EUROGRAPHICS 2004 / M.-P. Cani and M. Slater
(Guest Editors)

Volume 23 (2004), Number 3

SMARTPAPER: An Interactive and User Friendly Sketching System

Amit Shesh and Baoquan Chen

University of Minnesota at Twin Cities †

Abstract

This paper describes an interactive sketching system for 3D design/modeling that diverts from the conventional menu-and-button interfaces of CAD tools. The system, dubbed SMARTPAPER, offers a unified sketching environment that supports direct sketching as well as gestured sketching with more emphasis on the former to encourage natural sketching styles. SMARTPAPER also provides a unified 2D and 3D drawing domain by allowing the user to sketch directly on a 3D model in addition to the usual 2D sketching from scratch. A natural sketching experience is offered by supporting casual sketching consisting of wiggly, discontinuous, overlapping strokes. The system is empowered by an array of seamlessly integrated 2D and 3D features such as 2D sketch cleaning, 3D reconstruction from 2D sketch, 3D transformations, sketching on 3D, and conventional 3D CSG operations like cutting and joining. The key to the success of SMARTPAPER is efficient and robust 3D reconstruction from a single freehand 2D sketch with minimal hints. We have employed and improved Lipson's optimization method, originally designed for offline reconstruction of engineering drawings, in our interactive system by leveraging additional clues obtained by interaction during sketching.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Interaction Techniques, Pen-based Interaction

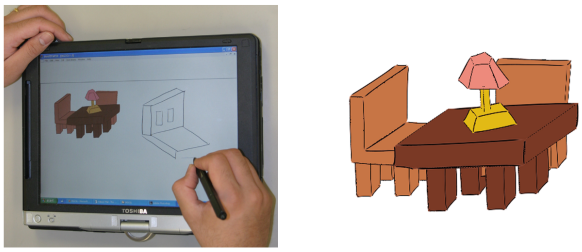
1. Introduction

Designers in almost all professions use a paper and pencil to make sketches during the early stages of design. However, designing by sketches on computer has been quite difficult because of hardware usability issues. Also, the lack of 3D geometric information in sketches and the imprecision associated with them makes them very difficult to interpret algorithmically. In particular, 3D model reconstruction from a single projective sketch is a mathematically insoluble problem.

Considerable research effort has been devoted to promote design by sketches. Work has been done to support sketching schematics [GD96] and actual 3D models in architecture design. Lipson and Shpitalni [LS00, SL97, LS02, SL96, LS96] generate 3D models from 2D sketches, but their systems are non-interactive in nature. SKETCH [ZHH96] represents a fully gesture-based sketching system, which however, can be unintuitive to use if the gestures are not carefully designed and are large in number. Teddy [IMT99] employs fewer and more intuitive gestures, but it focuses on design of free-form objects. We attempt to minimize these shortcomings by providing an interactive sketching system, SMARTPAPER, that is capable of reconstructing and operating on arbitrary rigid solid geometrical shapes and is intuitive to use. SMARTPAPER draws inspiration from all the seminal work in sketch recognition mentioned above to achieve these goals. The system is implemented on a Tablet PC for a more natural sketching experience (Figure 1(a)).

We make four significant contributions through SMARTPAPER. First, SMARTPAPER presents a unified sketching environment that supports both direct and gestured sketching, with emphasis on the former. Secondly, SMARTPAPER gives more freedom to the user by supporting casual sketching styles, where several overlapping discontinuous strokes

† Email: {ashesh,baoquan}@cs.umn.edu



(a) (b)

Figure 1: *SMARTPAPER at a glance: (a) SMARTPAPER on a Tablet PC, (b) a scene created using SMARTPAPER.*

could be sketched. Thirdly, it allows a user to sketch directly on a 3D model in addition to making a normal 2D sketch. Fourthly, SMARTPAPER provides a feedback system that allows a user to examine the interpretation made and provide hints accordingly to improve its performance, leading to greater user satisfaction. In addition to sketch recognition, SMARTPAPER offers a compendium of Computational Solid Geometry (CSG) operations, synergistically resulting in a practical proof-of-concept system. Also, it employs non-photorealistic rendering techniques to give the reconstructed objects a sketchy look. As will become evident in section 3, from a user interface perspective, the system combines seamlessly various 2D and 3D operations such 2D sketching, sketching on 3D, 3D transformations, cutting and joining.

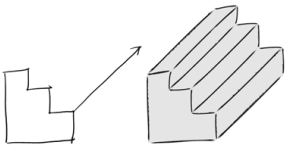
2. Previous Work

Most sketching systems focus on a particular class of entities according to their target application areas. Landay and Myers [LM95] study the application of sketch recognition in 2D user interface design. *Electronic Paper Napkin* [GD96] focusses on schematic diagrams in conceptual design, also in two dimensions. Teddy [IMT99] aims at design of 3D free-form objects using sketches. Many sketching systems [ZHH96, GD96, LS00, SL97, LS02, SL96, LS96] focus on 3D rigid objects. SMARTPAPER generates 3D models of arbitrary rigid solid objects.

