Integrating Constraints and Direct Manipulation

Michael Gleicher*
School of Computer Science
Carnegie Mellon University

ABSTRACT

In this paper, we present techniques for integrating constraint and direct manipulation approaches to geometric modeling. Direct manipulation positioning techniques are augmented to provide the option of making the relationships they establish persistent. *Differential constraint* techniques are used to maintain these relationships during subsequent editing. Issues in displaying and editing constraints are also addressed. By integrating constraints with direct manipulation, it is possible to build systems that provide the power of explicit representation of geometric relationships and the properties which make direct manipulation so attractive.

INTRODUCTION

Geometric relationships between parts are an important element in geometric models. From the earliest days of interactive systems[13], the benefits of using constraints to explicitly represent these relationships have been known. Although many have discussed the value of constraints, constraint-based approaches have not been successful in practical systems. Their success has been hindered by a large number of difficult issues.

In contrast to the failure of constraints, direct manipulation systems have been successful for geometric modeling tasks. Users control the geometry of objects by interactively grabbing and pulling them, with continuous update providing feedback. Such systems employ snapping techniques, such as grids, to aid in establishing relationships, but these relationships are immediately forgotten. They are neither explicitly represented nor automatically preserved. It is the user's job to maintain them during subsequent editing.

In this paper we combine the two approaches: snapping techniques establish relationships and constraint techniques maintain them during subsequent dragging. Our integrated approach distinguishes the problem of establishing relationships from that of maintaining them during subsequent editing. This separation allows us to skirt several difficult issues in constraint-based systems. Integration with direct manipulation addresses issues in solving, specifying, debugging, displaying and editing constraints.

The *Briar* drawing program demonstrates our approach. When direct manipulation snapping establishes a new relationship, *augmented snapping* provides the user with the option of transforming it into a persistent constraint. *Differential constraint* techniques can maintain these during dragging. Direct manipulation techniques also address editing constraints.

ESTABLISHING RELATIONSHIPS IN DRAWINGS

Previous constraint-based systems have operated in what we call a "specify-then-solve" approach to constraint usage. In such systems, the user describes the model by declaring relationships which must hold true and the system configures the model to meet these requirements. This approach allows a user to specify the important aspects of a design and have the system resolve the details. Because the system explicitly represents the relationships, it can insure that these constraints continue to hold during subsequent editing.

There are problems in using the specify-then-solve approach. One is "solving" the constraints – finding a new configuration of the geometric model which meets the set of requirements. This is difficult because, in general, systems of non-linear algebraic equations must be solved from arbitrary starting points. While this problem is intractable[11], systems can usually operate by limiting the class of constraints which can be handled (as done by [6, 14]) or using temperamental numerical techniques (as done by [9, 12]). If no configuration is found that satisfies the constraints, it can be difficult to determine whether none exists or if the solver was just unable to find one. If no solution exists, the conflicts must be diagnosed and debugged. If the solver does find a new configuration, it must help the user understand how and why it jumped to the new state.

These three challenges, solving constraint-satisfaction problems from arbitrary starting points, presenting state jumps to users, and coping with conflicts, must be addressed to build a specify-then-solve system. However, these issues only arise when the constraint mechanism is used to reconfigure the model to establish new relationships. To skirt these difficult issues, we separate the task of maintaining existing relationships from that of initially satisfying them.

Our systems use direct manipulation to establish relation-

* School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890. gleicher@cs.cmu.edu

Appears in:
Proceedings 1992 Symposium on Interactive 3D Graphics. pages 171–174

ships and use constraint techniques for maintaining them during subsequent editing. Constraints are only generated for relationships which exist in the drawing. They start out satisfied so there is never a need to jump from an arbitrary state to a consistent one. There are no constraint-satisfaction problems to solve or state jumps to explain. There is no concern about conflicting or unsatisfiable constraints, since there exists at least one configuration which meets the constraints.

