SKETCH: An Interface for Sketching 3D Scenes

Robert C. Zeleznik
Kenneth P. Herndon John F. Hughes
{bcz,kph,jfh}@cs.brown.edu
Brown University site of
the NSF Science and Technology Center
for Computer Graphics and Scientific Visualization
PO Box 1910, Providence, RI 02912

Abstract

Sketching communicates ideas rapidly through approximate visual images with low overhead (pencil and paper), no need for precision or specialized knowledge, and ease of low-level correction and revision. In contrast, most 3D computer modeling systems are good at generating arbitrary views of precise 3D models and support high-level editing and revision. The SKETCH application described in this paper attempts to combine the advantages of each in order to create an environment for *rapidly* conceptualizing and editing *approximate* 3D scenes. To achieve this, SKETCH uses simple non-photorealistic rendering and a purely gestural interface based on simplified line drawings of primitives that allows all operations to be specified *within* the 3D world.

Keywords: Interaction Techniques, 3D Modeling, Gestural Interface, Direct Manipulation, Sketching, Nonphotorealistic Rendering

CR Categories: I.3.8. [Computer Graphics]: Applications; I.3.6. [Computer Graphics]: Methodology and Techniques — Interaction Techniques

1 Introduction

SKETCH targets the exploration and communication of 3D geometric ideas. Traditionally, people have attacked conceptual design with paper and pencil, not with computers, even though computer models offer numerous advantages. The reasons for this include the low overhead of a single-tool interface (pencil), the lack of special knowledge needed to draw, the ease with which many kinds of changes can be made, and the fact that precision is not required to express an idea. Consider Ann sketching a table with an oval top for Joe. Joe gets an immediate sense of the object, without Ann having to indicate the precise locations of the legs, nor the exact shape of the top. By scribbling over what she has sketched, Ann can make the top round or square or freeform without affecting Joe's perception that the legs are attached to the top. (Imagine doing this in a typical CAD or drawing program.) Nevertheless, pencil and paper are still imperfect. After many changes, the paper can become cluttered. Drastic alterations such as showing the model from different viewpoints require new drawings, and collections of drawn objects cannot be transformed as a unit. While computer models do not have these disadvantages, they are typically considerably more difficult to create.

SKETCH is designed to bridge the gap between hand sketches and computer-based modeling programs, combining some of the features of pencil-and-paper sketching and some of the features of CAD systems to provide a lightweight, gesture-based interface to "approximate" 3D polyhedral modeling. Conceptually, our approach is very similar to Landay and Myers' use of sketching to support the early stages of conventional 2D interface design [16]. SKETCH uses a gestural mode of input in which all operations are available directly in the 3D scene through a three-button mouse. The user sketches the salient features of any of a variety of 3D primitives and, following four simple placement rules, SKETCH instantiates the corresponding 3D primitive in the 3D scene. SKETCH allows both geometry and the camera to be gesturally manipulated, and uses an automatic grouping mechanism, similar to that described by Bukowski and Sequin [6], to make it easier to transform aggregates of geometry. Since the set of geometric primitives is more restricted than those in most CAD systems, the user *approximates* complex shapes with aggregates of simpler primitives. Since we know these conceptual models are approximations (often to only partially formed mental images) SKETCH renders them with *non-photorealistic* rendering techniques designed to help viewers see what they want to see.

We also imagine that SKETCH might be used as part of a storyboarding system, for generating a series of scenes and camera views in planning a 3D animation.

The accompanying videotape[1] shows the features of SKETCH and indicates the utility of its simple approach in creating and editing 3D models.

2 Related work

A variety of efforts have been made to simplify the process of generating 3D models, including the "idea sketching" described by Akeo et al. [1]. Akeo allows users to scan real sketches into the computer where they are "marked-up" with perspective vanishing lines and 3D cross sections. The scanned data is then projected onto the 3D mark-up to complete the process.

Nearly all CAD applications employ some form of 2D sketching, although sketching is rarely used in 3D views. A notable exception is Artifice's Design Workshop [2], which allows cubes, walls, and constructive solid geometry (CSG) operations to be constructed directly in the 3D view. However, the overall style of interaction is still menu-oriented and the set of primitives is small.

