User Interface Design and Realization of a Design-by-Sketches System

Amit Shesh *Baoquan Chen*
University of Minnesota at Twin Cities
{ashesh|baoquan} @cs.umn.edu

ABSTRACT
This paper describes the user interface design techniques for a design-by-sketches system. The overall goal of the system is to support 3D object design through natural and interactive sketching processes. The system entails four unique features: (1) a tight coupling of direct and gestured sketching in both 2D and 3D domains; (2) a user invokable feedback system for supervised stroke interpretation, processing and 3D reconstruction; (3) harnessing both the hardware (pen device) and software (development toolkits) of the commodity Tablet PC; and finally (4) blending the new interface with limited amount of well accepted WIMP interface elements to ease a smooth adaptation to the new drawing conventions introduced in our system. We present issues and propose solutions to achieve these interface features and relevant sketching and construction operations. In addition to the interface design, we also discuss the reconstruction of sketched curved objects. Finally, we provide preliminary user evaluation and offer insights drawn from it.

KEYWORDS: Pen-based interaction, sketching, CAD

1 INTRODUCTION

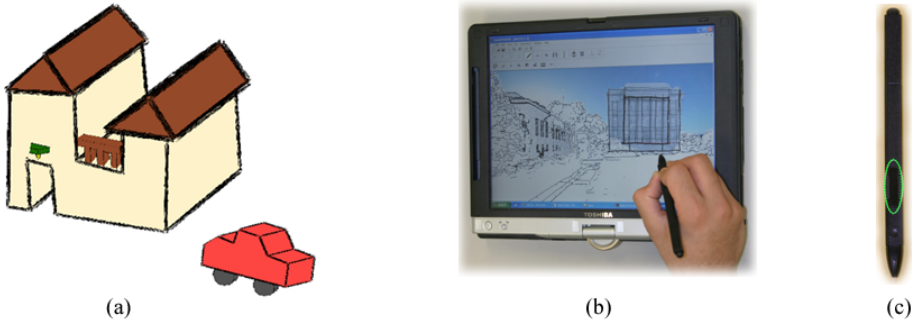
Design of objects and entities requires imagination and quick ways to record ideas, which is why designers prefer papers and pencils to computers even today. This is because the currently prevalent WIMP-style user interfaces do not offer room for more direct input like sketching, as they are based on selecting an option from a given set.

We present SMARTPAPER, an interactive and intuitive design-by-sketches system for creating and manipulating rigid 3D objects. There are two main avenues of research that we pursue through SMARTPAPER. The back-end sketch reconstruction module has been discussed in Shesh *et al* [11] and has been provided as supplemental material. This paper elaborates on its user interface design which makes it flexible and easy to use.

There are four aspects to the user interface of SMARTPAPER that make it unique and successful. First, SMARTPAPER contains a tight coupling of *direct* and *gestured* sketching in 2D and 3D domains. Direct sketching of objects of interest is given more importance when creating them from scratch, as the requirement of recalling learned gestures should be relaxed when the user is trying to record ideas. In addition to creating new objects, the user can directly sketch over existing 3D objects to augment them. Gestured sketching assumes more importance when reconstructed objects need to be modified as more specific meaning can be attached to strokes for specific operations. Secondly, SMARTPAPER facilitates interactive feedback for more active user participation in sketch reconstruction. SMARTPAPER does not strive to be a perfect, "oracular" reconstruction system that interprets sketches correctly every time. When the user is not satisfied with any output, s/he can explicitly invoke a feedback dialogue to monitor interpretation and quickly make changes through gestures. The feedback system is wholly optional and comes into play only when the user explicitly invokes it. Moreover, the feedback is unobtrusive and so the user is never distracted by unexpected pop-ups.

Thirdly we attempt to use features of commodity pen hardware to emulate the paper-and-pencil experience and create a better and more natural user interface for sketching. SMARTPAPER is implemented on a Tablet PC that makes it easy to design-on-the-go without elaborate hardware setup. Fourthly, we strive to create a blend of current user interface frameworks with new ones for sketch-based applications instead of rendering the former as irrelevant for such applications. Although most WIMP interfaces are not natural enough for sketching, some have been adapted well to suit the sketching domain. As most computer users are so used to them, adaptation to radically new ways of user interaction using specialized hardware will be difficult for everyday users.

