

# On the Utility of Semantic Constraints

G. Smith and W. Stuerzlinger  
Dept. of Computer Science, York University, Toronto, Canada  
{ gsmith | wolfgang }@cs.yorku.ca

**Abstract.** Content creation for computer graphics applications is a laborious process that requires skilled personnel. One fundamental problem is that manipulation of 3D objects with 2D user interfaces is very difficult for non-experienced users. In this paper, we describe a system that uses semantic constraints to restrict object motion in a 3D scene, making interaction much simpler and more intuitive. We compare three different levels of semantic constraints in a 3D scene manipulation program with a 2D user interface. We show that the presented techniques are significantly more efficient than alternate techniques, which do not use semantics in their constraints. To our knowledge, this is the first evaluation of 3D manipulation techniques with 2D devices and constraints.

## 1 Introduction

Many applications are readily available in the areas of 3D modeling and scene construction, but in general, these products are difficult to use and require many hours of training. For example, products such as *Maya* (Alias|wavefront) and *3D Studio Max* (Discreet), have dozens of menus, modes and widgets for scene creation and manipulation, which can be very intimidating for an untrained user. Our efforts address these difficulties.

The task of creating a 3D scene from scratch is very complex. To simplify the problem, we choose to focus on the creation of complete 3D scenes based on a library of existing objects. Here the challenge is to enable the user to easily add objects and to quickly position them in the environment. In general, positioning an object in a 3D scene is difficult as six independent variables must be controlled, three for positioning and three for orientation.

Our observations of humans rearranging furniture and planning environments indicate that humans do not think about scene manipulation as a problem with six degrees of freedom. The rationale is that most real objects are not placed arbitrarily in space, but are constrained by physics (e.g. gravity) and/or human conventions (ceiling lamps are almost never placed permanently onto the floor or onto chairs). This leads us to believe that an interface that exposes the full six degrees of freedom to the user makes it harder for the average person to interact with virtual environments. Many real objects have a maximum of three degrees of freedom in practice – e.g. all objects resting on a plane. In addition, many objects are often placed against walls or other objects, thus further reducing the available degrees of freedom. This implies that a

two-dimensional (2D) input device such as a mouse is sufficient to manipulate objects in a virtual environment.

In our system, information about how an object interacts with the physical world assists the user in placing and manipulating objects in virtual environments. Each object in a scene is given a set of rules, called constraints, which must be followed when the object is being manipulated.

## 2 Previous Work

Previous work on 3D object manipulation can be classified into two categories: those that use 2D and those that use 3D input devices.

The simplest solution for a 2D input device is to decompose the manipulation task into positioning and orientation. Unfortunately, there is no intuitive mapping of these tasks with three degrees of freedom each to a mouse with three buttons.

Bier introduced 'Snap-Dragging' [1] to simplify the creation of line drawings in a 2D interactive graphics program. The mouse cursor snaps to points and curves using a gravity function. Bier subsequently applied these ideas to placing and orienting objects in a 3D environment [2]. The main features of this system are a general-purpose gravity function, 3D alignment objects, and smooth motion affine transformations of objects. Gleicher [9] built on this work and introduced a method that can deal even with non-linear constraints.

For 3D scene construction Bukowski and Sequin [7] employ a combination of pseudo-physical and goal-oriented properties called 'Object Associations' to position objects in a 3D scene with 2D devices (mouse and monitor). A two-phase approach is used. First, a relocation procedure maps the 2D mouse motion into vertical or horizontal transformations of an object's position. Then association procedures align and position the object. Although intuitive, their approach has a few drawbacks. First, associations apply only to the object currently being moved and are not maintained after the current manipulation. In addition, when an object is selected for relocation, a local search for associated objects is performed. This can result in lag between the motion of the selected object and the motion of its associated objects. Cyclical constraints are not supported.

