The ModelCraft Framework: Capturing Freehand Annotations and Edits to Facilitate the 3D Model Design Process Using a Digital Pen

HYUNYOUNG SONG and FRANÇOIS GUIMBRETIÈRE
University of Maryland
and
HOD LIPSON
Cornell University

14

Recent advancements in rapid prototyping techniques such as 3D printing and laser cutting are changing the perception of physical 3D models in architecture and industrial design. Physical models are frequently created not only to finalize a project but also to demonstrate an idea in early design stages. For such tasks, models can easily be annotated to capture comments, edits, and other forms of feedback. Unfortunately, these annotations remain in the physical world and cannot easily be transferred back to the digital world. Our system, ModelCraft, addresses this problem by augmenting the surface of a model with a traceable pattern. Any sketch drawn on the surface of the model using a digital pen is recovered as part of a digital representation. Sketches can also be interpreted as edit marks that trigger the corresponding operations on the CAD model. ModelCraft supports a wide range of operations on complex models, from editing a model to assembling multiple models, and offers physical tools to capture free-space input. Several interviews and a formal study with the potential users of our system proved the ModelCraft system useful. Our system is inexpensive, requires no tracking infrastructure or per object calibration, and we show how it could be extended seamlessly to use current 3D printing technology.

Categories and Subject Descriptors: H.5.2 **Information Interfaces and Presentation**: User Interfaces—*Input devices and strategies*

General Terms: Design, Experimentation, Human Factors

Early results of this work appeared in *the Proceedings of the UIST'06* [Song et al. 2006].
This research was supported in part by the National Science Foundation under Grants IIS-0447703, IIS-0749094 and by Microsoft Research (as part of the Microsoft Center for Interaction Design and Visualization at the University of Maryland) and a graduate fellowship from the Department of Computer Science at the University of Maryland.
Authors' addresses: H. Song (contact author), F. Guimbretière, Department of Computer Science, University of Maryland, MD 20742; email: hsong@cs.umd.edu; H. Lipson, MAE, Cornell University, Ithaca, NY.

DOI 10.1145/1592440.1592443 http://doi.acm.org/10.1145/1592440.1592443

Additional Key Words and Phrases: Pen-based interactions, tangible interactions, rapid prototyping

ACM Reference Format:

Song, H., Guimbretière, F., and Lipson, H. 2009. The ModelCraft framework: Capturing freehand annotations and edits to facilitate the 3D model design process using a digital pen. ACM Trans. Comput.-Hum. Interact. 16, 3, Article 14 (September 2009), 33 pages.
DOI = 10.1145/1592440.1592443 http://doi.acm.org/10.1145/1592440.1592443

1. INTRODUCTION

In the process of designing artifacts, today's designers alternate between tangible, nondigital media such as paper or physical 3D models, and intangible, digital media such as CAD models. An architect may begin the design of a new building with sketches on paper, then, when her ideas solidify, create a rough model using cardboard, before creating the corresponding digital model. Once this model is finalized, it might be fabricated as a 3D object (either through rapid prototyping techniques or a modeling studio) so that her clients may have a better grasp of her vision. Although a fully digital design process has long been advocated, it still seems a distant goal because tangible, nondigital media models present unique affordances that are difficult to reproduce in digital media. For example, architectural models offer a unique presence that is difficult to reproduce on a screen. As a result, even projects that rely heavily on computer-assisted design techniques still employ tangible models, both for aesthetic and structural tasks (Figure 1, left).

Interacting with models is an intrinsic part of the design process for architects who see construction (and sometimes deconstruction) as a fundamental part of the idea forming process. For example, during the early phase of the design process called "massing," inexpensive, easy-to-build paper-based models are used extensively to better understand the shape requirements of a building. As rapid-prototyping technology has become more commonplace, other designers have employed physical models more extensively during their design process as well. Mechanical designers use models to check the form and functional compatibility of a given design with the context of its use. Frequently, sketches that describe the modification and edits to be made are drawn on the surface of the model (Figure 1, right). With current techniques, information described in this way on such models is difficult to integrate back into the digital world. While such annotations can be captured by a conventional tracking system (either magnetic or optical) as proposed by Agrawala et al. [1995], that approach is limited to a relatively small working volume, requires significant investment in infrastructure and a calibration process on a per-model basis, and is somewhat expensive. It is also difficult to deploy in the field where models are frequently tested. This limits widespread adoption by architects and designers.

