Efficient Manipulation of Object Groups in Virtual Environments

Wolfgang Stuerzlinger, Graham Smith
Dept. of Computer Science, York University, Toronto, Canada
http://www.cs.yorku.ca/~wolfgang

Abstract

In this paper, we describe simple techniques for object group manipulation, an important operation in user interaction with a Virtual Environment. All presented manipulation techniques exploit constraints to simplify user interaction. The techniques are based on how humans perceive groups and afford direct manipulation of such groups. Furthermore, we introduce two new intuitive ways to create a whole group of objects: drag-add and random drag-add. Finally, we present an evaluation of the presented techniques.

1. Introduction

There have been considerable advances in Virtual Reality (VR) in the last several years. Most application areas, such as walk-throughs (architectural, industrial, mechanical, etc.), require the creation and manipulation of three-dimensional (3D) scenes with a realistic complexity.

Although there are many modeling packages available, practically all focus on adding increasingly more features. Consequently, the user interface of such packages is very complex. Contrary to this trend, recently several 'home-designer' packages have simplified the process by allowing only limited manipulation of a 2D floor plan. A static 3D view can be displayed on demand. Unfortunately, the user can't usually move the objects in the 3D view, which would be the most intuitive solution.

Virtual Reality (desktop or immersive) seems to offer a solution to this dilemma. However, most VR systems offer only very rudimentary object manipulation possibilities that do not go beyond the operations afforded by the underlying primitive geometric operations. One of the few examples for an advanced VR system is the SmartScene system [19], which uses object 'behaviors' to simplify 3D scene interaction.

Construction in a 3D scene consists of two essential steps: First 3D objects must be created, and then these objects must be placed into the scene. To restrict the problem, we have chosen 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.

In our experience quick operations on groups of objects is one of the essential elements of faster scene construction. Currently grouping of objects in most 3D systems is done by explicitly selecting the objects that are to be in the group, then invoking a grouping command. Breaking up groups of objects proceeds in a similar manner. Although this behavior is flexible, our approach is superior because it uses knowledge about the real world to simplify and automate grouping operations. In other words we are trying to emulate how users perceive groups and base our direct manipulation techniques on this.

1.1. Previous Work

Most user interfaces for 2D and 3D applications are based on the use of an explicit grouping mechanism. I.e. if the user wants to manipulate a group of objects he/she must at some point select all objects within that group and invoke a group operation.

A few systems try to infer groups from the placement of objects in 2D. For example, the system presented in [12] tries to automatically find collections of objects that can subsequently be moved by the user. The Tivoli system [13] is an example of a system that introduces a set of simple group manipulation techniques for whiteboards. Very few systems have introduced similar ideas into 3D applications. One exception is the Sketch system [22], which automatically infers groups based on where objects are placed. This approach relies on a horizontal relationship between objects to decide when objects are to be placed in a group, and hence is restrictive.

The interactive grouping mechanisms presented here are based on the use of constraints to simplify the manipulation of objects in a scene. Several researchers have presented different forms of 2D constraint-based systems (see e.g. [2,4,6,11,15,21]). For an overview of constraint solvers see [8]. Several constraint-based systems have been presented for 3D modeling (see e.g. [1,3,5,9,10,14]). While these systems use the constraints to afford direct object manipulation, the manipulation of groups is not simplified similarly. E.g. while a connected

set of objects can be dragged as a whole, the user must explicitly break a constraint if he wants to split a group.

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). Although intuitive, their approach has a few drawbacks. Associations are not maintained after the current manipulation. Groups and other forms of cyclical constraints are not supported.

The MIVE system developed at York University follows the approach of the Object Associations system. First developed by T. Salzman [18] and later improved by G. Smith and others [20] it uses permanent constraints to simplify object manipulation. A brief overview over how the constraints are used for interaction in this system is given in section 2. While this system can deal with some situations that arise in practice, it is not general enough to handle a simple row of chairs.

1.2. Motivation

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. 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). Consequently, almost all objects have a maximum of three degrees of freedom in practice – e.g., all objects resting on a plane.

If fact, recent research argues that all manipulation techniques that work well for 3D applications are essentially 2D techniques [16]. This implies that a two-dimensional (2D) input device such as a mouse is very often sufficient to manipulate objects in a virtual environment. Indeed our previous work shows that a 2D device can indeed be used to intuitively and quickly manipulate a 3D environment [18].