Goesle and Stuerzlinger [8] built upon the ideas of Object Associations. Each scene object is given predefined offer and binding areas. These areas are used to define constraining surfaces between objects. For example, a lamp has a binding area at its base and a table has an offer area on its top. Consequently, a lamp can be constrained to a tabletop. To better simulate the way real world objects behave, a labeled constraint hierarchy adds semantics to the constraint process. Each constraint area is associated with a label from the hierarchy. A binding area constrains then only to offer areas whose label is equal to or is an ancestor in the constraint hierarchy. In this way, the legs of a chair can be constrained to the floor, or in front of a desk, but never to the wall. Collision detection is used to prevent objects from passing through each other.

Drawbacks of this approach include the following: Once a constraint has been satisfied, there are no means to re-constrain an object to another surface or to unconstrain it. Furthermore, the constraint satisfaction search is global, in that an object will be moved across the entire scene to satisfy a constraint. This has often-undesirable effects for the user, especially because constraints cannot be undone.

A number of authors have investigated the performance of object manipulation with 3D input devices, such as a space-ball or a six degree-of-freedom tracker. Such devices enable direct interaction with a 3D scene. In combination with devices that generate a 3D view, such systems can simulate Virtual Reality (VR).

One of the first researchers to use 3D devices to manipulate a 3D scene was Bolt in 1980 [3]. Subsequently many other researchers studied the creation and manipulation of 3D environments in VR (see e.g. [12][16]). Very few of these systems utilize constraints for object manipulation and even these support only the simplest geometric constraints (e.g. on-plane). Closest to the work discussed here is the 'SmartScene' system by Multigen [18]. This system uses tracked pinch-gloves in 3D as interaction devices.

More recently Bowman et al. [5], Mine et al. [12], and Pierce et al. [13] proposed different 3D manipulation methods that can be applied in a variety of settings. Poupyrev et al. recently also addressed the problem of 3D rotation [15]. For a more complete overview over previous work in this area, we refer the reader to [6].

While it may seem obvious that the introduction of constraints makes interaction in 3D easier, it is unclear how strong this effect is. An extensive search for literature was performed in this area, and no study that addresses was found.

## **2.1 Motivation**

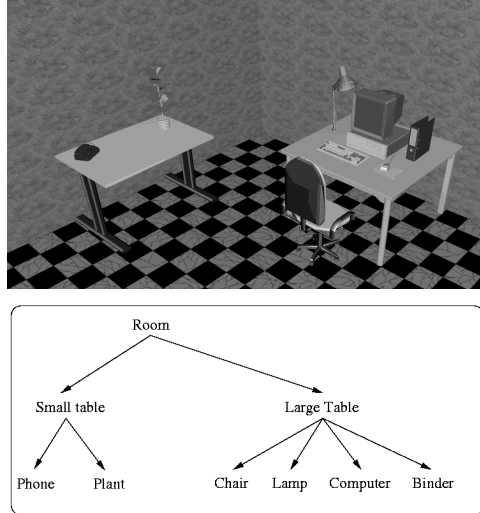
Our main motivation behind this work is that we wanted to analyze how semantic constraints affect user performance in a give task. Such constraints have been introduced to commercial products such as the Smartsce system [18], but to our knowledge, no evaluation has been performed to assess the value of these constraints. Our initial hypothesis was that semantic constraints would greatly simplify user interaction, as they make putting an object into the "right" place very simple.

## **3 The MIVE System**

The MIVE (Multi-user Intuitive Virtual Environment) system extends the work done in [8] by improving the way existing constraints behave, and adding new useful constraints. This work concerns only the interaction of a single user; therefore we disregard the multi-user aspects of the system here.

