Semantic Constraints for Scene Manipulation

Michael Gösele, Univ. of Ulm, Germany
Wolfgang Stuerzlinger*, York University, Toronto

Abstract

The creation of object models for computer graphics applications, such as interior design or the generation of animations is a labour-intensive process. Today's computer aided design (CAD) programs address the problem of creating geometric object models quite well. But almost all users find common tasks, such as quickly furnishing a room, hard to accomplish. One of the basic reasons is that manipulation of objects often does not yield the expected results.

This paper presents a new system that exploits knowledge about natural behavior of objects to provide simple and intuitive interaction techniques for object manipulation. Semantic constraints are introduced, which encapsulate such 'common knowledge' about objects. Furthermore, we present a new way to automatically infer a scene hierarchy by dynamically grouping objects according to their constraint relationships.

Keywords: Virtual Environments, User Interface for Modeling, Constraints

1 Introduction and Previous Work

Today's computer-aided design (CAD) programs address the problem of creating single geometric object models quite well. But CAD systems are not well suited for the quick creation of scenes populated with thousands of objects. One simple, illustrative task is the modeling of an attractive living room based on a library of pre-defined objects. While one can position the main objects like the table and couch in a reasonable short time it is much more work to generate a nice looking, detailed model with many common accessories.

Direct manipulation of objects with 6 degree-of-freedom devices is a very intuitive user interface. But it is hard to position objects precisely and user fatigue is an issue. For a discussion of the general problems see [HPGK94]. Due to the limitations of three-dimensional (3D) displays and 6 degree-of-freedom input devices, a great deal of research has been directed towards software-based techniques for manipulating 3D objects with standard, low-cost 2D input and output devices. One approach adds different 'widgets' to the displayed objects. To manipulate the object the user selects the appropriate widgets. But even the latest work [SHR+92] concedes that 3D widget sets do not approach the utility of their 2D counterpart.

Another approach that attacks the problem from a different angle uses constraints, which specify the relative position and orientation of objects. One simple example is 'on-plane'. This constraint limits translations to motions parallel to the plane and allows only rotation around the orthogonal axis. Many constraint systems force the user to specify all constraints beforehand and utilize a constraint solver to find a configuration that satisfies them. Examples of recent work can be found in [Bar94, Bor91, HHLM92]. One major problem is that constraint systems are hard to solve at interactive rates, especially for 3D scenes with many objects.

Bier presented a user interface for specifying constraints between objects [Bie86] and introduced the Snap-Dragging concept [Bie90] for easier selection of object features. Both systems use constraints only for the initial placement of objects and do not maintain them. Based on this work Gleicher [Gle93] introduced a system that incrementally maintains the constraints during manipulation. Houde [Hou92] utilized simple on-a-surface constraints for object manipulation in a 3D world and defined a resting plane or gravity direction for each object. Bukowski and Séquin [BS95] enhanced the user interface for this approach further and included the ability to stack objects. Common to all above approaches is that they are based only on simple geometric constraints. They utilize no or little knowledge about where an object is placed 'naturally'.

2 System Overview

The laws of physics determine the behavior of objects in real life. Most important for a static scene are gravity, friction, and the fact that objects do not penetrate each other.

However, in addition to the physical properties people expect objects in a natural environment to behave according to their experience. They know that a chair will be standing on the floor and a painting will be hung on a wall. Hanging the chair on the wall and placing the painting on the floor is against common

*http://www.cs.yorku.ca/~wolfgang

sense. Bukowski and Séquin defined a direction of gravity for each object [BS95]. But often more information about an object is available that is useful for positioning. A chair and a cup are both placed on horizontal surfaces. But a chair is usually standing on the floor whereas a cup is standing on top of a table or a shelf and not on the floor.

Based on this observation our system uses *surface constraints* to describe possible relations between objects. Our system combines several approaches: It enhances the object association concept [BS95] by adding declarative modeling style information. With this information we can create bindings between objects that reflect the natural behavior of the objects. Objects can be connected together, which makes them behave similar to Bier's snap dragging approach [Bie90]. Furthermore, these connections lead to a dynamic grouping technique, which is based on natural relations between objects. We are using *collision detection/avoidance* to prevent penetration of objects.

Both mechanisms (constraints with dynamic grouping and collision detection/avoidance) together cover a large part of the natural behavior of objects. This lets users interact naturally and *quickly* with objects in the virtual environment.

