Object Associations

A Simple and Practical Approach to Virtual 3D Manipulation

Richard W. Bukowski Carlo H. Séquin

University of California at Berkeley [‡]

Abstract

This paper describes a software framework to aid in designing and implementing convenient manipulation behaviors for objects in a 3D virtual environment. A combination of almost realistic-looking pseudo-physical behavior and idealized goal-oriented properties, called object associations, is used to disambiguate the mapping of the 2D cursor motion on the display screen into an appropriate object motion in the 3D virtual world and to determine a valid and desirable final location for the objects to be placed. Objects selected for relocation actively look for nearby objects to associate and align themselves with; an automated implicit grouping mechanism also falls out from this process. Concept, structure, and our implementation of such an object association framework in the context of the Berkeley Soda Hall WALKTHRU environment are presented.

1 Introduction

Creating a fully equipped model of a large, furnished building for virtual walkthroughs is an arduous task. Even assuming the availability of a good interactive 3D geometry editor with a friendly and efficient user interface, such tasks are inherently much more difficult than drafting and editing in only two dimensions. The problem with a 3D world is that it is impossible to exactly control all six degrees of freedom (DOF) at once with only 2-dimensional input and display devices. Typically, software solutions are used to map 2D cursor motion to limited 3D object space motion [12]. These can be cumbersome to use in complex environments, and do not address the fact that objects often require positioning with respect to objects around them. High-tech solutions such as the "SpaceBall" [2], "DataGlove," 3D mice [15], or virtual 3D displays do not solve the problem either; precise placement of objects in three dimensions is hard – even in the real world – unless we get help from the physical interactions of the objects we want to place. Consider positioning a picture frame one millimeter in front of a wall without touching the wall with the frame or with your hands; visual feedback alone cannot do a satisfactory job.

As part of the Berkeley WALKTHRU Project we have built a prototype version of an object manipulation system, called "WALKEDIT," tailored to populating large building models with furniture, personal items, books, coffee cups, and various trimmings and details that make such a building model look real and interesting (see Figure C1). Our approach is based on a system of "object associations," a software framework that supports simple and practical manipulation of 3D objects with 2D I/O devices via two special types of programmer-supplied procedures and an implicit grouping behavior. It gives the programmer the ability to specify object-dependent methods of disambiguating 2D gestures in a 3D world and allows association of suitable local behavior with database objects to make precise default placement easy. These associations usually fall somewhere between physical simulations and mathematical constraints, but can be less formal and more flexible than either.

2 Interactive Building Environments

In the process of developing an editor for our Soda Hall WALKTHRU program, we examined different methods for helping the user to move objects in 3D with 2D devices. We wanted the process of moving furniture in a 3D virtual environment to be as quick and easy as moving cut-out cardboard pieces on a floorplan. However, it should also be possible to force objects to align themselves nicely to walls and to one another, if the operator chooses such an option.

There is no single correct answer to the question of what "ideally" should happen in response to a user dragging an object across the 2D display screen. There is at least one uncontrolled degree of freedom (DOF) due to the third coordinate of the virtual world. Choosing the "right" value to be assigned to this coordinate becomes a contention between *realistic* (physically correct) and *teleological* (goal-oriented) models for the virtual world, and is strongly dependent on the specific application domain. Traditional tools tend to take an extreme stand on one or the other end of the spectrum.

Most 2D drafting tools provide an idealized goal-oriented behavior. Selected shapes freely follow the cursor "across" other objects and snap nicely into alignment with other features if *grids* or *gravity* have been turned on. Setting up these extra controls requires some overhead for activating alignment manifolds, setting up tugboats and orientation frames, changing editing modes, or grouping and un-grouping objects. In 3D virtual worlds, the situation is even worse; ob-

[‡]Computer Science Division, Soda Hall, Berkeley, CA 94720-1776; bukowski@cs.berkeley.edu and sequin@cs.berkeley.edu.

1995 Symposium on Interactive 3D Graphics, Monterey CA USA
© 1995 ACM 0-89791-736-7/95/0004...$3.50

jects now have twice the number of DOFs to be controlled. However, the real world being modeled can often provide disambiguating clues or implicitly desirable alignments. By exploiting these application-specific expectations, some of the teleological control overhead can be automated. In a simulation of a *physical* environment, it seems natural to exploit gravity and solidity to disambiguate the projection of the 2D input parameters into the 3D virtual world. By providing automated alignment with the surfaces on which objects come to rest, extraneous DOFs are removed, and the cognitive burden of specifying them is removed from the user. A complete, accurate, physical simulation, on the other hand, may be counter-productive to efficient 3D editing; we all know how hard it is to move real furniture through a small apartment.

Since we are working in a *virtual* world, we can adopt selectively some of the desirable characteristics, while ignoring others, and add some useful non-physical behavior on top. Such "magic" behavior can be *more* convenient than the behavior of real-world objects [13]. It is easier to move furniture without concern for temporary physical obstruction or inter-penetration; the notorious task of moving a piano through a staircase is no problem in our WALKEDIT environment, and pictures can hang on walls without physical hooks. On the other hand, extra physical or non-physical constraints can be imposed where they simplify manipulation tasks: e.g., pictures can be forced to hang perfectly level at all times, or chairs in a classroom can be made to snap into nicely aligned rows. Object associations provide the structure and the encouragement for the interface programmer to set up this balance between realism and virtual-world magic.