Many constraint-based systems expose the details of the constraint mechanisms (variables, equations) directly or in graphical form to the end user. Most users cannot deal with such a level of detail, which may explain why constraint-based systems are not as ubiquitous as one might expect. The goal of our system is to shield the user from the technical details as far as possible. One of the fundamental design decisions was therefore not to use a full constraint solver as such systems can easily generate object configurations, that do not conform to the user's expectations. For this reason, most constraint-based systems cannot automatically support direct manipulation of object groups – the solver is always free to re-arrange objects also within the group. This can be fixed only with an explicit grouping mechanism that 'freezes' a group during manipulation. We are trying to create manipulation techniques that work without explicit grouping techniques, and consequently chose to implement a deterministic and somewhat restricted form of a constraint-based system.

Similarly, groups are usually defined to contain objects that belong in a sense together, either by proximity or by similarity or by common properties [17]. In our system, simply moving 'group-able' objects close to each other will create a group. The breaking of a group is handled similarly by pulling objects away from the group. For the rare situations where the offered group mechanism is not sufficiently powerful, we discuss how an explicit grouping mechanism can be integrated seamlessly into the system.

1.3. Contributions

The main contributions of this work are:

- A new and simple direct manipulation technique for groups of 3D objects (the main contribution).
- Two new simple and effective techniques for creation of object groups: Drag-add and Random drag-add.
- A framework for explicit and implicit grouping in general scene graphs (directed a-cyclic graphs).

From the user's point of view, our main contribution can be roughly described as a powerful extension to 3D snap dragging [3]. Our novel extension uses the direction of the mouse movement to afford the direct manipulation of object groups. However, the technical realization is very different and we have chosen not to describe our approach as an extension to 3D snap dragging.

2. Constraints for Manipulation of Hierarchical Groups

The material presented in this section has been published previously [18]. However, some of the details in the work presented in this paper rely heavily on this previous work. Consequently, we give an overview of the relevant parts and refer the reader to [18] for details.

The constraint relationships for the 3D scene are stored in a directed a-cyclic graph (DAG) 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, due to the implicit group defined by the hierarchical relationship.

The scene graph edges 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 clicking on the desired object and pulling away from (or pushing towards) the existing constraint to break (or attach) it. This allows for fast changes to the scene graph structure. Figure 2 shows the same scene as Figure 1 after the chair has been pulled away from the large table towards the smaller table.

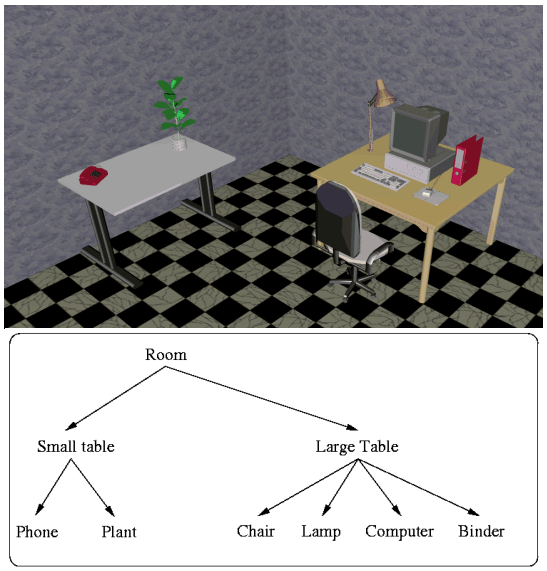


Figure 1: A Scene and its associated scene graph. Links describe constraint relations.

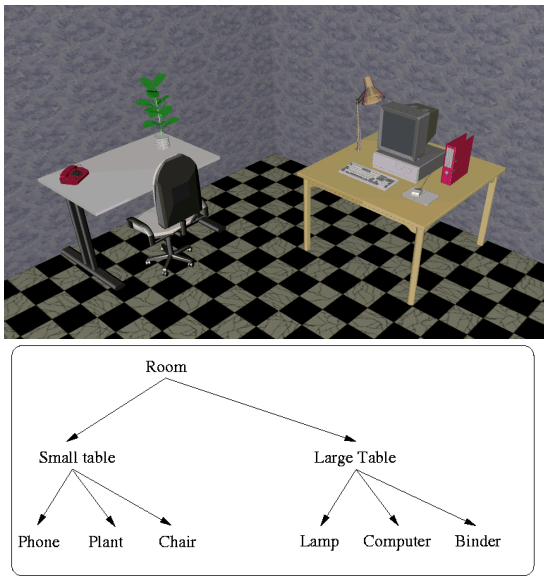


Figure 2: Scene after chair has been moved.

