An approach for a graphical CAD system that is capable of representing physical and geometric aspects of a design using high-level constraints (HLCs) is presented.

A prototype spatial planning system incorporating constraints is used in an interactive manner to refine designs by following an iterative approach which uses visual information to evaluate the design at each stage of iteration. High-level constraints aid this iterative approach by influencing (or constraining) the behavior of objects as they are interactively manipulated during the design stage of problem solving. High-level constraints also define the scaling properties of objects which are useful during the construction stage of problem solving. This system and its implications for the design of CAD systems incorporating HLCs are discussed.

Studying the Value of High-Level Constraints.

There are two main questions we undertake to answer [Papper 90].

1. We wish to determine a role that constraint-based CAD systems can play in the process of design. We hypothesize that adding constraints to highly interactive systems is useful for space planning applications.

2. We also wish to determine the advantages of using objects with predictable behavior during the design process. We are exploring HLCs as a method to describe complex interacting behaviors of objects.

Constraints are used to model various physical properties of solid objects (including gravity and friction) and to control the way in which objects scale. Constraints aid in manipulation during design by defining how objects should behave when they are manipulated.

A prototype system was constructed and used to solve a studio layout problem to ascertain the value of the system for design and the value of constraints for aiding in manipulation. The system uses direct manipulation [Shneiderman 83] of graphical objects in a graphical environment as an interaction paradigm. The system provides 3-dimensional perspective, 2-dimensional orthographic, and text views of objects, each updated in real-time. The problems handled by our prototype were restricted to objects representable with rectangular solids oriented along the Cartesian axes.

The test results indicate that constraints aid manipulation of objects as users capitalize on the predictable behavior of objects. Subjects learn the behavior of objects easily and use this information in useful ways. In addition, there are indications that constraints can reduce users' cognitive loads at the user interface level. The coupling of constraints with the interactive, 3-dimensional aspect of the system encourages the use of the system at earlier stages of the design process than simple geometric drafting systems.

The Application of Constraints to CAD.

A traditional CAD system uses a set of unrelated primitive entities and operations (move, scale, rotate) to model designs of physical objects. It also provides a grouping or layering mechanism to deal with abstract parts of designs. This allows a complex design to be created by building it out of primitive parts. One typical, general-purpose system is AutoCAD [Raker and Rice 85].
Parametric systems add a level of built-in knowledge of a specific set of parameterized objects. Design features are keyed as input parameters to objects, allowing users to create customized objects by substitution of specific values for the parameters [Gross 89]. The customized objects are used to construct designs.

Yesios suggests that for a CAD system to aid in design, entities must have semantically consistent operational behavior [Yesios 86]. A system should represent the constraints and limitations of designs. This may include constraints on price, size, strength, or clearance requirements. He suggests that this goal can be accomplished by: 1) lists of rules of behavior (constraints) among entities or; 2) incorporating information in the internal (data) structure of entities (parametric techniques).

Constraints can be used to specify relationships and provide design flexibility for users. A constraint is a user-defined rule or relationship created among one or more entities (such as architectural elements) that is maintained by a constraint satisfaction mechanism.

It is not at all obvious how we can effectively embed constraints into CAD systems. Early examples, including Sketchpad [Sutherland 63] and URBAN5 (Negroponte 70), suggest ways of incorporating constraints, but have not been implemented on today's faster computers that allow interactive 3-dimensional solid modelling. Recent increases in computational speed of computers and graphic computation performed in hardware allows constraints to be used in an interactive system.

Our system uses HLCs to model geometric and physical properties of objects so they behave in a predictable manner. Constrained manipulation coupled with a 3-dimensional, graphical, interactive interface encourages an iterative approach to design. However, in order to make the system computable for the purposes of this experiment the constraints are designed to address a limited application we call space planning.

Constraints.

