EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(Guest Editors)

Volume 27 (2008), Number 3

Sketching and Composing Widgets for 3D Manipulation

Ryan Schmidt[†] and Karan Singh[†] and Ravin Balakrishnan[†]

Dynamic Graphics Project
University of Toronto, Canada

Abstract

We present an interface for 3D object manipulation in which standard transformation tools are replaced with transient 3D widgets invoked by sketching context-dependent strokes. The widgets are automatically aligned to axes and planes determined by the user's stroke. Sketched pivot-points further expand the interaction vocabulary. Using gestural commands, these basic elements can be assembled into dynamic, user-constructed 3D transformation systems. We supplement precise widget interaction with techniques for coarse object positioning and snapping. Our approach, which is implemented within a broader sketch-based modeling system, also integrates an underlying "widget history" to enable the fluid transfer of widgets between objects. An evaluation indicates that users familiar with 3D manipulation concepts can be taught how to efficiently use our system in under an hour.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Interaction Techniques H.5.2 [Information Interfaces and Presentation]: User Interfaces

1. Introduction

Free-form 3D modeling has made a significant impact on the engineering, design, and entertainment industries. However, free-form modeling tools are not typically regarded as being easy to learn or use. Sketch-based interfaces [Ske07] can potentially improve this situation, as artists and designers respond very positively to the notion of simply drawing the desired 3D shape, rather than constructing it indirectly via control meshes and abstract parameters. However, some practical usability issues have been largely ignored in the transition from standard to sketch-based interfaces. One of these omissions is the lack of sketch-centric techniques for precisely positioning and orienting 3D objects.

Most sketch-based systems have included limited forms of direct 3D manipulation [ZHH96, JSC03, SWSJ05, NISA07]. However, as has been observed [IH01], even novice users will request precise 3D manipulation control. While the traditional 3D manipulation "widgets" found in commercial modeling packages (Figure 1) can be integrated into sketch-based interfacecs, as has been done in the ShapeShop system [SWSJ05], the design of these widgets is based on an explicit "tool" metaphor which conflicts with the more seamless tool-free philosophy of the sketch-modeling paradigm [Iga03]. More "philosophically correct" approaches, such as transformation strokes [SSS06], are currently too time-consuming for practical use, and are limited to coarse transformations.

A parallel thread in the evolution of computer interfaces is the growing availability of pen and touch-based input systems [vD97, DL01, Hal07]. Direct input technologies may appeal to the 3D modeler, but existing tools are mostly unusable on these devices. To cope with the massive complexity of modern 3D modeling software, designers rely on large sets of keyboard "hotkeys" and mode-switching buttons which are largely absent on pen-based computers. This is particularly apparent in 3D manipulation, which, due to it's frequency of use (Section 3), is usually allocated the most common buttons and keys. In addition, the effective "fingertip-blob" input resolution of touch devices makes the smaller widgets in standard interfaces difficult to operate.

We present a novel approach to 3D manipulation which is compatible with free-form sketch-based interfaces and supports buttonless, imprecise touch-based input without sacrificing usability. Using a carefully-designed mixture of 3D widgets, context-sensitive suggestions, and gestural commands, we support most features found in commercial interfaces. In addition to standard translation/rotation/scaling

[†] {rms | karan | ravin}@dgp.toronto.edu

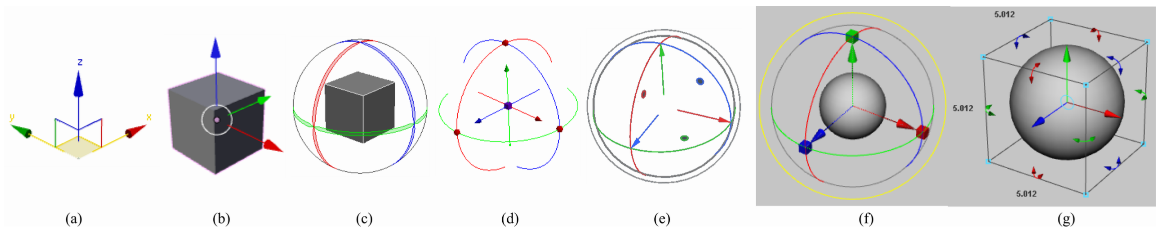

Figure 1: *3D transformation widgets used in other systems. Translation widgets from 3DS Max [Aut07a] (a) and Blender [Ble07] (b), Rotation widget from XSI [Avi07] (c), and combo-widgets from Houdini [Sid07] (d), Modo [Lux07] (e), and Maya [Aut07b] (f,g). While the visual design varies, functionality is largely identical.*

(Section 4), our system includes manipulation relative to arbitrary reference frames, quick free-form snapping, and a simplified approach to transformation strokes (Section 5). A novel "widget history" further simplifies many 3D manipulation tasks (Section 6). A pilot user evaluation (Section 7) provides preliminary evidence that the average 3D user will find our system intuitive and easy to learn, and also efficient enough to be competitive with the "standard" interface.

2. Related Work

Since the dawn of interactive computer graphics, researchers have been exploring the problem of manipulating 3D objects. Early systems used simple valuators [ETW81] to control transformation parameters. As computational power increased it became feasible to interactively manipulate 3D objects in real-time, leading to the development of basic 3D *widgets* such as Bier's Skitters and Jacks system [Bie86].

3D widgets are visual 3D elements which a designer uses to manipulate objects existing in the virtual environment. A variety of general frameworks for 3D widget design have been developed [CSH*92, SZH94], and several specific applications explored [GP95, CSB*05], but by far the most common application is basic 3D manipulation tasks. Although implementation details vary, virtually every commercial freeform modeling tool relies on similar three-axis widgets for translation, rotation, and scaling (Figure 1), with bounding-box handles [Hou92] being added in some cases.