Some of the key paradigms of 3D manipulation and some of the behavioral aspects of objects in a building that we found desirable when populating our Soda Hall model with furniture are summarized below, together with a reference to the object associations that provide the corresponding behavior:

- User-selected objects should follow the mouse pointer, so that "point and place" becomes an integral, intuitive operation. The *relocation procedure* (to be discussed below) provides the main mechanism for this behavior.
- Objects typically should not float in mid-air but rest on some supporting surface. If the cursor points to the surface of a desk or to a bookshelf, it can be implied that the user wants to move the selected object to that particular surface; an *association procedure*, "pseudo-gravity," supports that goal.
- Alternatively, many things, such as picture frames or light fixtures are attached to walls or other vertical surfaces; another *association procedure*, "on-wall," generates the desired behavior.
- Such implicit associations of objects with reference objects should be maintained even when the reference object moves or is changed in other ways; however, they must also be breakable so that objects can be lifted off a surface easily and moved somewhere else. An automatic *dynamic grouping mechanism* built into the object association framework provides that service.

We have also found visibility information to be an important tool. In our WALKEDIT environment, it is natural for the user to move in such a way that the destination point of the motion is visible and the object's final position can be defined precisely by direct pointing. Therefore, we find it acceptable to restrict object manipulations to locations that one can see, avoiding the complexity of user interfaces with which one can reach behind other objects (e.g. systems based on DataGloves or other 3D devices). This simplifies considerably the task of mapping 2D pointing to 3D motion.

3 Object Association Framework

3.1 Background

Our approach borrows heavily from several paradigms developed in the realm of interactive computer graphics over the last decades. It first has notions of *snap-dragging* [3], but without the need of explicitly dealing with visible alignment manifolds; most alignments are provided automatically by the association procedures rather than explicitly by the user. Second, while it can emulate some of the behavior of a *physical simulation* of the objects in the environment [1, 7], it can be less constraining than our every-day world; objects can pass through one another and remain in physically impossible non-equilibrium positions under the control of suitable associations, which may be application-specific or may depend on the editing mode. Third, while some associations could readily be described as *constraints*, our system does not require the rigid formality and associated solution machinery that one would find in a mechanism editor based on an underlying constraint system [11, 4, 10, 8, 6].

A novel feature that emerges naturally from our approach is an automated *implicit grouping mechanism*; it uses the relationships established between objects as they reposition themselves with respect to their environment.

3.2 Two-Phase Approach

The generic editing move in an interactive environment is to "grab" an object and then to "place" it (and any objects grouped with it) somewhere else. In our "WALKEDIT" program, a user clicking on an object selects both the object itself and a *selection point* on the object which makes a natural *handle* for further manipulation. Once the object and its selection point have been established, the user can apply either a *local* motion by dragging the mouse pointer, or a *remote* operation such as "picking up" the object and "placing" it at a different location. Control of these motions and final placement is handled by the procedures of the object association.

When dragging the object to some desirable final position, the intermediate path of the object should be as direct as possible to follow the goal-oriented directive of the user, yet the final position should be "realistic" within the defined simulation constraints of the virtual environment. This extracts the best of the two competing approaches discussed in Section 2; the user can move the object anywhere in a true teleological way, but then the object realigns itself to satisfy some of the physical realities of the environment. This leads to a two-phase approach to moving an object. During a first *relocation phase*, the object follows a trajectory free of physical or behavioral restrictions and which is a suitable disambiguation of the 2D path specification in screen space into a 3D motion in world space. During a second *association phase* the object uses its association rules to determine a good nearby position which best satisfies the stated behavioral conditions of the object in a rest state.

Object associations are thus based on two types of small procedures that are invoked when an object is selected. Each object is assigned one *relocation procedure* but may have a number of prioritized *association procedures*. The relocation procedure is used during local, interactive motion to disambiguate gestures made with the mouse pointer; it defines a mapping of incremental 2D mouse motion to incremental 3D object motion. Association procedures are used for both local and remote placement; they apply additional motion components to an object, based on the other objects in the area, with the goal to preserve the desired object behavior. In addition, objects will dynamically link themselves to the reference objects with respect to which they have aligned themselves; they will typically follow any movements of these reference objects.

3.3 Associations used in WALKEDIT

In WALKEDIT we are primarily concerned with keeping objects supported against gravity, having them attached – and thus properly aligned – to the ceiling, to walls, or to vertical surfaces of other objects, or having objects aligned with respect to each other. All this can be achieved with a remarkably small set of primitives. Different objects carry by design one or more (ordered) association attributes. The user can add or remove extra association attributes from the existing set to selected objects during the interactive walkthrough mode. When such an object is selected, its attributes will determine which relocation procedure applies to the object and which association procedures are used to determine the final placement of the object.

So far we have implemented two relocation procedures: the *on-horizontal* procedure is designed for objects that move primarily horizontally, while the *on-vertical* procedure is for moving along vertical surfaces. In both routines, the object moves along a piecewise continuous, polyhedral 2D manifold in space. The left mouse button *translates* the object along the manifold without changing its orientation relative to the manifold. The middle button *rotates* the object about a line through the center of its bounding box normal to the manifold section on which the object rests.

In addition, there are three association procedures: the *pseudo-gravity* procedure, the *anti-gravity* procedure, and the *on-wall* procedure. Pseudo-gravity simulates objects that normally rest on a supporting surface. Anti-gravity is used for attaching light fixtures, smoke detectors, sprinklers, and other such objects to a ceiling. On-wall is used for pictures, white boards, wall clocks, and other objects that hang on vertical surfaces. All of our WALKEDIT association searches use the same type of ray-based probing mechanism to find alignment objects. These ray-probes determine which nearby objects affect the alignment of the selected object.