A constraint is a relationship or rule that is always valid. A constraint satisfaction mechanism ensures that all constraints in a system are valid. As the state of the system changes, the constraint satisfaction mechanism alters variables (or data) in an attempt to keep all constraints satisfied.

An algebraic constraint can be defined as a set of variables, operations, and constants separated by a relation operator. The relationship is two way, meaning that variables on either side of the relational operator may be altered (and the proper variables will change to keep the constraint valid). For example: \( X > \text{or} = Y + Z \cdot 3 \) and \( X = Y + 5Z \) are two constraints. If \( X \) changes value the constraint satisfaction mechanism alters \( Y \) and/or \( Z \) until the constraints are valid. Note that there may be many possible solutions for the set of constraints. It is the constraint satisfaction mechanism's task to decide which one(s) are appropriate.

Algebraic constraints may be used to create objects with certain properties. A line can be kept horizontal by constraining its endpoints to have the same Y value. Two horizontal lines could be constrained to have equal length by constraining their magnitude in the X direction. Other uses of algebraic constraints include constraining variables to values of input control devices, solving simultaneous equations, and maintaining the integrity of an electrical circuit.

Constraint satisfaction systems generally attempt the relatively efficient methods of propagation of known states [Steele and Susman 79] or propagation of degrees of freedom [Borning 81] to solve a set of constraints. These methods operate by substituting completely known variables (of solved constraint relationships) into constraints that are almost solved, and continuing until all variables are known and all constraints solved.

If this fails, a much slower technique called relaxation [Sutherland 63] can be employed. A comprehensive discussion of constraints and constraint satisfaction can be found in [Leler 88]. Leler also discusses many other constraint satisfaction techniques and describes basic limitations of constraint satisfaction systems.

Other systems employing algebraic constraints include Thinglab [Borning 81], Sketchpad.
[Sutherland 63], a constraint language by Steele (Steele and Susman 79), another by Leler [Leler 81], and Co, a constraint language by Gross [Gross 89].

The HLCs in our system provide complex behavior for objects (gravity, anchoring, pushing, and friction) which can be specified by the end-user. Combinations of HLCs are used so objects suit a particular design application. Unfortunately, HLCs do not provide the range of flexibility that algebraic constraints provide (although both HLCs and algebraic constraints may coexist in one system). High-level constraints provide the quickness and complexity of parametric systems since the algorithms for HLCs are hard-coded into the system, while still allowing users control over the application of constraints. High-level constraints are easier to specify than algebraic constraints, but are not as straightforward as parametric systems.

Objectives.

The purpose of building this space planning system is to investigate the potential of using constraints in an interactive, graphical CAD system. Evaluation of the system to answer our questions is performed using an example 3D model of a house. The software is written in Common LISP and runs on a Silicon Graphics workstation, using a Silicon Graphics Personal Iris (4D-25G). The configuration of the system in terms of its key components: blocks, manipulation, graphical display, HLCs, and object types, is now described.

Prototype System Description.

Our system is an interactive 3-dimensional graphic interface and HLCs to aid in solving space planning problems by encouraging the use of an empirical approach to design. The system is based on: real-time interaction and visual feedback to ensure that changes in objects are reflected in the current design almost instantaneously; constraining the geometric scaling properties of objects to create objects that scale in reasonable and predictable manners; and constrained manipulation so objects behave similarly to the way they would when manipulated in the environment. The system is implemented on a Silicon Graphics Personal Iris (4D-25G). The configuration of the system in terms of its key components: blocks, manipulation, graphical display, HLCs, and object types, is now described.

Blocks: Blocks have attributes of length, width, height, position in space, color, and various physical constraints (physical constraints are a subset of HLCs). Blocks are combined to create objects. An object is a collection of blocks, assembled with relationship constraints (a subset of high-level constraints). Note that an object can consist of one block.
Manipulation: Blocks and objects can be moved, scaled, and rotated dynamically. The system maintains the integrity of objects subject to constraints and updates the display of objects in real-time. Manipulation can be carried out with constraint checking disabled. There are also accessory commands to precisely align, assemble, copy, create, and delete blocks and objects.