The user interface goals for a sketch-based application are aptly summarized in Xu *et al* [16]: it should be humanistic, intelligent and individualized. SMARTPAPER is humanistic because it allows the user to draw freely as s/he draws on paper, in addition to employing gestures which often decrease sketching effort at the expense of having to learn them. SMARTPAPER is intelligent because it allows user sketches to be imprecise and shabby, and yet generates plausible 3D models. This is a feature that all CAD systems lack, as they require precise line drawings, if not dimensions, to generate and display a 3D model.



(a) (b) (c)

Figure 1: *SMARTPAPER at a glance: (a) a sample scene generated by SMARTPAPER, (b) SMARTPAPER on a Tablet PC showing envisioned architecture design application, (c) the Tablet PC stylus (the button for right-click is marked)*

2 RELATED WORK

2.1 General overview

Sketch-based tools have found applications in diverse fields. Gross *et al* [3] interpret schematic diagrams in architectural design. Landay *et al* [6] investigate sketchy design of user interfaces in their SILK system, whereas Xu *et al* [16] design electronic circuits using sketching. Teddy [5] interprets sketches for the design of freeform objects. Others [4, 18, 14, 13] discuss sketchy design in the context of rigid 3D objects. SMARTPAPER facilitates design of arbitrary closed rigid 3D objects and is aimed at architectural design.

Most systems have a *theme of interaction* with the user, which affects their user interfaces as well as the classes of objects whose design they support. Some themes include an iterative process of sketching, scanning and aligning [14, 13], but these approaches are only suitable when a freehand design is to be formalized, and are not interactive when a design has to be created. Suggestive interfaces proposed by Igarashi *et al* [4] and Xu *et al* [16] generate suggestions as the user is sketching so that s/he can select what object s/he was trying to sketch. While this may decrease sketching effort and facilitate more accurate reconstruction, such feedback may be intrusive to the user. Also these approaches cannot be generalized enough without greatly increasing the complexity of the suggestion-generating engines. A more popular *theme* is to let the user sketch and then reconstruct the sketched object after s/he is done. SKETCH [18] is such a gesture-based sketching system, where design is specified by predefined gestures for various operations. Though gestures decrease sketching effort and simplify reconstruction, they are not the most natural forms of input. Moreover, as more and more objects are supported by such systems, the user is required to learn more and more gestures. Teddy [5] is a system for the design of freeform objects that uses gestures more implicitly. Gestures are not critical to their system as many operations can be performed by directly sketching on an object. This works well because they support only freeform objects whose reconstruction is more defined than 3D objects in general. Nevertheless, we draw inspiration from existing systems, especially SKETCH and Teddy, to effectively blend direct and gestured sketching.

Some systems allow sketching only in two dimensions. Others like the VIKING [9] system allow the user to augment an existing 3D object by directly sketching on it. SMARTPAPER supports both types of sketching.

There are certain problems specific to interfaces of sketching applications. Since a pen/stylus has fewer buttons and greater applications, it has to be multiplexed to work like a pen and an ordinary mouse. Thus, various modes of operation have to be explicitly indicated by the user, as in the VIKING system [9]. This problem is discussed by Saund *et al* [10] and a heuristic solution is provided to guess what a pen movement means in the current context. The solution proposed is very specific to and highly dependent on the operations and gestures offered by a system, and so may not scale well as more and more ink/gestured operations are supported. We have chosen to "hide" the gesture by minimizing the user effort required, by using available pen hardware, as explained in Section 5.

2.2 SMARTPAPER – A Primer

Our previous work focussed on sketch reconstruction in SMARTPAPER is discussed in a recent paper [11]. Its sketch processing and 3D reconstruction pipeline are summarized here for completeness.

Sketch Pre-processing: When strokes are placed on the tablet, some pre-processing is done irrespective of their purpose (creation of new object, cutting, etc.). A stroke is characterized by the ink produced between a pen-down and pen-up event. Short strokes that are intended to collectively represent larger strokes are combined by using heuristics on stroke slope and inter-stroke distance. The consolidated strokes are compiled into an edge-vertex graph.