2.1. Constraint Satisfaction

In our constraint system, each object can have zero or more binding and offer *areas*. Binding areas define where an object can attach to another object. An example is the base of a cabinet that has an 'on-floor' binding area. Offer area on the other hand define where other objects can attach to, e.g. the 'work-surface' offer area of a table. Binding and offer areas are both defined by a polygon and vector that represent their area and orientation. A binding area can be satisfied by an offer area by aligning their orientation vectors and by translating the binding polygon so that it lies within the offer polygon. If after rotation and translation the binding polygon is not completely enclosed by the offer polygon, then the object cannot be constrained there. In addition, a binding area cannot be bound to an offer area of the same object: an object cannot be constrained to itself.

Note that in contrast to some other approaches we do not use constraint points but *areas*, as only areas can guarantee that an object stands completely on another. Furthermore, many other useful forms of constraints, such as cylindrical or spherical constraints, have been introduced (see previous work). These can be added to our system. However, we choose not to do this to limit the complexity of our research prototype.

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. Proximity 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 distance threshold, 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.

3. Dual Constraints

One of the ways humans perceive groups is by proximity. Another way humans perceive objects as belonging together is similarity. Our target is to emulate these human notions with a computer, hence enabling similar objects that are close to each other to form a group automatically.

Since the scene graph is a-cyclic, only one-way relationships exist between constrained objects, a parent-child relationship. In order to facilitate the use of cyclical constraints, and to create a natural grouping mechanism, we created a new type of constraint, which we call the 'dual constraint'. The dual constraint is very useful in

situations where a sibling relationship between constraints makes more sense than a parent-child relationship, and provides a natural way to group objects together.

While an object is being moved, each of its dual constraints computes the distance to the dual constraints of nearby objects. If this distance falls below a threshold, the moved object will be repositioned next to the nearby object, and a group will be formed automatically. Dual groups are translated, rotated, and regrouped in the scene graph as if the group members were a single object. No explicit operation is necessary to create a dual group.

A dual constraint for an object is specified as a point, line, or polygon in 3D space along with an orientation vector. The constraint sits on or near the surface of the object, and the normal points away from the object.

A cabinet on the wall, for example, may have two dual constraints, one on its left side, and one on its right side. If such a cabinet is pulled toward another cabinet of the same type, the cabinet being pulled moves so that the sides of the two cabinets align. The cabinets are now a group, and stay together until the group is broken. If the dual constraint lies slightly off the surface of the object, space will be left between the object and the others in the group. With this, chairs can be grouped together into rows and/or columns. Figure 3 shows a dual group of cabinets on a wall, and a dual group of chairs on the floor.

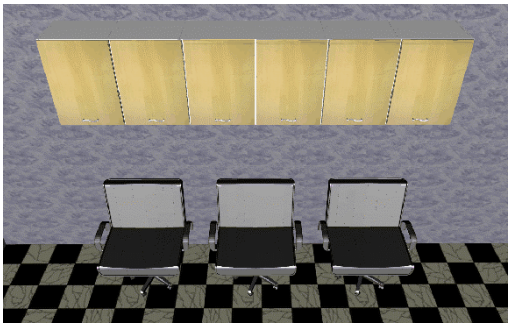


Figure 3: Two dual groups.

To break a dual group, the user simply selects an object within the group and moves the cursor in the desired direction. The motion simulates a pushing action, and all objects in contact in the direction of motion are grouped with the selected object and moved with it. Those objects that are not moved form their own new group(s) and stay in place. This has the desired effect that groups can be created and broken with extreme ease. Figure 4 visualizes the behavior of the manipulation technique.