Noting that most annotations take place on the surface of the model, we present an extended version of ModelCraft [Song et al. 2006], a system which uses the inexpensive, off-the-shelf Logitech io2^{TM} digital pen [Logitech] to track annotations and edits on the surface of models. This pen is equipped with

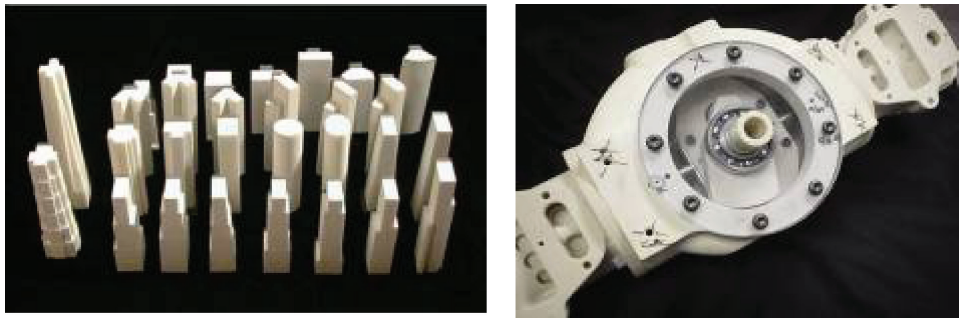

Fig. 1. 3D models are used extensively in design. Left: a structural model used during the design of the Hearst building (from Hart [2005]). Right: annotations on a 3D model from a ZCorp printer (from ZCorp [2005]).

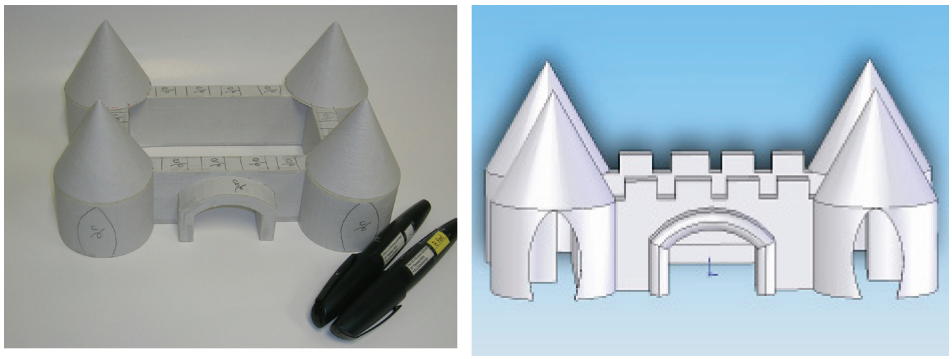

Fig. 2. Our system in action. Left: paper model of a castle with edits. Right: the same model in our rendering application showing the edits performed.

a built-in camera which captures position information by observing a digital pattern [Anoto 2002] printed on the surface of the model (Figure 2). The ModelCraft system, which is currently a plug-in for the commercial CAD application SolidWorks [2005], can capture both annotations and commands that are applied to the original digital model upon pen synchronization. Using our command system and auxiliary tools such as a ruler, protractor, and sketchpad, users can alter and adjust the shape of a model; available operations include modifying dimensions, filleting corners, creating holes, or extruding portions of a model based on requirements learned from the field. ModelCraft also lets users create relationships among several models to create a complex assembly from simpler models. Finally, ModelCraft can deal with nontrivial objects, such as the castle shown in Figure 2, as the expressiveness in the vocabulary of the models, the sizes of the models, and the basic geometry of the models were important issues during testing with architects and for future deployments.