While 3D manipulation widgets have been highly successful, the use of such manipulation "tools" conflicts with the sketch-based interaction philosophy [Iga03], as can be observed in sketch-based systems which have adopted this tool-based approach [SWSJ05]. The SKETCH system [ZHH96] integrated several alternate approaches to the manipulation task, including interactive shadows [HZR*92] and heuristic-based automatic grouping and snapping [Bie90, Gle92, JSC03, OS05]. These techniques make assumptions about scene geometry and viewpoint which are largely incompatible with freeform modeling. However, SKETCH also used gestures to specify transformation axes, an approach which we expand on in Section 4.

Perhaps the most "philosophically correct" solution to the manipulation problem in sketch-based systems is to simply re-draw the object, or it's silhouette, as has been recently proposed [SSS06]. However, because of the necessary approximations in fitting algorithms and the designer's input, such techniques are limited to coarse manipulation. While this may seem compatible with the loose nature of sketch-based interfaces [vD97, Iga03], the reality is that designers will inevitably be unsatisfied with having only imprecise controls. This problem has been observed in user studies of sketch-based tools, where the common request across participants was for direct manipulation control [IH01].

Although we limit our scope to manipulation techniques compatible with pen-based interaction, we note that there has been extensive work on performing 3D manipulation using multiple inputs [ZFS97, BK99] and input devices with additional degrees of freedom [Han97]. Exploring our approach in the context of such devices is left for future work.

3. Design Overview

We have two complementary goals. Primarily, we aim to integrate direct 3D manipulation control into a sketch-based free-form modeling system, using interaction styles compatible with the sketch-based interface philsophy. However, we also desire that our techniques be "competitive" with the 3D manipulation widgets found in existing tools. This not only sets the bar at an appropriate level for our work, but may also assist in making such systems (sketch-based or not) usable with pen and touch-based input systems. We also emphasize that we have focused on techniques applicable to *free-form* modeling. By limiting the user to geometry which is largely planar, tools like SKETCH [ZHH96] and SketchUp [Goo07] can make assumptions which simplify 3D manipulation. Unconstrained free-form modeling requires similarly unconstrained interaction techniques, and hence we design and evaluate accordingly.

When re-designing an interface, it is critical to analyze it's current use. In addition to discussing the problem with working 3D artists, we turned to the voluminous training video material available for 3D modeling software. In particular, many demonstration videos are posted by working 3D artists participating in community-driven "help" forums [Lux07, Ble07]. Often these videos are simply screen-captures of 3D modeling sessions, and even a cursory examination shows that three tasks - selection, 3D manipulation, and camera control - consume the majority of interaction time. In our experience, the same is true when building complex free-form models in sketch-based systems. This is not surprising, as free-form modeling largely consists of positioning objects via trial-and-error, iterating until the artist decides that the results "look good". One thing is clear - the design of any 3D modeling system is constrained by the high frequency with which these manipulation tools are used.

3.1. Design Guidelines

The design of our 3D manipulation techniques has been driven by the set of guidelines listed below.

Button-Free interaction should be designed with button-free input devices in mind. The "barrel-buttons" typically found on tablet pens are difficult to reliably use; hence it is worth focusing on designs that do not require a button.

Sketch-Compatible techniques must be compatible with free-form drawing interaction, minimizing the use of gestures which may conflict with what the user wants to draw.

Tool-Free the designer should be able to intermingle creation and manipulation strokes without switching "tools".

Non-Destructive single strokes should produce transient effects. Changes to the model should require a "two-step commit", to avoid confusing or surprising the designer.

Minimal Clutter manipulation widgets should only be visible when the user wants to use them, to minimize visual clutter. They should also be clearly visually distinct.

Speed Matters because 3D manipulation is so frequent a task, our interface must be competitive with existing approaches if designers are to adopt it.

We now give a brief overview of our system. To support fast, fine-grained manipulation, we utilize 3D widgets. The designer indicates an object to transform by explicitly selecting it with a tap of the pen. Instead of creating a default widget, which the user may have no need for, we treat 3D widgets as transient, context-sensitive responses to user strokes. This also minimizes clutter, as the widgets must be large enough to be easily selectable with a fingertip, and can be very distracting if the user has selected an object for another purpose (particularly if the object is small). We avoid creating large, complicated widgets for the same reason.

Based on the current selection, the system determines a set of candidate transformation axes. Then the user draws a stroke, and the system responds by automatically creating translation and rotation widgets based on the candidate axis nearest to the stroke. These initial widgets can be modifed using context-sensitive gestures, or the user can simply draw another axis. Unless the user gesturally indicates that the widgets should persist, they are automatically dismissed when the object is deselected.

These techniques are sufficient to specify any 3D manipulation. However, relative manipulation ("rotate this around that") can often accomplish the same task much more quickly. We support this in three ways. First, the designer can gesturally create pivot elements. Pivots are first class objects which can be independently manipulated. Objects bound to a pivot (using a crossing stroke) can be transformed relative to the pivot using sketched axis widgets, or precisely re-positioned using snapping. Branching out from snapping, we support fast relative screen-space rotation+translation using transformation strokes. By simplifying the notion of a transformation stroke down to it's most basic elements (an origin and direction), our technique is fast and quite accurate. Finally, we augment the entire system with an interactive transformation history, which allows widgets (and axes) to be efficiently transferred between objects.

3.2. Gestural Command Language

One challenge common to sketch-based interfaces is supporting a wide range of commands without increasing visual interface complexity. One approach is to generate context-dependent *suggestions* and allow the designer to choose from an iconic *suggestion list* [IMKT97, JSC03, SWSJ05]. However, such interfaces can lead to visual clutter, and picking suggestions is not particularly efficient.

Stroke-based command *gestures* provide an alternative to suggestions, and have been applied in a wide range of 2D and 3D interfaces. Designers of the SKETCH system [ZHH96] noted that it is very easy to over-load the user with too many gestures to learn and remember. Hence, we limited our system to 4 simple, orientation-independent gestures (Figure 2). These gestures are "overloaded" in the sense that they are context dependent, but their effect in each context is logically consistent, which may help avoid the "discoverability" problem [ZHH96].

Cross Gesture corresponds to an *Apply* or *Select* action, such as selecting a pivot point for relative transformation. Since our GUI is crossing-based [AG04], users of our system are already familiar with this gesture.