The considerable work done in the area of drawing interpretation, surveyed by Wang and Grinstein [28], focuses solely on interpreting an entire line drawing at once. In contrast, we attempt to provide a complete interface for progressively conceptualizing 3D scenes using aspects of drawing interpretation to recognize primitives from

[1] The videotape can be obtained upon request from the authors.

a gesture stream. Viking [20] uses a constraint based approach to derive 3D geometry from 2D sketches. In Viking, the user draws line segments, and the system automatically generates a number of constraints which then must be satisfied in order to re-create a 3D shape. The difficulty with these approaches is that even though they are generally restricted to polygonal objects, they are often slow and difficult to implement. In addition, they are often intolerant of noisy input and may either be unable to find a reasonable 3D solution, or may find an unexpected solution. Branco et al. [5] combine drawing interpretation with more traditional 3D modeling tools, like CSG operators in order to simplify the interpretation process; however, their system is limited by a menu-oriented interaction style and does not consider constructing and editing full 3D scenes.

Deering [10], Sachs et al. [22], Galyean and Hughes [11], and Butterworth et al. [7] take a very different approach to constructing 3D models that requires 3D input devices as the primary input mechanism. A variety of systems have incorporated gesture recognition into their user interfaces, including Rubine [21], who uses gesture recognition in a 2D drawing program, but we know of no systems that have extended the use of gesture recognition for 3D modeling.

We also use a variety of direct-manipulation interaction techniques for transforming 3D objects that are related to the work of Snibbe et al. [25], and Strauss and Cary [27]. In addition, we also exploit some very simple flexible constrained manipulation techniques that are similar to those described by Bukowski and Sequin [6]. The latter automatically generates motion constraints for an object directly from that object's semantics. Therefore, for example, when a picture frame is dragged around a room, the frame's back always remains flush with some wall in the room to avoid unnatural situations in which the picture frame might float in mid-air. Also, when a table is manipulated, all of the objects that are on top of the table are automatically moved as well.

In our system, since we have less semantic information than Bukowski, we have less opportunity to automatically generate appropriate constraints, and therefore we occasionally require the user to explicitly sketch constraints in addition to geometry. Our constraint techniques are fast, flexible and almost trivial to implement, but they are not as powerful as the constrained manipulation described by Gleicher [12] or Sistare [24]. Although Gleicher exploits the fact that constraints always start off satisfied, thereby reducing constraint satisfaction to constraint maintenance, he still must solve systems of equations during each manipulation which are often slow and subject to numerical instability. Other approaches like Bier's snap-dragging [4] are also related to our constrained manipulation, although we never present the user with a set of constraint choices from which to select.

Lansdown and Schofield [17] and Salisbury et al. [23] provide interesting techniques for non-photorealistic rendering, although none of these systems specifically targets interactive rendering.

3 The interface

All interaction with SKETCH is via a three-button mouse[2] with occasional use of one modifier key on the keyboard, and a single orthographic window onto the 3D scene. The mouse is used to generate gestures rather than to select operations from menus. Choosing an operation like object creation, transformation or grouping is seamlessly integrated with the natural expression of intent. SKETCH infers intended tools by recognizing gestures — sequences of two types of gestural elements — in its input stream.

Strokes, the first type of gestural element, are pixel-tracks on the film plane[3], made with the first mouse button. There are five classes of strokes shown in Table 1.

Each axis-aligned stroke is aligned with the projection of one the three principal axes of the world. We have also tried aligning strokes with the three principal axes of the surface over which the gesture is drawn. In general, this latter approach seems more effective, although it also presents some difficulties, especially for curved surfaces and for gestures which span over different surfaces. Since we have not yet adequately handled these concerns in our implementation, we will assume for the rest of the paper that all lines are aligned with the world's principal axes except those that are drawn with the "tearing" or freehand strokes.

mouse action	**stroke**
click and release	dot
click and drag without delaying	axis-aligned line: line follows axis whose screen projection is most nearly parallel to dragged-out segment
click and drag, then "tearing" motion to "rip" line from axis	non-axis-aligned line
click, pause, draw	freehand curve
click with Shift key pressed, draw	freehand curve drawn on *surface* of objects in scene

Table 1: The five stroke classes.

Interactors, the second type of gestural element, are made with the second mouse button. The two classes of interactors, a "click and drag" and a "click," have no visual representation.