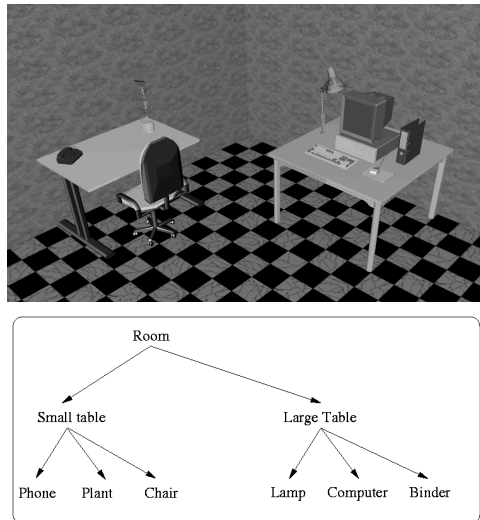
### **3.1 Semantic Constraints**

Every object in the MIVE system has constraints defined for it. Each constraint can be one of two types: offer or binding. Essentially, binding areas will "stick" to offer areas. Constraint relationships are stored in a directed a-cyclic graph called the scene graph. Figure 1 depicts a simple scene, and it's associated scene graph. When an object is moved in the scene, all of its descendants in the scene graph move with it.



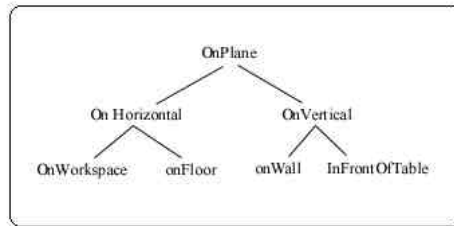
**Fig. 1.** A Scene and its associated Scene Graph.  
Links describe constraint relations

Notice that edges in the scene graph of Figure 1 correspond directly to satisfied constraints in the scene. The user can modify the scene graph structure by interacting with objects in the scene. Constraints can be broken and re-constrained with ease by simply clicking on the desired object, and pulling away from the existing constraint to break it. This allows us to dynamically change the structure of the scene graph. Figure 2 shows the same scene as Figure 1 after the chair has been pulled away from the large table, and dragged under the smaller table.



**Fig. 2.** Scene from Fig.1 after chair has been moved

A labeled constraint hierarchy is used to add semantics to the constraint process. The hierarchy is a tree structure, and defines the behavior of the constraint and offer areas. Each constraint area is associated with a label from the hierarchy, which defines what offer areas this constraint area is allowed to attach to. A binding area constrains only to offer areas whose label is equal to or is an ancestor in the constraint hierarchy tree.



**Fig. 3.** Semantic Constraint tree

A simplified version of the semantic constraint tree is shown in Figure 3. This tree is used to restrict object placements and to make interactions more intuitive. For example, the phone in fig 1 has a binding area on its base, which has the OnWorkspace label associated with it, hence it can be constrained to any offer area with an OnWorkspace label, OnHorizontal label, or OnPlane label. The top of the table has an OnWorkspace label associated with its offer area, so the phone can constrain there. The phone cannot be placed on the floor, which has an OnFloor label associated with its offer area, because the label of the floor's offer area is not an ancestor of OnWorkspace in the tree. Semantics are added in a similar manner to every binding and offer area.

#### 4 Constraint Satisfaction

For our constraint system, binding and offer areas both have a polygon and vector, which represent their effective areas and orientation. A binding area is satisfied by an offer area by aligning their orientation vectors and by translating the binding polygon so that it lies within the offer polygon (excluding the offer polygon edges). If after rotation and translation the binding polygon is not completely enclosed by the offer polygon, then the binding area is not constrained to the offer area. In addition, a binding area cannot be bound to an offer area of the same object: an object cannot be constrained to itself. In Borning's [4] terms, our system implements locally-predicate-better constraints.

To constrain an object, we attempt to satisfy all of its binding areas. For each binding area of an object, we search through the scene to find potential satisfying offer areas. Semantics restrict the offer areas that a binding area is allowed to constrain to. To prevent objects from jumping large distances to satisfy constraints, we only consider constraining an object to offer areas that are close to the object being constrained. Closeness is relative to object size, therefore we consider only objects that are within a sphere with a radius that is twice the radius of the sphere bound of the object.

Using this heuristic, constraints remain unsatisfied until an object is moved close to a valid offer area. It also ensures that objects are always locally constrained. For each binding area, if there are multiple satisfying offer areas, the closest satisfying offer

area found is chosen. The object is moved to connect the binding and offer areas. The bound object then becomes a child of the offering object in the scene graph, and the search is repeated for the next binding area.

