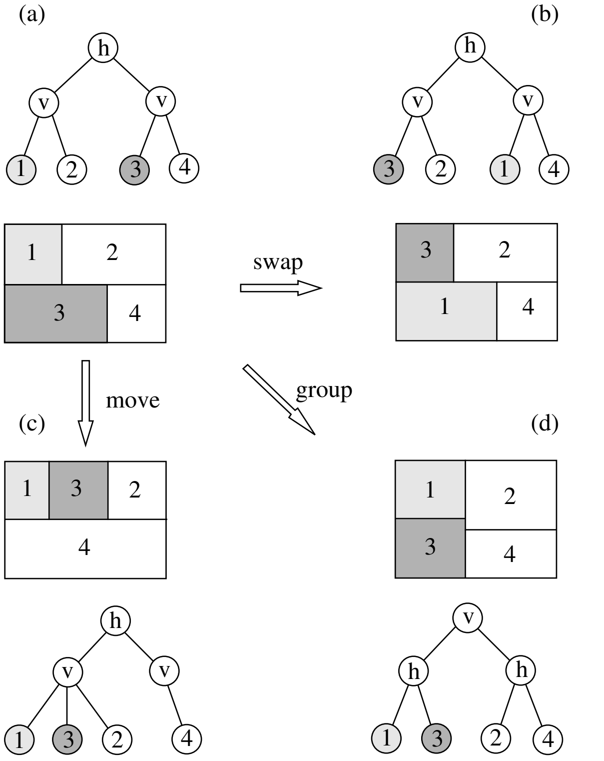

Figure 1: Rectangular dissections, subregion trees, and transformations. (a) A dissection and its subregion tree. (b–d) Transformations on (a).

To support discrete floor plan manipulations we have defined four transformations that operate on subregion trees: move, swap, group, and rotate. The first three of these are illustrated in figure 1.

- The *move* transformation takes as its arguments a node to be moved, the new node under which it should be placed, and an integer giving its position in the list of siblings at the new location. The target node is moved to the new location. If it is a nonterminal node, the subtree under it moves as well.

- The *swap* transformation takes as arguments two nodes, and exchanges their positions in the subregion tree, equivalent to two move operations.

- The *group* transformation takes as arguments two nodes, which must be spatially adjacent. The subregion tree is rearranged to make the nodes become siblings in the subregion tree. We use this operation when four subregions meet at or near one point to change the order of subdivisions (i.e., from vertical first and horizontal second to horizontal first and vertical second, or vice versa).

Figure 2: Architectural room layout. (top) The user pulls on the dining room, and a discrete change occurs. (middle) Two intermediate frames from the resulting animated transition. (bottom) The new configuration.

- The *rotate* transformation takes a single node as its argument. It does not change the structure of the subregion tree. Rather, it swaps the *x* and *y* values of the minimum and maximum dimensions assigned to the corresponding rectangle. This operation is only applicable to leaf nodes.

The search space for discrete changes is the set of subregion trees generated by applying sequences of these transformations to the original tree. We are interested in interactive manipulation rather than large-scale combinatorial optimization. To maintain interactivity we must restrict the search to a very small number of alternative states. We limit the search in two ways. First, we consider only transformations that operate on nodes which are directly involved with the rectangle or edge being dragged by the user. Second, we limit the depth of the search: in the examples presented here we consider only transformation sequences of length two or less.

For each new discrete state we consider, it is generally necessary to adjust the continuous parameters to satisfy the geometric constraints if possible, and to obtain a local optimum of whatever objective function is attached to the model. To perform the continuous constrained optimization we use a general purpose optimization package called NPSOL [9] to compute a new state vector $\mathbf{q}$. The NPSOL package optimizes smooth nonlinear functions subject to both linear and nonlinear constraints. We have considered several objective functions for the new state vector $\mathbf{q}$, including objective functions that seek to keep rectangular areas close to a desired size, functions that try to minimize the change in $\mathbf{q}$ from its prior value, and objective functions which minimize the total area of the rectangular dissection. Like constraints, new types of objective functions are easily added by supplying code which evaluates an objective function's value and gradient for a particular state $\mathbf{q}$.