Views: 3-dimensional perspective, 2-dimensional orthographic, and text views of objects are available. The design can be viewed from any point of view in the orthographic and perspective views. Blocks can be manipulated in any view.

High-Level Constraints: These can be broken into two categories: physical constraints which are applicable to individual blocks, and relationship constraints which define a relationship among two or more blocks.

Physical Constraints: These rules assigned to blocks so that they can be manipulated similarly to that in our environment include:

Gravity: Objects with gravity either rest on other objects or at some floor level.

Friction: All blocks resting on top of a selected block move with the selected block (as long as each block has the friction constraint).

Pushing: Since objects are solid, collisions among objects causes them to be pushed.

Anchor: Objects can be anchored in space so that they cannot be pushed by other blocks.

Fix: Objects that are fixed cannot change in size.

Relationship Constraints: These rules are used to create and maintain generic objects with certain scaling properties which are useful for specific design problems. Objects with these properties are called scalable. The relationship constraints are:

Attach: Blocks that are attached are constrained to move together. A set of attached blocks defines an object.

Edge: The edge constraint between two attached blocks constrains their positions (relative to the edge where the blocks are attached). Edge constrained blocks remain near the edges they were originally attached to during scaling operations. For example, consider a table leg edge attached to a table top. The leg maintains its position relative to the corner of the top nearest to where it was attached.

Resize: The resize constraint functions similarly to the edge constraint, but constrained blocks are scaled (edge constrained blocks are moved) as the selected block is scaled.

Scale: The scale constraint maintains the relative sizes of a set of blocks. If one block is scaled by an amount delta, then all constrained blocks are scaled by delta. A set of blocks that have the scale constraint amongst each other are called clones.

Object Types: An object (a set of attached blocks) can be named and stored as an object type so that many instances of it can be created. Instances are objects created as copies of a specific object type (instances are constrained to

Figure 1: Illustrates how the scaling function can operate.
each other with the scale constraint). Instances all have the same characteristics as the original object type. Thus, modifying any one instance’s size will cause all instances’ sizes to change. Copies are objects created from an object type that are not constrained to each other. An example object type may be a table. It is formed from four cloned legs, each edge attached to the table top. If two such tables are created as instances they behave the same. Thus, when one table is lengthened so is the other. If a third table is created as a copy of the first one, it will not behave the same. If the first table is lengthened, the third table will not lengthen.

A Typical Space Planning Exercise

The system is used in two stages in this exercise for space planning. The first (construction) stage is used to create the objects that are used for problem solving during the second (layout) stage.

During the construction stage, blocks and constraints are used to create object types at a suitable level of abstraction for the current design. Users require knowledge of the constraint mechanism to model the behavior of objects. They use constraints to model the manner in which objects are scaled so that one object type can easily be scaled into an object of exact dimensions.

In the layout stage of problem solving, designers manipulate objects to create and explore different space planning arrangements. Designers are mainly concerned with visual aspects of design and use the graphical views to evaluate designs.

A room (studio) layout problem is solved by creating object types (that represent furniture and a room) and then arranging instances of objects (furniture) to find a suitable layout. For this problem one can create a room object type and furniture object types. The furniture includes: wall-units, a computer, a table, and chairs. The creation of the room type is described first.

The room type is created by assembling wall and floor blocks with relationship constraints and putting physical constraints on these blocks. For our example, the front wall is longest along the X axis, the east wall is the right hand side wall (when the room is viewed from the front), vertical is the Y direction, and depth is the Z direction. Other aspects of the room, such as doors and windows, can be added to room instances to suit the particular room. The room type is created with the friction constraint so that furniture in the room moves with the room if the room is moved to a different location (rooms might be jiggled around during a floor plan problem). Walls can be displayed in wire-frame so contents of the room can be viewed. If a room of greater detail were required, the walls could be constructed from its component parts (for example, two-by-four studs or concrete blocks). The room type is created so it is easily scalable to different lengths, widths, and heights. For example, lengthening (scaling along the X dimension) the floor will cause the entire room to lengthen (the east and west walls remain at the edge of the floor, the front and back walls lengthen with the floor).