Once an object is constrained, its motion is restricted such that the binding areas of the object always remain in contact with the associated offer areas. This essentially removes degrees of freedom from object manipulations. Constraints can be broken with ease by simply pulling an object away from its associated offer area.

## 5 MIVE Constraint Environments

Previous systems have used a more general constraint environment, where objects only know that they must lie on a horizontal and/or vertical surface, such as the Object Association system [7]. We hypothesize that this makes interaction less intuitive because it gives the user less control over how objects behave in the scene. For example, the chair would have a horizontal constraint on its base, and could easily be placed on a table, bed, refrigerator, or any other horizontal surface. We have implemented such a constraint system in MIVE and call it the Partially Constrained (P) mode.

Each object in the MIVE system has a set of semantic constraints associated with it. This mode uses user-defined semantics to restrict object placement to valid locations. We call this the Fully Constrained (C) mode.

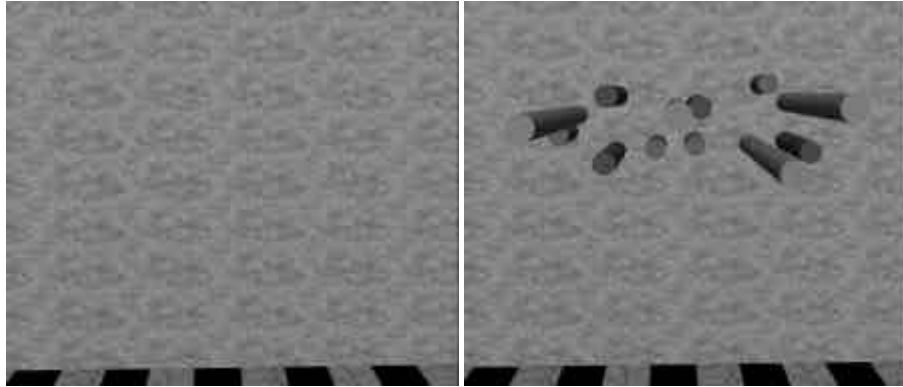
## 6 User Testing

We designed a test where significant semantic differences between object constraints exist to see if a significant difference between the PC and FC modes would occur. Twelve pipes (3 different sizes, 4 of each size) had to be placed onto a wall. The wall had 12 different receptacles where pipes could be attached in FC mode. The test is illustrated in Figure 4.

In the task, we evaluated 3 different modes:

- Partially constrained (P) mode, where the pipes connect to all vertical surfaces (i.e. anywhere on the wall). There is no visual indication where pipes should be attached.
- Non-semantic constrained with drawn offer areas (D) mode, where a pipe can connect to anywhere on wall (as in mode P), but the correct locations are indicated visually by a small transparent green square on the wall. However, each pipe could actually be placed anywhere on the wall.
- Fully constrained (C) mode, where pipes connect only to the correct receptacle. Three different kinds of pipes existed. This is a simplified analogy to having gas, hot water and cold water pipes in a mechanical construction task, where each type of pipe is only allowed to connect to a subset of the receptacles.

Tasks were set up so that no navigation was required for any of the tests. This avoids interference with problems of participants understanding navigation in 3D.



**Fig. 4.** The initial and target scenes for our user test

### **6.1 Participants**

Twelve volunteers participated in this experiment. Participants were computer science students with different experience and backgrounds, different computer skills and different degrees of exposure to 3D computer graphics.

### **6.2 Apparatus**

The MIVE interface was designed to very simple. Figure 5 shows the full user interface of the MIVE program running with its default object list.



**Fig. 5.** The MIVE interface

The MIVE interface consists of three windows: the scene window, the object selection window, and the button window. The scene window sits on the right hand side of the screen. The participant directly interacts with objects in the scene window by clicking and dragging them.

