A Suggestive Interface for 3D Drawing

Takeo Igarashi *John F. Hughes*
Computer Science Department, Brown University
115 Waterman Street, Providence, RI 02912, USA
takeo@acm.org, jfh@cs.brown.edu

ABSTRACT

This paper introduces a new type of interface for 3D drawings that improves the usability of gestural interfaces and augments typical command-based modeling systems. In our suggestive interface, the user gives hints about a desired operation to the system by highlighting related geometric components in the scene. The system then infers possible operations based on the hints and presents the results of these operations as small thumbnails. The user completes the editing operation simply by clicking on the desired thumbnail. The hinting mechanism lets the user specify geometric relations among graphical components in the scene, and the multiple thumbnail suggestions make it possible to define many operations with relatively few distinct hint patterns. The suggestive interface system is implemented as a set of suggestion engines working in parallel, and is easily extended by adding customized engines. Our prototype 3D drawing system, Chateau, shows that a suggestive interface can effectively support construction of various 3D drawings.

KEYWORDS: interaction technique, user interface design, 3D drawing, prediction, gestural interface.

INTRODUCTION

Typical 3D modeling tools are designed for precise control of complicated shapes, and the interfaces are generally hard for casual users to learn. To provide simplified interfaces for sketching 3D structures quickly, various gestural interfaces have been explored [12,25]. These let the user interact directly with 3D scenes without using buttons and menus and reduce the explicit control required by implementing various context-dependent rules.

Although gestural interfaces for 3D modeling have been successful as an experimental effort, they still have several limitations. First, they have been designed primarily for building approximate models rather than for the precise control used in traditional 3D modeling tools, and the realm between these, i.e., approximate modeling of objects with important symmetries and repeated substructures, has been largely unexplored. This makes it hard to sketch many interesting architectural forms, for example.[1] Second, they do not scale well because a system designer cannot define many gestures with limited combinations of gestural elements (stroke, click, modifier key, etc.). Third, it is difficult for novice users to learn a set of gestures because the user must complete a gesture to see the result, and must start over if it fails.

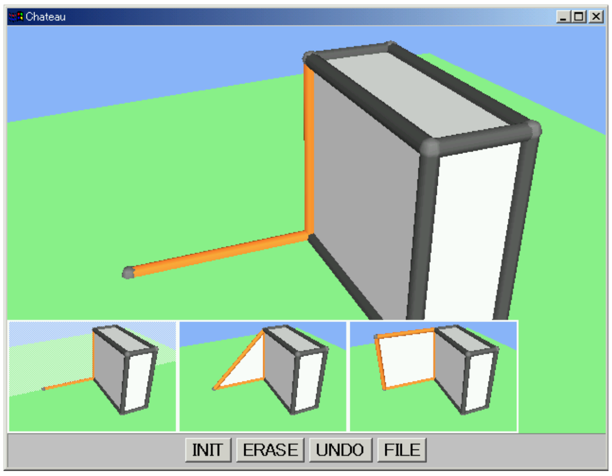

Figure 1: A screen snapshot of our prototype system Chateau. The user gives the hints to the system by highlighting related lines (red lines), and the system suggests possible operations (thumbnails at bottom) based on the hints.

This paper introduces a new type of interface that extends gestural interfaces to address these limitations. In the proposed *suggestive interface,* the user gives the system *hints* about the desired operation by highlighting related components in the scene, and the system *suggests* subsequent operations in an array of small thumbnails derived from the hints and the overall scene configuration (Figure 1). The user can complete an operation by choosing

[1] It was an interest in sketching French chateaus that originally motivated this work.

one of these suggestions, or can ignore them and continue constructing and/or hinting. Suggestions are generated by *suggestion engines,* each of which constantly observes the scene and generates a suggestion when the current hint configuration matches its *input pattern*.

A suggestive interface can be viewed as a mediated version of a gestural interface. Instead of responding to the user's input by updating the scene immediately, the system asks for the user's confirmation after showing multiple suggestions. This approach has several advantages over earlier gestural interfaces. First, the hinting mechanism lets us use existing components as input. This naturally helps in the specification of geometric relations among components in the scene. Second, because suggestions are merely *offered,* a single collection of hints can serve both as a gesture and as a *subset* of a more complex gesture; e.g., the new-drawing-plane engine responds to a single selected line, while the rectangle-creation engine responds to two connected perpendicular selected lines. Third, the suggestive interface helps the learning process because users can progressively refine their hints until the desired result appears among the suggestions. The suggestions themselves, even if not taken, may be helpful in the creative process.

In this paper we describe *Chateau*, a simple proof-of-concept 3D modeling tool we developed to explore the suggestive interface idea. Our experience shows that it is quite promising. Our initial concern was that too many suggestions might be generated, confusing the user. However, if the system is carefully designed so that most suggestion engines have mutually exclusive input patterns, users see only a few suggestions at a time and can control the system fluently. Our informal test users understood the interface quickly and created various 3D drawings successfully.

Although our original goal was to improve gestural interfaces, we believe suggestive interfaces can also be useful in augmenting traditional command-based interfaces for 3D modeling. Hinting and suggestions encourage novice users to explore a new system and find unknown operations, and some operations that require combinations of commands can be specified naturally by only a few hints.