To design a layout, the room is created, furniture is created and positioned in the room, and then furniture is manipulated to explore different layouts. Furniture is created as instances of object types and positioned in 3-dimensional space in the room. The cursor specifies two points of position and the view direction (top, front, etc.) determines the third point in space. For instance, in a top view the user can specify an X and Z point in space. The Y point is chosen by the system so that the object rests above the other objects in the view. In a top view, a table created and positioned over the floor will be positioned on top of the floor; if the table was positioned in a bottom view it would lie under the floor.

After creating furniture and placing it in the room, some object may require scaling so they match the relative size of the articles they represent. The relationship constraints ensure that objects can easily be scaled to the required dimensions. Some objects may be out of place when they are placed in the room. For example, the computer may be on the floor instead of on the table, or the chair may be on the table instead of the floor. The computer has the gravity constraint, so turning off constraints during manipulation will allow the computer to be moved vertically and positioned over the table. When manipulation is finished and the
physical constraints are turned on, the computer will fall to the table (see Figure 2). If a chair was accidently placed on the table, simply moving the chair off the table will cause it to fall to the floor because of the gravity constraint.

Different arrangements of furniture can be explored to evaluate possible layouts of furniture for the room. Arranging furniture using the system is similar to moving furniture at home. The physical constraints aid manipulation so users can concentrate on designing a layout. Objects stay on the floor due to gravity. Friction ensures that objects resting on tables remain there as the tables are moved. Objects that collide push one another until they hit walls or other anchored (stationary) objects. For instance, if all chairs are tucked into the table and one chair is moved, the chair will push the table, and subsequently all the chairs, along the floor. Also, the user can observe the effect of different sized wall units on the arrangement of the room by interactively scaling the units and evaluating the resulting design.

![Figure 2. Moving the computer on to the table](image)

**Figure 2.**

**User Testing**

User testing was carried out to better evaluate the system's usefulness for exploratory and iterative designing and the role of physical constraints for object manipulation.

We chose three fourth year landscape architecture students (these are called designers) and four novice computer users (these are called novices) to design a studio work area. Both these groups were familiar with the purpose of arranging the studio and were familiar with the studio itself. In addition, the designers were familiar with design in general and were familiar with the computer system we used (but not our particular software system). The novices had no experience with the computer system or formal training in any design procedures. Each subject was given 15 minutes of hands-on individual instruction, and 40 minutes to work on the studio layout problem.

![Figure 3: Example Studio Layout](image)

**Figure 3: Example Studio Layout**

The problem presented to the subjects was to design a suitable arrangement of furniture in their design studio, which occupied a section of a classroom. For the experiment, the subjects were given a representation of the studio itself on the computer screen and pre-defined object types to represent each type of furniture available for the studio. Figure #3 contains a screen picture of an example layout for the studio.

Each subject was taught the basic features of the
system (nine manipulation commands and the function of four [4] physical constraints). This took approximately 15 minutes. The commands taught were move, scale, rotate, move a set, place object, delete, create, color, and undo. They were told that there was an existing data base of objects, and each object behaved somewhat similar to the article it represented. They were also told that the behavior of the objects was due to the gravity, pushing, friction, and anchor constraints.

