A Browsing Interface for Exploring Constraints in Visualization Rules

Shin Takahashi (shin@is.titech.ac.jp)
Department of Mathematical & Computing Sciences, Tokyo Institute of Technology
2-12-1 Oookayama, Meguro, Tokyo, JAPAN 152-8552

Abstract

We built a prototype tool for browsing constraint systems for the layout of graphical objects. It has two views: In one view, the tool visualizes a constraint system as a three-dimensional graph structure, which shows the overall structure of the constraint system. The viewer can change the layout to focus on the part of constraints. The other view shows the target diagram. It also animates the diagram to show degrees of freedom in the constraints for the diagram. The cartoon technique of deforming graphical objects is utilized to depict whether a graphical object has a degree of freedom or not.

1 Introduction

The motivation of this paper is to help the programmer to debug a set of *visual mapping rules* of TRIP systems[4, 5]. TRIP systems are visualization tools for relational structures such as trees and graphs, and these rules specify how source data should be visualized into a diagram. The programmer represents a diagram as graphical objects such as boxes and spheres and graphical relations such as horizontal and vertical. The TRIP systems solve graphical relations, arrange graphical objects according to the result, and show them to the user. The visual mapping rules are difficult to debug because of the constraints used for the layout of the graphical objects.

This paper describes two approaches to visualize the constraints used in the visual mapping rules. Their purpose is to help the programmer to grasp the overall structure of constraints in rules. The first approach is to visualize constraints as an undirected three-dimensional graph structure. Constraints and constrained objects are the nodes. Edges connect constraints and constrained objects. The programmer can explore the structure of the graph by changing their layout in several ways so that the structure of the focused sub-graph becomes apparent.

The second approach is to represent constraint systems using animation. By using animation, the system can show the dynamic aspects of constraint systems. In particular, the system shows degrees of freedom of objects with an animation. When an attribute of a graphical object is not constrained by any constraint, it has a degree of freedom, that is, the system cannot determine its value. In that case, the graphical object is animated so that various valid values of the attribute are shown to the user. Other graphical objects are animated together satisfying the constraints. On the other hand, An attribute that has no degrees of freedom, i.e., an attribute that has a determined value, is animated (jerked or twitched) as if it is trying to change its value but cannot. Cartoon animation techniques are used to represent this situation.

We have implemented a browsing tool that can visualize graphical objects and constraints in these two ways. This tool has two windows. One window displays a resulting picture, and the other shows the corresponding three-dimensional constraint graph. The user browses and compares the target picture and the constraint graph with the viewer. The user can freely rotate and zoom in/out of the visualized picture and the 3D graph with a mouse.

2 Browsing Constraint Graphs in 3D

Basic Representation There are two types of nodes in constraint graphs. One type consists of geometric primitive objects, such as boxes, spheres, and lines. These are represented as sphere nodes in the visualized constraint graph. The color of a node shows the type of geometric object. The other type of nodes consists of geometric constraints (graphical relations) such as the constraint that graphical objects should be arranged horizontal and the constraint that the interval between graphical objects should be a certain value. Geometric constraints are represented as box nodes in the constraint graph. As with geometric objects, the colors of these nodes represent the types of constraints. Figure 1[1] shows a constraint graph for the layout of the tree in Figure 2 — also shown at the lower right corner in Figure 1, which contains the following constraints[2]:

```
x-parallel(V) : 1 2 3 4 8 9
x-average(Y)  : 0 1 2 3 4 8 9
x-parallel(V) : 5 6 7
x-average(Y)  : 4 5 6 7
z-relative(R)   0 : 0 1
y-relative(R) -30 : 4 5
```

[1] This screen snapshot of the system is modified by emphasizing edges so that it can be seen well in black and white. The label on each box is added as a substitution for the color.

[2] Constraints for line connections are necessary for the layout, but they are omitted.

```
x-relative(R)  15 : 1 2 3 5 6 7 8 9
place (B) 100 100 100 : 0
```

Here, the number from 0 to 9 is an ID of each graphical object represented as a sphere. In the figure, a box node is connected to spheres, i.e., a constraint node is connected to object nodes that are constrained by the constraint node.