A demonstration video and the prototype program are available at www.cs.brown.edu/research/chateau.html or www.mtl.t.u-tokyo.ac.jp/~takeo/.

RELATED WORK

Many researchers are exploring possible next-generation user interfaces beyond current WIMP-style GUIs [23]. A common observation is that next-generation user interfaces should be context-aware in order to reduce the number of explicit command operations required [21]. This paper reports our experimental effort to implement context-awareness in the domain of 3D modeling.

Our interface is similar to predictive interfaces [5][19] in that the system (or an agent) suggests possible subsequent operations, but prior efforts have focused on operation histories to facilitate repetitive operations, while our system suggests various predefined operations based on the static configuration of the user-provided hints.

Multiple candidates are commonly used in recognition-based systems such as handwriting or speech recognition to solve the inherent ambiguity problem [16]. Japanese text-entry interfaces rely on multiple candidates to input thousands of characters using a limited number of keys [18]. In computer graphics, multiple candidates have been used to find desired parameter settings in a large parameter space [22], most recently in the Design Galleries work [17]. Typically, however, galleries represent samples of a large continuous space of possibilities, while our suggestions work with a small discrete space of possibilities.

Some constraint-based drawing systems infer geometric constraints from the user's operations. Briar [7] and ROCKIT [13] infer graphical constraints based on a user's dragging operation, and allow the user to select from several candidate constraints. Hudson and Hsi presented a system that infers layout algorithms by generalizing examples provided by the user [9]. The system presents multiple candidates for the generalization and lets the user select the desired one. While these systems infer hidden relationships or rules in a programming-by-example manner [6], our system constructs static scenes using a simple pattern-matching method [14].

Suggestive user interfaces extend the notions of beautification and prediction introduced in the Pegasus system [10,11]. Pegasus beautifies hand-drawn strokes by inferring desired geometric relationships, and predicts the next operation based on the surrounding context. It also generates multiple candidates to facilitate these processes. One problem is that too many candidates are offered as the scene becomes complicated. We address this "candidate explosion" by introducing an explicit hinting mechanism. To prevent clutter, we also primarily use visual thumbnails instead of presenting candidates in the scene.

Gesture-based interfaces, frequently used in 2D pen-based applications [8,15,20], recognize specific stroke shapes as gestures and replace them with predefined primitives or invoke editing operations such as undo. The SKETCH system [25] introduced a gesture-based interface for making 3D scenes consisting of stacked geometric primitives. Teddy [12] used a gesture-based interface for freeform 3D modeling. Our goal is to extend these systems to increase scalability and to support geometric relations such as symmetry and congruence.

THE USER INTERFACE

The user constructs 3D scenes by drawing 2D lines on the screen. The system converts 2D lines on the screen into 3D lines by projecting them onto 3D elements already in the scene. Prediction and suggestion mechanisms facilitate this drawing process by inferring possible subsequent operations. Highlighting plays an essential role throughout; highlighted lines guide the snapping mechanism for drawing lines and provide hints for prediction and suggestions. This section introduces the basics of the modeling system and then describes the prediction and suggestion mechanisms in detail.

A First Example

Suppose that a user wants to create two adjoining walls of a room, i.e., the model shown in Figure 2h. We'll briefly describe WHAT she does and her intention at each stage (i.e., WHY), and then, in the following sections, give further details and examples.

At the start of a modeling session, the user sees a ground plane. She wants to create a wall that meets this plane, so she draws a line segment on the plane to begin with: she clicks at some point, drags to the right, and releases. This creates a segment on the ground and automatically highlights it. The single highlighted line causes a candidate operation to be offered: the system offers to create a drawing-plane that's perpendicular to the ground and passes through the line (Figure 2a). Because the user wants to draw a wall in just such a plane, she clicks on the candidate and the transparent drawing plane appears (Figure 2b). Now she again clicks on the same starting point, drags a line upwards on the screen and releases, which creates a second line perpendicular to the first and highlights it as well; because both segments are highlighted, the system offers a candidate operation — the creation of a rectangle in the drawing plane (Figure 2c). This candidate is ideal, so she clicks on the thumbnail to make it happen (Figure 2d). She now wants to draw a new "aseline" on the ground plane, so she first clicks on the background to unhighlight all lines (Figure 2e). She then clicks on the ground plane some distance in front of the first click point and drags back towards it and releases the mouse over it. A new line appears and is highlighted (Figure 2f). Finally, highlighting (by clicking) the first vertical line she drew makes the system offer a rectangle in the new drawing plane as a candidate (Figure 2g), which she selects by clicking on the thumbnail, resulting in the model shown in Figure 2h.

Thus the basic operations are "dragging out lines segments," clicking on things to highlight/unhighlight them, and clicking on thumbnail "candidates" to choose them.