The subjects were given the following information about the behavior caused by each physical constraint: that objects were solid and had the friction constraint, that if they collided during manipulation they would push each other; that some objects (the room and the dividers) were anchored so they couldn’t be pushed; and that all objects had gravity so they always rested on other objects or the floor. The subjects were instructed on how to by-pass constraint checking during manipulation and told that they could permanently add or remove constraints from objects. They were also told they could create any arrangement, for as many people as they wanted, using whatever space and objects they wanted. They were instructed to take into account what they already know about the situation, including the fact that half the room is occupied by a class, and that the studio area is used by several people. The subjects were informed that they could take into account any functional aspect of the design, including those not represented by the system (this would include glare from windows, or noise from the adjoining classroom area). In addition, subjects were told to think aloud, verbalizing any questions or plans and strategies. The subjects' verbal and video data was gathered to help answer our questions.

User Testing Results.

We wanted to discover how useful constrained manipulation and the interactive, visual aspects of the system were during the process of design. We used subjects' video and audio data to record observations and to determine their procedure for solving the studio problem. Specifically, we looked for how they used the system to refine a design. We viewed the data to determine the style of manipulation used during each of the two stages.

We also recorded the number of visits to each group of objects in the studio. A visit is the set of object manipulations performed consecutively on objects in one area of the studio. Data to determine the usefulness of constrained manipulation was gathered from observations of situations involving the use of each constraint. An applicable situation is one in which the subject uses a constraint with or without the knowledge that constraints are affecting the manipulation. We only collected data for situations involving the move command. We did not deal with rotate or scale commands because the scale command was not used, and the rotate command does not involve the physical constraints to a significant degree. An applicable situation is categorized as beneficial or accidental. A situation is accidental if the result of manipulation is unexpected, due to constraints, and the subject must repair the effects of the manipulation. For example, an accident occurs when a subject moves a block sitting on a desk and it unexpectedly falls off the desk due to gravity (and it must be placed back on the desk). Another example is when an object is accidentally pushed by another (and the subject must correct the position of the pushed object). Note that if no repair of the effect of the manipulation is required then it is not accidental. Otherwise, applicable situations are beneficial. For example, beneficial situations include subjects pushing a group of objects, using gravity to make an object fall to the floor, or using pushing as feedback to determine when one object has moved sufficiently to rest against another.

Each subject used the system for the full 40 minutes allotted. On average, subjects created 3.5 different groups or areas of the layout and visited each group 1.5 separate times. They took 7 distinct time-outs to view the scene (at which time they did no manipulation with the system). They used the viewing operations on average 6 times. There was a total of 25 move manipulations, 20 of them were applicable (involving the use of constraints). The average number of objects created for the studio layout was 23. Table 1 shows the breakdown of the average of all subjects' use of constraints during move operations.

217
Table 1: Averaged total beneficial and accidental situations for the move command (total number applicable situations is 20 from 25 total situations). Notes: Total number notices was 4, number of designers was 3. All values are rounded.

<table>
<thead>
<tr>
<th>Constraint type</th>
<th>Number</th>
<th>Designer 2</th>
<th>Mode 1</th>
<th>Mode 2</th>
<th>Mode 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beneficial push</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Accidental push</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Beneficial anchor</td>
<td>12</td>
<td>18</td>
<td>12</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>Accidental anchor</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>Beneficial gravity</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>17.5</td>
<td></td>
</tr>
<tr>
<td>Accidental gravity</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Beneficial friction</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Accidental friction</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The percentage of applicable pushing situations was 20% of the total situations (most objects were moved individually). Most (3 out of 4, or 75%) push situations were beneficial. These beneficial situations included using push to move a group of objects or as feedback to determine when two objects were in contact. The accidental situations occurred when subjects pushed objects out of position.

In general, subjects did not take advantage of the push constraint to move several objects at once; they tended to move blocks individually. This could have been caused by three possible factors: 1) the subjects were not familiar enough with the basics of the system to move objects any other way than in a straightforward manner; 2) the subjects were not familiar enough with the design to take advantage of moving groups of objects; and, 3) the subjects were not sure how a single object manipulation would affect the design, let alone the movement of many objects. Subjects did not think of pushing several objects as a natural operation. Even when the operation of pushing several objects was demonstrated, subjects forgot that objects could be manipulated in this manner (indicating that the operation was not natural since it was not remembered). However, users do realise that objects should not and will not violate each others' space. This was shown from observations of the use of push as a cue or special event that let subjects know when a moved block had contacted another block. The subjects recognized the cue, indicating that this type of behavior of objects was natural (they recognized the cue even if they didn't use pushing explicitly).