We cannot expect that these few simple procedures will take care of *all* editing needs in our building environment. The goal is to make 90% of the typically encountered operations easy and natural. For special needs we still can access traditional editor functions via pull-down menus. If one needs an exact rotation by 45 degrees, one opens the *rotation operator* menu; if one wants to create a perfect row of 20 chairs, the familiar *replicate* menu is perfectly appropriate. If, on the other hand, one finds that one often has to do a special task that is not well supported by classical editor menu commands, such as pushing furniture into corners, then it pays to write a new association procedure "in-corner." This procedure probes in all 4 directions, finds the *two* closest objects, and then does on-wall alignments in two directions, trying to satisfy them both at the same time.

If this is not good enough, because one frequently wants to crowd furniture together in less regular formations, then it is time to develop a more or less accurate pseudo-physical collision detection mechanism and add it to the collection of association procedures. Depending on the types of objects that need to be manipulated, this may simply be based on bounding boxes (good enough for file cabinets) or may use a more sophisticated algorithm that can handle concave objects (needed for grand pianos). We are currently experimenting with a prototype implementation of such a collision detection routine based on the Canny-Lin algorithm that quickly finds closest features in pairs of convex shapes [9, 1].

We have integrated these procedures with the user interface layer that controls all the major editing functions: selection, dynamic grouping, dragging, and detailed placement. In the following sections we review these tasks in detail and discuss our implementation of the procedures that constitute the object association framework.

4 Selection and Dynamic Gathering

In WALKEDIT, selection is performed by shift-clicking the object. There may be other objects that have been previously associated with the selected object; these other objects were positioned with respect to the selected object when they were last moved. For example, the reference object identified by the pseudo-gravity association is the surface on which the selected object came to rest. Since the position of the reference object influenced the position of the selected object, it makes sense to implicitly group the latter with the former and maintain that relative positioning when the reference object is moved. This means that all of these associated objects must be found and grouped with every new object selected; this grouping is maintained for the duration of the motion. An object can have multiple associations; it will then move when any of its reference objects moves.

Associations are *not* permanently maintained constraints; they are applied to the object that is currently being moved. Moving an object can cause other associations to disappear. Doing the group search dynamically ensures that each time an object is picked, the group that gets assigned to it is the right one at that point in time. Because associations are determined from a selected object towards potential reference objects, but are used in the opposite direction, valid associations between two objects may change by the motion of a third, unrelated object. For example, an alignment association between two concave objects may leave space between the two into which a third object can be inserted, thereby breaking the previous association. To allow for such changes and to ensure robust behavior of the object association framework, every time an object is selected we perform a local search for associated objects dynamically in real time and store them in a separate data structure. For efficiency, likely candidates (that is, those objects that were known to be associated with the selected object previously) are checked first. Then, a general search is started in the vicinity of the selected object, relying on our cell-based spatial subdivision structure used for visibility precomputation and observer tracking [14]. The association procedures (see below) are called for all objects incident to the subdivision cells occupied by the selected source object to see if they are associated with it; each object returns a set of association links, and all of these links together form a graph on the objects in that region. The search efficiently calculates a local closure on this graph to obtain the group of objects linked, directly or indirectly, to the selected object.

To keep the virtual environment interactive and the response to any mouse-directed motions instantaneous, we do not delay the interactive manipulation of the original selected object; we carry out the association search in the background. As soon as an associated object is found, it is subjected to the cumulative set of manipulation transformations applied so far to the source object. This approach has the somewhat startling effect, that when the user grabs and moves a fully loaded desk, some of the objects on the desk may at first remain behind, suspended in mid-air, and will then catch up with the new desk position within a few seconds as they are found to be associated with the desk. We found that most users quickly accept this behavior. To minimize this effect, the association closure graphs, once constructed, are cached in memory, so that any further moves of such a group of objects can be truly instantaneous. The closure process may be safely interrupted before closure is complete if the user decides not to move the chosen object but instead selects a different one. The cache holds whatever portion of the graph was completed, and this potentially useful work is saved; the next time an object in the area is selected, the system will simply pick up the search where it was left off.

This implicit grouping mechanism replaces both the explicit grouping mechanism found in many 2D editors and the inherent grouping resulting from setting constraints between objects. Our mechanism keeps the user focused on the actual positioning of the desired object, while automatically making many of the grouping connections the user would have to make by hand with either of the classical methods. Furthermore, breaking a connection between objects that have been implicitly associated is as simple as grasping the dependent (associated) object and moving it to a new location, at which point the association with the old reference object is broken and a new one is established. Of course, we also give the user the power to override the automatic grouping mechanism by turning it off, or to perform grouping manually by *alt*-clicking objects to explicitly add or subtract them from the current group. The two grouping mechanisms can be active simultaneously; adding an object to a group by *alt*-clicking will then also add any associated objects to that group.

5 Dragging with Relocation Procedures

The *local* motion paradigm – dragging the object with the mouse – is the basic editing move for fine-tuning the position of an object, or for moving objects over short distances; the user selects the object, then moves the mouse pointer in the desired direction. To generate each frame of the motion animation, the *relocation procedure* is first called to convert the cursor position into a constrained position on a suitable auxiliary manifold that depends on the type of association carried by the selected object. The relocation procedure moves the object along the manifold in such a way that the selection point maintains coincidence with the cursor. After the relocation procedure determines the base motion, any relevant *association procedures* are run to determine additional motions that the object must perform to maintain its desired behavior. The association procedures will normally move the object in degrees of freedom not controlled by the mouse; however, if a more constraining motion is desired, it may further restrict the motion on the surface of the 2D manifold. For instance, the association procedure may force an object to move along a 1D path as if dragged by an invisible rubber band between the mouse and the selection point.