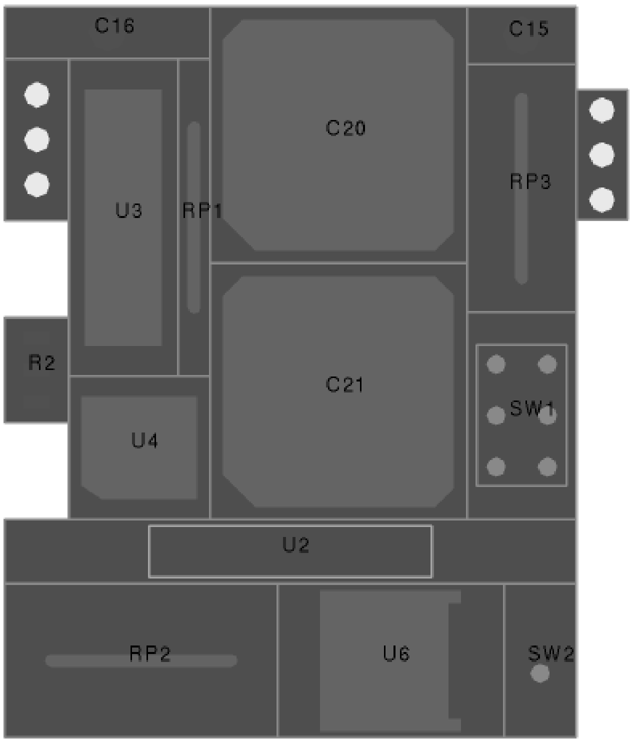

Figure 3: Circuit board layout.

5.2 Examples

Within the class of floor planning problems, we show examples from four domains: architectural room layout, circuit board layout, page layout, and stylistic analysis of abstract painting.

Architectural Room Layout

A primary motivation of our research has been to build a system for architectural layout planning. In our architectural examples, rooms or functional areas are modeled as rectangles. The sides of the rectangles are the walls surrounding a room. In addition to the constraints to set bounds on the lengths and areas of each room, and the overall floorspace occupied by the rooms, adjacency constraints can be defined between some rooms. For example, we might specify that a kitchen must remain adjacent to the dining room, while an entranceway must always be adjacent to an exterior wall of the layout.

Using the system, the user can pull on either a wall or the center of a room to change the position and size of a room. Constraints or preferences can be used to prevent rooms from becoming overly narrow during this manipulation. The user can also add constraints to fix walls in place and prevent them from moving from their current position. When the user pulls on an interior wall, the exterior walls of the layout remain fixed. When the user pulls on an exterior wall, the entire layout is scaled.

During manipulation, constraints are visually indicated. For example, if a room's width has reached its lower bound, a bright line is drawn horizontally across the room, to indicate that no more horizontal compaction of the room is possible. Even if the user continues pulling in the same direction, the constraint forces prevent the room's width from further decreasing. Similarly, a room whose area cannot be increased or decreased is bordered with a bright line to inform the user that the area of the room is currently bounded. When all progress is blocked by the constraints, a sufficiently hard pull on a blocked component of the layout activates a search for a discrete change. After the system finishes the search, a transition between the current state and the new state is shown as an "animation"; rooms fly from the current state to the new state (using simple linear interpolation).

For a proposed discrete change, the objective function on the state $\mathbf{q}$ is the difference of the desired total area of the house, and the total area occupied by the house in state $\mathbf{q}$. Two examples of architectural layout are shown; the first example shows a house with 14 rooms, while the second example has 24 rooms (21 actual rooms plus 3 extra rectangular areas to produce indentations, yielding a nonrectangular exterior shape for the house). A sample layout for the second house is shown in figure 2.

Circuit Board Layout

PC board layout is a heavily investigated application area of floor planning. In this context, the goal is to arrange a set of circuit modules on a board while minimizing the total interconnecting wire length. The objective of this problem sounds much clearer than the architectural examples. However, there are many flexible parameters that the designer must consider. The design process involves the selection of a representative among many alternatives. As an example, in designing a board layout for a wearable computer, the primary objective is to make the computer wearable; i.e., it must be small, and comfortable to carry around. The designer's concern then shifts to interface issues, such as the position of switch, the position of a strip, and the balance of weight.

We have taken a sample board layout from a design team working on a wearable computer. We have chosen an objective function that seeks to minimize the total area occupied by the circuit board. The basic interactions for the board layout are the same as in the architectural examples. We have added some icons to make it easier for users to recognize each module. Additionally, rotational transfor-

Figure 4: Page layout.