Anchor was the most commonly used constraint, as it was used in 17 out of 20 applicable move operations. 65% of the anchor situations were beneficial, being used to move an object against an anchored object (usually a wall, occasionally a divider). Accidental uses of anchor occurred when subjects wanted to move an object inside the studio (after accidentally creating the object outside the studio). In these situations, after realizing they couldn't push through the studio walls, subjects turned off constraints to move objects in the studio. In general these accidental situations did not hinder object manipulations.

Subjects took advantage of the anchor constraint without being fully aware of its use: they knew that even before moving any objects (they verbalised their intention to move something against a wall) that certain objects (such as walls) would stop the movement of other objects (showing that they were not aware of anchor, but of the properties of walls). Eventually they came to understand that only certain objects were anchored. As subjects learned the system, they did not slow down the movement of objects as they approached walls. Thus, the anchor constraint appears to aid manipulation by determining the extent, or magnitude, of operations.

Gravity was involved in about 25% of the situations, 17.5% of these were beneficial, 7.5% were accidental. Gravity was used beneficially to place objects on the floor that were originally (and unintentionally) created to rest on top of the walls of the studio. Most accidental situations occurred as objects fell off tables on to the floor and had to be placed back on the table.

Gravity was not immediately obvious to subjects, although once they realised its function, it was used to move objects to the floor. Sometimes they forget that they could use gravity to make objects rest on the floor, even when they had done this action before. The gravity constraint is implicitly used during each operation, making sure that objects are always resting on other objects or the floor. Without this constraint, objects may accidentally be positioned floating slightly above other objects.

Friction was used in 3 out of the 20 applicable situations, with no accidental uses. Friction was used to move a table with computers on top. Subjects expected the effects of friction (even
without being aware of the constraint). However, they still took a relatively long time to realize its effects. In fact, when subjects turned off constraint checking during a move operation (to move a desk through a divider for example), they were surprised when the computer on the desk no longer moved with the desk. These observations may indicate that the function provided by friction seemed natural to subjects.

Analysis of User Testing Results.

Results of user tests indicate that the system is not used to explore possible conceptual frameworks for potential designs. However, the system is used in an iterative manner to refine a conceptual framework into a solution. Both the presence of visual information and constrained manipulation (using HLCs) were used during this iterative process.

We found that constrained manipulation aids object manipulation in three important ways:

1) the manipulation of objects is easy to learn for beginners;
2) constrained object manipulation is helpful for expert users; and
3) users don’t have to fully concentrate on object manipulation tasks.

Our test results indicate that the behaviors incorporated into the objects for the studio problem are easily learned. The observations of subjects show that they learn to move objects very quickly, taking advantage of constraints without being aware of them. This is shown by the high percentage of non-explicit situations observed during user testing. The low percentage of accident situations shows that users do not make many errors. Subjects intuitively understand object behavior, so that once they are in the appropriate mode they know how to manipulate objects effectively. For instance, users presented with the studio problem assumed that objects were stopped by walls and that objects resting on desks moved with the desks.

Constrained manipulation can provide efficient object manipulation without an extensive set of manipulation commands (only move, scale, and rotate commands are required). Experts manipulate objects by using the constrained behavior of objects in novel ways. For instance, they use a block having the anchor constraint to line up objects against it. The power of constrained manipulation is shown by examples of experts using the system: experts use push and friction to move sets of objects in one operation; they line up objects by pushing them against an anchored object; they use one block as a pusher block to move several objects in the same mode; and they use gravity along with pushing, to move an unattached stack of objects up and down, while keeping the objects stacked on top of each other.