When the user initiates an interactive motion by holding down some shift/control key and clicking a mouse button, the relocation procedure is called with arguments corresponding to the current screen coordinates of the mouse, the user's view frustum, the particular drag mode being used (translate or rotate), the selection point on the object, and the original mouse screen coordinates where the object was selected. It first makes an *apriori* selection of one or two preferred DOFs that can be controlled directly and unambiguously with a mouse or with another 2-parameter input device, and which most naturally reflect the basic motion of the selected object. A simple, invisible, auxiliary 2-dimensional manifold, such as a plane, cylinder, or sphere, is established through the current selection point; the only requirement for the auxiliary manifold is that its projection into the view window maps points on the screen 1:1 onto points on the manifold. The object is then moved under mouse control in such a way that its selection point stays on the manifold. The mapping between the cursor motion on the screen and the relocation of the selection point in the 3D virtual world is obtained by intersecting the cursor ray from the eye point with the auxiliary manifold. This gives an intuitive behavior for direct control; the object, grabbed by the user-selected handle, will follow the projection of the mouse movement on a reasonable restricted manifold. In general, these manifolds should be piecewise continuous so that the object will move in a predictable local way for small movements of the mouse.

The manifold used in our *on-horizontal* procedure is simply a horizontal plane through the selection point. In the translation mode, the eye-cursor ray is intersected with the plane equation $z = \mathbf{s}_z$, where $\mathbf{s}$ is the original coordinate of the selection point. The ray-plane intersection returns some point $\mathbf{i}$; the procedure returns translation vector $\mathbf{i} - \mathbf{s}$. In the rotation mode, the eye-cursor ray is ignored; the x offset of the mouse pointer on the screen is used as an angle. A rotation by that angle about the plane normal is returned.

On-vertical uses a more complex manifold, composed of piece-wise planar offset surface segments situated in front of the faces of the visible walls in the scene. In the translation mode, the procedure uses the geometric database to intersect the eye-mouse ray with the first surface it hits. If this surface is a vertical one, the intersection point $\mathbf{i}$ of the ray with the surface is determined, and the translation vector $\mathbf{i} - \mathbf{s}$ is returned (where $\mathbf{s}$ is, again, the initial coordinate of the selection point). However, the algorithm also computes the rotation angle between the manifold's surface normal at the selection point and at the new point, and returns that rotation to maintain the orientation of the object's "back" with respect to the manifold. This makes wall hangings follow the changes in wall orientation; if a wall hanging is moved around a corner, the rotation causes it to turn its back toward the new wall as it moves. The on-vertical rotation mode simply rotates the object about the normal of the manifold.

After sliding the object along the alignment manifold, the relocation procedure returns a 3D *offset vector* in space, representing the difference between the original pose of the object when it was selected and the new pose indicated by the mouse motion; this represents the fundamental motion intended by the user. This offset position is what is passed on to the *association procedures* for the object.

6 Placement with Association Procedures

At the offset position, the association procedure needs to find the closest valid rest pose for the moving object, given that the latter is supposed to obey some particular behavior.

The first step is to find the possible candidates for alignment. All of our association procedures currently rely on ray projections. Pseudo-gravity and anti-gravity cast rays vertically downward and upward from the selection point, respectively; the objects that these rays hit are the objects with respect to which the selected object's position is adjusted, falling down or up respectively. The on-wall association casts rays in the major horizontal axis directions of the original definition of the object; the closest object in those four directions is the one used for alignment, as the object "falls" sideways against the closest vertical surface.

In these simple procedures, the object does not change its orientation. It is assumed that the object was suitably defined in its local coordinate system, i.e., in a horizontal, aligned position, so that by simply translating it, say, downwards onto (typically horizontal) floors, it will come to rest in the intended position.

Here is the pseudo-gravity procedure in pseudo code:

1. While the object O has changed height in the last iteration, do:
   (a) Project a ray from the selection point S on object O downward to hit some face F of some object A;
   (b) Determine if S is within the bounding box of some object B (the smallest bounding box if there is more than one);
   (c) if (B is NULL) or (B==A) or (S is visible), drop the bottom of O's bounding box to the height of F; else, lift the bottom of O's bounding box to the height of the top of B's bounding box;
2. Return the total motion of O and associate O with A;

In general, this procedure will place the selected object on top of another one that the user points at by using a combination of visibility cues and interference tests (see section 8.1 for discussion of visibility issues). The anti-gravity procedure, used for objects that stick to ceiling surfaces, is identical to pseudo-gravity with the vertical directions reversed ("upward" instead of "downward" and "bottom" for "top"). The on-wall procedure makes some additional assumptions. For an object to attach itself to a wall, it needs to have some notion of a "back-side" which is moved to be coincident with the closest vertical support surface. Since the Soda Hall object descriptions do not carry such a notion explicitly, we assume that the object is defined with its back's surface-normal in one of the major horizontal axis directions. These four directions are then checked for the closest vertical surface, and the pseudo-gravity algorithm is then run along the corresponding axis. Thus when the user first brings such an object into the Soda Hall environment, it needs to be placed close to some wall with its one side that is supposed to act as its back-side.

For every move generated from an offset vector along the relocation manifold, the association procedures decide what local fix-up motions must be made at the new position to implement the desired local behavior for the object (e.g., falling to a supporting surface, in the case of gravity). Each association procedure computes local components of the overall motion, commensurate with the desired object behavior. The motion generated by the association procedures may also cause the object to change from one supporting manifold to another, such as when the motion generated by the relocation procedure would move the object beyond the edge of the current support or into another solid object.