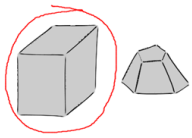
Since sketch recognition and interpretation is a mathematically insoluble problem, most sketching systems offer users some constrained drawing environment to ease reconstruction. In the system designed by Tolba *et al* [TDM01] the user iteratively sketches objects on paper, scans them into the system and aligns them on a provided "perspective grid". Igarashi *et al* [IH01] interactively generate suggestions as the user sketches. Although this may result in a greater recognition accuracy as it is based on selection instead of direct recognition after the sketch is complete, the frequent suggestions popping up on the screen while the user is sketching can be distracting. Also, these are rule-based systems and may not scale well to more general objects. Gestured interfaces like SKETCH [ZHH96] identify gestures from the input strokes and interpret them according to a fixed set of rules. The performance of these systems and their ease of use critically depend upon their ability to design intuitive and fewer gestures and accurately and faithfully recognize them respectively. Some systems [IMT99, LB90] allow direct sketching which does offer more flexibility to the user without having to learn many gestures. SMARTPAPER supports both direct and gestured sketching and reconstructs 3D objects from sketchy inputs.

Lipson *et al* [LS96] present an innovative approach to sketch reconstruction of 3D rigid objects based on optimization. The input is a scanned sketch. The sketch is processed and converted into a 2D graph. This approach then attempts to "inflate" the graph by giving suitable Z-coordinates to each vertex of the sketch, assuming that the drawing surface represents the x-y plane. These coordinates are determined by optimization based on geometric properties such as parallelism and perpendicularity of edges and faces estimated from the 2D sketch. Sometimes special properties, such as skewed symmetry, can be leveraged to make the optimization process faster [PMC03]. Later, Lipson *et al* [LS02] take a learning-based statistical approach in which correlations between 3D objects and their projections are set up and are used to identify 3D objects from arbitrary 2D projections. This approach requires considerable prior learning and works satisfactorily for mechanical engineering drawings that are more precise in nature. SMARTPAPER employs an array of modifications to Lipson's approach in [LS96] to deliver an interactive sketching system.

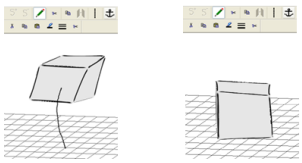
The rest of the paper is organized as follows: Section 3 summarizes a typical user experience with SMARTPAPER. Section 4 provides an overview of the system design, and Sections 5 and 6 provide details about sketch processing, representation and reconstruction. Section 7 describes the feedback system, CSG operations and provides a discussion on the NPR techniques used in SMARTPAPER. Section 8 gives the implementation details and provides a general discussion. Finally, Section 9 identifies avenues for future work.



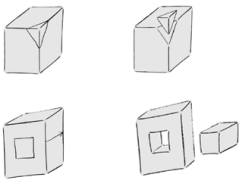
(a) Construction of an object by extrusion.



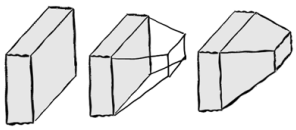
(b) Object selecting by circling.



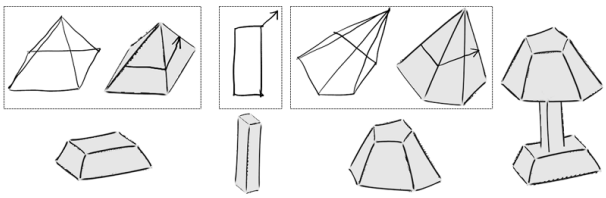
(c) Object anchoring by drawing a line between the ground and object. The line also indicates which face of the object touches the ground.



(d) Object cutting by specifying a cutting plane (top) and extrusion (bottom). Figures on the left show sketched input (triangle on top and profile and arrow on the bottom).



(e) Incremental construction of a 3D object by sketching directly on an existing 3D object.



(f) Construction of a slightly more complex object--a lamp; boxes show the steps involved in making the 3D components of the lamp: the lamp base, shaft and lamp shade, shown below them The lamp base is constructed by drawing a pyramid and then cutting it by extrusion. The shaft is constructed by drawing it by extrusion. The lamp shade is constructed by drawing a pentagonal pyramid and then cutting it by extrusion. The assembling step is not shown.