In addition to gestural elements, SKETCH supports direct-manipulation of camera parameters with the third mouse button, as outlined in Table 2. Third-button manipulations are not discussed further in this paper.

mouse action	**camera manipulation**
click and drag	*pan*: point on film plane beneath mouse remains beneath mouse
click, pause, drag	*zoom/vertical pan*: dragging horizontally zoom in/out towards clicked-on point, dragging vertically pan up/down
click near window boundary, drag	*rotate*: performs continuous XY controller rotation about center of screen [8]
click on object	*"focus"*: camera moves so that object is in center of view [18]
shift-click	*change rendering*: cycles through available rendering styles (see Section 5)

Table 2: Gestures for camera manipulation.

4 The implementation

SKETCH processes sequences of *strokes* and *interactors* to perform various modeling functions with a finite-state machine. The mapping between gestural input and modeling functions is easy to remember and gives the user a clear and direct correspondence. However, one of the principal difficulties in developing a good gesture-based interface is managing the delicate tradeoff among gestures that are natural, gestures that are effective, and gestures that are effective within a system that may already use similar gestures for other functions. For superior gestures to evolve, this tradeoff should continue to be explored especially with user studies.

4.1 Creating geometry

We believe gestures can be a natural interface for the fundamentally visual task of creating 3D geometry. The difficulty is choosing the "right" gesture for each geometric primitive. In SKETCH, we define "primary" gestures for instantiating primitives as sequences

[2] We think that a tablet/pen and an active LCD screen implementation might be even better. See Section 6.

[3] A plane perpendicular to the view direction and close enough to the eyepoint not to interfere with the objects in the scene.

of strokes that correspond to important visual features — generally edges — in partial drawings of the primitives. (see Figure 2 for an overview of all such gestures.) For instance, a drawing of three non-collinear line segments which meet at a point imply a corner, based on our visual understanding of drawings [19]; consequently, we interpret similar gestures composed of three line strokes as a cuboid construction operation.

We also provide alternate construction gestures using non-edge strokes. For example, an object of revolution is sketched via its profile and axis, and cuboids can be created by sketching a single edge and two "dimensioning segments" (perpendicular to the edge) that meet at a vertex lying anywhere along this edge. These alternative gestures take their structure from the notions of generative modeling [26].

SKETCH's other primitives — cones, cylinders, spheres, objects of revolution, prisms, extrusions, ducts and superquadrics — have their own gestures. For most, SKETCH forces some aspect of the shapes to be axis-aligned, so that the gestures are easier to both draw and recognize. For example, to create a duct, the user strokes a closed freehand curve for its cross section, and another freehand curve for its path of extrusion. However, an arbitrary 3D curve is not uniquely determined by a single 2D projection, so SKETCH's ducts must have extrusion paths that lie on an axis-aligned plane, specified by a third gesture — an axis-aligned line stroke normal to the plane on which the path of extrusion should be projected.

The small number of primitive objects sometimes requires the user to build up geometry from simpler pieces, and precludes some complex objects — freeform surfaces and true 3D ducts, for example — from being made at all. But in exchange for this, we believe that our small set of primitives minimizes cognitive load on the user and makes gesture recognition and disambiguation easier. Future work, including user studies, should explore this tradeoff.

4.2 Placing geometry

Object creation requires *placement* of the object in the scene. We base object placement on four rules: first, geometry is placed so that its salient features project onto their corresponding gesture strokes in the film plane; second, new objects are instantiated in contact with an existing object when possible; third, certain invariants of junctions in line drawings [9] that indicate the placement or dimension of geometry are exploited; and fourth, CSG subtraction is inferred automatically from the direction of gesture strokes (Figure 2).

These easy-to-understand rules often generate good placement choices; when they do not, users can edit the results. Furthermore, the few users that the system has had so far have rapidly learned to use the simple rules to their advantage, "tricking" the algorithm into doing what they want. (This may be a consequence of their programming background.)

Figure 1: A series of strokes is drawn in the film plane in red (left). The salient vertex is projected into the scene thus defining the placement of new geometry (green). Though this figure suggests a perspective camera, we use a parallel projection in our application.