Once the association has determined what local objects and forces affect the motion of the selected object, the offset vector from the relocation procedure is modified to reflect the local motion, and the new vector is returned from the association procedure. The procedure may also optionally return a set of one or more new local associations of the selected object with other objects in its new environment. When the user finalizes the motion by "releasing" the selected object, these new associations replace the original associations that were in effect when the object was selected.

Objects can be placed into the scene directly out of a *knapsack*. This is a standard inventory mechanism based on a temporary buffer with which users can pick up, put down, cut, copy, or paste the currently selected object or group. In case of such a direct placement from a knapsack, the user designates a destination point, but there exists no original object handle location in 3-space from which an offset vector can be calculated; thus, the relocation procedure is bypassed. In these cases, the eye-to-cursor ray is intersected with the first object that it hits, and a previously determined selection point of the object in the knapsack (or the center of the bounding box, as a default) is brought into coincidence with that 3D location. Normally the object to be placed will now be in an inconsistent physical state with respect to objects at the target position; the association procedure(s) for the selected object are called to correct the positioning in an appropriate way, such as lifting it to the surface of the target object under the influence of "pseudo-gravity" or pasting it to the target face if the primary association of the object to be placed is "on-wall." The object can now be re-selected and further fine-adjusted with local dragging motions.

7 Multiple Associations

Multiple association procedures may come into play for single objects. For example, objects like book cases are supposed to obey *pseudo gravity* and simultaneously fit snugly against walls. This may reduce the DOFs of an object to just one or even zero. In the latter case, the object may jump from one desirable location to the next one as the user moves the mouse pointer and the association procedure selects the closest location that fits the desired behavior.

Multiple associations attached to an object type are explicitly ordered. The corresponding procedures are called in a chain, each one receiving the cumulative associations and offsets generated by the one before. A systems programmer assigning combinations of associations to certain types of objects must consider their possible interactions. The interactions can potentially be very complicated since associations are described functionally rather than mathematically; an association procedure can conceivably do anything. Because of this, it is difficult, if not impossible, for the object association framework to generically resolve conflicts between all combinations of procedures. The associations implemented in WALKEDIT are simple and orthogonal and are particularly tailored to the rectilinear, axial environment of Soda Hall; thus, their interactions are easy to predict and not very problematic. The individual adjustments of all associations are gathered into a single cumulative transformation which is then uniformly applied to the selected object and all its dependent associated objects in the dynamically found group.

Figure 1 shows the flow of control, from the inputs to the object association mechanism to its output for an object with a relocation procedure and two association procedures. On the input side, the user selects the object (upper box) and

then moves it with the mouse pointer (lower box). Selecting the object launches the implicit grouping search, which proceeds simultaneously with the other operations. The original position of the object and the motion of the mouse are sent into the relocation procedure, which uses the initial position and the mouse motion to determine an offset which is sent through the chain of association procedures. Each association procedure modifies the offset and sends it to the next procedure, while outputting associations. The last procedure also outputs the final motion of the object in 3D space, which is applied to the list of objects output by the implicit grouping search.

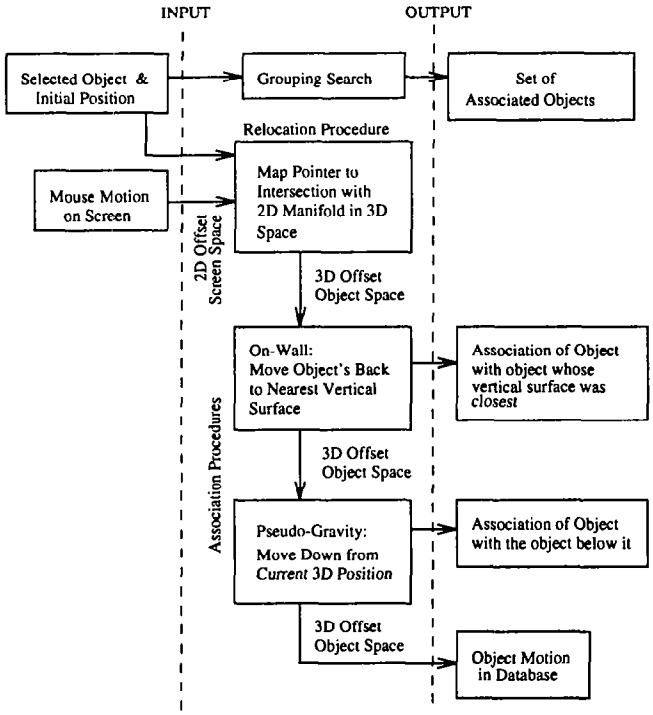

Figure 1: A flowchart showing the various procedures at work for an object that obeys on-wall and pseudo-gravity (for example, a bookcase).

An interesting algorithmic question is raised by cyclic constraints arising from the mutual associations of several objects. Imagine placing two "on-vertical" objects back-to-back in the middle of a room. Each object will associate with the other, thus forming a cycle. If object A is selected, object B will dynamically group with it, and will want to rigidly follow the motion of object A; however, object A will want to move along the surface of object B, because its association sees B as the closest vertical surface. Thus the two objects can never again be moved away from their joint back-to-back alignment plane. A similar situation could arise if an "on-ceiling" light fixture is attached to the underside of a table obeying pseudo-gravity. Our current solution involves breaking loops - once they have been detected - at the point where a large object would associate itself with a smaller object. This seems to provide the right feel in a building environment, but may not be a general enough answer.

8 User Interface Issues

While we can start from a few desirable paradigms (see Section 2) to define the user interface for object manipulation in a 3D virtual world, there will always be situations that will put some of these principles in conflict with one another and where there seems to be no obvious "right" answer. A few such tricky problems are raised in this section and our current solutions are discussed.