Figure 2: *Representative features of SMARTPAPER. Shaded figures are 3D objects while others are user sketches, unless otherwise specified.*

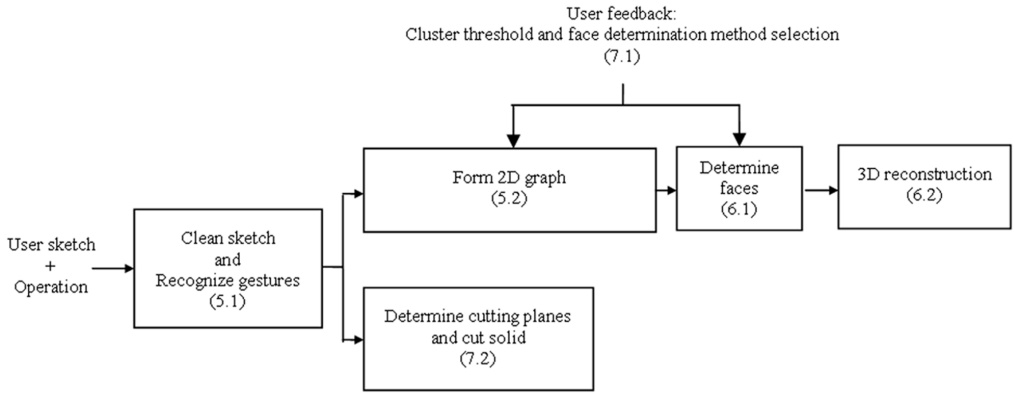


Figure 3: *The processing pipeline of SMARTPAPER.*

3. User Session Overview

SMARTPAPER is implemented on the Tablet PC, which creates an unparalleled paper-and-pencil experience. It presents a directly sketchable blank window. Alternatively, it can provide standard templates like a ground or a room as a starting point for the user. The user can directly begin to sketch on the window with a stylus. The stylus emulates well a standard pencil, by enabling the user to erase sketched strokes with its other end.

The general *theme* of working with SMARTPAPER is to sketch strokes and then specify what they mean. For example, to sketch an object the user places strokes as s/he likes and then specifies the ”recognize" operation. This allows imperfections in all operations requiring sketchy input.

To construct simple objects like pyramids, prisms, frustrums, etc. the user can directly sketch them. All edges of the object must be drawn in this case. But sometimes it is cumbersome for the user to draw all edges of an object. Objects that are extrusions of planar profiles can be constructed by sketching the closed profile and an extruding arrow, as is illustrated in Figure 2(a). To construct more complicated objects, the user can employ an incremental process by creating a simple object, transforming it, sketching directly on it to augment it and so on. This feature is illustrated in Figure 2(e). Whole objects may not be constructed in one session, making the process truly incremental.

Similar to drawing, a user can cut an object in two ways. S/he can directly specify how an object will be cut, by sketching lines on it. Alternatively, the cutting planes can be specified by drawing an open or closed profile, followed by an extruding arrow. Both forms of cutting are illustrated in Figure 2(d).

Two objects can be joined by first positioning them so that the faces to be joined are visible, sketching a line joining the two faces and then specifying the join operation. This sticks the two faces to each other, after which the user can position them as required and then commit the join operation. Anchoring an object to the ground is a special form of joining, and can be performed by sketching a line joining the ground and the face of the object to be kept on it and specifying the anchoring operation. This is illustrated in Figure 2(c).

Finally, when there are many objects on the screen, an object can be selected by circling the object to be selected and pressing the lasso selection button, as illustrated in Figure 2(b). Figure 2(f) shows how a lamp can be constructed using all of the above operations. The lamp base and lamp shade are created by drawing and cutting pyramids. These parts are assembled together by transformation and joining to create the lamp. This lamp is a part of a more complex scene shown in Figure 1(b), which is constructed incrementally.

4. System Overview

The block diagram in Figure 3 shows the processing pipeline of SMARTPAPER. This pipeline shows operations for which a sketched input is required, i.e. drawing a new object, augment existing ones, and cutting. Point-and-click operations such as joining two objects are not illustrated in this pipeline.

When a set of strokes and the user command is given, over tracing and other imperfections are removed, as explained in Section 5.1. This preprocessing is done on all sketched strokes irrespective of the operation to be performed. A direct consequence of this is that such sketching imperfections are allowed while drawing as well as cutting. Any gestures that are part of the set are recognized when queried.

The top part of the pipeline enumerates the 3D recognition operation, while the lower part represents the cutting operation. If the specified operation is 3D recognition, then 2D graphs are formed from the set of strokes (Section 5.2). A set of valid cycles forming faces of the sketched object is

then determined (Section 6.1). 3D-model reconstruction is then performed by optimization to reconstruct the final 3D object (Section 6.2). This process can be intervened by user feedback (Section 7.1). Figure 4 illustrates various steps in recognition.

If the specified operation is cutting, then ray casting is used to determine a set of cutting planes (Section 7.2). The object is then cut resulting in two or more objects.

Our system makes a **Closed Object assumption** about a drawn object: only solid objects that are homeomorphic to a sphere can be directly drawn. Objects that are not strictly closed (e.g. objects with holes) can be constructed by a series of operations. A series of planes resulting in an open object cannot be drawn.

This class of objects encompasses all objects that can be physically constructed, and hence does not impede applications like architectural design.