DoubleCross Gesture indicates that the crossed target should become *persistent*, until dismissed with another DoubleCross. Persistent widgets remain in the scene even when their target objects are de-selected.

Perp Gesture requests a "perpendicular" action, such as toggling between an axis constraint and the orthogonal planar constraint. The shape is meant to be analogous to the perpendicularity indicator used in 2D diagrams.

Pigtail Gesture used to disambiguate strokes which specify reference points and/or directions, such as a pivot point on a surface. The "pigtail" is the small self-intersecting loop drawn at the end of the stroke [HBRG05].

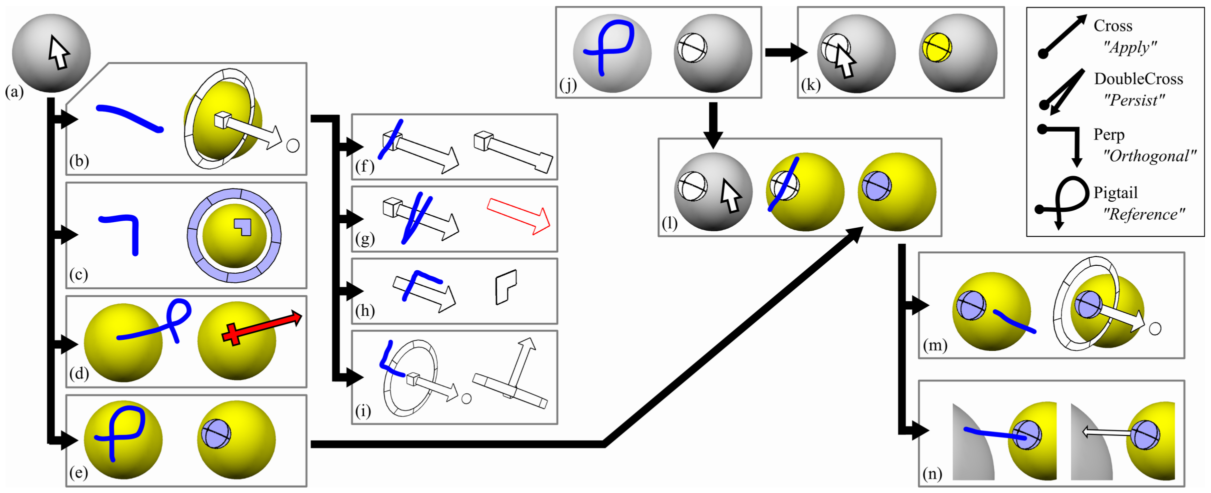

Figure 2: *After tapping an object to select it (a), a rotate/translate/scale widget (b) or screen-space widget (c) can be summoned with a stroke anywhere on the screen. A Pigtail stroke over the selection creates a transformation stroke (d) or pivot (e). Crossing the uniform scaling widget toggles between axis translation and scaling (f), a DoubleCross pins any widget (g), and a Perp creates orthogonal translation (h) and rotation (i) widgets. A pivot can be created at any surface point (j), and then directly selected (k), or Crossed to bind it to a selection (l). Bound objects are transformed relative to the pivot (m), and a line from the pivot to another point on the surface generates a snap suggestion (n). The gesture vocabulary is shown in the top right.*

4. Sketching 3D Widgets

Virtually every major commercial 3D modeling tool uses a 3D widget interface to support fine-grained 3D manipulation. Generally, such widgets act as visual handles for 1D (axis) and 2D (planar) constraints. Our general approach is to allow the user to draw axis constraints using line strokes, and use context-sensitive gestures to create planar constraints. Specific widgets are described in the following sections, but first we discuss some commonalities.

Since a 2D stroke does not define a unique 3D axis, we maintain a set of *candidate axes* generated from the current selection. The world and object *xyz* axes are always candidates, and if the stroke starts on the surface, so are the surface normal and principal curvature directions. Although our system has no knowledge of geometric features such as hard edges, they could easily be integrated into the candidate set. User-drawn 2D strokes are compared with the projected directions of the candidate axes, and the “nearest” axis used to instantiate 3D widgets (Figure 3).

As per our design guidelines, we render widgets using a distinct, “out-of-band” visual style to ensure legibility (Figure 4). Each widget function is represented by a single geometric entity, such as an arrow or cube, with a thick black border and interior color that indicates the axis direction (white for world-space, green for object-space, and so on). The user manipulates widgets using standard click-and-drag interaction. When possible, the clicked point remains under the cursor to enhance kinesthetic feedback.

4.1. Axis Translation

Translation along an axis is one of the simplest types of 3D manipulation, and perhaps the most critical. Hence, we assign a high priority to axis translation. When an object is selected, line strokes are interpreted as requests for axis-translation widgets unless they trigger some other context-dependent action, such as crossing (Figure 4). By default, the center of the selected object is taken as the widget orign. We experimented with using the stroke starting-point to set the widget origin explicitly, as is done in SKETCH, but found that it is difficult to judge where the stroke should begin on smoothly varying surfaces, and the planning required is often time-consuming. Most 3D modeling tools place widgets at object origins, so users are familiar with our approach.

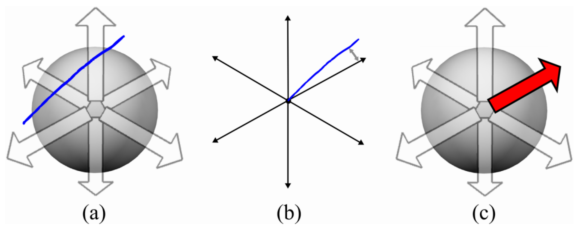

Figure 3: *The system tracks candidate axes for the selected object (a). Candidates are chosen based only on the angle between the 2D stroke and the 2D projections of the axes - stroke and axis position are ignored (b,c).*

Our translation widget is an arrow, positioned such that the tail begins just past the selection origin (this avoids hiding small selections such as individual vertices). Although the arrow is planar, it is dynamically oriented so that it always presents the largest clickable area. From viewpoints where the axis is nearly parallel to the view vector, this area becomes small, but translating along the axis also becomes unstable because each pixel corresponds to a large distance. We de-activate the widget in this case, and render it as semi-transparent to indicate that it is un-clickable.