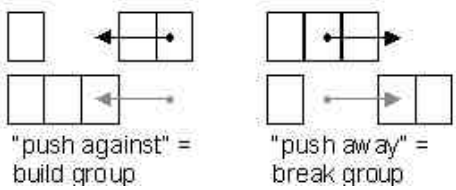


Figure 4: Building and breaking a group.

3.1. Scene Graph Manipulation

To initially constrain a dual constraint, the object(s) being moved must be repositioned so that they are aligned with the object(s) they are constrained to. The repositioning is simply a rotation of the moved object(s) such that the normals of the dual constraints are oriented in opposite directions and a translation such that one of the two dual constraints falls entirely within the other. Figure 5 illustrates the constraint process. When the chair on the right is dragged close to the chair on the left, it will snap in such a way that the dual constraint points (highlighted in yellow) are coincident and the normals are oriented in opposite directions.

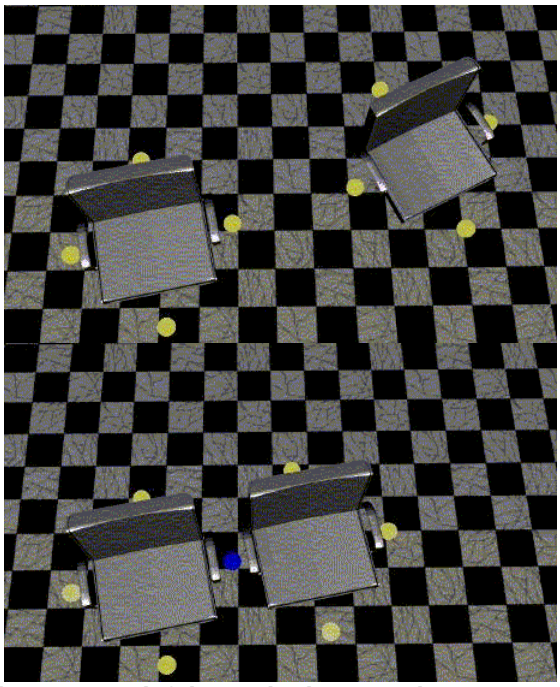
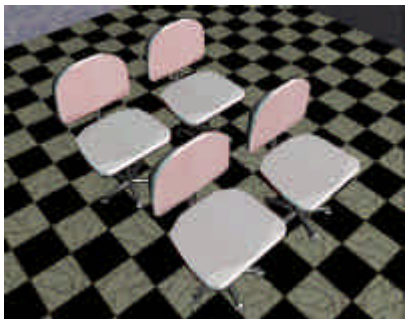


Figure 5: Satisfying a dual constraint.

When objects are dual constrained together, the scene graph must be modified to reflect this. A new node type was introduced, which we call a group node, to facilitate dual constraints in the scene graph. When two objects are first dual constrained together, a dual group node is created, and the nodes representing the two objects are placed under the dual group node in the scene graph. If any other objects are dual constrained to objects within this group, they will also be moved in the scene graph to be under the group node. Figure 6 shows a simple scene containing a dual group of four chairs along with the associated scene graph.

Translating and rotating a dual group of objects can be done by simply transforming the dual group node, as the transformation is automatically applied to all descendants when rendering. Rotations, while still applied to the group node, are actually performed around the center of the

selected object in the group, not the center of the group. This rotation behavior is more flexible, and allows the user more freedom to rotate objects as desired, without complicating the user interface.



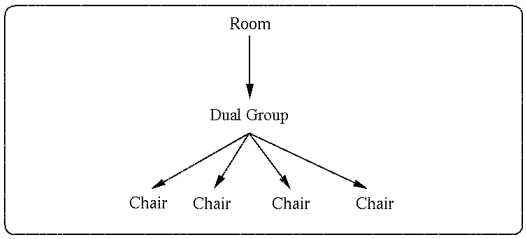


Figure 6: Scene graph for dual group.

3.2. Interaction Techniques with Dual Groups

Creating dual constraint groups is simple. If two objects or groups are moved close to each other the logic described in the previous section is used to create a new dual group node that has all objects involved as children.

Breaking dual constraints is more complicated. We aimed for an intuitive way to break apart groups, so that the user could simply predict how a group would be split. To describe this behavior, we introduce a dual group graph, which describes how objects are constrained to each other in 3D.