3 Constraint Mechanisms

3.1 Surface Constraints

A user of our system builds scenes based on a predefined library of objects. For each object two sets of areas are defined in this library: *offer areas* $A = \{a_1, a_2, \dots\}$ and *binding areas* $B = \{b_1, b_2, \dots\}$.

Offer areas mark places on the object surface where other objects can connect to the object. Typical offer areas are the top of a table, the top of a shelf, or the surface of a wall. Binding areas are their counterpart and mark areas on the object surface where it can connect to other objects. Typical binding areas are the bottom of a bottle, the sides of a die, or the standing area of a chair. Figure 1 shows examples for offer and binding areas.

Each offer area $a = (p, l)$ and each binding area $b = (p, l)$ consists of an oriented planar polygon p and a label l. p defines the geometry and orientation of the area. An orientation vector is associated with each offer and binding area. Most of the behavior of an object is encoded in the labels l. Each possible value for l stands for a class of surfaces. Examples for labels are *on-Workspace* or *on-Floor*. Intuitively an offer area and a binding area fit together if they both have the same label.

However some values for l are special cases of other values: a workspace can be (mis-)used to store things. Therefore we introduced the special case relation/hierarchy $\leq_L$ to avoid having multiple labels per area. An offer area a and a binding area b fit together if and only if $l(b) \leq_L l(a)$. Figure 2 shows as an example the built-in hierarchy $\leq_L$ of our system.

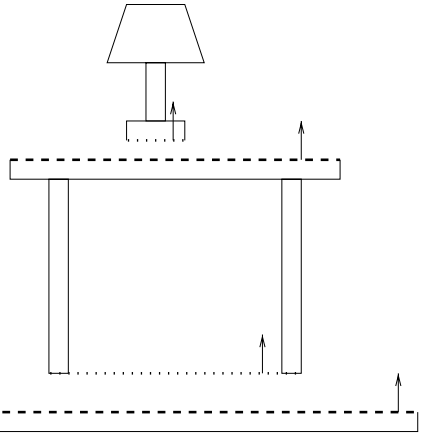

Figure 1: Table, lamp and floor with offer areas (dashed) and binding areas (dotted). Orientation vectors are perpendicular to the offer or binding areas and define their binding direction.

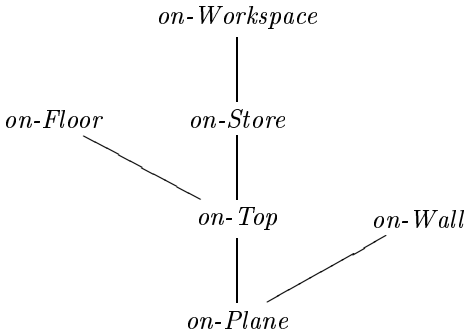

Figure 2: Partial order diagram $\leq_L$ of surface constraints.

Note that the definition of offer and binding areas depends on the (subjective) opinion of a user about the natural behavior of an object and not on the context in which an object is used. It is therefore possible to create a database of objects with corresponding definitions that can be used in many different application areas.

3.1.1 Satisfying a Surface Constraint

The process of creating a connection between an offer area and a binding area is called *satisfying* a constraint. A constraint between a binding area b and an offer area a can only be satisfied if their labels are partially ordered with $l(b) \leq_L l(a)$ and if they are geometrically aligned. A binding area b is geometrically aligned to an offer area a if they are coplanar, if b is totally included in a, and if their orientation vectors coincide.

The semantic criterion (the labels are partially ordered) is verified only for newly satisfied constraints. The geometric alignment criterion restricts the way objects can be moved relative to each other without breaking the corresponding surface constraints. It is tested whenever an objects in the scene is moved.

3.2 Collision Detection/Avoidance

In reality solid objects cannot interpenetrate each other. Our systems uses collision detection to ensure that no two objects occupy the same space. Due to numerical inaccuracies collision detection sometimes reports touching objects as intersecting. As constraints rely on touching surfaces we handle this case as follows:

Whenever two objects are bound together by a satisfied surface constraint then two or more of their surfaces are touching. We explicitly disable collision detection between two objects bound by a constraint to avoid problems with numerical accuracy. The downside of this is that breaking the constraint without moving any of the objects can lead to collisions between the two objects. This situation is handled as an exceptional case in our implementation by requiring that a surface constraint with a collision cannot be completely broken without moving at least one of the two objects until no collision occurs.