The first rule determines object placement except for translation along the view direction. This ambiguity is generally resolved by the second rule, implemented as follows: each gesture has a "most salient" vertex (the trivalent vertex for a cuboid, for example, or the first vertex of the two parallel strokes that indicate a cylinder); a ray is traced through this vertex to hit a surface at some point in the scene. The object is then instantiated so that the salient vertex is placed at the intersected surface point (Figure 1).[4]

The third placement rule exploits invariants of vertex junctions in line drawings, as described by Clowes [9]. However, our use of T junctions is related to the treatment given by Lamb and Bandopadhay [15]. In particular, T-shaped junctions arise in line drawings when a line indicating the edge of one surface, *Estem*, ends somewhere along a line segment indicating the edge of another surface, *Ebar*. These junctions generally signify that the surface associated with *Estem* is occluded by the surface associated with *Ebar*, although it does not necessarily indicate that the two surfaces meet. In SKETCH, a similar notion exists when a straight line segment (except for connected polyline segments) of a gesture ends along an edge of an object already in the scene (Figure 3). To uphold the intuition that such T junctions indicate the occlusion of one surface by another, SKETCH first places the gesture line into the 3D scene according to the previous two placement rules. Then, SKETCH sends a ray out along the gesture line (toward the T junction). If this ray intersects the object that defined the bar of the T junction and the normal at the point of intersection is pointed in approximately the opposite direction of the ray, then the gesture edge is extended so that it will meet that surface.

If the ray does not intersect the surface, then the object defined by the surface is translated along the viewing vector toward the viewer until its edge exactly meets the end of the gesture edge. If the end of the gesture edge is never met (because it was farther away from the viewer), then neither the gesture, nor the existing objects are modified. We never translate objects away from the viewer as a result of T junctions; tests of this behavior on a variety of users indicated that it was both unintuitive and undesirable.

The final rule determines whether the new geometry should be CSG-subtracted from the scene when added to it. If one or more of the gesture strokes are drawn *into* an existing surface (i.e., the dot product of a stroke and the normal to the existing surface on which it is drawn is negative), then the new piece of geometry is placed in the scene and subtracted from the existing object (Figure 2). CSG subtraction is recomputed each time the new geometry is manipulated. If the new geometry is moved out of the surface from which it was subtracted, CSG subtraction is no longer recomputed. This makes possible such constructions as the desk drawer in the Editing-Grouping-Copying section of the videotape.

4.3 Editing

SKETCH supports multiple techniques for editing geometry. Some exploit paper and pencil editing techniques by recognizing editing gestures composed of strokes (e.g., oversketching and drawing shadows). Others use gestures that contain an interactor to transform shapes as a whole by translation or rotation.

Resizing. A common way to "resize" a surface with pencil and paper is to sketch back and forth over its bounding lines until they are of the right size. SKETCH recognizes a similar "oversketching" gesture to reshape objects. If two approximately coincident lines are drawn in opposite directions nearly parallel to an existing edge, SKETCH infers a resizing operation (Figure 2). This sketching operation works for all primitives constructed of straight line segments, including cubes, cylinders and extrusions. Additionally, the two endpoints of an extrusion path can be attached to two objects

[4]If the ray intersects no surface (possible because we use a finite ground rectangle instead of an infinite ground plane), the object is placed in the plane perpendicular to the view direction that passes through the origin; this turns out in practice to be a reasonable compromise.

in the scene; whenever either object moves, the extrusion will resize to maintain the span. However, general reshaping of objects defined by freehand curves is more difficult and not yet fully implemented. We are currently adapting Baudel's mark-based interaction paradigm [3] for use in reshaping 3D objects.

Sketching shadows. Shadows are an important cue for determining the depth of an object in a scene [29]. In SKETCH, we exploit this relationship by allowing users to edit an object's position by drawing its shadow. The gesture for this is first to stroke a dot over an object, and then to stroke its approximate shadow — a set of impressionistic line strokes — on another surface using the Shift modifier key.[5] The dot indicates which object is being shadowed, and the displacement of the shadow from the object determines the new position for the object (as if there were a directional light source directed opposite to the normal of the surface on which the shadow is drawn). The resulting shadow is also "interactive" and can be manipulated as described by Herndon et al. [13].