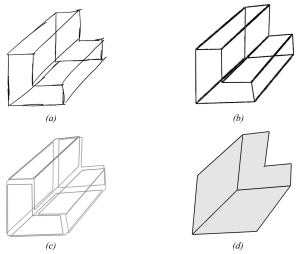


Figure 4: *3D reconstruction pipeline: (a) the input rough sketch, (b) the cleaned 2D graph, (c) the recognized faces, and (d) the recognized object.*

5. 2D Processing

The set of strokes is subject to initial pre-processing. This process achieves sketch cleaning and represents the strokes in a consistent format for 3D reconstruction or other operations.

5.1. Sketch Cleaning

Whenever strokes are drawn either for 3D reconstruction or cutting, some common preprocessing is done. Two functions are achieved in this block: over tracing consolidation and gesture recognition.

Over Tracing

A sketch is often drawn as a series of discontinuous strokes to illustrate edges (Figure 5). Such over tracing could either be done unintentionally, to highlight an edge or to simply complete an edge. These strokes are grouped together to form the continuous edge(s) that they collectively represent. This grouping is achieved in two passes over the set

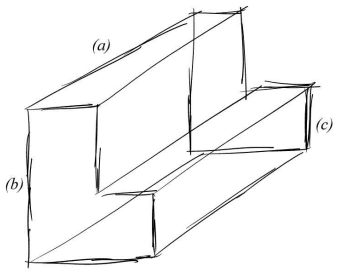


Figure 5: *Examples of over tracing: (a) over tracing done to complete an edge, (b) unintended over tracing, and (c) over tracing to highlight an edge.*

of strokes. In the first pass, we find pairs of strokes that qualify as over traced strokes. A pair (A,B) qualifies when they have nearly equal slopes, and at least one end point of B lies in the x and y ranges of the end points of A. Let $A(e_1,e_2)$ and $B(e_3,e_4)$ be the two strokes, and let e_3 be the end point of B lying in the range of e_1 and e_2. Now, let $length(e_2,e_4) < length(e_1,e_4)$. Then B is changed to $B'(e_2,e_4)$. At the end of this pass, overlapping segments become segments having one common end point. The second pass then culls all vertices, all edges incident to which have nearly equal slope. Thus, for example, if no other edges are incident on e_2, $A(e_1,e_2)$ and $B'(e_2,e_4)$ form a single stroke $A'(e_1,e_4)$. These vertices cannot be removed in the first pass itself because the incidence of *all* edges has to be known before a vertex can be culled. The over traced sketch in Figure 4(a) is cleaned and interpreted as Figure 4(b).

Gesture Recognition

SMARTPAPER recognizes standard gestures for gestured drawing and cutting. Gestures are also sketched and are part of the input set of strokes. The arrow gesture has been implemented as a proof of concept, and its usage for extrusion is illustrated in Figure 2(a) and (d). The drawing convention is to draw an arrow in two strokes, the first being the shaft and the second being the head, drawn from end to end. The recognizing and cutting modules query this gesture recognizer for gestures. The set of strokes is then passed to the recognizing or cutting module, depending upon the operation specified.

5.2. Graph Generation

Every sketched object is represented as a 2D graph of vertices and edges. A connectivity matrix is maintained for each object. Each vertex stores (x,y) coordinates.

The set of input strokes is distributed among the existing set of objects depending on their proximity with the projection of existing objects in the current viewing plane, as the user does not explicitly specify whether the strokes specify new objects or augmentation to existing ones. Strokes that do

not augment existing objects create new graphs. Two graphs are merged if a stroke with one end point in each of the graphs occurs. If an object is drawn by extrusion, then two copies of the profile are made and are connected by edges parallel to the direction of extrusion.

Clustering as mentioned in [SL97] is used to group vertices close to each other in the graph. As edges are added to the graph, all end points within a distance of δ from an existing vertex are grouped with it. This clustering threshold δ can be changed in the feedback system to allow sketches with lesser precision (Section 7.1).

To uphold the Closed Object assumption, each vertex must have degree at least 3. This check is used to clean unnecessary vertices, which may be created when a single stroke is incorrectly interpreted as two or more edges due to its slope. The final representation is a 2D graph with vertex degree at least 3 and order at least 4. Augmented and newly sketched objects undergo 3D geometry reconstruction.

6. 3D Geometry Reconstruction

This section encapsulates the functionality for determining the 3D aspects of each unrecognized object, namely face determination and iterative 3D reconstruction. For the following discussion, for a given graph G, $V(G)$ and $E(G)$ represent the vertex set and edge set of G, respectively, while $G-e$ is the sub-graph obtained by deleting the edge e from G.

6.1. Face Determination

We determine the faces of the object from its representative graph G as illustrated in Figure 4(c). All faces are cycles of G; however the converse is not true. Shpitalni *et al* [SL96] discusses a face determination algorithm based on an A* or branch-and-bound search. Their algorithm is too slow because it performs an exhaustive search on the set of all possible cycles, and the Closed Object assumption allows us to formulate a definition which simplifies face determination. In the definition, graph G represents the 2D graph of an unrecognized object:

Definition A: All edges of graph G are part of exactly two faces. Every valid face F of G is such that for all pairs $v_1, v_2 \in V(G)$ that are in F, the shortest v_1, v_2-path in G is of the same length as the v_1, v_2-path in F.