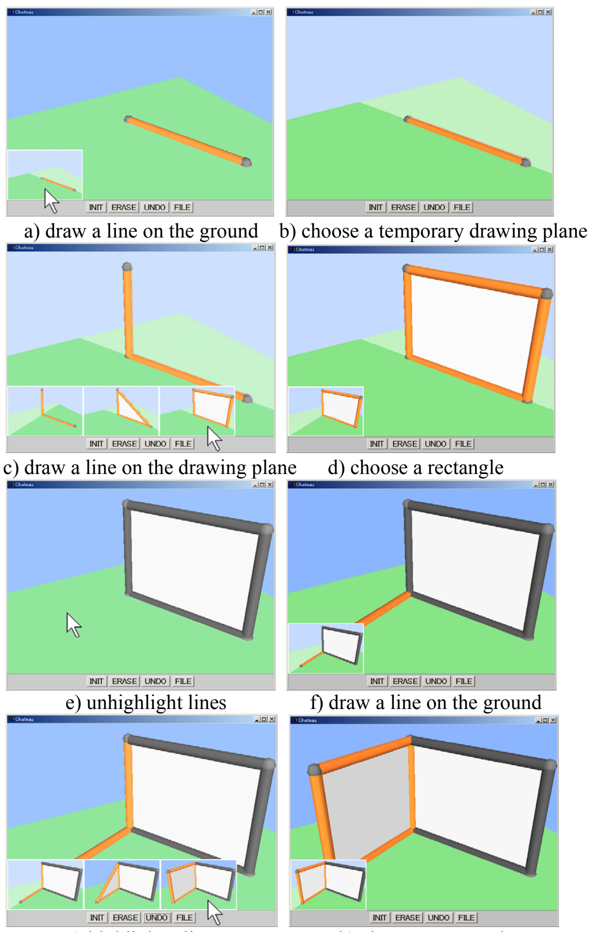
a) draw a line on the ground
b) choose a temporary drawing plane
c) draw a line on the drawing plane
d) choose a rectangle
e) unhighlight lines
f) draw a line on the ground
g) highlight a line
h) choose a rectangle
INIT
ERASE
UNDO
FILE

Figure 2: A first example.

Basics

Chateau currently supports the construction of 3D scenes consisting of straight line segments and planar polygons (curves and circles are not yet supported). Each line segment (called a *line*) is defined by two terminal vertices (called *joints*). Polygons (called *plates*) are always surrounded by lines. The ground plane is always visible and the user begins construction of every model by drawing a line on the ground.

All modeling operations are effected by left-mouse-button clicks and drags in the main screen. The right mouse button is reserved for camera control, for which we use the UniCam interface [24]. Only a few GUI buttons (clear, erase, undo) are provided on the screen. Our system requires no keyboard operation, and hence supports one-handed operation on devices like hand-held notepads.

Highlighting plays an essential role in our system: the user controls snapping, prediction, and suggestion by highlighting appropriate lines as hints. The user highlights a line on the screen by clicking on it. If the user clicks an already highlighted line, it is unhighlighted. When the user double-clicks a line, the system highlights all lines connected to it. Any newly drawn lines are automatically highlighted. The user can unhighlight all lines by clicking on the ground or the background. When the user clicks on a plate, the system highlights all its edges.

The user draws a new line on a plate or the ground plane in the 3D scene by a dragging operation. To be precise, the system first finds the foremost plate or plane under the mouse cursor at the beginning of dragging, and projects the line on the plate or plane. The end points snap to existing lines and their end points [2] on the plate or plane. We also implemented a “drafting assistant” mechanism [1] whereby the user can activate additional snapping constraints by touching a line during the dragging operation. For example, if the user touches the midpoint of a line, the mouse cursor starts to snap to the perpendicular bisector of the line. Furthermore, snapping is affected by the highlighted lines; it guides the user to draw lines that are parallel or congruent[2] to the highlighted lines. In addition to the visible plates and the ground plane, the user can draw a line on a *temporary drawing plane*, so that lines can be drawn floating in the air [3]. A temporary drawing plane is activated by the suggestive interface mechanism described later, and appears as an translucent plane in the display. The user erases a line or plate by a scribbling gesture (moving the mouse cursor back and forth while dragging). The “erase” button on the screen erases all highlighted lines at once.

Predictions

A prediction mechanism like that in the Pegasus system [11] predicts the next lines to be drawn around the most recently highlighted line and presents multiple candidates as purple lines in the 3D scene. (This can be seen as a very specialized version of suggestion; its rules are so simple and it’s so often applicable that its candidates are shown in the 3D scene rather than as thumbnails.) While Pegasus uses all lines in the scene as the context information for prediction, Chateau uses only the highlighted lines, which significantly reduces the number of candidates generated. Specifically, Chateau generates the flipped duplications of the highlighted lines connected to the most recently highlighted line (Figure 3a-c). It also searches for a reference line that is congruent to the most recently highlighted line, and copies the lines connected to the reference line around the most recently highlighted line (Figure 3d-f). The user can click a candidate to adopt it or simply proceed to the next operation to ignore the prediction. This prediction mechanism helps users draw locally symmetric or congruent structures. Prediction and suggestion are always active, but for clarity we suppress prediction in the remaining figures.

[2] Here, congruence means translational congruence and does not include rotational congruence.