Annotations, edits, and assemblies instructions are naturally captured in the reference frame of the model, without the need to worry about scale or orientation. Our approach does not have a predefined working volume, and

can easily scale in terms of the number of objects traced, number of pens used, and locations of usage. Furthermore, it does not require a per-model calibration. ModelCraft integrates seamlessly with the current usage patterns among architects and mechanical designers. By capturing annotations and edits on physical 3D models, our system streamlines the design process, simplifies documentation of the design history of a given project, and supports design education [Song et al. 2007]. This vision of streamlining the design process will be completed if the traceable pattern is printed as the physical models are constructed. We present in detail possible paths and challenges to be solved for the implementation of such a system using current 3D prototyping technology.

2. PREVIOUS WORK

Our work extends the idea of capturing and tracking sketches on the surface of a 3D object (Section 2.1). As we allow the user to not only annotate but also use sketches as commands (Section 2.2), our system selectively borrows ideas from sketch-based systems. While most sketch-based systems have an indirect relationship between the 3D representation and the sketch input, our system provides the user with an actual physical proxy when executing an operation, similar to many Tangible User Interfaces (TUI) (Section 2.3).

2.1 3D Painting Systems

Several systems have been proposed that allow users to draw (or paint) on digital models. Hanrahan and Haeberli [1990] described a WYSIWYG system to paint on 3D models using a standard workstation. This approach has also been adapted to annotate CAD drawings [Jung et al. 2002; Solid Concepts, Inc. 2004]. While drawing on a virtual object has many advantages, such as the ability to work at any scale, we believe that physical models will always play an important role in the design process because of their appeal to designers (Figure 1). In that respect, our system is closely related to Agrawala et al.'s [1995] 3D painting system. Our approach extends this work in several ways: By using a tracking system based on an optical pattern printed on the surface of the object, we offer a very short setup time requiring no calibration on a per-object basis. Our tracking approach also provides greater flexibility for users as annotations can be captured at any location. Finally, our approach is inherently scalable, both in the number of models and annotating devices, a property difficult to achieve by either optical or magnetic tracking techniques. Using a different approach, several systems propose using augmented reality techniques to annotate objects [Grasset et al. 2005]. On one hand, by relying on passive props, our system is less powerful than such systems as it does not offer direct feedback. On the other hand, the simplicity of our system keeps its cost of use low; there is no need to wear or set up any equipment, a key aspect for acceptance by designers and architects.

We believe that future systems should allow users to interact directly with the representation of a given object that is most convenient for the task at hand, be it a digital model on a screen or a 3D printout of that model, or a combination of the type of sketching proposed here with augmented reality feedback. In that respect, our work is similar in spirit to the work by Guimbretière [2003] on digital annotation of document printouts.

2.2 Sketch Interfaces

Our work is also related to the large body of work on 3D sketching in systems like Sketch [Zeleznik et al. 1996], Teddy [Igarashi et al. 1999], SketchUp [Google 2006], and the 3D Journal project [Masry et al. 2005]. These systems interpret 2D sketches and transform them into 3D volumes by using curvature information [Igarashi et al. 1999], by using the angular distribution of the strokes to predict the three axes [Masry et al. 2005] or by providing an interactive toolkit [Google 2006]. ModelCraft complements these systems by addressing the need to capture modifications sketched directly on the models at later stages of the design process. In particular, our approach makes it easy for users to capture real-world geometric information such as the length or angle of the surrounding physical context. Our command system is also quite extensive and accommodates a variety of different operations. While the systems mentioned before focus on a gesture-based interface, we adopt a syntax-based approach inspired by recent work on Tablet-PC-based interfaces such as Scriboli [Hinckley et al. 2005], Fluid Inking [Zeleznik et al. 2004], and paper-based interfaces such as PapierCraft [Liao et al. 2005]. We believe that this approach allows for a more flexible and extensive command set while retaining a sketching-like style.