MAINTAINING RELATIONSHIPS

When initial solutions are provided, the task of constraint techniques changes; instead of establishing the relationships, constraint-based techniques are used to maintain them. Rather than jumping from an inconsistent state to one where the constraints are met, constraint techniques permit users to drag models and have the constraints enforced as the drawing follows with continuous motion. We call this facility to drag constrained models *Differential Constraints.*

Unlike solving non-linear algebraic equations, good techniques exist for maintaining constraints during dragging. We use techniques which treat the motion of the model as a differential equation and provide methods for maintaining sets of non-linear constraints by solving systems of sparse linear equations[4]. Alternatively, solving can be accomplished using a standard constraint-solving approach: the model is repeatedly perturbed slightly, then re-solved.

Fast computers and good algorithms allow update rates which give the appearance of continuous motion. This rapid feedback is essential. Although the trajectory the model follows is not part of the resulting drawing, this animation makes it possible for users to employ their perceptual skills to connect states of the drawing with many things changing between them[2].

Differential constraints provide a natural way to incorporate constraints into a conventional drawing system. Objects are dragged the same way, except that relationships can be maintained among them. This allows the drawing process to be incremental: each new relationship added to a drawing does not disturb previously established ones.

The ability to directly manipulate constrained models helps address many of the issues in constraint-based systems. It provides an easy way for the user to explore underconstrained spaces, permitting them to experiment with models to understand how they work, or why they do not. The existence of a direct manipulation facility means that all parts of the model do not need to be specified by constraints. If it is difficult to devise a way to describe an aspect of a drawing with constraints, direct manipulation can be used instead.

Constraints can aid in the direct manipulation process by providing the user with "extra hands" to hold things in place. Providing the user with "lightweight constraints" which are easy to place temporarily to aid in manipulation is a useful feature in modeling systems.

SPECIFYING CONSTRAINED MODELS

Rather than using constraints, our approach, like most direct manipulation systems, uses *gravity* to help users establish relationships in models. The drawing cursor follows the motion of the pointing device, but snaps to locations which will establish relationships in the model when it is close to them. This idea of gravity has existed for a long time, having been demonstrated as early as Sketchpad[13]. The most common variant of gravity is the uniform grid. A more interesting technique is Snap-Dragging[1] which extends gravity by expanding the set of snapping targets to include intersections and construction lines.

Gravity is successful at helping a user establish relationships in models, but previous systems promptly forget these relationships once the positioning operation is complete. To employ constraint maintenance, these newly established relationships must be made into persistent constraints. The user could be required to explicitly identify the constraints, but this creates excess work: each relationship is specified twice, once to establish it and once to identify it as a constraint. Previous systems have attempted to infer constraints after drawing operations by looking at the resulting drawing[10], or at a trace of user actions[7]. Because this information typically does not specify the relationships unambiguously, these systems relied on heuristics or asked the user to resolve the ambiguity[8]. Our approach augments positioning methods so they, in addition to location, unambiguously specify the relationships which are being established.

Our *augmented snapping* technique lets direct manipulation positioning specify constraints as well as location. It enhances the snapping operation so that it generates constraints. The basic idea is that cursor placement operations contain information about why an object was positioned where it was, and can, therefore, also provide a constraint specification. Suppose the user, while dragging an object, moves the pointer near another object so that the cursor and the point being dragged snap to the second object. Snapping has helped the user establish a relationship between the dragged point and the target object. We provide the user with the option of making this relationship persistent so it can be preserved during subsequent editing.

When a snapping operation occurs, the system acknowledges it by showing the newly established relationship to the user. The user has the opportunity to accept the new relationship, transforming it into a persistent constraint. To make the constraint creation process more transparent, the default can be to accept new constraints. [1]

Augmented snapping permits direct manipulation techniques to be integrated with constraints. Since snapping is used for all drawing operations, such as creating and moving objects, all of these operations can specify constraints. Constraint generation is opportunistic, as the user draws, constraints are

[1] Although we provide an "accident-prone" mode where acceptance is not the default, we find that it is seldom used.

created when relationships are established. Aside from the occasional rejection (or acceptance, if that is not the default) of a constraint, the interface should not require any additional effort from the user. Such an interface feels just as fast and clean as the non-augmented version.

As a constraint specification technique, augmented snapping offers other advantages. It does not require the user to learn new commands for each type of constraint since a uniform interface creates all constraints. Because it provides both constraints and an initial configuration that satisfies them, augmented snapping cannot create conflicting constraints.