Justification: The first statement is a property of closed, non-laminar, rigid objects. If the second statement is not true, let P be the shortest v_1, v_2-path in G and let P' be the shorter v_1, v_2-path in F. There is at least one edge in P not in P'. PP' thus creates a smaller closed walk and hence a smaller cycle C containing v_1 and v_2. The edge set $E(C) - (E(C) \cap E(F))$ divides face F into two or more different planes, which is a contradiction as F is a valid face and hence is planar.

We propose two algorithms for face determination that directly determine all valid faces of G instead of examining all possible cycles of G for validity. While the first algorithm takes advantage of interactive drawing cues and is fast, the second algorithm is theoretically more robust albeit slower.

Algorithm 1: Edge Coherence algorithm

Humans draw objects according to how they perceive them. Often our drawing styles construct the object part by part. This algorithm examines the sequence S in which strokes are drawn to search for cycles that form valid faces. Note that consolidating over traced strokes does not disturb this sequence, as the earliest stroke in a series of over tracing strokes is used to determine the order of the consolidated stroke.

If the object is drawn face by face, then edges of such faces are adjacent in S and thus, a linear traversal of edges in their order directly recognizes some faces. If two adjacent edges in this order do not have a common end vertex, the algorithm "looks ahead" in S to find an edge connected to the current edge. Our preliminary tests showed that a look ahead of 1 was sufficient for simple primitives like prisms, pyramids, etc. This algorithm works in two passes. Pseudo code for it is provided in the appendix and is illustrated in Figure 9. The main advantage of this algorithm is its speed. The amortized cost of the first pass is $O(e)$, while the second pass takes $O(e1 \cdot n^2)$ time, where e1 is the number of edges that are not part of two faces and n is the number of vertices of G. The first pass amortizes the second pass depending upon how the object is drawn. This is the default face determination algorithm in SMARTPAPER.

The speed of this algorithm is due to the fact that humans intuitively draw objects according to how they perceive them and not in a completely random fashion. However, this assumption is not theoretically strong. Also, if all the strokes collectively representing an edge of the object are erased and redrawn, this order may be disturbed. To improve the performance of SMARTPAPER, we have devised a second algorithm for face determination. This is a theoretically more robust albeit considerably slower algorithm, but it still directly determines all valid faces and hence is an improvement over [SL96]. The user can explicitly switch algorithms as explained in section 7.1.

Algorithm 2: Modified Dijkstra's algorithm

This algorithm finds all faces for each edge in the graph such that definition A is satisfied. For every edge e, we remove the edge from the graph and find a shortest path between its end points. We employ Dijkstra's algorithm for this purpose.

We maintain a set R of edges for which all faces containing those edges have been found. If any edge e' in this set shares an end vertex with the current edge e, we mark the other end vertex of e' as traversed. This prevents the Dijkstra's algorithm from finding a shortest path which contains

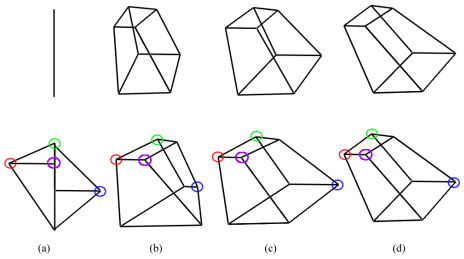


Figure 6: *Inflation of sketch by optimization. Upper and lower rows show how inflation without and with layering respectively: (a) initial condition, (b) and (c) intermediate states, and (d) the final object. The colored circles show how vertices move during inflation.*

any edge in R in it. Intuitively, once all the faces of which an edge is part of have been determined, the edge cannot be part of any faace determined in future, even if it is on some shortest path. Thus, this satisfies definition A and hence the algorithm is correct. Pseudo code for the algorithm is provided in the appendix. The last step in the pseudo code removes any fictitious "internal" faces as illustrated in Figure 10. Thus, the algorithm finally produces two faces per edge.

Both algorithms update face lists of objects incrementally. It is important to note that if a sketch augments an existing object, only faces containing vertices to which new edges are incident are determined.

6.2. Iterative 3D Reconstruction

SMARTPAPER uses a modification of the optimization process proposed in [LS96], which we summarize here for completeness. This step "inflates" the planar sketch by assigning suitable Z-coordinates to each vertex of the graph of the object, which are determined using a set of geometric properties. Some properties used are planarity of faces, parallelism between edges, etc. The general idea of this method is to duplicate the properties available in the 2D sketch in the 3D object. Each constraint is expressed as a factor relating a 2D property to a 3D property. A compliance function $F(Z)$ is computed for an 3D configuration by summing the contribution of the factors. The final compliance function to be optimized takes the form

$$F(Z) = W \sum A$$

where A is the vector of all factors and W is a weighting function. This is an n-dimensional optimization problem. We use Brent's minimization technique to solve it as a set of 1-dimensional optimization problems by cycling through all vertices. For a detailed discussion on factors and formulation of the problem, please refer to [LS96].