As the number of candidate axes increases, it becomes difficult to draw a stroke which picks the desired axis, particularly if it is nearly coincident with other axes from the current viewpoint. To avoid frustrating trial-and-error, we only consider world and object axes when interpreting line strokes. Less frequently used axes are accessed via a small *axis-dragging* widget at the end of the translation arrow. Clicking on this disc shows the available candidates, and dragging it snaps the visible widgets to the nearest candidate (Figure 5). Still, from some viewpoints, multiple axes will overlap, forcing the user to rotate the view to choose them. An alternative visualization could improve this interface.

We have found this approach to be highly efficient. Drawing a quick stroke in the desired translation direction is much faster than a round-trip to a button palette, and simultaneously supports multiple coordinate systems. The frequent use of axis-constrained translation justifies our dedicating the straight-line gesture to this task (although only when an object is selected, and context-sensitivity can always be used to overload the stroke). One problem is that in some cases the designer may simply want to draw a line. We experimented with a two-stage process, where the stroke generates a set of axis "suggestion" widgets, which the user crosses to create translation widgets (or ignores if the stroke is a line). Unfortunately this was too slow for frequent use. A hardware "gesture-mode" button efficiently solves the problem, but does not support touch-based input. We currently use a two-second *dwell* to indicate that a stroke is not a gesture [HBRG05], which is slightly annoying but does preserve functionality.

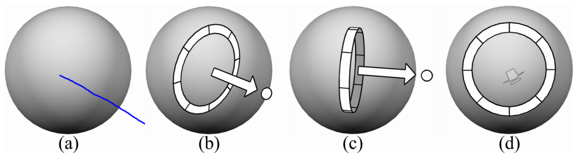

Figure 4: *When a straight-line stroke is drawn (a), the system generates translation and rotation widgets using the closest candidate axis (b). The widgets dynamically re-orient themselves as the view changes to ensure that a large clickable area is presented (c), and disappear if stable operation is impossible from the current viewpoint (d).*

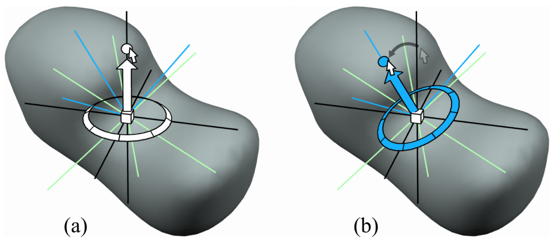

Figure 5: *Only world and object axes are considered for automatic widget generation. Pressing down on the disc at the end of the translation widget (a) shows all candidate axes, and dragging the disc to another axis re-orients the translation and rotation widgets.*

4.2. Planar Translation

While any 3D translation can be accomplished by axis translations, translation constrained to a 3D plane can be precisely controlled by a 2D input, and is often more efficient. The SKETCH system allowed users to create planar constraints by literally drawing the basis vectors of the plane, but this was found to be difficult to do in practice [ZHH96]. In our system, the user toggles between orthogonal axis and planar constraints by crossing them with the *Perp* gesture (Figure 2h). Another common interaction is translating objects in a view-parallel plane - conceptually this is "translation in the screen". A *Perp* gesture drawn anywhere on the screen will summon this frequently-used widget (Figure 2c).

4.3. Axis Rotation

Orienting 3D objects using a 2D input device is a challenge, as evidenced by the number of studies which have investigated the problem [CMS88, JO95, HTP*97, Par99, BRP05]. The various virtual trackballs [CMS88, Sho92, HSH04] seem to perform equally well, however in our experience they are also equally difficult to use. There is quantitive evidence that users are significantly faster and more accurate when performing "simple" single-axis rotation [CMS88]. Commercial virtual trackballs uniformly include constrained rotation widgets (Figure 1), and during our pilot studies (Section 7) we observed that even expert users avoided the virtual trackball, favoring constrained rotation. One commented that the trackball was a "last resort", and another stated that it was only useful for giving objects a "random rotation".

Based on this experience, we have taken the potentially radical step of only supporting axis-constrained rotation in our system. Since any 3D orientation can be specified by rotation around an axis, our approach is to try to make it possible for the designer to specify the "right" rotation axis. Unfortunately, the right axis is not necessarily one of our candidate axes. In this case, the user can draw a *Perp* gesture across any rotation widget to create another rotation widget which is orthogonal to the plane of the first and aligned with the stroke. This supports the turntable-like interaction of first rotating in one plane, and then "up" along an arbitrary arc, which is difficult to perform accurately with a virtual trackball. An example is shown in Figure 6.

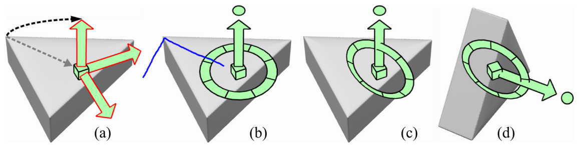

Figure 6: *Often a desired rotation cannot be directly specified using any of the candidate axes (a). By drawing a Perp stroke across a rotation widget (b), a specific orthogonal plane-of-rotation can be specified (c), allowing the goal to be reached with a single rotation (d).*

To avoid the tool switching found in most systems, when an axis-translation widget is generated, so is the corresponding axis-rotation widget (Figure 4b). Similarly, a viewplane-parallel rotation widget is generated along with the planar constraint (Figure 2c). Our axis-rotation widget is represented by a wide planar circle oriented perpendicular to the rotation axis. However, when looking edge-on, the circle is difficult or impossible to hit. To avoid this, we smoothly transition to a circular widget that is extended perpendicular to the plane (Figure 4c) as the viewpoint changes, ensuring that a large clickable area is always presented to the user (various states of the transition can be seen in Figures 2 and 8).