Careful attention to the user interface is crucial to making augmented snapping work. Feedback must show the snapping operations to the user so it is clear what relationship is being established. When a new relationship is established, it must be displayed prominently enough that it is clear what constraint will be created, but not be obtrusive to hinder the drawing process.

Augmented snapping only generates constraints for relationships which are unambiguously specified by the user's actions. A snapping operation unambiguously specifies a relationship, but if multiple objects coincide, it can be ambiguous which to snap to. Feedback, which clearly shows which object is snapped to, and a cycling mechanism to choose between potential snapping targets resolves this problem. Pruning the set of objects that are snapped to (for example, avoiding snaps which would create a redundant constraint) avoids excess cycling.

Augmented snapping does not guess about the user's intentions. It relies on the construction process to obtain constraints. The user may construct a model in a manner which does not convey the desired constraints. To curtail this, it is important to design modeling operations which make it easy to convey what is intended, rather than just what is convenient to express. For example, making two objects be the same size should be no more work than making them both be the same fixed size.

VISUAL REPRESENTATIONS FOR CONSTRAINTS

Constrained drawings have more state that must be displayed to the user than non-constrained ones do. A system must convey to the user not only the geometry of the model, but also the constraints. The user must be able to edit this structural information as well as the geometry. Although textual languages for describing constraints, such as in [9, 14] are easy to edit, they are distinct from the drawing and can be difficult to connect to their corresponding places in the model. Visual representations[5, 12] superimpose symbols for constraints directly on the model. Unfortunately, devising clear visual representations is challenging and editing such representations is often difficult.

When differential constraints are used, the continuous motion and ability for users to experiment with models can convey much of the information about the constraints. We also use a visual representation for constraints.

The problem of editing constraints transcends visual representations. Before being able to delete or modify a constraint, the user must figure out which constraints to alter. We have developed methods for editing constraints which avoid this problem by having the users edit constraints by referring to the desired effects, not to the constraints themselves. Instead of pointing at constraints, users directly manipulate objects to show how they are to move. For example, constraint maintenance can be disabled so objects move freely. Constraints which are broken are clearly noted to the user. When maintenance is restarted, violated constraints are removed. A variant is a "rip" command which allows the user to pull part of an object free from its constraints.

Designing the semantics of the constraints properly can also reduce problems in the visual language. For example, when a group of points is connected together, an equivalence class is used rather than a large number of binary connection relations. This is also significant since it removes the need to remember which point is connected to which other point if some are to be disconnected.

DRAWING WITH CONSTRAINTS

To explore the integration of constraints and direct manipulation, we have built a drawing program called *Briar*[2][3]. A direct manipulation drawing technique called Snap-Dragging[1] is augmented to specify constraints. Differential constraint techniques are used to maintain these relationships as the user modifies the drawing. Augmented Snap-Dragging also serves as the basis for a visual representation for the constraints.

Snap-Dragging enhances the usefulness of gravity. The cursor snaps not only to the edges of objects, but also to interesting points in the scene such as intersections and vertices of objects. Relations other than contact are created in Snap-Dragging through *alignment objects:* objects that are not part of the drawing *per se,* but exist only to be snapped to. The original Snap-Dragging work includes several types of alignment objects, each corresponding to types of relationships which are useful in drawings. The usefulness of alignment objects is further enhanced by making them easy to place.

Snap-Dragging provides two operations for positioning points in two dimensions: snapping the cursor to a point, such as a vertex, and snapping the cursor to an object's edge or curve. These operations correspond directly to *Briar's* two basic constraints, "points-coincident" and "point-on-object" respectively. The two snapping operations combined with alignment objects allow a user to establish a wide variety of relationships. Similarly, the two basic constraints are combined with alignment objects to enforce a similarly large set of relationships. For example, a distance constraint can be expressed using a fixed size circle.

[2]It is called Briar because, like the plant it is named for, things stick together inside it.