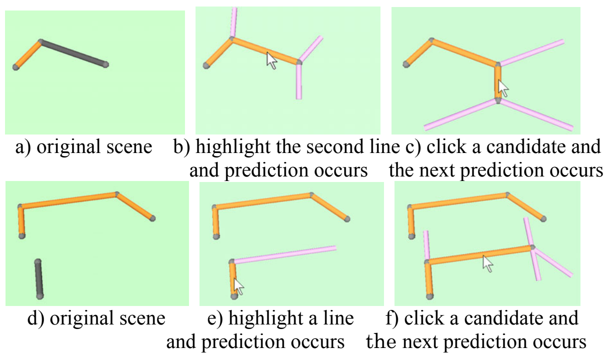
a) original scene
b) highlight the second line and prediction occurs
c) click a candidate and the next prediction occurs
d) original scene
e) highlight a line and prediction occurs
f) click a candidate and the next prediction occurs

Figure 3: The prediction mechanism.

Suggestions

Chateau generates suggestions whenever the user adds, erases, highlights, or unhighlights a line. The system automatically infers possible next operations based on the configuration of the highlighted lines, and presents the results of the operations as an array of thumbnails (Figure 1). The user can either ignore these or adopt one by clicking the thumbnail. The user can also “preview” the result as a large image in the main screen by dragging the mouse cursor across the thumbnails. The operation is finalized when the user releases the mouse button over the desired thumbnail.

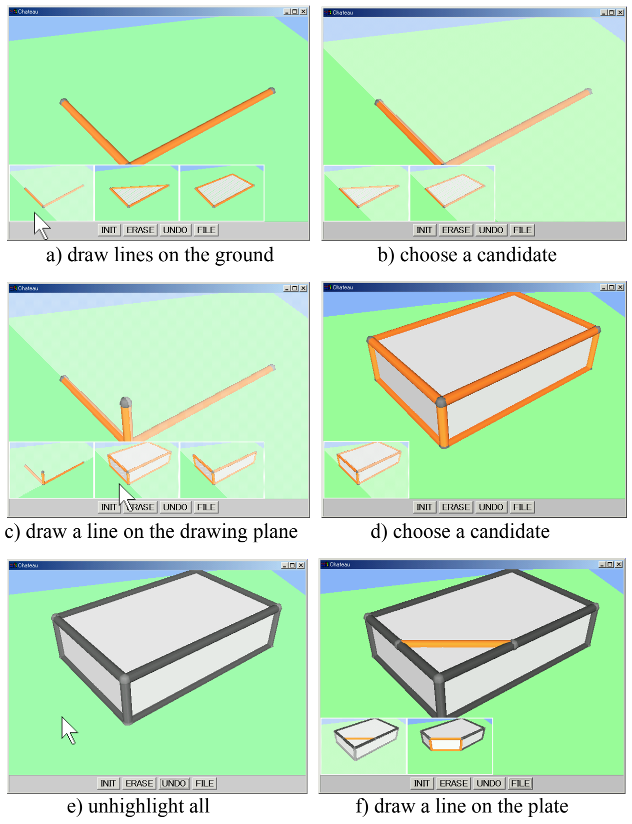
Chateau
INIT ERASE UNDO FILE
a) draw lines on the ground
b) choose a candidate
c) draw a line on the drawing plane
d) choose a candidate
e) unhighlight all
f) draw a line on the plate

Figure 4: Example operation sequence.

Figure 4 shows an example operation sequence. The user first draws two lines on the ground and the system presents three suggestions (a). Then she chooses the leftmost suggestion, which creates a new drawing plane (b). She draws the third line on the drawing plane and the system presents three new suggestions (c). She chooses to make a box (d). She unhighlights everything by clicking on the ground (e). She draws a line on the box, and the system shows two candidates (f), including one that suggests chamfering.

Candidates are generated by a set of *suggestion engines*. Each engine observes the scene, and when the current scene configuration matches its input test pattern it returns the updated scene as a candidate (Figure 5). The current implementation duplicates the entire scene for each candidate instead of maintaining a progressive data structure. The behavior of an individual suggestion engine can be seen as a variation of the constraint-based search-and-replace operation in the Chimera system [14], but our engines focus only on the highlighted lines instead of pattern-matching against the entire scene. When a suggestion is created, a thumbnail is rendered as an offscreen image, using the same camera parameters (i.e., view) as in the main window. For efficiency, we use fixed bitmaps for the thumbnails, which therefore do not update as the main-window view is changed.

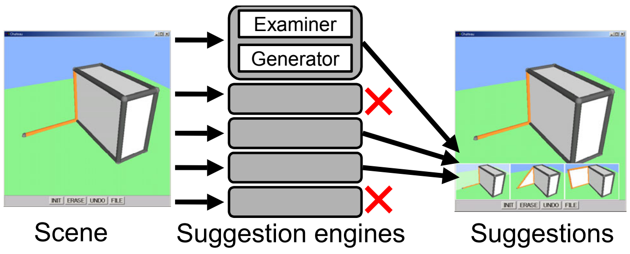

Figure 5: Suggestion engines observe the scene and return candidates when the scene matches their input patterns.

Figure 6 shows our current list of engines, S1 to S20. The first two suggestion engines support fundamental operations. S1 creates a temporary drawing plane to let the user draw lines in the air. If the most recently highlighted two lines are on a single plane, the system offers it as the next drawing plane. If not, the system offers a plane that contains the last-highlighted line and is perpendicular to the current drawing plane. S1 always returns a suggestion unless the resulting plane is identical to the ground plane. S2 creates a plate in a planar loop of highlighted lines.