3.3 Consistency of Scenes

We can now define a predicate $consistent(s)$ that is true for all valid scenes. A scene is consistent if all satisfied constraints are valid and no two objects are colliding that are not connected by a satisfied constraint:

$$
\begin{aligned}
&consistent(s) \Leftrightarrow \\
&\quad (\forall a, b \in s : satisfied(a, b) \rightarrow \\
&\qquad (aligned(a, b) \wedge l(b) \leq_L l(a))) \wedge \\
&\quad (\forall o, o' \in s : collision(o, o') \rightarrow \\
&\qquad (\exists a \in A(o), b \in B(o') : satisfied(a, b) \vee \\
&\qquad \exists a \in A(o'), b \in B(o) : satisfied(a, b)))
\end{aligned}
$$

Our goal is to keep the scene consistent at all times.

The system starts with an empty scene. According to the above definition an empty scene is always consistent, because there are no objects in the scene. If a scene is loaded from file, it is checked for consistency during loading. Therefore, it is sufficient to show that all our operations transform a consistent scene into a consistent scene.

When an object is added to the scene it has to be placed at a position where it does not collide with any other object in the scene. Removing an object from a scene is a trivial task: satisfied surface constraints can be broken up and collisions can be removed from the scene. Both operations cannot render a consistent scene inconsistent.

It remains to be shown that moving an object and breaking or satisfying a surface constraint in our system maintains consistency. This is shown in section 5 after the concept of dynamic grouping is introduced in the following section.

4 Dynamic Grouping

Each satisfied surface constraint implies also a hierarchy onto the objects involved. The object that belongs to the binding area is grouped to the offering object. If the offering object is moved the bound object should be moved, too. But the bound object can move independently of the offer object. Consider the example of a cup standing on a table: If the table is moved the cup will move along, too. Note that it is still possible to move the cup without moving the table.

The satisfied constraints within a scene form a directed graph. This graph can be interpreted as a hierarchical grouping graph. Figures 3 and 4 show an example of a scene and the corresponding grouping graph. The chair is floating in the air and not yet bound by a satisfied surface constraint. It is therefore a singular node in the grouping graph.

Figure 3: Picture of a scene. Figure 4 shows the corresponding grouping graph.

Note that we use existing information from the objects and their constraints to build this graph. We do not have to infer the grouping graph from the position of the objects in the scene. This is a very robust and reliable way of automatically deriving a scene hierarchy as it does not rely on geometric computations.

Whenever the user moves an object and a constraint is satisfied or broken the method re-structures the internal representation to maintain the correct hierarchical grouping. Details of this are described in section 6.1.

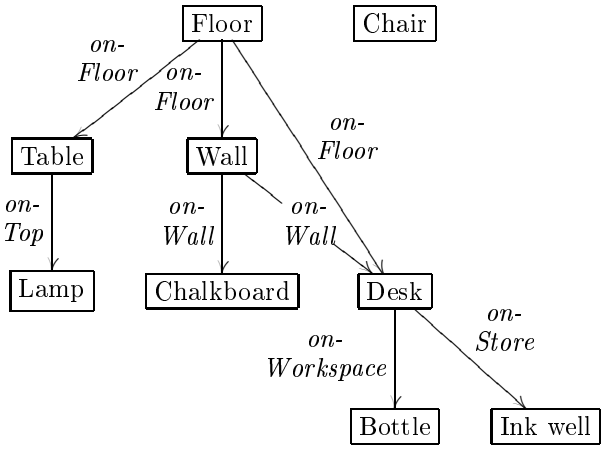

Figure 4: Grouping graph for the scene in figure 3. The label of satisfied surface constraints is given in italics.

4.1 Special Cases

The grouping graph has an arbitrary structure. There are two special cases when a circle exists in the graph.

A *directed* circle in the graph implies that the objects in the circle are not hierarchically arranged but are completely grouped. This can happen for example when four walls are grouped together to form a room. If one of the objects is moved all other objects have to be moved, too. At present our implementation does not allow directed circles due to the way the hierarchical grouping mechanism is implemented (for details see section 6.1).

If the grouping graph contains an *undirected* circle there may be objects with a common successor for which no order is defined. Figure 6 shows an example where no ordering between wall and table exists.

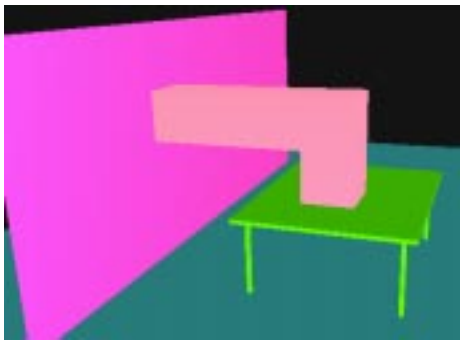

Figure 5: Picture of a scene. Figure 6 shows the corresponding grouping graph.