The dual constraint graph used in MIVE is symmetric and three-dimensional. For simplicity, examples are shown here using objects that are constrained to a plane, making interactions essentially 2D. Although the examples are 2D, the ideas presented are directly applicable to 3D, as all the underlying data structures are 3D as well.

Figure 7 illustrates a dual group of chairs being split into two groups by a mouse move. For clarity, the gesture is visualized by an arrow in the figures. The user drags the middle chair towards the upper right in this example.

The nodes that will split from the group are called the Directional Dual Component (DDC). The black center node is the object that was selected at the start of the mouse movement. The circle in the bottom left corner of Figure 8 illustrates the associated dual group graphs and the movement direction for reference.

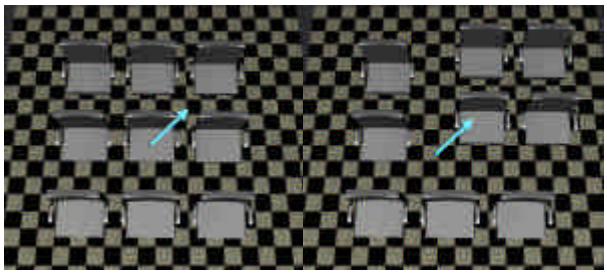


Figure 7: Splitting of a dual group.

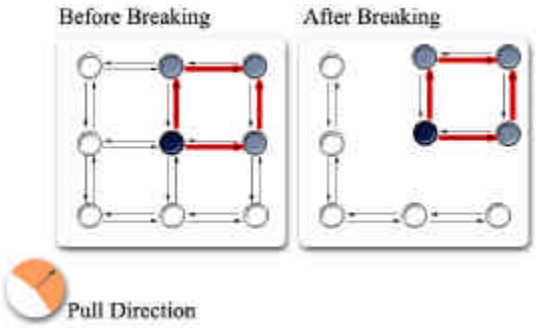


Figure 8: Dual group graphs for Figure 7.

We use a recursive algorithm to decide which nodes make up the DDC. First, the selected node is added to the DDC, as it will always be split from the group. The graph is searched in a depth first manner, where only edges that make (approximately, i.e. using a tolerance) a right angle with the pull direction are followed, and the nodes reached are added to the DDC. This process continues until no new nodes are added to the DDC. In Figure 8, the followed edges are highlighted in red, and the reached nodes are shaded gray. Algorithm 1 shows pseudo-code.

```
# findDDC()takes a node, pull direction
# and a tolerance (usually 10 degrees)
# It returns the vertices that should
# break off the dual group.

findDDC(node,pullDir,tolerance,result)
{
   append node to result
   for (each edge E leaving node) {
      if (angle between E and
          pullDir > 90 + tolerance)
              continue     # skip this edge
      # otherwise, we recurse here
      neighbor = vertex reached when E
                                 is followed
      if (neighbor is not in result)
          findDDC(neighbor, pullDir,
                        tolerance, result)
      } #end for
   }
```

Algorithm 1: Dual component algorithm.

The above mentioned tolerance is necessary for vertical and horizontal mouse drags. Figure 9 shows the dual

group graph and DDC if the user drags the center chair to the right. The tolerance ensures that the dual group breaks in a way consistent with user expectations. Without the tolerance, the user would have to pull exactly in the horizontal direction (a very difficult task) if they wished to split all six objects from the whole group with one drag. In our experience, a tolerance of 10 degrees works very well.

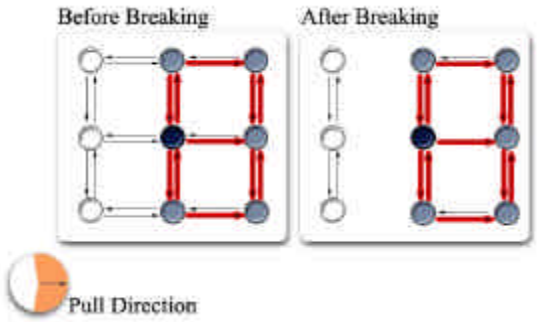


Figure 9: Dual group graphs for a horizontal mouse drag.

To ensure that the user does not accidentally split off an object from a group we activate this logic only if the user has moved for a certain minimum distance.