The lower left-hand corner shows the object selection window. Objects are positioned on an invisible cylinder, which is rotated by clicking any mouse button within the

window and dragging left and right. Objects are added to the scene window by simply clicking on the desired object in the object selection window, and clicking on the desired location to add it in the scene window. Drag & Drop is supported as well. To facilitate selection of small objects all objects are scaled logarithmically.

The upper left-hand corner of the window contains buttons for performing tasks such as loading or saving the scene, deleting an object, undoing the previous operation, or quitting the program. There is also a radio button, which can be used to switch between interaction and navigation mode. This functionality was disabled for the tests in this publication.

The MIVE system is implemented in C++ and runs on an SGI Onyx2 running IRIX 6.5. It is based on the Cosmo3D [10] scene graph API.

### **6.3 Interaction**

The interface for MIVE was designed to be as simple and uncluttered as possible. All interactions between the participant and the program are done using a 3-button mouse.

For this test, the three modes use only two of the three buttons. The left mouse button is used to move objects by clicking and dragging them to the desired new location. The middle mouse button is used to rotate the objects. The third mouse button is currently unused in these modes.

### **6.4 Procedure**

A three-minute tutorial was given prior to the testing, at which time the experimenter gave the participant instructions on how to use the system. Each participant was then allowed to experiment less than two minutes with the system before testing started.

Each test began with the participant sitting in front of a computer monitor with a scene displayed. A target scene was displayed on an adjacent monitor, and the participant was instructed to make the scene on their screen look like that in the target scene. The experimenter supervised the participant, and when the task was deemed complete the supervisor instructed the participant to continue to the next task.

Each participant was asked to perform the task in three different modes. The order that the participant performed the modes was chosen using a Latin square method.

For each of the tests, we recorded the time taken by the participant, and the accuracy of object placement compared to the target scene. Accuracy was measured by summing the distances in centimeters between each of the object centers in the participant's final result and the target scene.

## **7 Analysis**

At the end of each experiment task completion time and the modified scene was stored. The Euclidean distance between the participant's solution and the reference solution was computed later on.

### **7.1 Adjustments to Data**

No adjustments were made to the collected data and no trials were excluded.

## 7.2 Computed Formulas

Errors are sums of Euclidean distances. We ignore rotation because no ideal measure for rotation differences exists to our knowledge. Moreover it is hard to find a meaningful combination of translation and rotation errors into one number.

## 7.3 Results

Figure 6 summarizes the results of the test. The thick center line of a box shows the median, the second line is the mean, the box itself indicates the 25<sup>th</sup> and 75<sup>th</sup> percentile, and the 'tails' specify the 10<sup>th</sup> and 90<sup>th</sup> percentile.

The analysis indicates a significant effect between modes C and P ( $p < 0.001$ ). Mode P is clearly slower than mode C, by a factor of 2.5. More detailed analysis reveals that mode C is also significantly faster than mode D by a factor of 1.5. In accuracy, modes C and P do not have a significant difference, but mode C is significantly better than mode P by a factor of 8.6 ( $p < 0.001$ ).

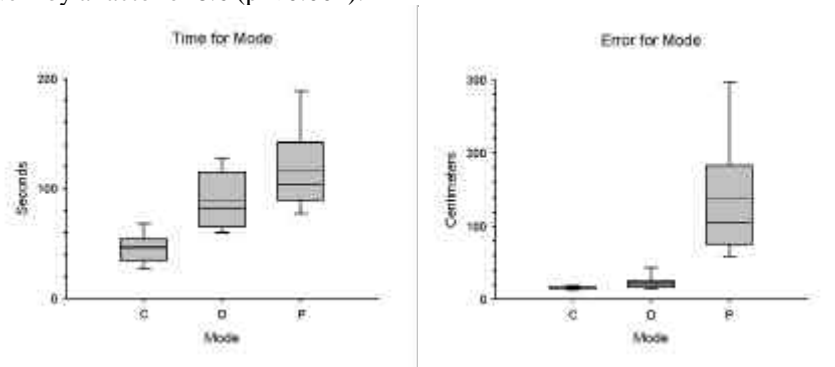


Fig. 6. User test results

Although only a small number of participants took part in this test, the statistical test has an extremely high power ( $>0.99$ ) and we are confident that further testing would only re-confirm these results. The results for the tasks show that the semantic constraints of mode C can provide benefits for scene manipulation in environments where semantic differences among objects exist.