As no ordering information exists it is not clear whether or not the wall moves when the table is moved. Both alternatives are equally viable and there is no one correct answer as different persons will expect a different solutions.

In our implementation we chose to group only an object's successors to the object. This is consistent with the behavior of the system in other cases. It is a viable solution and leads only to a small work overhead for the user in cases where it is the wrong choice.

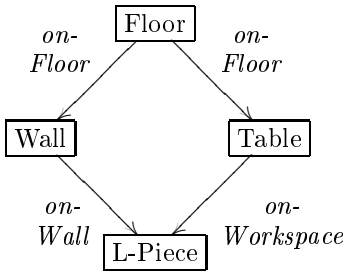

Figure 6: Grouping graph with undirected circle for the scene in figure 5. There is no ordering between table and wall.

5 Algorithms

5.1 Constraint Satisfaction Algorithm

The formal conditions for satisfying a surface constraint were already described in the previous sections: They deal with the position and shape of the offer and binding areas, their labels, collisions between objects and the resulting structure of the grouping graph. Based on these conditions a declarative modeling system can try to find a configuration of objects where all possible surface constraints are satisfied. But due to the potentially large number of combinations this is a very time-consuming option. Furthermore, there are often many possible solutions and not all of them match the user's expectations. For an overview over research on constraint satisfaction see e.g. [Doh95].

An interactive system allows the user to manipulate objects directly and provides much better feedback to the user. It is very important that the user can predict and understand the behavior of the objects - in other words the objects should behave according to the user's experience. Furthermore, objects should only be moved to satisfy a surface constraint when they are already close to the desired location and orientation.

As discussed above a constraint is satisfied if offer and binding areas are coplanar and share the same orientation and if the binding area is included in the offer area. In general an object has to be rotated and translated to satisfy these requirements.

First our search algorithm rotates the binding area object so that the orientation vectors of offer and binding area coincide. The rotation axis is chosen to be parallel to the offer area. This avoids unnatural rotations of the binding object during the alignment

procedure.

The binding area object is called close, if the perpendicular projection of the binding polygon on the offer polygon is completely included in the offer polygon. To keep objects from moving through the whole scene a maximal distance between offer and binding area is defined in the system.

Further tests ensure that moving the binding object maintains the alignment of other satisfied constraints in the scene and that the object does not collide with other objects at the new position.

The search is performed by algorithm 1. The function *checkMove*(o, t, r) (see section 5.2) checks whether a given translation t and rotation r can be applied to a scene without making it inconsistent. The function *searchCloseBindings*(b, s) returns all spatially close offer areas a of objects o' that can satisfy the given binding area and the necessary translation t and rotation r. The algorithm guarantees that the scene remains consistent whenever a surface constraint is newly satisfied.

Algorithm 1 satisfying binding area b of object o in scene s

satisfyConstraint(b, o, s)

1: *offers* $\Leftarrow$ *searchCloseBindings*(b, s)

2: **while** $|$*offers*$| > 0$ **do**
3: choose $x = (o', a, t, r) \in$ *offers*
4: *offers* $\Leftarrow$ *offers* $\setminus \{x\}$
5: disable collision test between o and o'

6: **if** *checkMove*(o, t, r) does not change (t, r) **then**
7: *satisfied*$(a, b) \Leftarrow$ true
8: **if** grouping graph contains no directed circle **then**
9: *offers* $\Leftarrow \emptyset$
10: **for all** $o'' \in O(s)$ **do**
11: **if** *ancestor*(o, o'') **then**
12: apply (t, r) to o''
13: **else**
14: *satisfied*$(a, b) \Leftarrow$ false
15: enable collision test between o and o'
16: **else**
17: enable collision test between o and o'

5.2 Restriction on the movement of objects by satisfied surface constraints

Before applying any transformation to an object in a scene algorithm 2 is used to check whether this can be done without introducing an inconsistency. Depending on the context a transformation onto an object can be either modified or completely rejected. The system modifies the current transformation whenever the user is moving a constrained object so that the constraint is maintained. Whenever the system tries to satisfy a new constraint it will reject the transformation if no solution can be found. Note that it is possible that an object cannot be moved anymore if it is bound by several constraints simultaneously.