The constrained behavior of objects did not change for our beginner or expert users. Thus, HLCs seem to combine ease of learning (because it is a simulation users already know about) with power (by using the behavior in novel ways).

To use HLCs effectively users must discover novel ways to put constraints to use to aid manipulation. It is hypothesized that as users experiment with the system they will develop a useful set of techniques that use HLCs for problem solving. They do not need to be taught specific techniques, but learn them on their own by experimenting. The fact that beginning users did not use constraints in novel ways may be explained by assuming that they are concentrating too hard on the basics of the system to experiment.

Constrained manipulation has the potential of freeing users from tedious aspects of design (such as specifying the magnitude, or direction of an operation). Freeing the users may allow them to concern themselves with more important aspects of designs. This behavior was observed when subjects placed objects against a wall; they knew the wall would stop the object at the correct position. The behavior caused by the anchor constraint effectively frees users from having to specify an exact magnitude for the move operation; subjects simply moved the cursor by an extreme amount and let the system delimit the operation. Other (more subtle) aspects of constrained manipulation may also free users from tedious tasks, although these aspects of the system were not tested. For instance, the gravity constraint continuously ensures that an object cannot be displaced upward, ensuring that an object will always rest on another. This would be important for maintaining the integrity of the
design (tables that floated above the ground would provide invalid visual information) and to ensure correct object manipulation (if objects floated over each other the friction constraint would not work). Continued user testing, looking at manipulation with constraints on and off, may confirm the hypothesis about the subtle aspects of constrained manipulation.

Summary.

Our results appear to indicate that the visual, graphical, 3D nature of the system encourages its use in an iterative manner. A better interface may encourage the use of the system in a more exploratory manner. High-level constraints provide an extensible method of incorporating behavior into objects. They operate at the user interface level providing useful tools for manipulating objects.

The constraints are not currently capable of providing insight into functional aspects of design. It is the 3-dimensional graphical nature of this system that gives users insight into their problems. The incorporation of constraints for increased functionality, although computationally expensive, may increase the usefulness of the system in a way that allows the system to reduce user's cognitive load during design, not only at the user interface level.

Future Enhancements and Research.

To enhance the current system, skewed [5] blocks, additional high-level constraints, constraint macros, algebraic constraints, organisation tools, a diagram mode, and better object construction tools could be added.

Skewed blocks: Providing for skewed blocks would increase the applicability of the system. It would be useful for almost any room design problem. If the number of blocks in the system were large, then skewed objects could be built up with many blocks.

Additional high-level constraints: Another useful high-level constraint is "N" objects each at "delta L" spacing. This constraint would be useful to constrain functional aspects of a design to object manipulation. For instance, a wall stud could be generated for every 16 inches of the wall or a lamp post may be created for every 10 meters of sidewalk space. As the "L" value (ie. total length of wall) increases (during an operation) more objects in correct locations are automatically added to the scene. This is an example of a high-level constraint that specifies the behavior of a set of objects. It cannot be duplicated with algebraic constraints.

A constraint similar to gravity, but operational in the X and Z directions (as well as the Y direction), would be useful and easy to implement. This constraint would ensure that objects are positioned as far as possible to the right, left, back, or front of a design. A constraint similar to friction, but also operating in the X and Z directions would also be useful. This constraint could be used to temporarily drag objects around. Consider moving an object, brushing up against another object and dragging (as opposed to pushing) it to a new location. The constraint could be turned off and on during manipulation.

Macro: Macros could be incorporated into the system for heavily used combinations of constraints. For instance, a leg-tabletop macro would describe the appropriate edge and attach constraints to correctly constrain a leg and a top to create a table. Macros would also enable novice users to create their own constrained object types because constraint relationships could be set up quickly, and easily. The novices would not have to know what particular constraints were set up, or how the relationships constraints work. Rather, they would only have to know about high-level descriptions of objects, like table leg and tabletop.