4.4. Scaling

Like most 3D modeling systems, we support two different types of scaling widget. The first, *uniform* scaling, scales simultaneously in all three dimensions. To convey that uniform scaling is a three-dimensional manipulation, we use a small cube-shaped widget. The cube widget also provides a convenient handle for creating constrained scaling widgets, which would otherwise conflict with our translation widgets. The user simply creates translation widgets in the desired axes or planes, then crosses the cube to toggle between scaling and translation (Figure 2f).

4.5. Persistent Widgets

To avoid the overhead of widget management, we treat automatically-generated widgets as transient, and dismiss them whenever a new axis-stroke is drawn or the selection is changed. However, in some cases the user may wish to compose multiple widgets, for example to construct a standard 3-axis triad widget (Figure 7c). This requires the notion of widget persistence.

The *DoubleCross* stroke (Figure 2) can be used to make a widget persistent, or *pin* it to the selected object. Pinned widgets remain visible and active as new axis-strokes are drawn. In addition to enabling the construction of the standard 3-orthogonal-axis widgets seen in other systems (Figure 1), arbitrary widget components lying in various coordinate systems can be combined.

Most 3D modeling systems treat 3D widgets as a sort of modifier interface for the current selection. However, many 3D manipulation tasks involve positioning multiple objects with respect to each other. If these objects are to be manipulated using different widgets (for example, object A must be translated, and object B rotated relative to object C), significant widget-switching overhead is incurred, especially if different coordinate systems are to be used. Widget pinning can improve this situation, simply by keeping pinned widgets active when the selected object changes. This allows the designer to quickly construct sets of widgets which can be used simultaneously, as shown in Figure 7d.

Given a system of pinned widgets, an obvious extension would be to set up constraints between them. For example, the heights of the ears in Figure 7d could be constrained to the head. While we do not yet support such interaction, we plan on exploring it in the future.

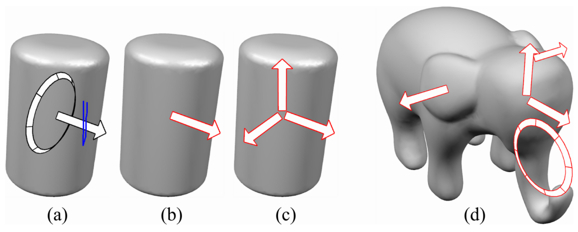

Figure 7: *Widgets can be "pinned" to the selected object using a DoubleCross stroke (a). Pinned widgets have a red border (b), and remain in the scene as other widgets are added (c). Pinned widgets persist across selection changes, allowing arbitrary systems of widgets to be assembled (d).*

5. Relative Manipulation

Three-dimensional manipulations such as translation and rotation are applied relative to some orthogonal coordinate frame centered at a specific point in space - the *origin* of the frame. Each object in our system has an implicit frame (the object coordinate system) and origin (the center of the object bounding box). We have described methods for aligning widgets with different frames, but so far the widget origin has been fixed to some object origin.

This is not inherently limiting - any 3D manipulation can be constructed by translations and rotations around the object origin. However, it can be quite cumbersome. For example, a simple task such as rotating an arm at the shoulder must be decomposed into multiple translations and rotations. To simplify these tasks, we add interaction techniques which support manipulation relative to arbitrary reference frames.

5.1. Pivots

Bier [Bie86] introduced the notion of reference frames as first-class scene objects. His *Jacks* could be directly manipulated, in addition to being used for relative 3D interaction. We adapt this approach - our *pivot* widgets are persistent scene objects which can be directly selected and manipulated to construct arbitrary reference frames. Drawing a *Pigtail* stroke which starts on, and crosses over top of, the surface creates a pivot underneath the first stroke vertex. We experimented with trying to automatically "center" the pivot inside objects, but found that simply placing it on the nearest surface gave the most predictable behaviour.

Selected objects can be *bound* to a pivot by drawing a *Cross* stroke across the pivot. Once bound, 3D manipulations are applied relative to the pivot (Figure 8). By default, pivots are not associated with specific objects, and can even be bound to other pivots. Binding is transient, and objects are automatically un-bound on deselection. We experimented with automatically binding a pivot when an axis stroke crosses it, however we found that it was easy to cross a pivot by accident, resulting in a contextual mis-interpretation that is very confusing for the user. Instead, a *DoubleCross* stroke creates a persistent binding which is maintained even when the object is de-selected.

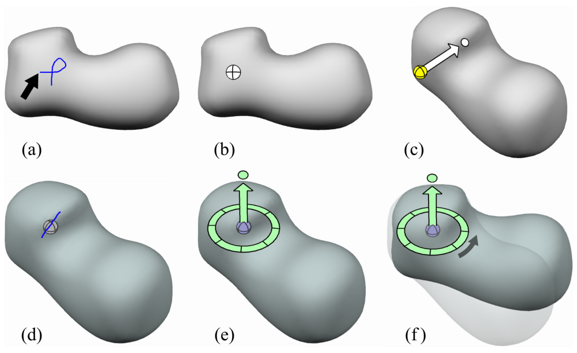

Figure 8: *Drawing a Pigtail gesture over top of an object in the scene (a) drops a Pivot point onto the surface at the start of the stroke (b). Pivots are first-class scene objects which can be directly manipulated (c). A selected object can be* bound *to a pivot using the Cross gesture (d). Once bound, the object is transformed relative to the pivot (e,f).*

5.2. Snapping

Snapping is one of the most effective ways of specifying highly accurate 3D modeling transformations. Interactive snap-dragging techniques [Bie90] can be used to avoid tedious menu and dialog navigation. Engineering-style CAD modeling tools rely heavily on such techniques to precisely position 3D objects. However, snapping interfaces assume the existence of salient features - corners, edges, and faces - which can be used to specify snapping actions. Free-form surfaces are often completely smooth, limiting automatic snapping because the user must manually specify snap points. Our pivot widgets, combined with gestural commands, provide a quick snapping interface which supports both precise and coarse object positioning.