2.3 Tangible User Interfaces

Our work is also closely related to tangible interfaces [Hinckley et al. 1994; Ishii and Ullmer 1997; Ullmer and Ishii 1997; Underkoffler and Ishii 1999, 1998], which let users interact with digital information through the use of tangible artifacts. All these systems leverage users' familiarity with spatial interactions to allow them to perform complex interactions with ease. Our system extends and complements these systems by offering a tight correspondence between the tangible proxy and its digital representation. In doing so, we offer users the opportunity to modify the digital representation in the real world. In that respect, our system is also closely related to the Illuminating Clay system [Piper et al. 2002] and Liu's work on editing digital models using physical material [Liu 2004], as they allow users to see modifications made in the real world applied to the equivalent 3D model. Approaches that model shapes using fingers and physical props [Sheng et al. 2006] or using hand gestures [Schkolne et al. 2001] are also related to our approach, but focus on constructing free-form shapes such as clay models. All of these systems require the use of somewhat complex tracking equipment that is only available in a lab setting, while our approach is very lightweight.

Block-based tangible interfaces [Anderson et al. 2000; Sharlin et al. 2002] propose to extend standard building blocks with a set of sensors so that it is

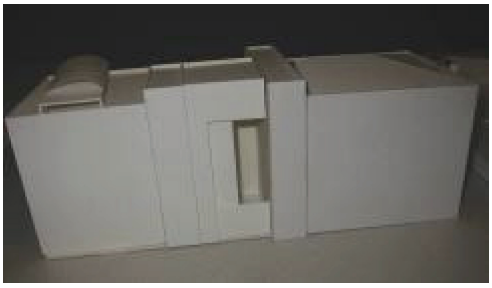

Fig. 3. A typical paper-based massing model used by an architectural firm to refine the shape of a building.

possible for the system to sense the relationship between the different blocks connected to each other. The relationship is then used to render the connected blocks on a nearby computer. Digital block interfaces trade ease of assembly for a somewhat limited vocabulary of possible shapes and connections (which have to follow the orientation predefined by the connection mechanism). Here we explore the opposite end of the expressiveness versus ease of assembly trade-off. In ModelCraft, a wide variety of building blocks can be assembled in complex configurations, but ModelCraft requires users to actively mark the connectivity information in addition to physically connecting them together. We believe that our work will provide the opportunity to identify which approach fits best for different design tasks.

3. THE SYSTEM IN ACTION

ModelCraft is implemented as a plug-in to the CAD program SolidWorks [2005] that helps users create a traceable 3D model from a virtual model and integrates the captured strokes from the physical model back to the original virtual 3D model. The plug-in produces traceable 3D models by printing (using a standard desktop printer) the 2D layout of a 3D model on top of one or more sheets of Anoto pattern [Anoto 2002] paper. This pattern provides a very large space of uniquely identifiable pages (in excess of 2^{48} letter-sized pages). This makes it possible to interact with a large number of printed objects at once, including different printouts of the same object. The pages are then folded along guidelines into a physical representation of the model. While building a paper model seems arduous, interviews with architects confirmed that they often build models out of paper (Figure 3). For example, architects sometimes opt to create a 2D printout of an unfolded 3D CAD model and then cut it out using a laser cutter. Hence our pattern mapping process on 2D unfolded layout augments the current practice of creating 3D models by folding 2D printout. Practical paper models are usually quite simple since early designs often rely on a vocabulary of basic shapes (cube, cylinder, pyramid, cone, sphere) as proposed by Ching [1996]. For more complex shapes, the traceable pattern can be applied directly on top of a model printed with a 3D printer by using a water transfer decal printed with the pattern. This approach is supported by our current system, which provides a way to generate pattern patches for each face and a

map to simplify assembly. Our vision is to create models using a 3D printer and have a traceable pattern generated automatically during printing. Automatic pattern mapping is currently a work in progress that will be described in Section 5.3.1.

All interactions are carried out with the Logitech io2™ pen [Logitech 2005], a commercial implementation of the Anoto system [Anoto 2002]. As each Anoto digital pen has a unique ID, it is possible to distinguish between several different pens interacting with one object. Our system also lets us designate special objects as tools. For example, we instrumented conventional design tools such as rulers, protractors, and a sketchpad by taping a strip of Anoto pattern onto them (Section 3.3). Marks made on these tools are used by the system as measurements and guidelines when entering commands.