8.1 Use of Visibility Information

One of the main cues used to disambiguate the depth coordinate during object manipulation is the intersection of the cursor ray with a visible support surface. Thus when moving an object obeying pseudo-gravity, one would typically grab it near its "foot" while looking downwards onto the supporting surface. This establishes a relocation manifold with a reasonable intersection angle with the cursor ray and gives the user good interactive control over the motion. It raises the issue what should happen when the object is dragged beyond the visible range of the support surface or outside the extent of the support altogether. It also raises the issue how one can ever lift an object *off* such a support surface, e.g., to place a book onto a higher shelf.

Figure 2 illustrates a first typical situation. It should be possible to slide a coffee cup underneath a table; thus, we can not simply lift it to the top of the table when the bounding boxes of the cup and of the table start to intersect. Here we use visibility information and our *pointing* paradigm to resolve the issue. As long as the cursor ray clears the table top, the cup stays on the floor. Since no part of the table is between the cup and the floor, and the cup is not actually intersecting the table, the association procedure has no difficulties settling the cup in a valid position on the floor. However, when the ray intersects any part of the table, *and* the bounding boxes of the cup and the table intersect, the cup gets lifted to the top of the table.

Another critical situation is shown in Figure 3. When the cup is dragged beyond the edge of the table top, a non-physical situation occurs. This could be resolved in two ways. The system could try to place the cup where the cursor ray hits a valid support surface. Since the ray may still hit the table top, or perhaps end in a vertical surface, this will not always lead to a useful answer. Thus we have

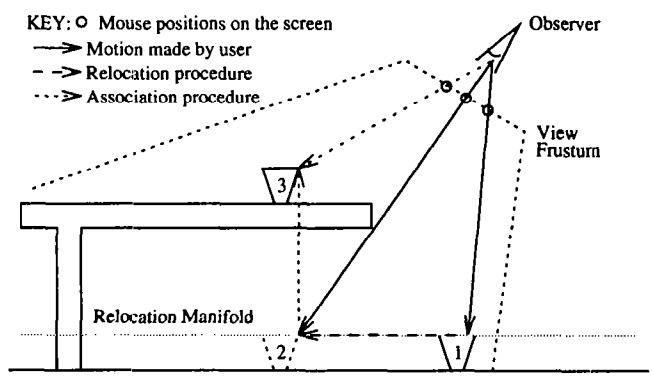

Figure 2: A selected cup (1) is dragged under a table. Visibility information is used to determine when it rises to the tabletop (2); the association procedure modifies both the object position and mouse cursor position (3).

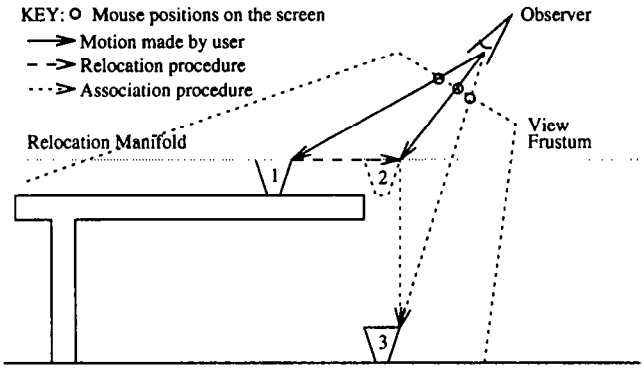

Figure 3: A selected cup (1) is dragged off a table's supporting surface. The cup falls (2) onto the lower surface (3).

found that it makes more sense to give priority to the physical view of the world and drop the cup straight down from the spot where it left the table top to the floor, which then acts as its new support surface.

In all these situations we have an interesting interplay between the teleological and the physical view of our virtual world; visibility information and the intersection of cursor ray with a particular objects are used as additional cues to infer the intent of the user.

8.2 Mouse-Cursor Correspondence

Another key paradigm of the desired user interface is that the object should follow the cursor as directly as possible. This principle needs to be violated necessarily in situations such as the ones above, where establishing a physically valid position may result in a dramatic (vertical) adjustment. As long as the association procedure doesn't add any motion to the object, the relocation procedure usually maintains correspondence. However, the association procedure has no responsibility to maintain the connection between the mouse pointer and the selection point. This then raises the issue whether in such situations the cursor should stay where the user last moved it, or should be "warped" along with the extra motion given to the object by the association procedure. While it is generally preferable to keep the cursor point attached to the handle established at the selection point on the object, this has the consequence that the cursor - and the object itself - may disappear from the screen altogether. Consider the situation in Figure 4 where the cup is moved beyond the *back* end of the table, and where the cursor ray hits no suitable support. The gravity procedure will drop the cup behind the table and possibly out of sight, and the cursor may vanish with it if the floor lies below the lower edge of the viewport. If the fall happens too quickly, the user might not know where the cup has gone and what should be done to bring it back. We have introduced several remedies for this unacceptable situation. First, the cup is made to fall slowly, to imitate reality to some degree and to give the user time to see what is happening. Second, the cursor never disappears entirely from the screen; in the above situation it would be clamped at the lower edge of the viewport. Third, we maintain three axial lines through the selection point on the object to give the user better insight into its position in 3-space. In the above case, the user would thus still see a vertical line emanating from behind the table, giving a clear cue of where the cup currently lies.