Sketch Reconstruction: The following operations are performed in sketch reconstruction:

1. Clustering [12] is performed in order to remove sketch imperfections such as two edges not intersecting at a vertex or going beyond it. In this operation, end points of edges that are within a certain distance from each other are merged.

2. The clustered graph is processed to determine all faces of the object using two algorithms devised by us. The graph is then "inflated" by determining depth coordinates for all its vertices. Various constraints like parallelism and

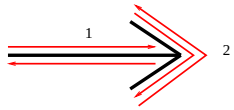
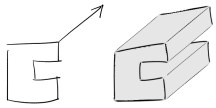
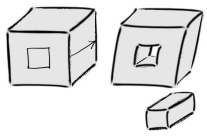
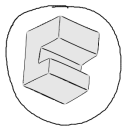
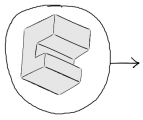
	Arrow gesture: numbers indicate sequence of drawing and red arrows indicate possible pen paths.
	Use of the arrow gesture in drawing by extrusion.
	Use of the arrow gesture in cutting by extrusion.
	Selection of an object.
	Combined gesture for transferring an object to the feedback window.

Figure 2: *Gestures in drawing and editing modes of SMARTPAPER.*

perpendicularity of edges and faces are used to set up a constrained optimization system as originally proposed by Lipson *et al* [7] and modified by us.

If an object is drawn by extrusion, then a new graph of the bounding cube is formed and the original object is "carved" out of it after reconstruction of the cube. Thus all objects specified by extrusion take the same time approximately for reconstruction irrespective of their complexity.

All reconstructed objects are projected in the viewing planes so that their augmentation is possible by directly placing strokes on or near them. If the user sketches to augment an existing 3D object, the unaffected vertices are marked and are not moved in the optimization process.

3 SMARTPAPER INTERFACE OVERVIEW

3.1 User Interface Theme

The general theme of the user interface is that after the required sketching is done, the user signals completion of sketching and the operation to be performed by making a single menu selection or pressing a single button from the toolbar. A direct consequence of this is that the user is free to sketch in neat lines as well as short discontinuous strokes without worrying about misinterpretation caused by such a style of sketching, as SMARTPAPER does not make an attempt to interpret the strokes until explicitly signalled to do so. This also adds uniformity to the kind of user input required to perform various operations. For example, SMARTPAPER does not count pen-up and pen-down events for gestures and so the user is allowed to over trace gestures as well, within certain limits.

3.2 User Walkthrough

SMARTPAPER starts by providing a blank window to the user. The user can immediately start sketching on it with the pen. The other end of the pen can be used as an eraser to erase sketched strokes.

If the user wishes to create a new object, s/he must draw it from a view in which edges and vertices do not hide behind each other, and must draw all edges, whether visible or invisible. Alternatively, the user can specify an object by drawing a closed profile followed by an arrow gesture as shown in Figure 2 to extrude it. The closed profile may be curved. The user can also sketch directly on a previous reconstructed 3D object to augment it. Sketches are reconstructed and the resulting 3D model is rendered using non-photorealistic techniques.

To cut an object, the user either draws the cutting lines directly or specifies the cutting planes by an open or closed profile followed by an extruding arrow and presses the "cut" button. To join two objects, the user first positions them so that the faces to be joined are visible, then draws a stroke between them and finally presses the "join" button. To select one or more objects, the user draws an enclosing circle around them and presses the lasso button (Figure 2).

In order to modify the structure of an existing object or to diagnose an error in reconstruction, the user first opens the feedback window. Then s/he encircles the object in question and draws an arrow gesture as shown in Figure 2. This results in three views of the object in the feedback window as shown in Figure 3. The user then specifies strokes in each view as required, as explained in Section 4. In this window too, the user presses the OK button to signal completion of sketching. After the user is satisfied with the changes in the feedback system, s/he can transfer it back to the original scene in the left window.

4 FEEDBACK SYSTEM

There are many uncertainties in sketch reconstruction and sketching systems in general. The reconstruction process is based on optimization and heuristics and hence results may not always satisfy the user. Also, the sketch can be unpredictably dirty leading to misinterpretation.