To apply a snap action to an object, it is first bound to a pivot (Figure 9a). Drawing a line from this pivot to another point on the surface generates a snapping suggestion, visualized as a thin arrow (Figure 9b). The user can then cross this widget to translate the pivot to the target point, or Perp-cross it to apply translation plus an additional alignment of the pivot to the target normal (Figure 9c). Alternately, the target can be another pivot, providing more precise snapping.

The "cost" of this snapping operation is much lower than in commercial free-form modeling tools, where a snapping task first requires selection of the snap targets (which itself is non-trivial if arbitrary points on a smooth surface are to be used), and then navigation through menus and dialogs. In the best case, our technique requires only 3 strokes - a Pigtail to place the pivot, a line to specify the target point, and a Cross to apply the snap. As a result, it can be efficient to formulate even relatively simple tasks as snapping actions, followed by minor adjustments using 3D widgets.

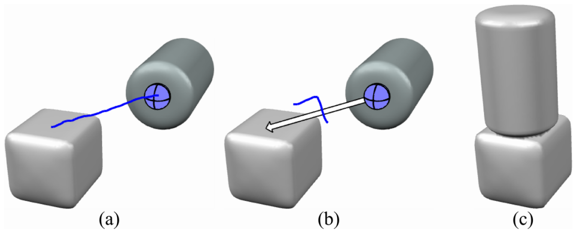

Figure 9: *A snapping suggestion is generated by drawing a stroke which starts on a pivot and ends on the surface, or another pivot (a). Crossing this widget with a Perp stroke (b) snaps the pivot to the target point while aligning the two surface normals (c).*

5.3. Fast 2D Orientation Strokes

One of the common manipulation strategies described by several 3D artists we talked to was to use the 2D screen-space widgets, to quickly (but only approximately) orient objects, and then use the constrained widgets to tweak the 3D orientation. This often involves re-positioning the rotation pivot, to avoid repeated rotation-translation cycles. While our pivots and widgets can be used in the same way, we designed another interaction technique specifically targetting this 2D orientation task.

The idea is to make it quick to apply 2D translation and rotation to an object. In it's simplest form, this is a task of transforming one 2D line (position + vector) into another.

Hence, this is our interface. The designer draws a Pigtail stroke which starts on the selected object and crosses off of it. The first vertex of the stroke specifies the 2D position, and the line segment between the start and crossing points of the stroke specify the direction (Figure 10). This fully determines the *source* line, which is visualized with a dark red arrow. The dark color indicates that the system has entered a modal state, in which any straight-line stroke is interpreted as the destination line, and the object immediately transformed. The destination then becomes the new source line, allowing the user to repeatedly draw strokes to experiment with different orientations. To exit this mode and fix the position of the object, the user de-selects the object.

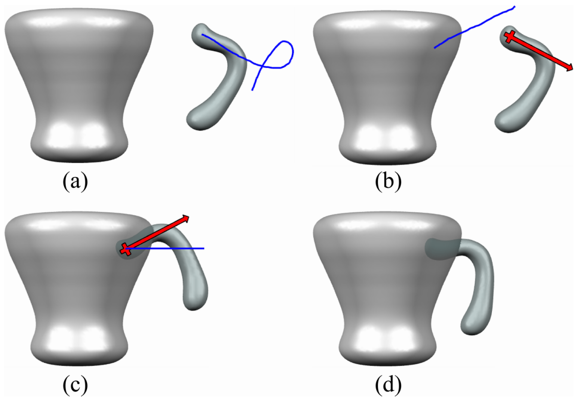

Figure 10: *A pigtail stroke (a) creates a transformation stroke widget, which can be repreatedly re-drawn to quickly orient an object in the view plane (b,c).*

6. Widget History

One limitation of most 3D manipulation systems, ours included, is that while canonical axes can be easily specified, setting up arbitrary axes often requires time-consuming use of pivots. However, it is often the case that the "right" axis is a canonical axis of another object. For example, in Figure 11, the horizontal rungs of the ladder need to be translated in the coordinate system of the vertical beams. Generally, it would be useful to be able to *transfer* axes between objects.

Instead of adding an explicit axis-transfer tool, we provide a more general solution, in the form of a *widget history*. Our system tracks each change to the current widget set, but instead of an undo / redo interface, the widget history is represented as a timeline. Scrubbing through the timeline causes widgets from the past to be re-instantiated at the currently-selected object. Hence, to transfer the current widget to a new object, one simply selects the new object and scrubs back to the most recent state in the history (Figure 11).

The widget history provides a solution to several other problems. For example, we allow designers to construct arbitarary sets of 3D widgets via composition. There is a natural desire to somehow save these widget sets, but an explicit widget-saving system increases UI complexity. Instead, if the widget history stores the associations between pinned widgets and their target objects, complex widget sets are easily recovered. Pivots are also stored in the history, allowing designers to "clean the palette" without worrying about losing carefully-constructed pivot axes. Note that pivots have fixed world coordinates, and hence are not re-instantiated relative to selected objects.

The widget timeline also provides a simple way to remove any existing widgets or pivots without having to explicitly dismiss each one. The designer simply scrubs the timeline "into the future", where no widgets or pivots exist.

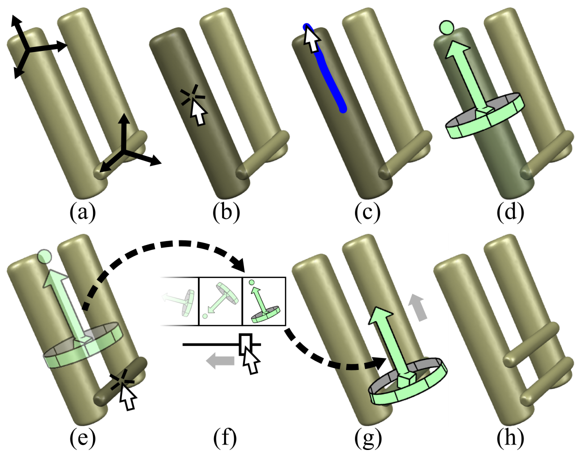

Figure 11: *The user has duplicated the rung and wants to translate it up the ladder. It has no suitable axis, but the beam does (a), so the user selects the beam (b), strokes an axis widget (c,d), and then selects the rung (e), placing the beam axis in the history. Scrubbing back in the history timeline (f) transfers the beam axis to the rung (g), allowing the new rung to be accurately translated (h).*

7. Evaluation

Our interaction design has in part been guided by a series of pilot experiments. These informal tests involved six graduate students from HCI and graphics backgrounds performing a chair assembly task (Figure 12) with our system and with standard 3D widgets in Maya [Aut07b]. Participants used various input devices, including a mouse, pen tablet, and direct input display-tablet. There were no accuracy requirements, participants were encouraged to "think aloud" as they worked, and we discussed the two approaches with each participant after they completed the session. Note that subjects were only instructed on basic translation and rotation widgets, as Maya's pivot and snapping techniques are much more difficult to use.