The surface constraints influence the way an object o can be moved based on the following restrictions: All objects that are grouped to o are moved together with o. Thus satisfied surface constraints between two objects grouped together cannot be corrupted. Special precautions are necessary if the binding object of a satisfied surface constraint is moved without the offer object being moved, too.

Therefore algorithm 2 first collects all orientation vectors of satisfied surface constraints whose binding objects are grouped to o and whose offer objects are not grouped to o. The object o can only be translated perpendicular to this set of orientation vectors and only rotated along an axis parallel to the set. t and r are modified accordingly. Furthermore, the system has to test if the bound polygon is still included in the offer polygon and if no collisions occur. Note that in the description $t \Leftarrow \vec{0}$ and $r \Leftarrow \vec{0}$ both describe an identical transformation.

Algorithm 2 check whether translation t and rotation r can be applied to object o in a scene without making it inconsistent

checkMove(o, t, r)

1: *group* $= \{o$ and all objects grouped to $o\}$
2: *ovs* $= \{$all orientation vectors of satisfied surface constraints *satisfied*(a, b) with $b \in$ *group* and $a \notin$ *group*$\}$
3: remove duplicate directions (parallel or antiparallel) from *ovs*

4: **if** $|$*ovs*$| = 1$ **then**
5: set t perpendicular to *ovs*[0], axis$(r) \Leftarrow$ *ovs*[0]
6: **else if** $|$*ovs*$| = 2$ **then**
7: set t perpendicular to *ovs*[0] and *ovs*[1], $r \Leftarrow \vec{0}$
8: **else if** $|$*ovs*$| \geq 3$ **then**
9: $t \Leftarrow \vec{0}$, $r \Leftarrow \vec{0}$

10: **if** $t \neq \vec{0} \vee r \neq \vec{0}$ **then**
11: **if** t and r do not preserve polygon inclusion for satisfied surface constraints **then**
12: $t \Leftarrow \vec{0}$, $r \Leftarrow \vec{0}$
13: **if** applying t and r to all objects in *group* leads to a collision **then**
14: $t \Leftarrow \vec{0}$, $r \Leftarrow \vec{0}$

15: **return** (t, r)

5.3 Breaking a surface constraint

Breaking a satisfied surface constraint requires two steps: the data structures describing the satisfied constraint have to be updated and the collision test between the two objects has to be re-enabled. However, due to numerical inaccuracies it is possible that breaking the constraint leads to a collision.

If a constraint is broken we add the corresponding object pair to an exception list. All collisions between pairs on this list are ignored to keep the scene consistent. As soon as an object pair is no longer colliding it is removed from the list and the constraint is removed.

6 Implementation Issues

The IConS system is based on SGI's OpenGL Optimizer 1.1 (see [Eck97]) and implemented in C++. Objects are represented as polygonal models. IConS uses a simple two-dimensional interface. The main interaction device is the mouse, keyboard commands are used only for switching modes and organizational tasks (such as deleting an object). Currently three modes exist: constrained and un-constrained object movement and viewer navigation.

6.1 Implementation of Dynamic Grouping

The grouping graph as introduced in section 4 in which all objects have one or no satisfied surface constraints is a tree (or a set of trees). In Optimizer 1.1 we can represent the scene in a scene graph that has the same structure as the grouping graph. As all transformations of one node apply also to all of its children we can maintain dynamic grouping with no additional cost.

Whenever a surface constraint is newly satisfied we restructure the scene graph to make the binding object a child of the offer object (see figure 7). Each transformation of the offer object is then automatically applied to the binding object. Conversely, whenever a surface constraint is broken we attach the previously bound object to the root node of the tree.

If an object o is bound by more than one constraint it will also appear at several places in the *grouping* graph. If all offer objects are ordered in the grouping graph the object o is simply inserted in the scene graph as a child of the offer object lowest in this order But if the offer objects are not ordered the scene graph has to be restructured before one of the offer objects is moved to ensure that o is one of its successors.

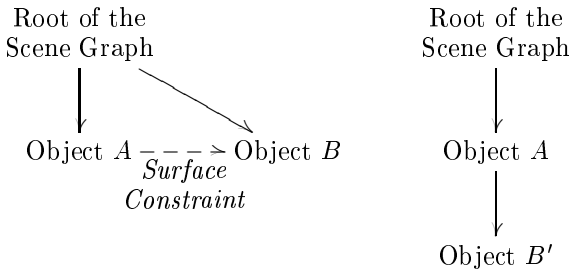

Figure 7: Changes in the scene graph when a surface constraint is satisfied. B becomes a successor of A. The change from B to B' symbolizes the changed transformationof Object B.