We feel that the best way to improve the performance of a system like SMARTPAPER is to facilitate user feedback and dialogue. If the user sees what the problem is, s/he can greatly assist in quickly correcting it. This is the motivation behind the feedback system in SMARTPAPER.

The goals of this system are effective visualization to enable the user to pinpoint the problem, allowance to changes and suggestions and quick response to them, and the provision of ways to refine a correctly reconstructed object. Visualization is important because an erroneous result often offers little insight into its cause. Therefore, three views of an object are offered (Figure 3).

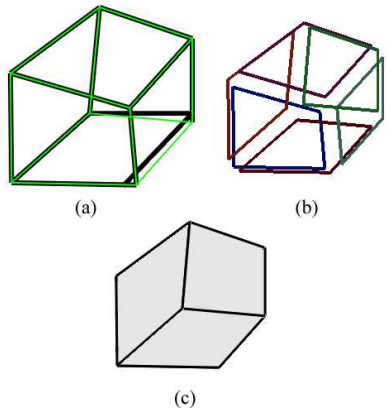


Figure 3: *A single object shown in three views in the feedback system: (a) Sketch view: The thick black lines show strokes entered by the user after preprocessing and the green lines show result of clustering, (b) Face view: an exploded view of the object, (c) Object view: the reconstructed 3D object.*

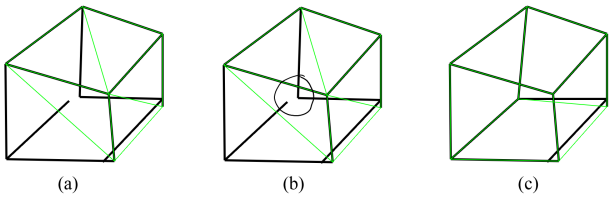


Figure 4: *Example of clustering by giving a hint. The thick black lines show the original sketch and the green lines show result of automatic clustering: (a) Initial sketch and clustering output, (b) the vertices to be clustered are marked by encircling, (c) the result of the hint, showing overall correct clustering by just one hint.*

4.1 Sketch View

This view shows how the system interpreted the user's strokes. If the sketch is dirty, then clustering may be incorrect leading to an erroneous reconstruction. This view shows the cleaned graph superimposed on the user's input (after preprocessing as described in Section 2.2). This makes any incorrect clustering obvious.

The user can manually suggest clustering by enclosing all the vertices to be clustered as shown in Figure 4 (b). When s/he presses OK, the points are combined, the whole graph is re-clustered and the results are immediately shown as in Figure 4 (c). It can be seen from these figures that correcting one clustering can produce wholly correct results. The user can then press the refresh button to make the other views consistent with this view.

4.2 Face View

The face view shows an exploded view of the object(Figure 3). This view is useful in verifying if the faces have been determined correctly. If any faces have been incompletely or incorrectly determined, the user can sketch a face directly in this view (Figure 5). Whenever a new face is sketched by the user, all edges that are part of more than two faces are marked, and all faces made of only marked edges are deleted.

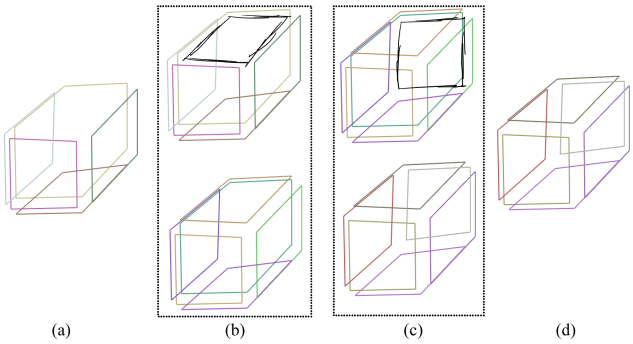


Figure 5: *Gestures in the face view: (a) a sample object with incorrectly determined faces, (b) and (c)upper figures show a face sketched and lower figures show their respective results, (d) the resultant object (with all faces correctly determined.)*

4.3 Object View

This view shows the reconstructed object, which is topologically correct if the clustering and face determination was done correctly. This view is used to further refine the object by specifying additional constraints on its structure. All algorithms use and modify the properties of the underlying graph of the object. To visualize these algorithms it is helpful to imagine the solid as a region enclosed by several bounded or unbounded planes and move them as per the operation. Our current implementation supports the following structural changes:

4.3.1 Making two edges of a face congruent to each other

It is difficult to sketch two congruent lines, and a gesture is used to specify such a constraint. The edges are specified by drawing one line across each of them (Figure 6). By default, the length of the second edge is modified to match that of the first. The algorithm *MakeEdgesCongruent* is provided in appendix A.1.