The results of these pilot studies indicate that our system

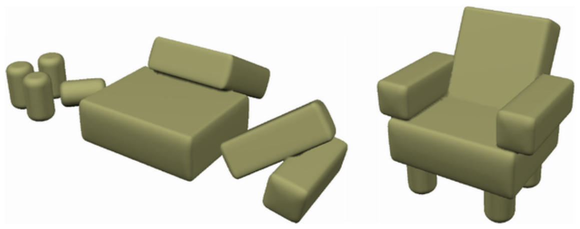

Figure 12: *Subjects were asked to assemble a set of parts (left) into a chair model (right) using our system and Maya.*

is likely to be quite effective. Our two 'Expert' participants who regularly use 3D modeling tools took roughly twice as long to complete the task using our system as they did with the standard Maya widgets. We see this as a very positive result, given that the experts had years of training with standard widgets in Maya, and less than an hour with our modeling system and new manipulation interface.

The two participants with 'Average' 3D modeling skills - meaning that they were familiar with 3D manipulation concepts, but did not regularly use 3D modeling tools - were roughly as efficient with our system as with the Maya widgets. Note that these users required extensive instruction with both tools, indicating that neither is particularly "learnable" without some form of tutorial. Surprisingly, these users seemed to become comfortable with our system faster than some of the expert modelers, who tried to force our interface to behave like the tools they were more familar with.

Two novice participants who had no familiarity with 3D modeling were essentially unable to complete the task, requiring heavy prompting in both our system and Maya. In both cases they positioned the chair parts as if they were 2D objects. They would then rotate the camera and become frustrated upon discovering that some parts were randomly floating in space, exhibiting a fundamental lack of understanding of basic 3D orientation concepts. This observation provided a strong indication that unconstrained 3D manipulation is an expert task, and we have accordingly directed our efforts towards expert use.

We found two major issues that recurred between most participants. The first is that there is no way to draw a rotation axis which is nearly parallel to the view vector. This is problematic because it is often the desired rotation axis. Although screen-space rotation can handle this issue, we discovered that some experienced 3D modelers insisted on using axis-constrained rotation, which required rotating the view, drawing the axis, and then rotating back. Other than relying on the axis-dragging widget (Figure 5), we have not yet found a satisfactory resolution to this problem.

We also found that the *discoverability* problems often seen in gesture interfaces [ZHH96] can arise due to context-sensitivity. For example, some users assumed that axis strokes should be drawn starting on the selected object, and them made the same (incorrect) assumption about pivots. However, a line from the pivot to another surface point produces a snapping suggestion instead of an axis widget, leading to user confusion. Once specific context-sensitive interactions were explained, users rarely repeated their mistakes. This leads us to suggest that context-dependence may be easier to learn (and remember) than a larger gesture vocabulary.

8. Concluding Remarks

We have proposed a new approach to interactive transformation of 3D objects, including translation / rotation / scaling widgets, pivots, snapping, and transformation strokes. To evaluate the system, we implemented it the ShapeShop 3D modeling tool [SWSJ05]. This useful exercise lead to the discovery of various issues in our early designs, such as an over-reliance on subtle differences in context which users initially misunderstood. In our evaluation, we found that context-sensitivity had discoverability problems similar to those found in gesture interfaces [ZHH96], but that once problems were explained, users quickly adapted. Visualization techniques that hint at "hidden" context-dependent actions are a future direction for formal study.

The ability to compose systems of widgets was a surprising emergent property of our persistence mechanism. This composability aspect would be interesting to explore further. One can imagine constructing complex object-specific manipulation systems, and several users requested this functionality specifically. Combining this idea with interactively-defined constraints [Gle92] could be very effective.

Finally, while our *widget history* already adds powerful functionality at little UI cost, it can be improved substantially. Better filtering could reduce the number of redundant "events". It may be useful to consider machine-learning techniques in an attempt to predict which history events are useful, or even to improve automatic widget generation.

Although our pilot studies have been very informative, they do not constitute a formal evaluation. Given that we are proposing a comprehensive re-design of the 3D manipulation workflow, isolating independent variables will be difficult. Furthermore, our experience with novice users suggests that isolated testing will not be indicative of integrated performance. Instead, since our interface has been implemented in the publicly-available ShapeShop system [SWSJ05], we will test it "in the wild" in an upcoming release (and expect to receive extensive feedback from the user population).

Acknowledgements

The authors wish to thank the anonymous reviewers, Tomer Moscovich, and the rest of the DGP for their invaluable discussions and feedback. This work was funded by MITACS and NSERC.

References

[AG04] Apitz G., Guimbretière F.: Crossy: a crossing-based drawing application. In *Proc. UIST 2004* (2004), pp. 3–12.

[Aut07a] Autodesk Inc.: 3ds Max 2008, September 2007. http://www.autodesk.com/3dsmax.

[Aut07b] Autodesk Inc.: Maya 8.5, September 2007. http://www.autodesk.com/maya.

[Avi07] Avid Technology Inc.: SOFTIMAGE|XSI 6.5, September 2007. http://www.softimage.com/products/xsi.

[Bie86] Bier E. A.: Skitters and jacks: interactive 3d positioning tools. In *Proc. I3D 86* (1986), pp. 183–196.