All modeling operations can be achieved using just the basic drawing operations and the two engines just described. All the other suggestions can be seen as “assistants” that facilitate typical modeling tasks. For example, instead of using S4, the user *could* draw a box by drawing 12 lines and making 6 plates manually. We briefly describe the behavior of the suggestion engines to supplement the visual description in Figure 6.

S3 and S4 respond to two/three highlighted lines connected perpendicularly to one another. S5 and S6 respond to a highlighted line that is perpendicular to the plane containing all other highlighted lines; S6 responds only when the remaining lines form a loop, in which case the top vertex is positioned over the loop’s center. S7 responds when the last-highlighted line overlaps a line in the highlighted group. (A group is a set of highlighted lines and plates connected to one another.) S8 responds to two sets of highlighted lines when each set lies on a plane and each line in a set has a parallel partner in the other set. S9 responds when the extrusions from the planar highlighted lines hit an existing plate (this is useful, for example, in making the legs of a table). S10 and S11 respond to highlighted lines that touch the edges of a polyhedron. Specifically, S10 requires that the two edges touched by the highlighted line share a vertex and that the vertex be shared by three plates. There must also be another plate at the opposite side. S11 requires that the highlighted lines form a planar loop and that all highlighted lines be on plates surrounding a corner. S12 responds to two parallel highlighted lines, of which one is an edge of a plate and the other touches the edges next to it. S13 responds to two intersecting lines (this is useful for trimming operations). S14 responds when the last-highlighted line is isolated from the highlighted group and is congruent with a line in the group. S15 is similar, but responds when the highlighted line is the mirror copy of the corresponding line. S16 responds when two congruent groups are highlighted, and therefore appears whenever the user has adopted an S14 suggestion. S17 responds to sequences of parallel lines such that the gaps between corresponding segments are nearly equal. S18 responds when three congruent groups or lines are linearly aligned. It generates equally spaced copies of the group between the external two as hinted by the middle one. S19 responds to irregular “stairs” (a repeated sequence of mutually perpendicular lines). S20 responds to three lines of equal length sharing a vertex when two of the joint angles are equal. This engine is useful in drawing regular polygons.

The particular choice of engines was determined by our needs as we experimented with the system and is clearly application-dependent. In a plumbing application, for example, it would be natural to have engines that created standard junctions (tees, unions, couplers, elbows, etc.).

In the current implementation, engines require exact matching in the examination phase. For example, S4 requires that all three edges to be exactly perpendicular and S17 requires that the pairs be exactly parallel. Alternatively, one can allow small deviations and *beautify* them after the operation [14]. We did not adopt this scheme in order to

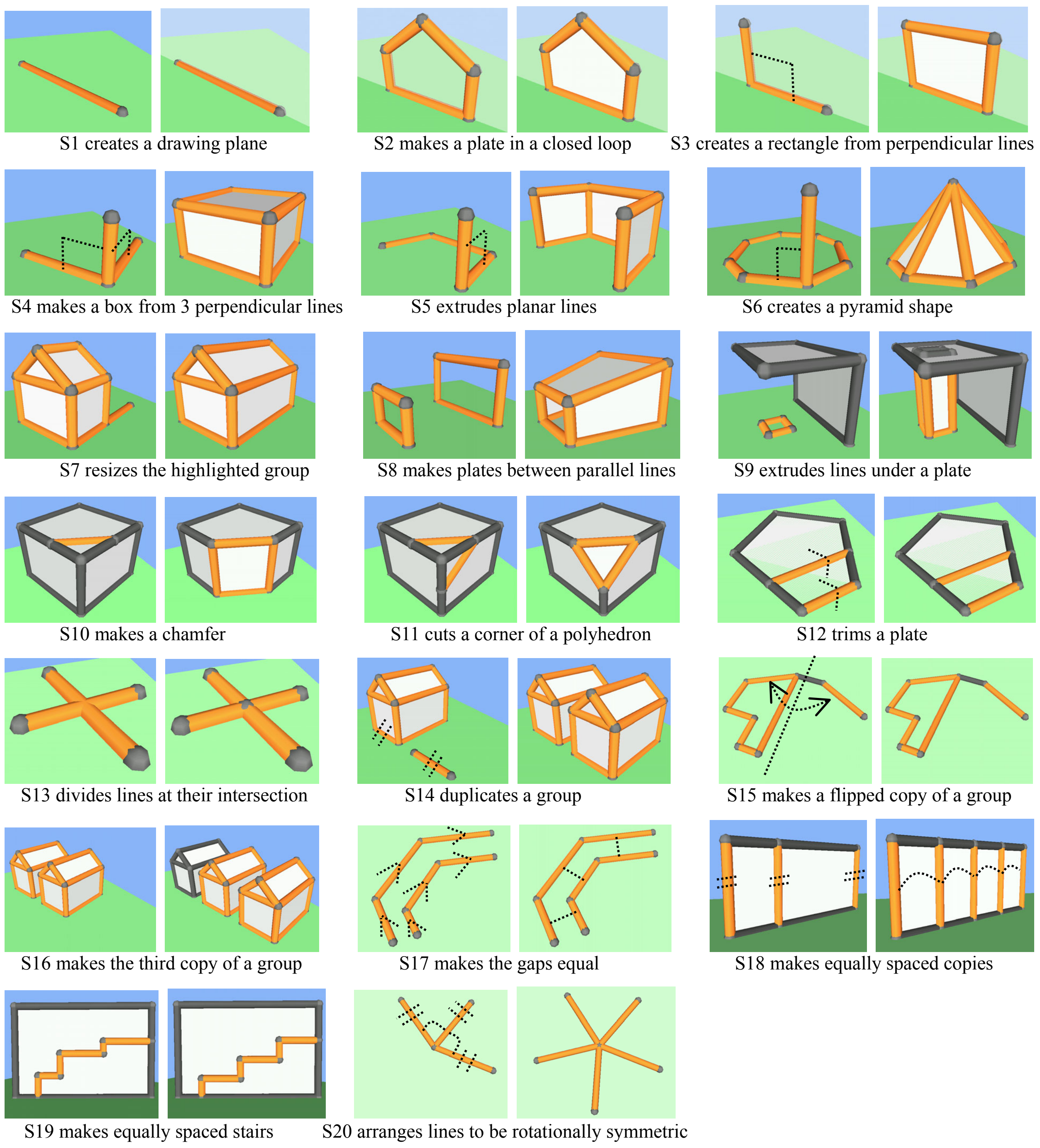
S1 creates a drawing plane
S2 makes a plate in a closed loop
S3 creates a rectangle from perpendicular lines
S4 makes a box from 3 perpendicular lines
S5 extrudes planar lines
S6 creates a pyramid shape
S7 resizes the highlighted group
S8 makes plates between parallel lines
S9 extrudes lines under a plate
S10 makes a chamfer
S11 cuts a corner of a polyhedron
S12 trims a plate
S13 divides lines at their intersection
S14 duplicates a group
S15 makes a flipped copy of a group
S16 makes the third copy of a group
S17 makes the gaps equal
S18 makes equally spaced copies
S19 makes equally spaced stairs
S20 arranges lines to be rotationally symmetric