When a group is split, the user can continue to move the nodes that were split from the group if the mouse button is not released. The group itself, however, may have been split into multiple connected components, as in the example shown in Figure 10. To ensure consistency these connected components must form their own groups in the scene graph. Hence, after the DDC has been split off, the rest of the dual group graph is searched for connected components. If there is more than one remaining connected component, the scene graph is modified to reflect this.

While dual groups share a common parent in the scene graph, it is still possible to place objects as children of the objects in a dual group. Figure 11 illustrates a group of tables, where each table has objects constrained to it. Since these other objects are also descendants of the group node, they will move when the group is moved, but moving them will not move the group.

3.3. Generalization to Arbitrary Scene Graphs

The techniques presented so far have been based on the assumption that the scene graph is in fact a tree structure. If the scene is represented in a directed a-cyclic graph (DAG) where nodes are instantiated multiple times, all the presented techniques are applicable but need slight changes. When a group is being moved and the objects have different parents in the scene graph, then the dual group node will have multiple parents. In this case, we have to use the union of the constraints imposed by all parents on the members of the group to restrict the degrees of freedom when the group is being moved. In addition, we have to make sure to update all transformations correctly when moving an object.

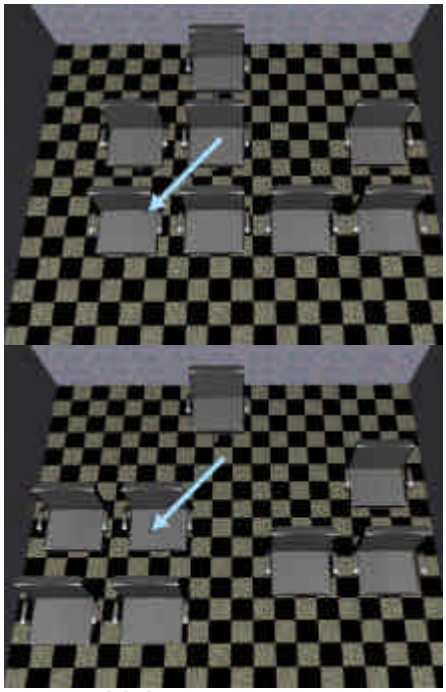


Figure 10: Multiple components after splitting a group.

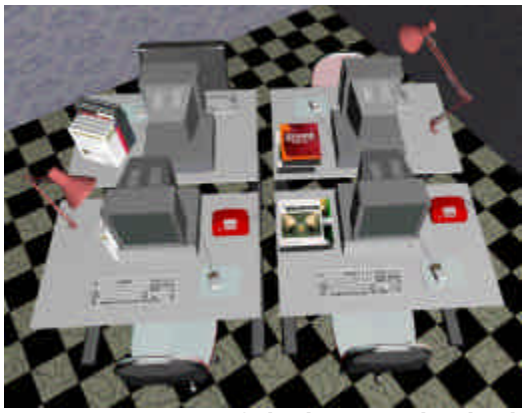


Figure 11: A group of dual constrained tables with objects constrained to each table.

3.4. Explicit Grouping Operations

Sometimes, the user may wish to create a group that cannot simply be described with a hierarchical group (see section 2.2) or with a dual group (see section 3.2). For example, the user may wish to group a bed, a television, and a plant together.

To handle such operations, an explicit grouping mode is provided by the system. A group can be formed by clicking on a button in the function window, then clicking

on objects in the scene. A dual group node is placed in the scene graph as described above and all newly selected members of the group are put under this node.

To remove objects from the group, we can simply use an explicit ungroup operation. Even simpler is to use the methods described in section 3.2. In other words to split objects from a group by dragging them out of the group.

4. Creation of Object Groups

To make the addition of several objects to the scene easier, we allow the user to add multiple objects with a single mouse operation. When an object is selected in the objects window for addition to the scene, the user can click and drag the mouse button in the scene window. This will add multiple instances of the selected object to the scene. We call this technique 'drag-add'. The collision detection together with the logic how constrained objects are added to a scene prevents objects from appearing in the same location and will place objects side by side.

The concept of drag-add is even more powerful, if the individual objects have dual constraints associated with them. Then the collection of objects that was created with the mouse drag forms automatically a dual group and thus can be manipulated efficiently with the techniques described in the previous sections. For example, the scene in Figure 3 was created using two mouse drags, one for the chairs on the floor, and one for the cabinets along the wall.