4.3.2 Making two edges of an object parallel to each other

Again, as it is difficult to sketch two perfectly parallel edges, this operation is useful in making two edges parallel. The two edges are specified by drawing two parallel lines across them as shown in Figure 6. By default, the second edge is made parallel to the first. The affected vertices are moved as explained in the algorithm *MakeEdgesParallel* given in appendix A.2.

4.3.3 Making two edges of a face perpendicular to each other

This operation is useful to make a face rectangular. It is specified by drawing a bracket between the two edges as shown in Figure 6. The two edges specified have to belong to the same face. After determining the edges from the bracket gesture, they are made perpendicular by moving vertices of the graph using the algorithm *MakeEdgesPerpendicular* provided in appendix A.3.

4.3.4 Making two faces of the object perpendicular to each other

This is specified by drawing two lines along the two faces meeting near their common edges, and a bracket between them as shown in Figure 6. The two faces must share an edge. These faces are determined from the end points of

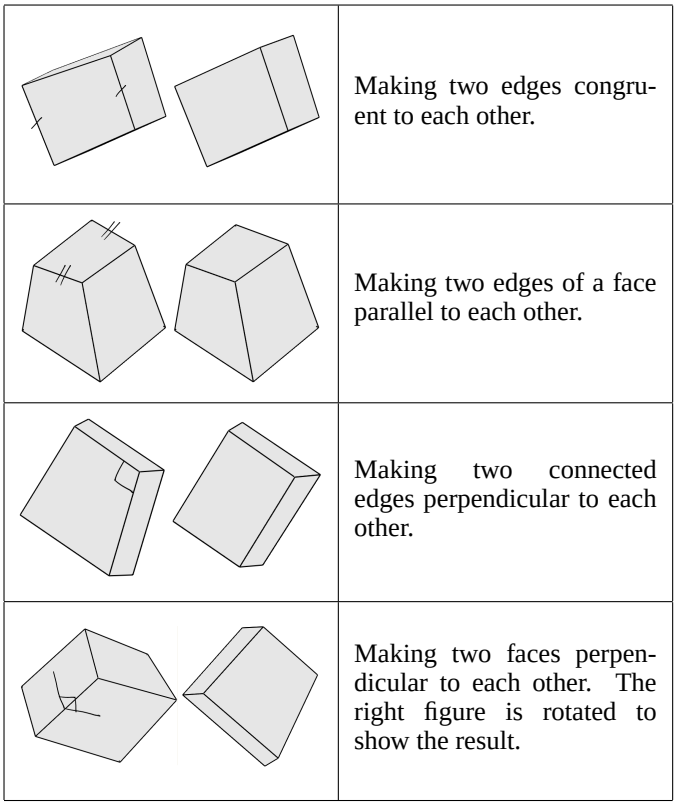


Figure 6: *Gestures in feedback system (In the first column, figures on the right show the result of the respective operation).*

the first two strokes by traditional ray-casting. The algorithm *MakeFacesPerpendicular* given in appendix A.4 is then used to accordingly move the affected vertices.

4.3.5 Making two faces of an object parallel to each other The two faces are specified by drawing two parallel lines on them (one face is marked, the object is rotated and then the second face is marked). By default, the second face is changed to make it parallel to the first. Affected vertices are moved according to the algorithm *MakeFaceParallel* given in appendix A.5 to complete the operation.

Our experience shows that an object can be quickly refined to regular shapes using these gestures. Regularity cannot be implicitly assumed in the sketch because it will artificially classify all sketches into a small class of regular objects. However, it can be achieved easily through operations like these.