To bring the object back into view, the user can move the cursor so that the (invisible) cup moves into the bounding box of the table, whereupon it jumps back to the table top. Alternatively, the user may go to a new location from where the cup is visible, and then continue moving it from its current location on the floor behind the table. Finally, if the object seems totally lost, it can readily be brought into the knapsack while it is still selected, and from there it can be placed directly at the current cursor position. A keyboard shortcut permits to "warp" the object directly from any (possibly hidden) position to the cursor position with a single *ctrl*-click. This operation is also a very efficient way to quickly populate a room with furniture. It takes three mouse operations to place an object in a desired spot: one click to select it, a *ctrl*-click to warp it into the neighborhood of the desired spot, and one *shift*-click-and-drag operation to fine-tune the final position.

9 Software Engineering Concerns

Providing desired object behaviors in 3D virtual worlds is in principle not an easy task. Many nitty-gritty problems concerning data structures and efficient representations must be addressed in order to keep the environment truly interactive. Creating a cohesive framework of object associations is our attempt at keeping this overhead concentrated in one place, so that it can be amortized more easily by the systems programmer with each new object behavior introduced, and so that the user can be given the flexibility of easily choosing the types of behaviors for each object that are most appropriate for the manipulation tasks at hand.

The descriptions of the association and relocation procedures used in the Soda Hall walkthrough look very simple in pseudo-code. It is important to note that the pseudo-code is very close to the level of the actual C code used for the implemented procedures. This is because the WALKTHRU program system provides a rich set of libraries including a complete geometric computation package that operates on vectors, rays, points, planes, and other objects. It also provides the mechanisms to easily search the local area of an object for other objects, to find the objects whose bounding boxes contain a given point, and to quickly find the first object intersected by some space ray. Thus, most lines of pseudo code convert to a few lines of actual C code, making implementation rather straightforward. In such an environment, object associations are most naturally implemented

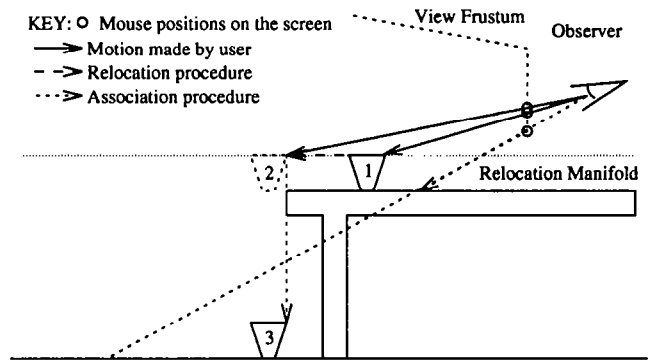

Figure 4: A selected cup (1) is moved off the back of a table (2), falling completely out of the view window (3). The cursor is clamped to the lower edge of the window.

with additional C routines; the C language is more flexible and powerful than any higher level geometric scripting language we could design ourselves.

10 Results

We have constructed a placement editor for real-time interactive walkthrough of large building databases. One of our primary goals was to work with off-the-shelf input and display hardware, a goal which required the use of a software framework to allow the user to perform unambiguous 3D manipulation with 2D devices.

Our solution is based on *object associations*, a framework that provides the flexibility to combine pseudo-physical properties with convenient teleological behavior in a mixture tailor-made for a particular application domain or a special set of tasks. We have found that such a mixture of the "magical" capabilities of geometric editing systems with some partial simulations of real, physical behavior makes a very attractive and easy-to-use editing system for 3D virtual environments. The combination of goal-oriented alignments, such as snap-dragging, with application specific physical behavior, such as gravity and solidity, reduce the degrees of freedom the user has to deal with explicitly while maintaining most of the convenience of a good geometrical drafting program.

We found it to be practical to separate into two types of procedures the mapping of 2D pointing to 3D motion and the enforcement of the desired object placement behavior. These procedures are clearly defined and easy to implement as small add-on functions in C. Geometric and database toolkits allow high-level coding and ease of modification. Our object associations normally cause little computational overhead to the WALKTHRU system. This is an important concern, since keeping the response time of the system fast and interactive is a crucial aspect of its usability and user-friendliness [5].

The result is a technique that makes object placement quick and accurate, works with "drag-and-drop" as well as "cut and paste" interaction techniques, can provide desirable local object behavior and an automated grouping facility, and greatly reduces the need for multiple editing modes in the user interface. The resulting environment is devoid of fancy widgets, sophisticated measuring bars, or multiple view windows. To the novice user it seem that not much is happening – objects simply follow the mouse to reasonable, realistic locations. And that is how ideally it should be: any additional gimmick is an indication that the paradigm has not yet been pushed to its full potential. Some issues remain to be fully resolved, such as dealing with association loops, but our prototype demonstrates that this approach provides a simple, flexible, and practical approach to constructing easy-to-use 3D manipulation interfaces.

A prototype implementation in the context of a model of a building with more than 100 rooms has proven to be attractive and has reduced by a large factor the tedium of placing furniture and wall decorations. One of the authors has constructed scenes of rather cluttered offices with many pieces of furniture, fully loaded with books, pencils, coffee cups, etc. in five to ten minutes (see Figure C2). The implementation in our specific WALKEDIT application domain required only 5 programmer-defined procedures to fully characterize most of the desired object behavior.

References

[1] Baraff, D. Fast Contact Force Computation for Nonpenetrating Rigid Bodies. *Proc. of SIGGRAPH '94* (Orlando, FL, Jul. 1994), pp. 23-34.

[2] Barlow, M. Of Mice and 3D Input Devices. *Computer-Aided Engineering* 12, 4 (Apr. 1993), pp. 54-56.

[3] Bier, E.A. Snap-Dragging in Three Dimensions. *Proc. of the 1990 Symposium on Interactive 3D Graphics* (Snowbird, UT, Mar. 1990), pp. 193-204.