Transforming. Objects can be transformed as a unit by using a "click-and-drag" interactor (with the second mouse button): the click determines the object to manipulate, and the drag determines the amount and direction of the manipulation. By default, objects are constrained to translate, while following the cursor, along the locally planar surface on which they were created. However, this motion can be further constrained or can be converted to a rotational motion.

It is important to keep in mind that our interaction constraints are all very simple. In SKETCH, instead of using a constraint solver capable of handling a wide variety of constraints, we associate an interaction handler with each geometric object. This handler contains constraint information including which plane or axis an object is constrained to translate along, or which axis an object is constrained to rotate about. Then when the user manipulates an object, all of the mouse motion data is channeled through that object's handler which converts the 2D mouse data into constrained 3D transformations. To define which of our simple constraints is active, we require that the user explicitly specify the constraint with a gesture. Whenever a new constraint is specified for an object it will persist until another constraint is specified for that object. Each new constraint for an object overwrites any previous constraint on the object.

The advantages of such a simple constraint system are that it is robust, fast, and easy to understand. A more sophisticated constraint engine would allow a greater variety of constrained manipulation, but it would also require that the user be aware of which constraints were active and how each constraint worked. It would also require additional gestures so that the user could specify these other constraints.

Systems such as Kurlander and Feiner's [14] attempt to infer constraints from multiple drawings, but this approach has the drawback that multiple valid configurations of the system need to be made in order to define a constraint. Such approaches may also infer constraints that the user never intended, or may be too limited to be able to infer constraints that a user wants.

Constrained transformation. The gestures for constraining object transformations to single-axis translation or rotation, or to plane-aligned translation are composed of a series of strokes that define the constraint, followed by a "click-and-drag" interactor to perform the actual manipulation (Figure 2). To constrain the motion of an object to an axis-aligned axis, the user first strokes a constraint axis, then translates the object with an interactor by clicking and dragging parallel to the constraint axis. The constraint axis is stroked just as if a new piece of geometry were being constructed; however, since this stroke is followed by an interactor, a translation gesture is recognized and no geometry is created. Similarly, if the user drags perpendicular to the constraint axis instead of parallel to it, the gesture is interpreted as a single axis rotation. (This gesture roughly corresponds to the motion one would use in the real world to rotate an object about an axis.)

To translate in one of the three axis-aligned planes, two perpendicular lines must be stroked on an object. The directions of these two lines determine the plane in which the object is constrained to translate. If the two perpendicular lines are drawn over a different object from the one manipulated, they are interpreted as a *contact* constraint (although non-intuitive, this gesture is effective in practice). This forces the manipulated object to move so that it is always in contact with *some* surface in the scene (but not necessarily the object over which the gesture was drawn) while tracking the cursor. Finally, a dot stroke drawn on an object before using an interactor is interpreted as the viewing vector; the object will be constrained to translate along this vector. This constraint is particularly useful for fine-tuning the placement of an object if SKETCH has placed it at the "wrong" depth; however, since we use an orthographic view that does not automatically generate shadows, feedback for this motion is limited to seeing the object penetrate other objects in the scene. We believe that a rendering mode in which shadows were automatically generated for all objects would be beneficial, although we have not implemented such a mode because of the expected computational overhead. We did, however, mock up a rendering mode in which just the manipulated object automatically cast its shadow on the rest of the scene. People in our group generally found the shadow helpful, but were slightly disturbed that none of the other objects cast shadows.

In each case, the manipulation constraint, once established, is maintained during subsequent interactions until a new constraint is drawn for that object. The only exception is that single axis rotation and single axis translation constraints can both be active at the same time; depending on how the user gestures — either mostly parallel to the translation axis or mostly perpendicular to the rotation axis — a translation or rotation operation, respectively, is chosen.

Finally, objects are removed from the scene by clicking on them with an interactor gesture. In early versions of SKETCH we used an apparently more natural gesture to remove objects: the user "tossed" them away by translating them with a quick throwing motion, as one might brush crumbs from a table. We found, however, that this gesture presented a complication: it was too easy to toss out the wrong object, especially if its screen size were small.