5 SWITCHING MODES AND GROUPING OBJECTS

The pen is used both as a sketching and a pointing device, leading to the ambiguity problem discussed earlier. The right click button on the pen is conveniently located near the grip (Figure 1 (c)). Pressing it and tapping the screen changes the mode and the change is visible by depression of a button. This is a semi-implicit gesture because the user does not have to move the pen away from the sketching surface nor does s/he have to change the hand position or grip. Pens that do not have this feature can use the button provided for this purpose.

For moving an object, we define an "interactor" that manages and supplies the transformation matrices for movement by mouse/pen. Each such interactor contains a queue which points to the objects under its control. Every object stores its transformation matrices. Thus, grouping objects consists of identifying them and attaching them to the same interactor. Henceforth, when this group is moved, the interactor passes the relevant transformation matrices to all the objects in its queue. This creates an implicit object and transformation hierarchy.

6 RECONSTRUCTION OF CURVED OBJECTS

We support drawing objects with curved edges by extrusion, which is not part of our prior system [11]. Extrusion requires a closed profile. However, the user may draw a curve whose end points either do not meet or cross each other. To generate a smooth closed curve from such input, we employ a modified version of the Snake algorithm [15, 2].

The original Snake algorithm is interactive in semi-automatic conditions where the user explicitly inputs an approximated closed curve that is used as an initial guess, resulting in fast convergence. This is not feasible in our case as this problem is transparent to the user. Therefore we fit a circle using least-square approximation to the stroke data and use it as the initial guess. This potentially decreases the rate of convergence and hence we modify the implementation of the Snake algorithm in the following ways:

1. It is required to compute the nearest point in the input data (edge data) to a point on the closed curve in a particular direction during each iteration. This is an expensive operation if there are a lot of points in the input data. Therefore, we allow 3 initial expensive iterations to "guess" a subset of the input for every point on the closed curve, and cache it for further iterations. This greatly speeds up the program.
2. The original condition for convergence in the Snake algorithm is the number of points moved in one iteration. This causes points to oscillate if the input data is not smooth, as in the case of strokes. Hence, we eliminate points on the closed curve from further consideration if they are within a certain distance to the input data. Thus more and more points are culled as the iterations proceed, leading to an increase in speed.

The wheels of the car in Figure 1 are cylinders constructed by this process.

7 DISCUSSION AND USER EVALUATION

SMARTPAPER has been implemented on a Toshiba Portege 3500, which runs Windows XP on a 700 MHz PIII Mobile processor with 512 MB RAM. It has been coded in Microsoft VC++.NET.

SMARTPAPER was demonstrated to 10 students of the Department of Architecture at the University of Minnesota. Two graduate students from the Department of Architecture and

four computer science graduate students participated in a detailed user evaluation session. The users were briefed about its functionality, gestures and usage and then were asked to perform some pre-defined tasks.

Support for over traced sketching was appreciated by the students, as a lot of them indulged in similar sketching practices. They were asked to construct primitive objects like cubes and prisms and curved objects like cylinders which they did with reasonable ease.

In an effort to confirm the intuitiveness of the gestures in the feedback system, they were not explained to the architecture students beforehand and were asked to do whatever they deemed intuitive. Some gestures were guessed correctly by them, which showed that they were fairly intuitive to users of non-computer science background. Other gestures were learnt easily when they were explained. The users received the seemingly "new" theme of explicit gestures and direct sketching very well, compared to their experience with mouse-based CAD software. The overall theme of the feedback system and the feature of instant shape modification found immediate support among our users. In an attempt to evaluate how all the gestures worked in tandem towards a common goal, the users were asked to make the object shown in Figure 3 into a uniform cube. The total time inclusive of the time taken to learn the gestures by both users was approximately 2.5 minutes. Some useful suggestions made by them are discussed in the next section.

8 CONCLUSION AND FUTURE WORK

In summary, the general goal of the user interface features in SMARTPAPER is to emulate the paper and pencil in usability. The first step towards this goal has been achieved by using convenient, portable and easily available pen hardware like a Tablet PC. We believe that the user interface can be made intuitive by striving to create a mix of direct and gestured sketching. More and more domain-based practices must be learned and incorporated to provide a user-centered practical sketching system.