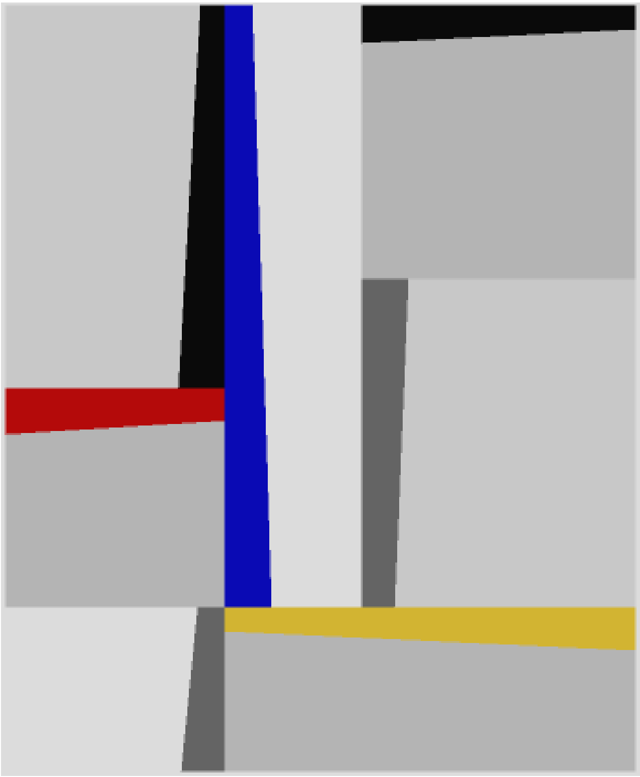

Figure 5: De Stijl style art.

mations are part of the allowable set of transformations during discrete search. A sample configuration of the circuit board is shown in figure 3.

Page Layout

Another potential application area of our approach is in graphic design for posters, covers and page layout. Figure 4 shows a sample image of typographic design, in which headlines, picture areas, and text areas are considered. In this example we apply aspect-ratio constraints to headlines and pictures, and area as well as dimension constraints to text blocks.

Analysis of Abstract Painting

Constraints are known to be one of the essential elements in defining an artistic style. In the rich combinatorial space of design, an artist's method of selective search through that space is thought to be a key element of style [23]. Our last example focuses on images from abstract painting. This example is taken from Knight [16], who has analyzed stylistic changes in the paintings by De Stijl artists. The purpose of this example is to show the usefulness of exploratory manipulation as a tool to study "styles." Knight studied artists' paintings over periods of time. Dividing them into several stages, she defined generative grammars for each stage of each artist. Our example is based on her analysis on the first stage (1945–50) of Frits Glarner's work.

Primary elements of Glarner's stage I paintings are rectangular divisions; oblique divisions in each rectangle; and grey, black, white, and primary colors (red, yellow, and blue). The constraints in Glarner's work are characterized as follows:

1. The oblique division is angled slightly from 90 degrees.
2. The placement of the oblique is near the edge of a rectangle, forming a wedge shaped "strip."
3. For each rectangular division, the larger wedge is grey or white, and the narrower wedge is grey, black, white or a primary color.
4. The rectangular divisions appear regular.
5. Among six combinatorially possible relationships between two obliquely divided rectangles, only three are used.

Currently, our system is limited to the constraints listed above. Whatever changes a user makes, continuous or discrete, the image keeps the original style, as defined by the constraints. In future work, we would want to let the user easily impose new stylistic constraints if they encounter transformations which are not consistent with the desired style. Conversely, if the constraints prevent us from seeing a legal instance, we may wish to modify or delete some of the constraints. According to the work by Knight, for example, a modification to the second constraint above is one of the changes needed to go on to the next stage of Glarner's work. A sample piece of artwork in the De Stijl style generated by our system is shown in figure 5.

5.3 Results

Overall, our initial experiments in continuous and discrete manipulation have been successful. The system runs at a pleasingly interactive speed on the examples discussed. Continuous and discrete manipulations are integrated seamlessly. The usefulness of direct continuous manipulation was expected. We were gratified to find that continuous manipulation, combined with a variety of continuous constraints, yielded a powerful and simple style of manipulation. In particular, the complexity of interrelated constraints generated continuous movements of the layout that were far too complicated for the user to have formulated on their own.