4.4 Grouping and copying

By default, objects are automatically unidirectionally grouped with the surface on which they were created, an idea borrowed from Bukowski and Sequin [6], generally resulting in hierarchical scenes. Each geometric object in SKETCH contains a list of objects that are grouped to it. Whenever an object is transformed, that object will also apply the same transformation to all other objects that are grouped to it; each grouped object will in turn transform all objects grouped to itself. Cycles can occur and are handled by allowing each object to move only once for each mouse motion event.

This kind of hierarchical scene is generally easier to manipulate than a scene without groupings since the grouping behavior typically corresponds to both expected and useful relationships among objects. For example, objects drawn on top of a table move whenever the table is manipulated, but when the objects are manipulated, the table does not follow. Grouping also applies to non-vertical relationships, so a picture frame drawn on a wall is grouped with the wall.

In some cases, grouping is bidirectional. The choice of bidirectional and uni-directional grouping is guided by what we be-

[5] Recall from Table 1 that Shift-modified strokes normally produce lines drawn on the surface of objects without special interpretation.

lieve is an inherent difference in the way people interpret relationships between certain horizontal versus vertical drawing elements. When an object is drawn that extends horizontally between two surfaces, like a rung on a ladder, the two surfaces that are spanned are grouped bidirectionally, so that if one rail of the ladder moves so does the other. Although the rung moves whenever either rail is manipulated, the rails do not move when the rung is manipulated. The grouping relationship for objects that span vertically, however, establishes only one-way relationships: the topmost object is unidirectionally grouped to the bottommost object and the spanning object is similarly grouped to the topmost object. Thus, a table leg that spans between a floor and a table top causes the top to be grouped to the floor and the leg to be grouped to the top, but the floor is *not* grouped to the top. We only exploit the difference between horizontal and vertical elements to distinguish these two grouping relationships. However we believe it is important to study with user tests how effective this automatic grouping approach actually is, and perhaps to determine as well if there are other ways that we might be able to exploit the differences between vertical and horizontal elements.

Unlike Bukowski, object grouping is not automatically recomputed as objects are moved around the scene. Therefore, if an object is moved away from a surface, it will still be grouped with the surface. Grouping relationships are recomputed only when objects are moved using the contact constraint mentioned in Section 4.3 — the moved object is grouped to the surface it now contacts and ungrouped from any surface it no longer contacts. We have found this approach to automatic grouping to be simple and effective, although in some environments, Bukowski's approach may be more appropriate.

Lassoing groups. SKETCH also allows the user to explicitly establish groups by stroking a *lasso* around them (Figure 2).[6] Deciding which objects are considered *inside* the lasso is based on the heuristic that the geometric center and all of the visible corners of the object must be inside the lasso; shapes like spheres must be completely contained in the lasso. SKETCH currently approximates this heuristic by first projecting an object's geometric center and all of its crease vertices (where there is a discontinuity in the derivative of the surface) and silhouette vertices into the film plane, then testing whether all these projected points are contained within the lasso. Currently, no test is made for whether objects are occluded or not; future work should address the ambiguities that arise in using the lasso operation.

All lassoed objects are copied if they are manipulated while the Shift modifier is used. Lassoed objects can be scaled by dragging the lasso itself.

Repeating gestures. A different form of copying is used when a user wants to repeat the last geometry-creation operation. After SKETCH recognizes gesture strokes corresponding to a geometric construction, it creates and places the new geometry in the scene, but *does not erase* the gesture strokes. Thus, the user can click on any of these strokes (using button 1) in order to "drag and drop" (re-execute) them elsewhere. Gesture strokes are erased when a new gesture is started or when any object is manipulated. These techniques are shown in the videotape.

5 Rendering

SKETCH renders orthographic views of 3D scenes using a conventional z-buffer. Color Plates I-VI show some of the rendering techniques that SKETCH supports.

"Sketchy" rendering styles are essential because they often enable users to focus on the essence of a problem rather than unimportant details. Non-photorealistic rendering draws a user's attention away from imperfections in the *approximate* scenes she creates while also increasing the scene's apparent complexity and ambiguity. By making scenes more ambiguous, users can get beyond SKETCH's approximate polygonal models to see what they want to see. This is an important concept: we do not believe that sketchy rendering adds noise to a signal; rather we believe that it conveys the very wide tolerance in the user's initial estimates of shape. The user is saying "I want a box *about* this long by *about* that high and *about* that deep." Showing a picture of a box with *exactly* those dimensions is misleading, because it hides the important information that the dimensions are not yet completely determined.