Instead of having the same geometry appear when a drag-add operation occurs, we could alternately add different object to the scene during the move. We call this 'random drag-add'. It allows the user to add a collection of objects with a single mouse movement. This technique facilitates quick population of a scene with a minimum of interactions. For example, filling a closet with clothes, a shelf with books, or a park with people can be achieved simply with a mouse drag.

Objects that support random drag-add are selected and added to the scene exactly the same way as regular objects. The only difference is that if the left mouse button is held down during the addition of the model, and if the mouse moved over a new location, another object will pop up in that space. For example, to fill a shelf with books, the user simply clicks and drags across the shelf while the book that supports random drag-add is selected in the object selection window (see Figure 12).

In our implementation, the models that make up a particular random drag-add object are stored in an array for efficiency. When the user attempts to add the object to the scene, one of the elements of the array is randomly selected and an attempt is made to constrain the object. If this fails or a collision occurs, the object is not added to the scene. Otherwise, the object is added and another object is randomly selected for the next addition until the user releases the mouse button.

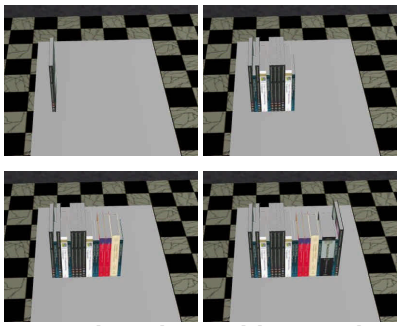


Figure 12: Random drag-add example.

5. Evaluation and Discussion

Although the interaction techniques presented above seem very intuitive, we wanted to confirm this. We chose to evaluate the techniques with a user study that asks participants to create, split and merge groups of object with and without dual constraints.

Space restrictions prohibit us from reporting all details here and consequently, only the most salient results are outlined. Eight participants (6 male, 2 female, average age 28) volunteered for a 15-minute session. All were regular computer users, but had various 3D experience. Each participant was asked to perform three tasks in the dual constraint (or group) mode as well as the non-grouping mode. Trials were counterbalanced to cancel learning effects. The three tasks were to create a 3x3 block of chairs, to split it into four separate groups, and to merge the four groups together into the original group.

The overall result indicates that the dual constraint mode is 1.6 times faster than the non-grouping mode. In fact, the result of the repeated measures ANOVA is highly significant ($F_{1,7} = 42.12$, $p < 0.0005$). For each task, the difference between the two modes is again highly significant. For task 1 (creation), the group mode is 1.3 times faster ($F_{1,7} = 17.15$, $p < 0.005$). For task 2 (split), the group mode is 1.8 times faster ($F_{1,7} = 22.03$, $p < 0.005$), for task 3 (merge) 2.3 times faster ($F_{1,7} = 53.36$, $p < 0.0005$). This is a strong indication for the effectiveness of dual constraints.

For the creation task drag-add was available in both versions (with or without dual constraints). We choose this comparison to highlight the effectiveness of dual constraints. However, based on our observations we are confident that the performance difference would be even larger if a system without drag-add is compared against our system.

All participants commented very positively on the intuitiveness of the manipulation techniques. Further evidence for the intuitiveness of the presented techniques is that most users understood the manipulation technique for dual constraints immediately during the training period, which lasted three minutes on average. Participants

did comment about the lack of feedback (e.g. auditory), when a group was created in our implementation, but were still able to successfully perform the tasks. Some participants even wished to see this manipulation technique in other applications, too!

6. Conclusions and Future Work

Dual constraints provide a new and intuitive way to automatically group and un-group objects in a 3D scene. Furthermore, they afford quick and simple manipulation of object groups. They model the way humans perceive a group of objects belonging together, and afford for direct manipulation of groups. We described the technical details for the implementation of dual constraints. And discussed how this technique can be used with general scene graphs. Furthermore, we presented two ways to create whole groups of objects quickly: drag add and random drag-add. Our evaluation of dual constraints showed that they are both effective and intuitive mechanisms for 3D scene construction and manipulation.