The feeling when discrete changes are made is much the same: the user is freed from the burden of remembering and maintaining a large and complicated set of constraints. The work so far definitely encourages us to try more changes in the system. The conditions which activate and set the direction of discrete search work well. The set of transformations we implemented seemed trivial at first, but we found that they were powerful enough to generate many alternate solutions. Although the system makes structural changes relatively quickly, the evaluation of each possible alternative slows down as we work with more complex objective functions. The visualization of discrete changes using an interpolating animation is very effective in helping the user to understand what discrete changes have occurred. Without this visualization, users hardly recognize when and where changes have occurred.

6 Conclusion

In this paper, we have introduced a new technique for interaction with discrete/continuous geometric models, allowing users to explore problem spaces having both continuous and discrete parameters. The user can continuously manipulate an object as long as the given constraints are satisfied. When a point is reached at which no further continuous movement in the desired direction is possible, a slightly stronger pull on the mouse triggers a discrete change. This change is based on a rapid, behind-the-scenes local search, constrained by the user's pulling action. A small number of transformation rules are defined to perform actual changes. Alternatives are compared in terms of goodness of the overall design and the magnitude of the change from the previous state, subject to continuous and discrete constraints. The best among the alternatives becomes the new state.

Our approach is most useful for discrete/continuous exploration problems which are not governed by a cut-and-dried objective function, but involve aesthetic or other subjective judgements as well, and that therefore cannot be solved by conventional combinatorial optimization methods.

We have implemented a system that demonstrates seamlessly integrated continuous and discrete manipulation. We have applied the technique to planar layout problems in architecture, PC board layout, page layout, and analysis of art.

Our plans for additional work focus on ways to improve performance, and on applications to a broader class of models and problem domains. From a performance standpoint, the discrete evaluation phase is the largest bottleneck, even though the discrete search is local and limited. We currently call an external nonlinear constrained optimization package at each discrete evaluation; it may be that we can enhance performance by writing custom numerics code, or by using heuristics to further prune the search. Our current system is limited to axis-aligned rectangles; we have been able to handle somewhat more general shapes, such as L-shapes, as constrained sets of rectangles. We intend to move to much more general shapes and shape grammars.

We are currently investigating scheduling problems as a an additional application area. For example, in a resource allocation problem, resources are usually limited in quantity, and can be discrete, such as humans and machines. The activities in a project have constraints on sequential order (discrete) and completion time (continuous). We believe that our methods will carry over directly to scheduling projects such as those that arise in construction project management.

Another direction for future work is the extension of our system to 3D models. A first step is to handle floor plans for multi-story houses, which can be treated as a set of 2-D plans which are linked by constraints. For instance, locations of stairways and structural walls must be consistent.

Acknowledgements

This research was supported in part by a Science and Technology Center Grant (#BIR-8920118) and a Research Initiation Award (#CCR-9308353) from the National Science Foundation, by the Engineering Design Research Center, an NSF Engineering Research Center at Carnegie Mellon University, by the Phillips Laboratory, Air Force Material Command, USAF, under cooperative agreement number F29601-93-2-0001, by Apple Computer, Inc, and by an equipment grant from Silicon Graphics, Inc.

References

[1] David Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics* (*Proc. SIGGRAPH*), 23:223–232, 1989.

[2] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. *Computer Graphics* (*Proc. SIGGRAPH*), 28:23–34, 1994.

[3] David Baraff. Interactive simulation of solid rigid bodies. *IEEE Computer Graphics and Applications*, 15:63–75, 1995.

[4] Christopher Carlson. Structure grammars and their application to design. Technical Report EDRC–01–09–89, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA, 1989.

[5] George Celniker and Dave Gossard. Deformable curve and surface finite-elements for free-form shape design. *Computer Graphics*, 25(4), July 1991. Proceedings SIGGRAPH '91.

[6] Ulrich Flemming. Wall representations of rectangular dissections and their use in automated space allocation. *Environment and Planning B: Planning and Design*, 5:215–232, 1978.

[7] Ulrich Flemming. More than the sum of parts: the grammar of Queen Anne houses. *Environment and Planning B: Planning and Design*, 14:323–350, 1987.

[8] Ulrich Flemming and Robert F. Coyne. A design system with multiple abstraction capabilities. In *Avignon '90: Tools, Techniques & Applications*, volume 1, pages 107–122. EC2, 1990.