Figure 6: Complete list of suggestions implemented in the current prototype (left: hints, right: result). (The dotted lines are added for clarity; they do not appear in the actual system.)

clearly distinguish the role of snapping/prediction and suggestions. Our design principle is to use snapping and prediction for satisfying basic relations such as congruence and parallelism, and to use suggestions for completing construction tasks. Another reason is that small deviations in the hints can make the result of suggestions ambiguous. For example, in the case of S4, the system has three options for positioning the resulting box if the three lines are not exactly perpendicular each other, and thus must ask the user to choose one among them.

To investigate the capability of a *pure* suggestive interface, we intentionally excluded traditional editing operations such as translation, rotation, and duplication. However, it is natural and useful to provide both command-based and suggestion-based operations in a single system. We envision that in practical applications, suggestive user interfaces will *augment* command-based interfaces.

IMPLEMENTATION

The Chateau system is implemented in Java (JDK1.1.5), and uses directX3 for 3D rendering. Suggestion engines (Java class files) are implemented as listeners that respond to changes in the scene configuration. An engine has an input examination part that determines whether it responds to the scene, and a suggestion generator that edits a copy of the scene to construct an updated scene. When the current scene matches an engine's input pattern, the engine returns the updated scene object and a thumbnail image (Figure 5). The implementation of suggestion engines is relatively simple because standard routines are provided by the base system. In the examination part, an engine checks the scene based using such criteria as the number of highlighted lines, connectivity, and spatial interrelationships. A typical suggestion engine's source code is between 100 and 200 lines.

It is essential to design suggestion engines carefully so that their input conditions are as mutually exclusive as possible. If many suggestion engines match a single scene configuration, they will generate many suggestions, confusing the user and cluttering the screen. With our current choice of engines, the system generates only a few suggestions at a time, showing that careful design can help prevent suggestion explosion. In the future, we will investigate the feasibility of the interface with many more suggestion engines.

USER EXPERIENCES

We have started an informal user study using the prototype system. Figure 7 shows examples of 3D models created by our test users, all of whom are graduate students in computer science. They learned the behavior of the system in approximately 30 minutes of tutorial and practice and created various models, including those shown in Figure 7, within a few hours. Test users were generally satisfied with the interface, but they wanted simple direct manipulation functions such as "move" and "rotate." Because of the limitations of the current implementation, the system gets too slow when the model becomes more complicated than these examples.