[Bie90] Bier E. A.: Snap-dragging in three dimensions. In *Proc. I3D 1990* (1990), pp. 193–204.

[BK99] Balakrishnan R., Kurtenbach G.: Exploring bimanual camera control and object manipulation in 3d graphics interfaces. In *Proc. CHI '99* (1999), pp. 56–62.

[Ble07] Blender Foundation: Blender 2.44, September 2007. http://www.blender.org.

[BRP05] Bade R., Ritter F., Preim B.: Usability comparison of mouse-based interaction techniques for predictable 3d rotation. In *Smart Graphics* (2005), pp. 138–150.

[CMS88] Chen M., Mountford S. J., Sellen A.: A study in interactive 3-d rotation using 2-d control devices. In *Proc. of SIGGRAPH 88)* (1988), pp. 121–129.

[CSB*05] Coleman P., Singh K., Barrett L., Sudarsanam N., Grimm C.: 3d screen-space widgets for non-linear projection. In *GRAPHITE 05* (2005), pp. 221–228.

[CSH*92] Conner D. B., Snibbe S., Herndon K., Robbins D., Zeleznik R., van Dam A.: Three-dimensional widgets. In *Proc. I3D 92* (1992), pp. 183–188.

[DL01] Dietz P., Leigh D.: Diamondtouch: a multi-user touch technology. In *Proc. UIST 01* (2001), pp. 219–226.

[ETW81] Evans K., Tanner P., Wein M.: Tablet based valuators that provide one, two, or three degrees of freedom. In *Proc. of SIGGRAPH 81)* (1981), pp. 91–97.

[Gle92] Gleicher M.: Integrating constraints and direct manipulation. In *Proc I3D '92* (1992), pp. 171–174.

[Goo07] Google Inc.: SketchUp 6, December 2007. http://www.sketchup.com.

[GP95] Grimm C., Pugmire D.: Visual interfaces for solids modeling. In *Proc. UIST 95* (1995), pp. 51–60.

[Hal07] Interaction tomorrow, 2007. ACM SIGGRAPH 2007 courses.

[Han97] Hand C.: A survey of 3d interaction techniques. *Comp. Graph. Forum 16*, 5 (1997), 269–281.

[HBRG05] Hinckley K., Baudisch P., Ramos G., Guimbretiere F.: Design and analysis of delimiters for selection-action pen gesture phrases in scriboli. In *Proc. CHI '05* (2005), pp. 451–460.

[Hou92] Houde S.: Iterative design of an interface for easy 3-d direct manipulation. In *Proc. CHI '92* (1992), pp. 135–142.

[HSH04] Henriksen K., Sporring J., Hornbaek K.: Virtual trackballs revisited. *IEEE Trans. Vis. and Comp. Graph. 10*, 2 (2004), 206–216.

[HTP*97] Hinckley K., Tullio J., Pausch R., Proffitt D., Kassell N.: Usability analysis of 3d rotation techniques. In *Proc. UIST '97* (1997), pp. 1–10.

[HZR*92] Herndon K., Zeleznik R., Robbins D., Conner D. B., Snibbe S., van Dam A.: Interactive shadows. In *Proc. UIST 92* (1992), pp. 1–6.

[Iga03] Igarashi T.: Freeform user interfaces for graphical computing. In *Proc. Smart Graphics 03* (2003), pp. 39–48.

[IH01] Igarashi T., Hughes J.: A suggestive interface for 3d drawing. In *Proc. UIST '01* (2001), pp. 173–181.

[IMKT97] Igarashi T., Matsuoka S., Kawachiya S., Tanaka H.: Interactive beautification: a technique for rapid geometric design. In *Proc. UIST 97* (1997), pp. 105–114.

[JO95] Jacob I., Oliver J.: Evaluation of techniques for specifying 3d rotations with a 2d input device. In *Proc. HCI '95* (1995), pp. 63–76.

[JSC03] Jorge J., Silva N., Cardoso T.: Gides++. In *Proc. 12th Encontro Português de Computação Gráfica* (2003).

[Lux07] Luxology LLC: Modo community forum, September 2007. http://www.luxology.com/community/tutorials.

[NISA07] Nealen A., Igarashi T., Sorkine O., Alexa M.: Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph. 26*, 3 (2007).

[OS05] Oh J.-Y., Stuerzlinger W.: Moving objects with 2d input devices in cad systems and desktop virtual environments. In *Proc. Graphics Interface 05* (2005), pp. 195–202.

[Par99] Partala T.: Controlling a single 3d object: Viewpoint metaphors, speed and subjective satisfaction. In *Proc. INTERACT '99* (1999), pp. 486–493.

[Sho92] Shoemake K.: ARCBALL: A user interface for specifying three-dimensinal orientation using a mouse. In *Graphics Interface '92* (1992), pp. 151–156.

[Sid07] Side Effects Software Inc.: Houdini 9, September 2007. http://www.sidefx.com.

[Ske07] Sketch-based interfaces: techniques and applications, 2007. ACM SIGGRAPH 2007 courses.

[SSS06] Severn A., Samavati F., Sousa M. C.: Transformation strokes. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2006), pp. 75–82.

[SWSJ05] Schmidt R., Wyvill B., Sousa M. C., Jorge J. A.: Shapeshop: Sketch-based solid modeling with blobtrees. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2005), pp. 53–62.

[SZH94] Stevens M., Zeleznik R., Hughes J.: An architecture for an extensible 3d interface toolkit. In *Proc. UIST '94* (1994), pp. 59–67.

[vD97] van Dam A.: Post-wimp user interfaces. *Commun. ACM 40*, 2 (1997), 63–67.

[ZFS97] Zeleznik R., Forsberg A., Strauss P.: Two pointer input for 3d interaction. In *Proc. I3D '97* (1997).

[ZHH96] Zeleznik R. C., Herndon K. P., Hughes J. F.: Sketch: An interface for sketching 3d scenes. In *Proc. SIGGRAPH 96* (1996), pp. 163–170.