A line drawing effect is achieved by rendering all polygonal objects completely white, and then rendering the outlines and prominent edges of the scene geometry with multiple deliberately jittered lines; the z-buffer therefore handles hidden-line removal. A charcoal effect is created by mapping colors to grayscale and increasing the ambient light in the scene; a watercolor effect that washes out colors is created by increasing the scene's ambient light. There are a number of other techniques that we would like to explore, including pen and ink style textures, and drawing hidden edges with dashed lines.

Objects are assigned a default random color when they are created to help differentiate them from the scenery. We can also copy colors from one object to another. By just placing the cursor on top of one object and pressing the Shift modifier, we can "pick up" that object's color. Then, we can "drop" this color on another object by placing the cursor over it and releasing the modifier. We can also explicitly specify colors or textures for objects. In our present implementation, we do this by placing the cursor over the object and typing the name of the color or texture. Although this interface requires the keyboard, it is consistent with SKETCH's interface philosophy of not making users search through a 2D interface for tools to create particular effects. In the future, we expect that voice recognition, perhaps in conjunction with gesturing, will be a more effective way to establish surface properties for objects (and perhaps other operations as well).

6 Future Work

We regard SKETCH as a proof-of-concept application, but it has many flaws. Many of the gestures were based on an ad hoc trial and error approach, and some of the gestures still do not satisfy us. For example, the pause in the freehand curve gesture rapidly becomes annoying in practice — the user wants to do something, and is forced to wait. Possible solutions of course include using more modifier keys, although we would rather find a solution that preserves the simplicity of the interface.

SKETCH is based on an interface that is stretched to its limits. We expect that adding just a few more gestures will make the system hard to learn and hard to use. We'd like to perform user studies on ease of use, ease of learning, and expressive power for novice users as a function of the number of gestures. We're also interested in trying to determine to what extent artistic and spatial abilities influence users' preference for sketching over other modeling interfaces.

We have begun to implement a tablet-based version of SKETCH. The current generation of tablet pens include pressure sensitivity in addition to a single finger-controlled button, and one "eraser-like" button. In order to develop an equivalent interface for the tablet, we simply need to treat a specific pressure level as a button click to achieve the equivalent of three buttons. Therefore, the button 1 drawing interactions described for the mouse are done by simply pressing hard enough with the penpoint of the tablet pen. To achieve the button 2 operations of the mouse, the user simply

[6] The lasso gesture is similar to the sphere gesture. We differentiate between them by requiring that the sphere gesture be followed by a dot stroke, whereas the lasso is simply a free-hand closed curve stroke followed by a manipulation gesture.

presses the finger controlled button on the tablet pen. Finally, to effect camera motion, the user turns the pen over and uses its "eraser" to manipulate the camera. Our initial efforts with a tablet based interface lead us to believe that a tablet based system could be far more effective than a mouse based system, especially if pressure sensitivity is cleverly exploited.

SKETCH is a tool for initial design — the "doodling" stage, where things are deliberately imprecise. But initial design work should not be cast away, and we are examining ways to export models from SKETCH to modelers that support more precise editing, so that the sketch can be moved towards a final design. Since subsequent analysis and design often requires re-thinking of some initial choices, we are also interested in the far more difficult task of re-importing refined models into SKETCH and then re-editing them, without losing the high-precision information in the models except in the newly-sketched areas.

The scenes shown here and in the video are relatively simple. Will sketching still work in a complex or cluttered environments? We do not yet have enough experience to know. Perhaps gestures to indicate an "area of interest," which cause the remainder of the scene to become muted and un-touchable might help.

The tradeoffs in gesture design described in Section 4 must be further explored, especially with user-studies.

7 Acknowledgments

Thanks to Dan Robbins, Tim Miller, and Lee Markosian for many helpful discussions. Thanks also to Andries van Dam and the Graphics Group, as well as our sponsors: grants from NASA, NSF, Microsoft, Sun and Taco; hardware from SGI, HP and Sun.

References

[1] M. Akeo, H. Hashimoto, T. Kobayashi, and T. Shibusawa. Computer graphics system for reproducing three-dimensional shape from idea sketch. *Eurographics '94 Proceedings*, 13(3):477–488, 1994.