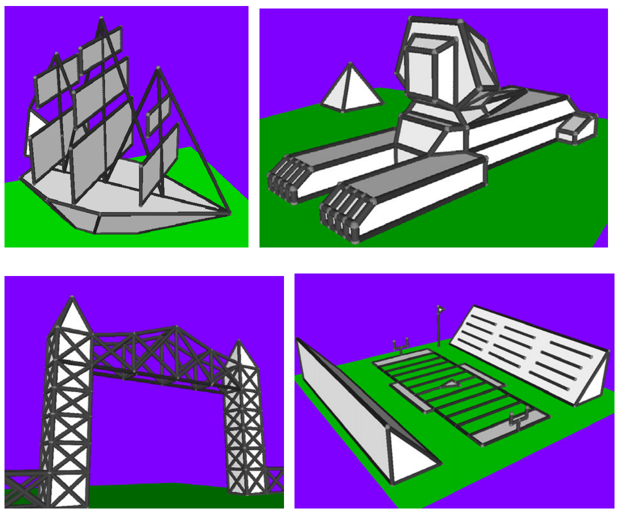
Figure 7: 3D drawings created by test users using Chateau.

We also asked students in an advanced computer graphics class (including both graduate and undergraduate students) to test the prototype system and to implement their own suggestion engines as a part of an assignment. In general, they found the idea of a suggestive interface attractive and useful, but also felt that the current implementation requires substantial improvements. They wanted a better interface for controlling temporary drawing planes, appropriate feedback for camera control and snapping, the ability to turn off/on each feature, keyboard shortcuts for frequent operations, and command-based direct manipulation or 3D widgets for translation and rotation. This result suggests that a *pure* suggestive interface is not very practical, and that suggestion may be most effective when combined with traditional interfaces. We also asked them to list suggestion engines that they evaluated positively (useful) and negatively (useless or difficult to use). Table 1 summarizes the results. The basic engines (S1-S6) were popular, but other engines received mixed reactions reflecting large diversity in personal preferences.

Table 1: Subjective evaluation of suggestion engines. The table shows the number of subjects who evaluated each suggestion positively or negatively. Six subjects provided answers.

S	1	2	3	4	5	6	7	8	9
positive	3	5	6	5	5	3	1	1	0
negative	0	0	0	0	0	1	2	3	5

10	11	12	13	14	15	16	17	18	19	20
3	3	3	3	2	1	2	0	4	0	1
3	2	1	1	2	3	2	3	1	2	4

Figure 8 shows some suggestion engines implemented by the students. S21 takes a structure on the ground plane and a vertical line, and hoists the structure. S22 takes two closed loops that are not parallel, and makes a *tube* between the loops. S23 takes connected lines and returns a spline curve. S24 takes three lines in Y shape and fractalizes the Y. Overall, students implemented their own suggestion engines successfully, showing that one can extend the system as desired with reasonable effort.

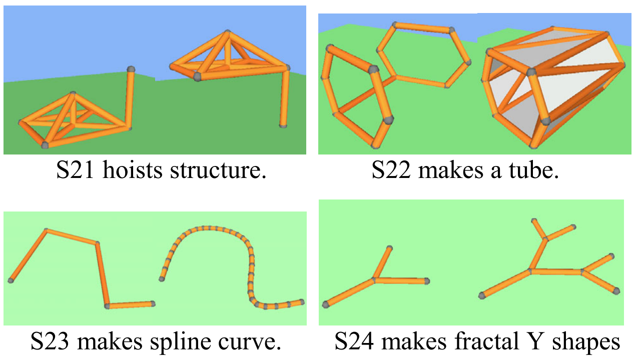
S21 hoists structure. S22 makes a tube. S23 makes spline curve. S24 makes fractal Y shapes

Figure 8: Examples of suggestion engines implemented by test users.

LIMITATIONS AND FUTURE WORK

Suggestive interfaces have some drawbacks: they can help promote serendipitous discovery of available operations, but they give a user no way to discover all operations directly, as "browsing the menus" can in a WIMP interface. If the hints given are inadequate, the system never responds and it is unclear to the user why the system is failing. A visual summary of suggestions, such as shown in Figure 6, is necessary for learning and reference. For operations with continuous parameters (e.g., shearing), there is no opportunity for partial feedback (such as a highlighted bounding box or parallelogram) during parameter adjustment. These operations may be best supported by a traditional direct-manipulation approach such as 3D widgets [4].

As with any experimental interaction technique, scalability is a major concern with the suggestive interface. One scalability problem is the complexity of the 3D scene. Although the hinting mechanism effectively limits the number of candidate suggestions compared with the simple search-entire-scene approach [11][14], complicated 3D scenes can make it difficult to specify hints and to find the desired one among small thumbnails. We need some advanced mechanisms such as grouping and locking to deal with complicated scenes.

Another scalability issue is related to the number of engines. The current suggestive interface system may not work well when hundreds of engines are implemented since the system may generate too many suggestions and confuse the user. We need refined mechanisms that automatically suppress inadequate engines based on the user's preferences, or let the user manually activate/inactivate specific engines. We also need to provide traditional command-based interfaces to perform complicated tasks.

The order of suggestion presentation is fixed in the current implementation: it is determined by the order in which the suggestion engines are implemented in the system, so S1 always appears first, S2 (if appropriate) second, and so on. We could instead first display recently used suggestions, or sort the suggestions based on the current context, or organize suggestions into a hierarchy. The value of such approaches will have to be determined through careful user studies.

In the near future, we will extend the current interface to support circles and curves. We plan to implement suggestion engines that construct cylinders, revolved surfaces, and rounded corners. In the longer term, we hope to use a suggestive user interface in a sketch-based freeform modeling system [12].