## 8 Conclusion

In this publication we presented a system that allows users to easily manipulate a 3D scene with traditional 2D devices. The MIVE system is based on semantic constraints, which enable an intuitive mapping from 2D interactions to 3D manipulations. The semantic constraints and manipulation techniques encapsulate the user's expectations of how objects move in an environment. Based on user tests we showed that the use of semantic constraints, as opposed to more general constraints without semantics, provide clear benefits for manipulation of 3D objects in a 2D user interface.

The benefits of our interaction techniques become very apparent when one compares the simple MIVE user interface with the complex 3D user interface in commercial packages such as AutoCAD, or Maya that are also based on 2D input devices. We can only hypothesize at the outcome of a test comparing our system with e.g. Maya, but

are confident that it is clearly easier to learn our user interface due to the reduced complexity. In fairness, we need to point out that these packages are also capable of object creation and the specification of animations, which our system does not currently address.

## References

1. Bier, E.A., and Stone, M.C. Snap-dragging. *SIGGRAPH 1986 proceedings*, ACM Press, pp. 233-240.
2. Bier, E.A. Snap dragging in three dimensions, *SIGGRAPH 1990*, pp. 193-204.
3. Bolt, R., Put-that-there, *SIGGRAPH '80*, 262-270.
4. Borning, A., Freeman, B., Ultraviolet: A Constraint Satisfaction Algorithm for Interactive Graphics, *Constraints: An International Journal*, 3, 1-26, 1998.
5. Bowman, D., Hodges, L. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. *Proceedings of ACM Symp. on Interactive 3D Graphics*, 1997, pp. 35-38.
6. Bowman, D., Kruijff, E., LaViola, J., Mine, M., Poupyrev, I., 3D user interface design, *ACM SIGGRAPH 2000, Course notes # 36*, 2000.
7. Bukowski, R., and Sequin, C. Object associations. *ACM Symp. Interactive 3D Graphics 1995*, 131-138.
8. Goesele, M, Stuerzlinger, W. Semantic constraints for scene manipulation. *Proc. Spring Conference in Computer Graphics 1999*, pp. 140-146.
9. Gleicher, M, A Graphics Toolkit Based on Differential Constraints. *Proc. UIST 93*, 109-120.
10. Eckel, G., Cosmo 3D programmers guide. Silicon Graphics Inc. 1998.
11. Mine, M., ISAAC: A Meta-CAD System for Virtual Environments. *Computer-Aided Design*, 29(8), 97.
12. Mine, M., Brooks, F., Sequin, C. Moving Objects in Space: Exploiting proprioception in virtual-environment interaction. *SIGGRAPH 1997*, pp. 19-26.
13. Pierce, J., Forsberg, A., Conway, M., Hong, S., Zeleznik, R. *et al.*, Image plane interaction techniques in 3D immersive environments. *Proceedings of ACM Symp. on Interactive 3D Graphics*. 1997. pp. 39-43.
14. Poupyrev, I., Weghorst, S., Billinghurst, M., Ichikawa, T., Egocentric object manipulation in virtual environments: empirical evaluation of interaction techniques. *Computer Graphics Forum*, 17(3), 1998, 41-52.
15. Poupyrev, I., Weghorst, S., Fels, S. Non-isomorphic 3D rotational techniques. *ACM CHI'2000*, pp. 546-547.
16. Shaw, C., Green, M., THRED: A Two-Handed Design System, *Multimedia Systems Journal*, 5(2), 1997.
17. Shoemake, K., ARCBALL: A user interface for specifying three-dimensional orientation using a mouse, *Graphics Interface*, 1992, pp. 151-156.
18. SmartScene promotional material, Multigen (San Jose, CA), 1999.