A critical issue is the dependence of the result of optimization on the initial guess. If all Z values are initialized to 0, incorrect local minima are often reached, which is visually indicated by a deformed or collapsed reconstructed object (Figure 6 (a) top). The user can provide hints (dotted lines) in the sketch by specifying hidden edges. If such a hint is available, then the object is divided into 3 Z-layers. One layer consists of vertices to which only hidden edges are incident ($Z = -10$, say), the second layer consists of vertices to which only visible edges are incident ($Z = 10$, say) and a third layer consists of all remaining vertices on silhouettes $(Z = 0)$, as illustrated in Figure 6 (a) bottom. This partially inflates the object and our tests show that this produces a better initial guess leading to fewer cases of convergence to incorrect local minima. It is important to note that 3D information of vertices of existing objects to which new edges are not incident is retained, and hence 3D reconstruction is incrementally performed.

The final representation of the object is a graph with augmented 3D information. This representation is similar to a boundary representation (B-Rep). When the user wishes to sketch from a new viewpoint, all objects are re-projected onto the current viewing plane. These projections are used to determine if strokes drawn augment existing objects or create new ones, and for object selection.

7. Other Features

7.1. Feedback system

The sketch recognition system may occasionally fail. SMARTPAPER offers visual feedback for the user to determine the source of error and repair it. The user explicitly invokes the feedback system. There are two types of feedback.

The clustering threshold mentioned in section 5.2 leads to an incorrect interpretation of the sketch if it clusters too many or too few vertices. This is possible if the object is sketched such that different vertices appear too close or if end points of different strokes meant to be the same vertex are too far apart. Figure 7(a) shows how a sketch looks if clustering is wrong. Figure 7(b) shows the result of changing the threshold value and previewing the changes.

If the order of drawn edges is random, then the edge coherence algorithm finds faces incompletely/incorrectly. Figure 7(c) shows an exploded view of the recognized object showing incorrect faces. The Modified Dijkstra's algorithm can be switched to and results previewed immediately, as is illustrated in Figure 7(d). Our tests show that most erroneous recognitions are caused by the above two miscalculations and thus, the feedback system is very effective.

7.2. CSG Operations

SMARTPAPER supports limited CSG operations, such as cutting and joining.

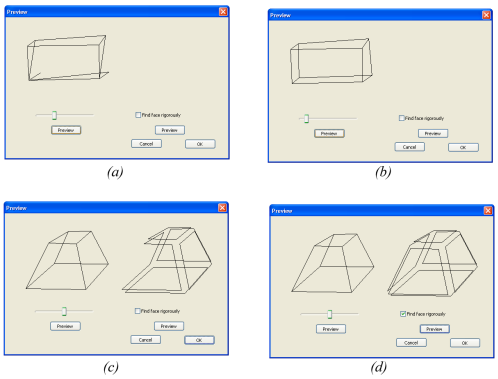


Figure 7: *Feedback system: (a) incorrect clustering of a sketch, (b) the correct clustering after the clustering threshold is decreased by moving the slider, (c) incorrectly determined faces, (d) the correct faces when the Modified Dijkstra's algorithm is selected by checking the option.*

Cutting

The user can specify cutting in two ways. A cutting plane can be directly drawn on the object. Alternatively, an open or closed profile and an extruding arrow can also be drawn. Due to the unified preprocessing in the pipeline, over tracing is allowed in these specifications as well.

Because our representation scheme is similar to B-reps, the actual cutting algorithm is similar to that proposed in [Män86]. The user strokes are converted into a 2D graph after preprocessing. A ray is cast from the eye position into the scene through both end vertices of each edge. For the extruding arrow, the shaft edge is used to determine the direction of extrusion by ray casting. A set of cutting planes is obtained and a set F of faces created by them is determined. An algorithm similar in idea to that discussed in [Män86] then completes this operation.

Joining

Two objects can be joined face-to-face by selecting each object and then selecting the face that is to be joined to the selected face of the other. A simple coordinate transformation "sticks" the two faces together. Now one object can be translated in the plane of the selected face or rotated about its normal for positioning. The join operation joins the two objects after the user commits this positioning. Since objects are drawn roughly, two faces are seldom congruent to each other. Therefore the user can choose to deform one of the selected faces so that it coincides seamlessly with the other selected face. Alternatively, the user can choose to simply stick the two faces without deformation. This is desirable when the two faces are meant to be different in size, like those of a table top and a rectangular leg while constructing a table.