There are several avenues which we plan to pursue in the future towards a smarter SMARTPAPER. We envision SMARTPAPER to work in conjunction with another project independently pursued by our group, which deals with acquisition, digitization and rendering of data obtained by scanning outdoor environments [1]. An architect would be able to view a design site through this work and then sketch ideas and buildings directly in this site model using SMARTPAPER (Figure 1 (b)). The ability to design "on site" has been highly desired by the architecture community, and this will be a very captivating experience and will provide an ideal design environment for conceptual design in architecture.

Secondly, we are working closely with the students and faculty of the Department of Architecture at our university to gain more knowledge about the practices of architectural design. We plan to incorporate such domain knowledge in SMARTPAPER to make it more practical for architects and students of architecture. In particular, the practice of drawing "construction lines" for reference in sketching, the ability to draw open objects and environments and drawing only the visible parts of an object, etc. are included in our immediate research plans. We also plan to investigate the importing and exporting of data in compatible formats so that SMARTPAPER can be used in real design environments in conjunction with other design software.

The initial positive feedback from our users about our feedback system has encouraged us to strive for supporting more and more "prototyping" operations. A few suggestions like making all sides of a face mutually parallel and perpendicular, making one face congruent to another and making an edge perpendicular to a face made by our users will be incorporated in SMARTPAPER shortly. More operations like subtraction and union of objects will also be incorporated and the user interface will be modified to specify these in an easy manner.

Navigation in 3D environments using 2D pointing devices has been identified as a problem, and this problem is somewhat more obvious in a pen-based application than a mouse-based one. The degrees of freedom (DOF) of 3D operations cannot be mapped to the limited DOF of 2D pointing devices like mice and pens. Zeleznik *et al* [17] propose two cursors to achieve this mapping. However, their recommendation of using a mouse with a keyboard is not feasible for hand-held devices like the folded Tablet PC. Also, we feel that this method is significantly intuitive in transforming operations like rotation, etc. but is not as effective in sketching operations. The method proposed by Llamas *et al* [8] is worth consideration if it can be generalized to operations other than sculpting.

A Algorithms for 3D Operations

A.1 MakeEdgesCongruent

Input: Edges e and e' of face F.

//By default make $l(e') = l(e)$.

Let $e'(v, v')$. Let F' be the face sharing e' with F. Let e'' be the edge incident on v' in face F. Let $e''(v, v'')$. Let F'' be the face sharing edge e'' with F. Let e_1 be the edge incident on v' not in F. Let $e_1(v', v_1)$. Finally, let $e_2(v_1, v_2)$ be the edge in F' incident on v_1 and is not e_1.

Move v' along e' till the length is equal to e. Now move v_1 an equal distance along e_2 in F' by an equal distance to keep the slope of e_1 unchanged. Face F' has a new plane formed by new v', new v_1 and unchanged v''.

For all vertices V in F'' that are not $v', v'' and v_1$:

Find the edge E incident on V that is not in F''. Find the intersection of E with new F'' to find the new position of V.

A.2 MakeEdgesParallel

Input: Edges e and e' of face F.

//By default make e' parallel to e.

Let F' be the face sharing edge e' with F. Let $e'(v, v')$. Fix v. Let e'' be the edge incident to v' and in F. Let $e''(v, v'')$. Let $e_1(v, v_1)$ be the edge incident at v in F'.

Move v' along e'' till e and e' are parallel. Face F' has a new plane formed by v,v_1 and the new v'.

For all vertices V in F' that are not v and v':

Find the edge E incident on V that is not in F'. Find the intersection of E and the new F' to find the new position of V.

A.3 MakeEdgesPerpendicular

Input: Edges e and e' of face F.

Let v be the common vertex between e and e'. If there is no such vertex, then the operation cannot be performed.

Let $e'(v, v')$. Let F' be the face sharing e'. Let e'' be the edge in F other than e' incident on v'. Let $e_1(v, v_1)$ be the edge incident at v which is not in F. Move v' along e'' till e and e' are perpendicular. Face F' has a new plane formed by v, new v' and v_1.

For all vertices V of F' that are not v,v' and v_1:

Find the edge E incident on V that is not in F'. Find the intersection of E and the new F' to find the new position of V.