[2] Artifice, Inc. Design Workshop. Macintosh application.

[3] T. Baudel. A mark-based interaction paradigm for free-hand drawing. *UIST '94 Proceedings*, pages 185–192, Nov. 1994.

[4] E.A. Bier. Snap-dragging in three dimensions. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):193–204, Mar. 1990.

[5] V. Branco, A. Costa, and F.N. Ferriera. Sketching 3D models with 2D interaction devices. *Eurographics '94 Proceedings*, 13(3):489–502, 1994.

[6] R. Bukowski and C. Séquin. Object associations: A simple and practical approach to virtual 3D manipulation. *Computer Graphics (1995 Symposium on Interactive 3D Graphics)*, pages 131–138, Apr. 1995.

[7] J. Butterworth, A. Davidson, S. Hench, and T.M. Olano. 3DM: A three dimensional modeler using a head-mounted display. *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, 25(2):135–138, Mar. 1992.

[8] M. Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3-D rotation using 2-D control devices. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):121–129, August 1988.

[9] M. Clowes. On seeing things. *Artificial Intelligence*, (2):79–116, 1971. North-Holland.

[10] M. Deering. Holosketch: A virtual reality sketching/animation tool. *ACM Transactions on Computer-Human Interaction*, 2(3):220–238, 1995.

[11] T. Galyean and J. Hughes. Sculpting: An interactive volumetric modeling technique. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):267–274, July 1991.

[12] M. Gleicher. Integrating constraints and direct manipulation. *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, 25(2):171–174, March 1992.

[13] K.P. Herndon, R.C. Zeleznik, D.C. Robbins, D.B. Conner, S.S. Snibbe, and A.van Dam. Interactive shadows. *UIST '92 Proceedings*, pages 1–6, Nov. 1992.

[14] D. Kurlander and S. Feiner. Inferring constraints from multiple snapshots. *ACM Transactions on Graphics*, 12(4):277–304, Oct. 1993.

[15] D. Lamb and A. Bandopadhay. Interpreting a 3D object from a rough 2D line drawing. *Visualization '90 Proceedings*, pages 59–66, 1990.

[16] J.A. Landay and B.A. Myers. Interactive sketching for the early stages of user interface design. *Proceedings of CHI'95*, pages 43–50, 1995.

[17] J. Lansdown and S. Schofield. Expressive rendering: A review of nonphotorealistic techniques. *IEEE Computer Graphics & Applications*, pages 29–37, May 1995.

[18] J.D. Mackinlay, S.K. Card, and G.G. Robertson. Rapid controlled movement through a virtual 3d workspace. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, pages 171–176, October 1986.

[19] G. Magnan. *Using technical art: An industry guide*. John Wiley and Sons, Inc., 1970.

[20] D. Pugh. Designing solid objects using interactive sketch interpretation. *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, 25(2):117–126, Mar. 1992.

[21] D. Rubine. Specifying gestures by example. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):329–337, July 1991.

[22] E. Sachs, A. Roberts, and D. Stoops. 3-draw: A tool for designing 3D shapes. *IEEE Computer Graphics & Applications*, pages 18–25, Nov. 1991.

[23] M. Salisbury, S. Anderson, R. Barzel, and D. Salesin. Interactive pen–and–ink illustration. *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 101–108, July 1994.

[24] S. Sistare. Graphical interaction techniques in constraint-based geometric modeling. *Proceedings of Graphics Interface '91*, pages 85–92, June 1991.

[25] S.S. Snibbe, K.P. Herndon, D.C. Robbins, D.B. Conner, and A. van Dam. Using deformations to explore 3D widget design. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):351–352, July 1992.

[26] J.M. Snyder and J.T. Kajiya. Generative modeling: A symbolic system for geometric modeling. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):369–378, July 1992.

[27] P. Strauss and R. Carey. An object-oriented 3D graphics toolkit. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):341–349, July 1992.

[28] W. Wang and G. Grinstein. A survey of 3D solid reconstruction from 2D projection line drawings. *Computer Graphics Forum*, 12(2):137–158, June 1993.

[29] L.R. Wanger, J.A. Ferwerda, and D.P. Greenberg. Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications*, 12(3):44–58, May 1992.