It is important to note that, while the examples in this paper show an interior design scenario, the presented techniques are much more widely applicable. For example, consider the piping and machinery in a power plant. All objects in such a plant have constraints (e.g. pipes fit together). Consequently, all techniques presented here can be applied directly to engineering scenarios and other areas, such as urban design, as well.

The user interface techniques of most current VR systems violate some of the assumptions of the user about how objects can be manipulated, thus creating surprises and frustration. The system presented here is more consistent with the way users think about their environment. This leads to much more intuitive object manipulation techniques.

Due to the use of a 2D input device, we effectively support only 2D operations in this desktop prototype. We are currently porting our system into a full VR setup. Note that we cannot foresee any problems with porting the presented techniques to the full 3D case (i.e. with a 3D tracker as input device), as the techniques rely only on direction vectors and the angles between them.

We are also currently re-implementing this framework for a 2D drawing application to see how well these techniques work for 2D applications.

7. Acknowledgements

The authors want to thank all participants for donating their time for the user study. Furthermore, we want to thank A. Vorozcovs and S. MacKenzie, who have aided this research in several ways. Thanks also to Alias|Wavefront and NSERC for their generous support.

8. REFERENCES

[1] R. Anantha, G. Kramer, R. Crawford, Assembly Modeling by Geometric Constraint Satisfaction. *CAD*, 1996, 707-722.

[2] E. A. Bier, M. C. Stone, Snap-dragging. *SIGGRAPH*, 1986, 233-240.

[3] E. A. Bier, Snap dragging in three dimensions, *SIGGRAPH* 1990, 193-204.

[4] A. Borning, B. Freeman, Ultraviolet: A Constraint Satisfaction Algorithm for Interactive Graphics, *Constraints: An International Journal*, 3, 1998, 1-26.

[5] W. Bouma, *et al.* A Geometric Constraint Solver. *Computer Aided Design*, June 1995, 487-501.

[6] A. Bourning, B. Freeman, M. Wilson, Constraint Hierarchies, *Lisp and Symbolic Computation: An International Journal*, 1992, 223-270.

[7] R. Bukowski, C. Sequin, Object associations, *ACM Symposium on Interactive 3D Graphics*, 1995, 131-138.

[8] M. Dohmen, A Survey of Constraint Satisfaction Techniques for Geometric Modeling. *Computers & Graphics*, 1995, 831-845.

[9] T. Fernando, *et al.* Software Architecture for a Constraint-based Virtual Environment. *VRST*, 1999, 147-153.

[10] M. Gleicher, A Graphics Toolkit Based on Differential Constraints. *UIST*, 1993, 109-120.

[11] S. Hudson, I. Smith, Ultra-Lightweight Constraints, *UIST*, 1996, 147-155.

[12] T. Igarashi, *et al.* Adaptive Recognition of Human-Organized Implicit Structures, *Visual Languages '95*, 258-266.

[13] T. Moran, *et al.* Implicit Structures for Pen-Based Systems Within a Freeform Interaction Paradigm, *CHI*, 1995, 487-494.

[14] E. Lamounier, T. Fernando, P. Dew, Incremental Constraint Satisfaction for Variational Design Systems. *Univ. of Leeds Research Report Series*, Report 95.31.

[15] D. Olson, Inductive Groups, *UIST*, 1996, 193-199.

[16] I. Poupyrev, *et al.* Egocentric object manipulation in virtual environments: empirical evaluation of interaction techniques. *Computer Graphics Forum*, 17(3), 1998, 41-52.

[17] I. Roth, J. P. Frisby, Perception and Representation: A Cognitive Approach. *The Open University*, 1986.

[18] T. Salzman, S. Stachniak, W. Stuerzlinger, Unconstrained vs. Constrained 3D Scene Manipulation, *EHCI*, 2001, 321-333.

[19] SmartScene promotional material Multigen, San Jose, 1999.

[20] G. Smith, T. Salzman, W. Stuerzlinger, 3D Scene Manipulation with 2D Devices and Constraints, *Graphics Interface*, 2001, 135-142.

[21] B. Zanden, B. Myers, Demonstrational and constraint-Based Techniques for Pictorially Specifying Application Objects and Behaviors, *CHI*, 1995, 308-356.

[22] R. Zeleznik, K. Herndon, J. Hughes, SKETCH: An Interface for Sketching 3D Scenes. *SIGGRAPH*, 1996, 163-170.