Augmented Snap-Dragging also provides the basis for *Briar's* visual representation of constraints. Constraints are displayed just as they are specified: using the two basic elements along with alignment objects. Although *Briar* can handle a wide variety of relationships, users need not learn a large number of constraint creation commands or display symbols. *Briar* provides several methods for altering constraints by direct manipulation of objects, including disabling constraint maintenance and commands to "rip" parts of objects free of their constraints.

Briar's display employs many mechanisms to convey its state to the user. Objects light up when snapped to and the cursor changes shape to indicate the type of snapping operation. Newly established relationships are shown in distinctive colors which signify whether or not they will become constraints.

THREE DIMENSIONAL SYSTEMS

Extending a system like *Briar* to three dimensional modeling poses a new set of challenges. For modeling tasks, the set of possible spatial relationships between objects is much richer, and more complex, than in 2D. However, this richness and complexity is also a strong motivation for the development of constrained interaction techniques for 3D. Direct manipulation techniques to establish spatial relationships are not as developed as their two dimensional counterparts. A more pragmatic concern is that our reliance on feedback already causes *Briar* to use almost all available perceptual cues, such as texture, hue, brightness, size, and motion, leaving little for the increased visual demands of 3D.

Techniques such as augmented Snap-Dragging, differential constraints, and visual alteration of constraints make it possible to build systems which integrate constraints and direct manipulation. Such systems can combine the power of representing geometric relationships with the fluency and intuitive interfaces which have made direct manipulation so successful.

Acknowledgments

This research was funded in part by Apple Computer, a fellowship from the Schlumberger Foundation, and an equipment grant from Silicon Graphics. I would like to thank Andy Witkin for his assistance with this work. I would also like to thank everyone who has commented on this paper, especially Will Welch, David Pugh, and Bruce Horn, for helping make it more readable.

REFERENCES

[1] Eric Bier and Maureen Stone. Snap-dragging. *Computer Graphics*, 20(4):233–240, 1986. Proceedings SIGGRAPH '86.

[2] Jock Machinlay George Robertson and Stuart Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proceedings CHI '91*, pages 189–194, May 1991.

[3] Michael Gleicher. Briar - a constraint-based drawing program. In *CHI '92 Formal Video Program*, 1992. SIGGRAPH video review, in press.

[4] Michael Gleicher and Andrew Witkin. Differential manipulation. *Graphics Interface*, pages 61–67, June 1991.

[5] Mark Gross. Relational modeling: A basis for computer-assisted design. In Maclcolm McCullough, Williadm J. Mitchell, and Patrick Purcell, editors, *The Electronic Design Studio (Proc. CAAD Futrues '89)*, pages 123–146. MIT Press, 1989.

[6] Jiarong Li. Using constraints in interactive text and graphics editing. In P. A. Duce and P. Jancene, editors, *Eurographics*, 1988.

[7] D. L. Maulsby, K. A. Kittlinz, and I. H. Witten. Metamouse: Specifying graphical procedures by example. *Computer Graphics*, 23(3):127–136, July 1989. Proceedings SIGGRAPH '89.

[8] Brad A. Myers and William Buxton. Creating highly-interactive and graphical user interfaces by demonstration. *Computer Graphics*, 20(4):249–258, 1986. Proceedings SIGGRAPH '86.

[9] Greg Nelson. Juno, a constraint based graphics system. *Computer Graphics*, 19(3):235–243, 1985. Proceedings SIGGRAPH '85.

[10] Theo Pavlidis and Christopher Van Wyk. An automatic beautifier for drawings and illustrations. *Computer Graphics*, 19(3):225–234, 1985. Proceedings SIGGRAPH '85.

[11] William Press, Brian Flannery, Saul Teukolsky, and William Vetterling. *Numerical Recipes in C*, chapter 9.6, page 286. Cambridge University Press, Cambridge, England, 1986.

[12] Steven Sistare. Interaction techniques in constraint-based geometric modeling. In *Proceedings Graphics Interface '91*, pages 85–92, June 1991.

[13] Ivan Sutherland. *Sketchpad: A Man Machine Graphical Communication System*. PhD thesis, Massachusetts Institute of Technology, January 1963.

[14] Christopher J. Van Wyk. A high level language for specifying pictures. *ACM Transactions on Graphics*, 1(2):163–182, April 1982.