[4] Borning, A. The Programming Aspects of Thinglab, a Constraint-Oriented Simulation Laboratory. *ACM Trans. on Programming Languages and Systems* 3, 4, pp. 353-387.

[5] Funkhouser, T.A. and Séquin, C.H. Adaptive Display Algorithm for Interactive Frame Rates during Visualization of Complex Virtual Environments. *Proc. of SIGGRAPH '93* (Anaheim, CA, Aug. 1993), pp. 247-254.

[6] Gleicher, M. Briar: A Constraint-Based Drawing Program. *Proc. of the ACM Conference on Human Factors in Computing Systems – CHI '92* (Monterey, CA, May 1992), pp. 661-662.

[7] Hahn, J.K. Realistic Animation of Rigid Bodies. *Computer Graphics* 22, 4 (Aug. 1988), pp. 299-208.

[8] Helm, R., Huynh, T., Lassez, C., and Marriott, K. Linear Constraint Technology for Interactive Graphic Systems. *Proc. of Graphics Interface '92* (Vancouver, BC, Canada, May 1992).

[9] Lin, M.C. and Canny, J.F. A fast algorithm for incremental distance calculation. *International Conference on Robotics and Automation*, IEEE (May 1991), pp. 1008-1014.

[10] Myers, B.A. Creating User Interfaces using Programming by Example, Visual Programming, and Constraints. *ACM Trans. on Programming Languages and Systems*, 12, 2 (Apr. 1990), pp. 143-177.

[11] Nelson, G. Juno, a Constraint-Based Graphics System. *Proc. of SIGGRAPH '85* (San Fransisco, CA, Jul. 22-26, 1985). In *Computer Graphics* 19, 3 (Jul. 1985), pp. 235-243.

[12] Nielson, G. and Olsen, D. Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices. *Proc. of the 1986 Workshop on Interactive 3-D Graphics* (Chapel Hill, NC, Oct. 1986), pp. 175-182.

[13] Smith, R.B. Experiences with the Alternate Reality Kit: An Example of the Tension between Literalism and Magic. *IEEE Computer Graphics and Applications* 7, 9 (Sep. 1987), pp. 42-50.

[14] Teller, S.J., and Séquin, C.H. Visibility Preprocessing for Interactive Walkthroughs. *Proc. of SIGGRAPH '91* (Las Vegas, Nevada, Jul. 28-Aug. 2, 1991). In *Computer Graphics*, 25, 4 (Jul. 1991), pp. 61-69.

[15] Venolia, D. Facile 3D Direct Manipulation. *Proc. of the ACM Conference on Human Factors in Computing Systems – CHI 93* (Amsterdam, Netherlands, Apr. 1993), pp. 31-36.

Figure C1.

A snapshot of WALKEDIT in use. A desk is selected. Note the selection point (a small blue-green octahedron) and the fact that the desk contents are implicitly grouped to the desk.

Figure C2.

A scene created with WALKEDIT. There are approximately fifty objects, all resting on valid surfaces and none interpenetrating. This scene took about six minutes to create.

Bukowski and Séquin, "Object Associations: A Simple and Practical Approach to Virtual 3D Manipulation"

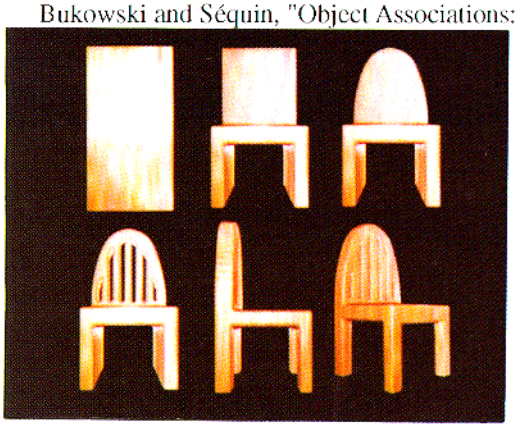

Figure 3: Sculpting of a chair from a block of wood.

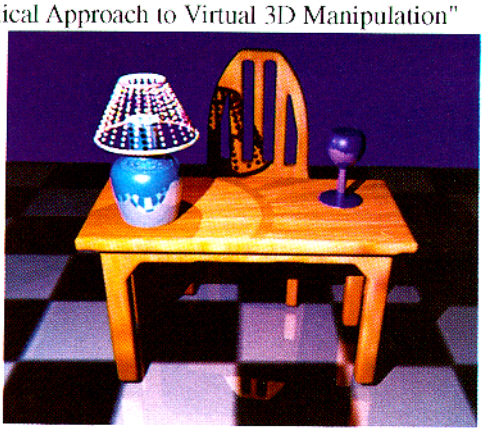

Figure 4: A volumetric ray traced scene of a room.

Figure 5: Sculpted cello and chair.

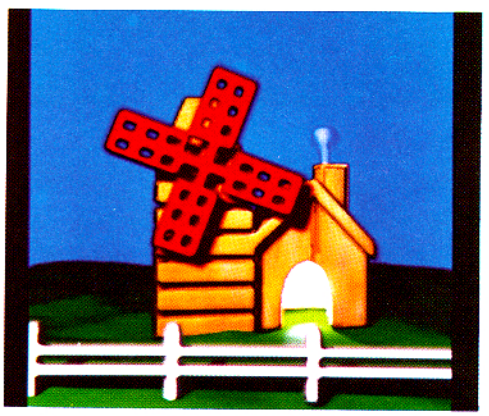

Figure 7: Sculpted windmill on a fractal terrain.

Wang and Kaufman, "Volume Sculpting"