7 Results

In the presented system the interaction speed depends mainly on the following three factors: First, Each satisfied constraint between the moved object and the rest of the scene has to be checked for validity. This is quite fast because an object or a group of objects is usually bound by less than three constraints (otherwise it would be immovable). Moreover, for each non-satisfied binding area of the moved object all offer areas with matching labels within close proximity have to be checked. Lastly, a collision check for the moved object is necessary.

These factors depend mainly on the local complexity of the scene. Consequently, our approach scales well with scene size.

A simple, informal user test was performed using Gomoll's methodology [Gom90]. Six subjects were asked to perform simple tasks with our system. We observed their performance and collected their comments about the program. No timing tests were performed. The test showed that even inexperienced users could quickly construct a virtual environment with the IConS system. One user even remarked that objects behaved quite naturally.

8 Conclusion and Future Work

We presented the IConS modelling system, which uses semantic constraints to facilitate user interaction with predefined objects in a scene. The constraints encapsulate (part of) the natural behaviour of objects, which ensures that objects are placed according to the user's expectations. The current prototype uses a traditional 2D mouse and screen interface.

During manipulation the semantic constraints cull most potential alternatives from consideration. Only a small number of geometric constraint checks are performed, which ensures the system's interactivity.

In the future we want to extend the constraint system to encapsulate object behavior even more accu-

rately and to support complete grouping of objects in the presence of cycles. Furthermore, we need to ensure that the system can deal efficiently with large scenes with thousands of objects. And finally, we want to test the semantic constraint concept with a different user interfaces such as an immersive VR system or even a system with haptic feedback.

9 Acknowledgments

We would like to thank the Baden-Württemberg Program, the FWF, the computing centre and the Dept. of Programming Methodology at the Univ. of Ulm, and especially all people at the Dept. of Computer Science at the Univ. of North Carolina at Chapel Hill for their support.

References

[Bar94] D. Baraff. Fast Contact Force Computation for Non-penetrating Rigid Bodies. In *Computer Graphics (SIGGRAPH 1994)*, pages 311–318, 1994.

[Bie86] Eric A. Bier. Skitters and Jacks: Interactive 3D positioning tools. In *Proc. 1986 Workshop on Interactive 3D Graphics*, pages 183–196, 1986.

[Bie90] Eric A. Bier. Snap-Dragging in Three Dimensions. In *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, pages 193–204, 1990.

[Bor91] A. Borning. Constraint Hierarchies and Their Applications. In *IEEE CompCon Spring 1991, Digest of Papers*, pages 376–387. IEEE Computer Society Press, Los Alamos, 1991.

[BS95] Richard W. Bukowski and Carlo H. Séquin. Object Associations: A Simple and Practical Approach to Virtual 3D Manipulation. In *Computer Graphics (1995 Symposium on Interactive 3D Graphics)*, pages 131–138, 1995.

[Doh95] M. Dohmen. A Survey of Constraint Satisfaction Techniques for Geometric Modeling. *Computers & Graphics*, 19(6):831–845, 1995.

[Eck97] George Eckel. *Cosmo 3D Programmer's Guide*. Silicon Graphics, Inc., 1997.

[Gle93] M. Gleicher. A Graphics Toolkit Based on Differential Constraints. In *ACM UIST '93*, pages 109–120, 1993.

[Gom90] Kathleen Gomoll. Some Techniques for Observing Users. In B. Laurel, editor, *The Art of Human-Computer Interface Design*, pages 85–90. Addison-Wesley, 1990.

[HHLM92] R. Helm, T. Huynh, C. Lassez, and K. Marriott. Linear Constraint Technology for Interactive Graphic Systems. In *Graphics Interface '92*, pages 301–309, 1992.

[Hou92] Stephanie Houde. Iterative Design of an Interface for Easy 3-D Direct Manipulation. In *Proceedings of the ACM Conference on Human Factors in Operating Systems - CHI '92*, pages 135–141, 1992.

[HPGK94] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. Design Hints for Spatial Input, Course Notes – Developing Advanced VR Applications. In *SIGGRAPH 1994*, 1994.

[SHR+92] S. Snibbe, K. Herndon, D. Robbins, D. Conner, and A. van Dam. Using Deformations to Explore 3D Widget Design. In *Computer Graphics (SIGGRAPH 1992)*, pages 351–352, 1992.