7.3. Rendering

Non-photorealistic rendering techniques are employed to retain the sketchy look of an object in order to remind the user of its ambiguous shape and dimensions [KMM*02]. We extend the approach taken by [ZHH96] to render the recognized objects in sketchy styles. The rendering is achieved in two passes. In the first pass, faces of all objects are rendered as colored polygons and the depth buffer is modified accordingly. In the second pass, all the edges are rendered as textured quads with the depth testing turned on. Different NPR styles are achieved by changing the stroke textures, as illustrated in Figure 8.

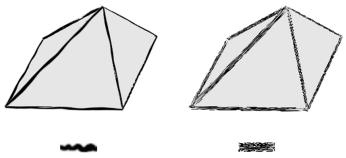


Figure 8: *Various NPR styles. The thumbnails below each figure show the stroke texture used for the corresponding NPR style.*

8. Discussion

SMARTPAPER has been implemented on the Toshiba Portégé 3500 Tablet PC. The programming has been done using Microsoft Visual C++ .NET with the Tablet PC SDK on the Windows XP platform.

SMARTPAPER is a design-by-sketches system for applications like architectural design. A user meeting and discussion was held with a group of 10 students and one faculty member from the Department of Architecture. The users experimented with SMARTPAPER and appreciated its ability to allow sketching freely without much learning. The idea of drawing by gestures, which seemed somewhat nonintuitive to us initially, found support among our users, due to which it was incorporated in SMARTPAPER. They found the Tablet PC environment a very refreshing alternative to the mouse-based CAD experience that some of them had. Some suggestions regarding support for architectural drawing conventions were made, which we have striven to incorporate in SMARTPAPER.

In summary, SMARTPAPER offers a natural and effective sketching experience because of its support for natural drawing styles, easy and intuitive specification of operations, integrated 2D and 3D drawing domains, active user involvement in 3D reconstruction by providing feedback, and its implementation on the Tablet PC. This does away with the

artificial fully menu-based and mouse operated CAD interface.

9. Future Work

Some limitations and ideas for future work are being worked upon. Currently, all drawn edges must be straight lines. We plan to support the drawing of curved objects by recognizing curves as primitives in strokes and maintaining their characteristics in the 3D reconstruction process.

In order to sketch an object directly the user needs to draw all edges, whether visible or invisible from the assumed viewpoint. Drawing by extrusion already circumvents this problem. We strive to devise methods to relax this constraint [LB90].

The problem of the optimization procedure reaching a local minimum would be investigated further. Alternatives for minimizing this problem include using genetic algorithms for solving the optimization, but these are usually very slow. A few other optimization methods are being experimented with.

We recognize the power of gestured drawing shown by existing systems like SKETCH [ZHH96], and plan to investigate further in this direction while striving to maintain a balance between direct and gestured sketching. Gestures will be used to specify geometric properties like parallelism and perpendicularity in rough sketches to obtain more regular shapes. For example, a bracket can be drawn between two edges to show mutual perpendicularity, etc.

The sketch reconstruction and interpretation are critically dependent on a set of thresholds and constants, like the clustering threshold. We plan to build a threshold management system to maintain individual user profiles and drawing styles through these parameters and dynamically change them by learning from user sketching styles and habits. This will make the overall drawing process more adaptive to individual users. In general, we plan to conduct more comprehensive usability studies to improve upon our current interface design. In particular, we wish to make the feedback system more intuitive to use and transformation and placement of objects easier to perform during construction of complex environments.

In order to focus more on our target application of architectural design, we plan to incorporate more domain-specific drawing knowledge obtained from our user sessions. For example, architecture designers use construction lines in drawing, which do not have any physical meaning but merely act as scaffolding to the actual object sketch. We plan to support the user of such construction lines.

We continue to draw inspiration from Tablet PC devices and are limited by their hardware capabilities. We plan to develop more strategies leveraging hardware capabilities for a better drawing experience as and when these limitations are minimized by the advent of newer and more powerful Tablet PC devices.

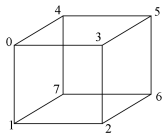


Figure 9: *Edge coherence algorithm: edges are drawn in order 01-12-23-30-04-45-35-56-62-47-76-71 (a typical way of drawing a cube). The first pass determines faces 01-12-23-30 and 04-45-35-03 and partial faces 56-62, 47-71 and 76. The second pass completes these faces.*

The ability to draw directly on existing 3D models offers a plethora of ideas for future design applications. Models of existing sites obtained from 3D scanning can be used in conjunction with SMARTPAPER to design buildings directly in their target sites. Efforts to scan and process such environments are being independently pursued by our research group. We envision SMARTPAPER to be used in such design applications in future.

10. Acknowledgements

Support for this work includes a University of Minnesota Digital Technology Center Seed Grant 2003, a Microsoft Gift, and NSF CAREER ACI-0238486. We thank Dr. Andrzej Piotrowski for his resourceful comments and evaluation on our system and Dr. Ravi Janardan for his inputs towards design of the Modified Dijkstra's algorithm.

Appendix

Pseudo code for the Edge Coherence algorithm:

Input: Set of edges in order specified in S

Output: Complete set of faces L and incomplete set of faces V

Pass 1:

//for each edge e in S if the next edge in S is connected to it, add in temporary face F, and if F is closed, add F to L.