One advantage of the suggestive interface is extensibility. Users can customize the interface by adding their own special-purpose engines to the system. In the current implementation the user must write Java code, but we hope to provide an end-user programming environment, possibly an example-based framework [6].

Suggestive interfaces can be useful in various other graphical applications such as 2D bitmap editors and graph drawing programs. For example, if the user highlights almost-aligned objects in a 2D drawing program, the system might suggest an aligning operation, and it would be natural in a graph-drawing program to support even spacing of nodes or replication of selected subgraphs. Indeed, we believe that the ease of describing useful suggestions for a variety of applications indicates the promise of suggestive interfaces.

ACKNOWLEDGMENTS

We would like to thank the Brown University computer graphics group, especially Bob Zeleznik and Andy Forsberg, for thoughtful discussions, and the CMU stage3 research group, especially Dennis Cosgrove, for allowing us to use their Jalice scenegraph. We also thank Brown CS224 students for testing the system and providing valuable feedback.

REFERENCES

1. Ashlar Vellum Products, Ashlar Inc., http://www.ashlar.com/
2. E.A. Bier and M.C. Stone. Snap Dragging. *Computer Graphics*, Vol. 20, No. 4, pp. 233-240, 1986.
3. J.M. Cohen, L. Markosian, R.C. Zeleznik, J.F. Hughes, and R. Barzel. An Interface for Sketching 3D Curves.

1999 Symposium on Interactive 3D Graphics, pp. 17-21, 1999.

4. D.B. Conner, S.S. Snibbe, K.P. Herndon, D.C. Robbins, R.C. Zeleznik, and A. van Dam. Three-Dimensional Widgets. *1992 Symposium on Interactive 3D Graphics*, pp. 183-188, 1992.

5. A. Cypher. Eager: Programming Repetitive Tasks by Example. *Proceedings of CHI'91,* pp.33-39, 1991.

6. A. Cypher. *Watch What I Do: Programming by Demonstration*. Cambridge, MA: MIT Press. 1993.

7. M. Gleicher and A. Witkin. Drawing with constraints. *The Visual Computer*, Vol. 11, No. 1, pp. 39-51, 1994.

8. M.D. Gross and E.Y.L. Do. Ambiguous Intentions: A Paper-like Interface for Creative Design. *Proceedings of UIST'96*, pp. 183-192, 1996.

9. S. Hudson and C. Hsi. A Synergistic Approach to Specifying Simple Number Independent Layouts by Example, *Proceedings of INTERCHI'93*, pp. 285-292, 1993.

10. T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka. Interactive Beautification: A Technique for Rapid Geometric Design. *Proceedings of UIST'97*, pp. 105-114, 1997.

11. T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka. Pegasus: A Drawing System for Rapid Geometric Design. *CHI'98 Summary*, pp. 24-25, 1998.

12. T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. *SIGGRAPH 99 Conference Proceedings*, pp. 409-416, 1999.

13. S. Karsenty, J.A. Landay, and C. Weikart. Inferring Graphical Constraints with Rockit, *Proceedings of HCI'92*, 1992.

14. D. Kurlander and S. Feiner. Interactive Constraint-Based Search and Replace. *Proceedings of CHI'92*, pp. 609-618, 1992.

15. J.A. Landay and B.A. Myers. Interactive Sketching for the Early Stages of User Interface Design. *Proceedings of CHI'95*, pp. 43-50, 1995.

16. J. Mankoff, S.E. Hudson and G.D. Abowd. Interaction Techniques for Ambiguity Resolution in Recognition-based Interfaces. *Proceedings of UIST'00*, pp. 11-20, 2000.

17. J. Marks, B. Andalman, P. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation. *SIGGRAPH 97 Conference Proceedings*, pp. 389-400, 1997.

18. T. Masui. An Efficient Text Input Method for Pen-based Computers. *Proceedings of CHI'98*, pp. 328-335, 1998.

19. D. Maulsby, I.H. Witten and K.A. Kittlitz. Metamouse: Specifying Graphical Procedures by Example. *Proceedings SIGGRAPH'89*, pp. 127-136, 1989.

20. T.P. Moran, P. Chiu, W. van Melle, and G. Kurtenbach. Pen-based Interaction Techniques for Organizing Material on an Electronic Whiteboard. *Proceedings of UIST'97*, pp. 45-54, 1997.

21. J. Nielsen. Noncommand User Interfaces. *Communications of the ACM*, Vol. 36, No. 4, pp. 83-99, 1993.

22. K. Sims. Artificial Evolution for Computer Graphics. *SIGGRAPH 91 Conference Proceedings*, pp. 319-328, 1991.

23. A. van Dam. Post-WIMP User Interfaces, *Communications of the ACM*, Vol. 40, No. 2, pp. 63-67, 1997.

24. R.C. Zeleznik and A. Forsberg. UniCam — 2D Gestural Camera Controls for 3D Environments. *Proceedings of 1999 Symposium on Interactive 3D Graphics*, 1999.

25. R.C. Zeleznik, K.P. Herndon, and J.F. Hughes. SKETCH: An Interface for Sketching 3D Scenes. *SIGGRAPH 96 Conference Proceedings*, pp. 163-170, 1996.