[9] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. User's guide for NPSOL (version 4.0): A fortran package for nonlinear programming. Technical Report SOL 86-2, Stanford University, Stanford, California, 1986.

[10] Michael Gleicher and Andrew Witkin. Through-the-lens camera control. *Computer Graphics*, 26, 1992. Proc. Siggraph '92.

[11] Michael Gleicher and Andrew Witkin. Drawing with constraints. *The Visual Computer*, 1994.

[12] John Grason. *Methods for the computer-implemented solution of a class of "floor plan"*. PhD thesis, Department of Electrical Engineering, Carnegie Mellon University, 1970.

[13] Jeff Heisserman. *Generative geometric design and boundary solid grammars*. PhD thesis, Department of Architecture, Carnegie Mellon University, 1991.

[14] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *Int. J. of Computer Vision*, 1(4), 1987.

[15] Terry W. Knight. Designing with grammars. In Gerhard N. Schmitt, editor, *CAAD Futures '91 Proceedings*, pages 19–34, 1991.

[16] T.W. Knight. Transformations of De Stijl art: the paintings of Georges Vantongerloo and Fritz Glarner. *Environment and Planning B: Planning and Design*, 16:51–98, 1989.

[17] H. Koning and J. Eizenberg. The language of the prairie: Frank Lloyd Wright's prairie houses. *Environment and Planning B: Planning and Design*, 8:295–323, 1981.

[18] Sukhamay Kundu. The equivalence of the subregion representation and the wall representation for a certain class of rectangular dissections. *Communications of the ACM*, 31(6):752–763, 1988.

[19] Cornelius Lanczos. *The Variational Principles of Mechanics*. Dover Publications, Inc., 1970.

[20] David P. LaPotin. Mason: A global floor-planning approach for VLSI design. Technical Report IBMC–11657, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, January 1986.

[21] William J. Mitchell, Robin S. Ligget, Spiro N. Pollalis, and Milton Tan. Integrating shape grammars and design analysis. In *CAAD futures '91 proceedings*, pages 1–18, 1991.

[22] W.J. Mitchell, J.P. Steadman, and Robin S. Ligget. Synthesis and optimization of small rectangular floor plans. *Environment and Planning B: Planning and Design*, 3(1):37–70, 1976.

[23] Herbert A. Simon. Style in design. In Charles M. Eastman, editor, *Spatial Synthesis in Computer-Aided Building Design*, chapter 9, pages 287–309. Applied Science Publishers LTD, Ripple Road, Barking, Essex, England, 1975.

[24] G. Stiny. Introduction to shape and shape grammars. *Environment and Planning B: Planning and Design*, 7(3):343–351, 1980.

[25] G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture. *Information Processing*, pages 1460–1465, 1972.

[26] G. Stiny and W.J. Mitchell. The Palladian grammar. *Environment and Planning B: Planning and Design*, 5(1):5–18, 1978.

[27] Mark C. Surles. An algorithm with linear complexity for interactive, physically-based modeling of large proteins. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 221–230, July 1992.

[28] Demetri Terzopoulos, Andrew Witkin, and Michael Kass. Energy constraints on deformable models: recovering shape and non-rigid motion. In *Proc. AAAI-87*, Seattle, 1987.

[29] Demetri Terzopoulos, Andrew Witkin, and Michael Kass. Symmetry-seeking models for 3D object reconstruction. *International Journal of Computer Vision*, 3(1), 1987.

[30] Jeffrey A. Thingvold and Elaine Cohen. Physical modeling with B-spline surfaces for interactive design and animation. *Computer Graphics*, 24(2):129–138, March 1990.

[31] William Welch and Andrew Witkin. Variational surface modeling. *Computer Graphics*, 26, 1992. Proc. Siggraph '92.

[32] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. *Computer Graphics*, 24(2):11–21, March 1990. Proc. 1990 Symposium on 3-D Interactive Graphics.

[33] Andrew Witkin and William Welch. Fast animation and control of non-rigid structures. *Computer Graphics*, 24(4):243–252, July 1990. Proc. Siggraph '90.

[34] D.F. Wong, H.W. Leong, and C.L. Liu. *Simulated Annealing for VLSI Design*. Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061, 1988.

[35] Lin S. Woo, C.K. Wong, and D.T. Tang. Pioneer: a macro-based floor-planning design system. *VLSI System Design*, pages 32–43, August 1986.