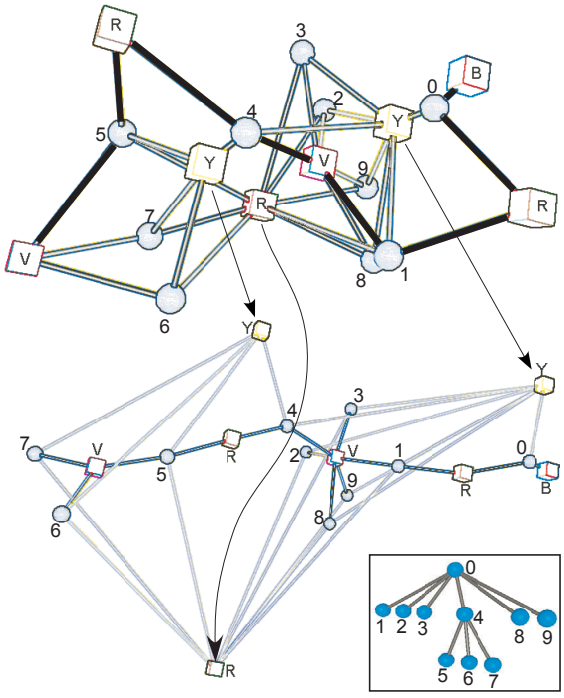

Figure 1: Normal and modified layout of a constraint graph.

Focusing on Constraints We provide two ways to simplify constraint graphs. One way is to lengthen the edges of the selected constraints, which makes them unfocused and simplifies the layout of the constraint graph. Stretching the edges of a constraint makes the layout of the graph as if it is removed from the graph, but the connections still remain as pale translucent lines. The graph at the bottom in Figure 1 shows a simplified constraint graph unfocusing a `relative` (the R box) and two `average` constraints (the Y boxes). We can see that the layout of this graph is governed by `relative` (R) and `parallel` (V) constraints. One `parallel` (V) constraint constrains three sphere object (blue sphere nodes), and another `parallel` (V) constraint constrains six sphere objects. Such structure is shown also in the graph at the top of Figure 1 (represented as thick black lines), but it is clearly shown in the graph at the bottom. By selecting other constraints to be unfocused, we can get different layout that focus on a different structure. The user can click these constraint nodes to highlight the constrained graphical objects in the diagram, which will help the users to find the correspondence between the diagram and the constraint graph. Another method of exploring 3D constraint graphs is to bind up the constraint nodes that constrain the same set of graphical objects. Such a set of constraints can be thought of as a compound constraint. Our tool provides a command to change the layout of the graph so that the grouped constraints are positioned at almost same place. They are shown to the user as if it is one constraint. This is a way of abstracting constraint graphs. We are planning to provide more ways to abstract the graphs, such as to classify and color the groups of constraints, or to abstract the hierarchical structures of constraint graphs.

Implementation issues The graph layout module of our system uses the three-dimensional version of Kamada's graph-drawing algorithm[2], which tries to make the geometric distance between each pair of vertices in the graph close to the logical graph-theoretic distance between them. By utilizing the features of this algorithm, the change of layout described in this section can be easily achieved. That is, by setting the default length of all edges from a node very long causes the difference of graph-theoretic distance unimportant, which makes the effect of the node to the layout very little. Binding up groups of constraints can be achieved by setting very short edges among a group of constraint nodes.

3 Animating DOF in Constraint Systems

Visualization of constraint graphs shows their structure directly. However, it is still difficult to understand the role of each constraint in the whole graph, because constraints and objects are represented abstractly. For example, a lack of constraints may be evident in the visualized constraint graph, but how it affects the result is difficult to guess. To represent the behavior of constraints more directly, we propose another method of visualizing constraint systems. This method animates the target picture itself, and shows degrees of freedom in a constraint system.

For example, if the x and y positions of a sphere are determined but the z position is not constrained, the sphere can move along the z-axis while satisfying all constraints. In this case, the system shows an animation that moves along the z-axis and comes back to the original position. The user knows immediately from the animation that the z position of the object is not constrained. Figure 2 shows screenshots of the animation when the system pulls a node. In Figure 2(b), a node is pulled downward. The node stretches and moves downward, which means that the node has a degree of freedom in this direction. The other nodes except the root node also move in the same direction, which means that there are constraints that keep their relative positions. The lines that connect them and their parent are also animated, because there are "connect" constraints that connect these nodes. On the other hand, in Figure 2(a), the system tries to move a node in another direction, but the node is only stretched and does not move. This indicates that the

node is constrained and does not have freedom to move in that direction.