A.4 MakeFacesPerpendicular

Input: Faces A and B with common edge e.

Let n_A and n_B be the normals of A and B respectively. Move B' by computing a new normal n'_B which is perpendicular to n_A.

For all vertices V of B not in edge e:

Let e' be the edge at v which is not in B. Find the intersection of e' and new B to find the new position of V.

A.5 MakeFacesParallel

Input: Faces A and B with face F which shares an edge each with A and B.

Let n_A and n_B be the normals of A and B. Let $e(v, v')$ be the edge in F which is shared with B. Let e' be the other edge in F that is incident on v'. Let e'' be the edge in B incident at v and not in F. Let $e''(v, v'')$. Move v' along e' till n_B equals n_A. Face B has a new plane formed by v, new v' and v''.

For all vertices V in B that are not v, v' and v'':

Find the edge E incident at V that is not in face B. Find the intersection of E and B' to get the new position of V.

REFERENCES

1. 3d scanning project. *www.cs.umn.edu/ baoquan/scan.html*.
2. Active image contours. *www.cs.wisc.edu/ pingelm/Algo.html*, Date retrieved: April 6, 2004.
3. Mark D. Gross and Ellen Yi-Luen Do. Ambiguous intentions: a paper-like interface for creative design. In *Proc. ACM Conference on User Interface Software Technology (UIST) 1996*, pages 183–192, 1996.
4. Takeo Igarashi and John F. Hughes. A suggestive interface for 3d drawing. In *14th Annual Symposium on User Interface Software and Technology, ACM UIST 2001*, pages 173–181, 2001.
5. Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *Proc. SIGGRAPH 1999*, pages 409–416, 1999.
6. James A. Landay and Brad A. Myers. Interactive sketching for the early stages of user interface design. In *Proc. SIGCHI 1995*, pages 43–50, 1995.
7. Hod Lipson and Moshe Shpitalni. Optimization-based reconstruction of a 3d object from a single freehand line drawing. *Journal of Computer Aided Design*, 28(8):651–663, 1996.
8. Ignacio Llamas, Byungmoon Kim, Joshua Gargus, Jarek Rossignac, and Chris D. Shaw. Twister: A space-warp operator for the two-handed editing of 3d shapes. In *Proc. SIGGRAPH 2003*, pages 663,668, 2003.
9. David Pugh. Designing solid objects using interactive sketch interpretation. In *Proc. of 1992 Symposium on Interactive 3D Graphics*, pages 117–126, 1992.
10. Eric Saund and Edward Lank. Stylus input and editing without prior selection of mode. In *Proc. of the 16th annual ACM symposium on User Interface Software and Technology (UIST) 2003*, pages 213,216, 2003.
11. Amit Shesh and Baoquan Chen. Smartpaper–an interactive and intuitive sketching system. *Accepted and to appear in Eurographics 2004*, 2004.
12. Moshe Shpitalni and Hod Lipson. Classification of sketch strokes and corner detection using conic sections and adaptive clustering. *ASME Journal of Mechanical Design*, 119(2):131–135, 1997.
13. Osama Tolba, Julie Dorsey, and Leonard McMillan. Sketching with projective 2d strokes. In *Proc. UIST 1999*, pages 149–157, 1999.
14. Osama Tolba, Julie Dorsey, and Leonard McMillan. A projective drawing system. In *Proc. I3D Symposium on Interactive 3D Graphics*, 2001.
15. Donna J. Williams and Mubarak Shah. A fast algorithm for active contours and curvature estimation. *CVIGP Computer Vision Graphics Image Processing:Image Understanding*, 55(1):14–26, 1992.
16. Xiaogang Xu, Wenyin Liu, Xiangyu Jin, and Zhengxing Sun. Sketch-based user interface for creative tasks. In *Proc. 5th Asia Pacific Conference on Computer Human Interaction (APCHI2002)*, pages 560,570, 2002.
17. Robert C. Zeleznik, Andrew S. Forsberg, and Paul S. Strauss. Two pointer input for 3d interaction. In *Symposium on Interactive 3D Graphics*, pages 115,120, 1997.
18. Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: An interface for sketching 3d scenes. In *Proc. SIGGRAPH 1996*, pages 163–170, 1996.