//If next edge in S does not have a common vertex with e, look ahead "k" steps for such an edge. If such an edge is found, add in temporary face F and if F is closed, add F to L, else add F to V.

Pass 2:

//Construct sub-graph G', *such that* $V(G') = V(G)$ $E(G') = e : e \in E(G) and e is not part of 2 faces$

//For every edge $e \in E(G')$, *if* $e = (v_1, v_2)$, *then find a shortest* v_1, v_2*-path in* $G' - e$. *This forms a cycle with e in* G'. *If e is now part of two faces, delete e from* G'.

Pseudo code for the Modified Dijkstra's algorithm:

S: Set of edges that are not part of at least 2 faces R: Set of edges that are part of at least 2 faces M: Matrix having faces as rows and edges as columns.

//Initialize S to all edges that are not part of at least 2 faces. If a current object is being augmented, this set is a proper subset of the edge set of the graph.

while S is not empty do

$e \in S, e = (v_1, v_2)$

//if $e' \in R$ shares an end vertex with e, then mark end vertices of e' as traversed

//Find all edge-disjoint v_1, v_2-paths, mark M accordingly.

//Traverse M column-wise and mark all edges are that are part of more than 2 faces. Then traverse M row-wise to delete all faces consisting of only marked edges.

$G = G - e$ end while

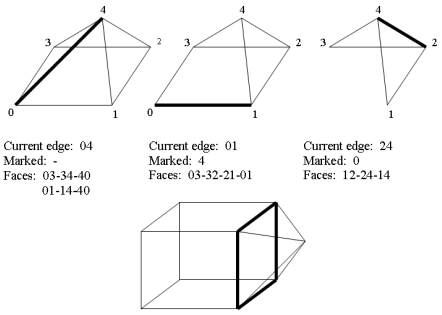


Figure 10: *Modified Dijkstra's algorithm. The top figures show iterations. The edge for each iteration is shown by a bold line. The bottom figure shows how an "internal face" may arise by first drawing the cube and then augmenting the pyramid to it.*

References

[GD96] GROSS M. D., DO E. Y.-L.: Ambiguous intentions: a paper-like interface for creative design. In *Proc. UIST 1996* (1996), pp. 183–192.

[IH01] IGARASHI T., HUGHES J. F.: A suggestive interface for 3d drawing. In *14th Annual Symposium on User Interface Software and Technology, ACM UIST 2001* (2001), pp. 173–181.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: A sketching interface for 3d freeform design. In *Proc. SIGGRAPH 1999* (1999), pp. 409–416.

[KMM*02] KALNINS R. D., MARKOSIAN L., MEIER B. J., KOWALSKI M. A., LEE J. C., DAVIDSON P. L., WEBB M., HUGHES J. F., FINKELSTEIN A.: Wysiwyg npr: Drawing strokes directly on 3d models. In *Proc. SIGGRAPH 2002* (2002), pp. 755–762.

[LB90] LAMB D., BANDOPADHAY A.: Interpreting a 3d object from a rough 2d line drawing. In *Proc. IEEE Visualization 1990* (1990), pp. 59–66.

[LM95] LANDAY J. A., MYERS B. A.: Interactive sketching for the early stages of user interface design. In *Proc. SIGCHI 1995* (1995), pp. 43–50.

[LS96] LIPSON H., SHPITALNI M.: Optimization-based reconstruction of a 3d object from a single freehand line drawing. *Journal of Computer Aided Design 28*, 8 (1996), 651–663.

[LS00] LIPSON H., SHPITALNI M.: Conceptual design and analysis by sketching. *Journal of AI in Design and Manufacturing (AIEDAM) 14* (2000), 391–401.

[LS02] LIPSON H., SHPITALNI M.: Correlation-based reconstruction of a 3d object from a single freehand sketch. *AAAI Spring Symposium on Sketch Understanding* (2002), 99–104.

[Män86] MÄNTYLÄ M.: Boolean operations of 2-manifolds through vertex neighborhood classification. *ACM Transactions on Graphics 5*, 1 (1986), 1–29.

[PMC03] PIQUER A., MARTIN R., COMPANY P.: Using skewed mirror symmetry for optimisation-based 3d line-drawing recognition. In *Proc. 5th IAPR International Workshop on Graphics Recognition* (2003), pp. 182–193.

[SL96] SHPITALNI M., LIPSON H.: Identification of faces in a 2d line drawing projection of a wireframe object. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 18*, 10 (1996), 99–104.

[SL97] SHPITALNI M., LIPSON H.: Classification of sketch strokes and corner detection using conic sections and adaptive clustering. *ASME Journal of Mechanical Design 119*, 2 (1997), 131–135.

[TDM01] TOLBA O., DORSEY J., MCMILLAN L.: A projective drawing system. In *Proc. I3D Symposium on Interactive 3D Graphics* (2001).

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