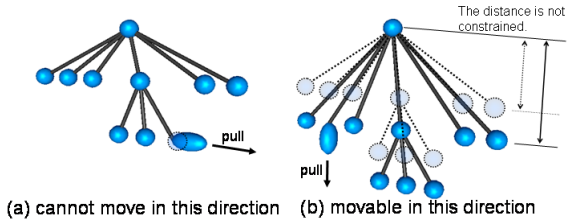

Figure 2: Pulling a node in a tree.

As shown in these figures, even when the object does not move, the object is stretched in the direction in which it is pulled. This shows effectively that the system is trying to pull the object, but that the object cannot move in this direction. If the system showed only the animation of objects that have a degree of freedom, the user might think that all objects have a degree of freedom to move.

3.1 Implementation of Freedom Animation

Visualizing degrees of freedom (DOF) is achieved in two steps: (1) detecting DOF in a constraint system; and (2) showing the detected DOF.

Detecting DOF in a Constraint System Detecting DOF in a constraint system means searching for the less constrained variables in a constraint system. Our system utilizes the constraint hierarchy mechanism of the HiRise constraint solver[1] to search for such variables. The system randomly or successively selects a variable (attribute) of a graphical object in a picture, and checks whether it has DOF or not. Currently, the system selects only the variables that represent the x-, y-, and z- coordinates of objects.

Showing DOF as an Animation According to the checked DOF of each variable, the system shows an animation to indicate the DOF of each variable. How an object is moved differs according to whether the variable has a DOF or not. When a variable has a DOF, the system generates an animation by gradually changing the value of the target variable to a slightly changed value, and then gradually changing it back to the former value. For example, Figure 2 shows a screenshot of an animation that shows pull-and-release of a node in a tree. During this animation, the system is solving the entire constraint system repeatedly. This is done by adding an *edit* constraint[1] that is stronger than the other constraints. Using an edit constraint, HiRise can efficiently solve the constraints repeatedly with the value of the target variable changing gradually.

Even if a variable does not have a degree of freedom, the system shows an animation indicating this. In Figure 2, the object is stretched in the direction of the pull, but the position of the object does not change. This animation implies that the system tries to pull the object, but the object does not move because it is constrained. This is a kind of cartoon technique used to distort characters in cartoon animations[3].

Besides animating DOF in a constraint system, the system allows the user to drag graphical objects directly with a mouse. The dragging of a graphical object is executed with satisfying constraints on it. The well-constrained objects cannot be dragged. The system repeatedly solves the constraint system during the user's dragging.

4 Related Work

In Thomas's work [6], the distortion effect is used when dragging objects in drawing editors, which makes users feel as if they are dragging "soft" objects. For example, if the vertex of an object is pinned at a point, the user cannot drag it freely but can pull the object to stretch or squash it. After releasing the mouse button, the object returns to its normal shape. Without such an effect, users cannot easily determine whether an object is constrained and therefore unable to move, or the system is not responding to the user's operation. Our system uses similar techniques, but is extended to handle constraint systems. In addition, our system animates a constraint system without user operations.

5 Concluding Remarks

We have described two approaches to visualize constraint systems in the visualization rules of TRIP systems. One is to draw a three-dimensional graph structure of the constraint system in the rules, and the other is to animate the target picture to show the degrees of freedom of graphical objects. We have prototyped these two approaches and applied them to the tree example. Although we have not yet evaluated these approaches, they both help to clarify the structure and behavior of constraint systems.

References

[1] Hiroshi Hosobe. A scalable linear constraint solver for user interface construction. *Proc. 6th Int'l Conf. on Principles and Practice of Constraint Programming (CP2000)*, Sep. 2000. Lecture Notes in Computer Science.

[2] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, April 1989.

[3] John Lasseter. Principles of traditional animation applied to 3D computer animation. *ACM Computer Graphics*, 21(4):35–44, July 1987.

[4] Shin Takahashi et al. A constraint-based approach for visualization and animation. *Constraints: An International Journal*, 3(1):61–86, 1998.

[5] Shin Takahashi et al. A framework for constructing animations via declarative mapping rules. In *Proceedings of the 1994 IEEE Symposium on Visual Languages*, volume 10, pages 314–322, 1994.

[6] Bruce H. Thomas and Paul Calder. Animating direct manipulation interfaces. In *UIST'95*, pages 3–12, 1995.