Algebraic constraints: Algebraic constraints could be added to the existing system. A macro could represent a set of algebraic constraints that collectively represented a high-level concept. Thus, functional aspects of designs could be constrained allowing the system to address many more design issues than the current set of HLCs. For instance, they could represent hallways that met fire regulations for width, urban design masking constraints such as relationships between building height and land area, or constraints on the entrances for a house and a garage.

220
Organization tools: A mechanism to help the user organize and switch among different levels of representation of an object would be a useful part of a constraint envelope mechanism. The user would be able to quickly switch between different user-defined abstractions of an object. Any unused data in the current representation of the object would be retained, but not in main memory (a mechanism that is transparent like virtual memory could be used to keep track of data for objects).

Diagram mode: A diagram mode could be used to display and allow direct manipulation of constraints. Relationship constraints could be represented with lines among blocks and physical constraints could be represented with symbols. This view would speed up the process of assigning constraints among blocks. Time consuming menus are currently used to set up each constraint.

Object construction tools: A mechanism to aid in assembling blocks into objects would make the precise alignment of blocks at their edges easy to accomplish. Since all objects are blocks, the system should be able to automatically line up blocks at their edges as they are moved into place to create an object. Different styles of this mechanism would be used to get different effects. For instance, squareX may ensure that a manipulated block would line up with surrounding blocks along the X edges.

The manipulation of objects during the layout stage can be accomplished by non-computer users with little effort. However, manipulation of blocks to create objects during the construction stage is fairly difficult. Using the system during this stage requires understanding the system concepts and user interface. The addition of the enhancements described in this section should make it easier to construct objects using the system.

Conclusions.

One goal for the interface of our CAD system was to allow users to interact with their designs in a free-form manner, giving the user a sense of play. The high-level physical constraints attempted to provide this type of interaction. In one sense they have worked well as beginning computer users have moved objects in space planning situations with less than 15 minutes of instruction. Unfortunately, the creation of scalable object types is much more tedious and difficult for users. Other aspects of the interface do not work so well, the command interface is cumbersome, and the response time is too slow for a truly usable system (due to the drawing time). The quality of the scenes is only good enough to get an idea of the fit of a group of objects. Detailed geometric representation, lighting, and shadow aspects of scenes are not represented in the current system. The 3-dimensional aspects of the system require a relatively long time to redraw scenes in order to simulate motion. We will, therefore, have to wait for faster graphics capability before these tools can be applied to general design problems.

The constraints in our system can only represent a fraction of the interesting aspects of space planning problems. We need a much richer set of constraints than those provided with our system. The use of our system for complex problems has not been tested, and it is exactly this type of problem where constraints would be most useful. Only when the complexity of the problem gets beyond the designer will constraints be truly useful for design. We will have to wait for faster computers, alternative constraint solving algorithms, or hybrid constraints (like our physical constraints) to test the use of constraints for complex problems.

Nevertheless, this work shows that constraints are beneficial at the user interface level. Constraints may reduce the cognitive aspects of object manipulation. They may be useful to represent functional aspects of designs. The system also shows that response time is the major difficulty of incorporating constraints into a design system such as ours. However, when faster computers are available, the rewards of using constraints in a design system may outweigh their implementation problems.

Acknowledgements.

The authors would like to acknowledge the funding support of this work by: The Information Technology Research Center of the Province of Ontario, Apple Computer Inc., and Silicon Graphics Inc.
Footnotes.

[1] Mike Papper is currently a researcher at the Royal Melbourne Institute of Technology, Australia.


[3] Enough so that the constraints aid space planning, while still being solvable in real-time.

[4] These were pushing, anchor, gravity, and friction. Fix was not applicable to the layout problem.

[5] Skewed blocks can be positioned at arbitrary angles relative to the Cartesian axes.

Bibliography.