3.1 Annotations

Annotating a model is straightforward: The user simply picks up the model and writes directly on any surface. Annotations can be characters, marks, or guidelines for a shape. Upon pen synchronization, the marks will be merged onto the corresponding surface of the SolidWorks model. Users can use several pens for different colors of annotations. Marks created by annotation pens are not interpreted by the system as commands.

3.2 Form Editing Command Syntax

The pen can also be used to capture edit commands to be executed directly on the digital CAD models. Our objective is not to replace the standard (and far more accurate) CAD construction process, but instead to address two disparate needs. First, in the early stages of design, a rough prototype is often sufficient to present or verify a designer's idea. For instance, if after 3D printing it is found that a piece conflicts with another element in the design, simply marking the conflicting area and cutting it away may be all that is needed. Second, we found that when a large number of marks are made on the prototype, it is somewhat difficult upon review to understand how the marks relate to each other. In that context, providing tentative feedback to the executed operations helps the users to understand the structure of the marks. Furthermore, as all annotations and command parameters are created as first-class objects inside the SolidWorks features tree, they can easily be modified inside SolidWorks [2005]. A simple update of the model will automatically reflect any such changes.

All commands are performed with a command pen which lays ink in a different color (red or pencil lead in the figures). We choose a "command" pen approach as it fits well with the current practice of using color-coded annotations. Other solutions such as having a command mode, triggered by a button on the pen, are also possible.

All commands follow a uniform syntax (Figure 4) inspired by Scriboli [Hinckley et al. 2005] and PapierCraft [Liao et al. 2005]. To issue a command, users first draw the necessary parameters directly on the surface of the model (Figure 4(a)). Next, they draw a pigtail gesture, which is used as a separator between the parameter strokes and the command identifier (Figure 4(b)). Next,

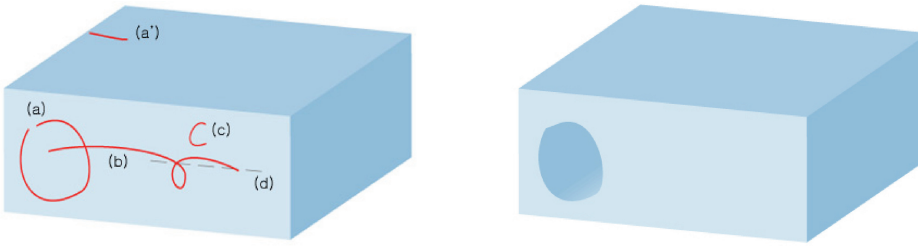

Fig. 4. Command syntax for editing a single object. Left: (a) main parameter; (a') additional parameter (in this case, the depth of the cut); (b) pigtail delimiter; (c) command name; (d) reference line for character recognition. Right: the result of command execution.

they indicate which command they wish to execute by drawing a letter or a simple word on top of the pigtail (Figure 4(c)). During pen synchronization, the command is then executed using the area on which the pigtail started as the primary shape parameter. For example, to create a hole though an object, the user would draw the shape of the hole on the object's surface, then draw a pigtail starting inside the shape, and then write a C (for cut) on top of the pigtail. Starting the pigtail outside of the shape complements the selected area and would have created a cylinder instead.

The use of the pigtail proved to be very reliable for pen-based interaction [Hinckley et al. 2005] and is well adapted to our case, as it does not require any feedback besides the ink laid on the surface [Liao et al. 2005]. For our system, the pigtail has two advantages. First, it serves as a natural callout mark when one needs to execute a command on a small area such as cutting a hole for a screw. Under such conditions, it would be difficult to write the name of the command directly on the area of interest because the area is too small or too close to the surface border. Second, the pigtail provides a natural orientation for the surface. While up and down are well understood in a Tablet-PC context, this is not the case on 3D objects, which people may place in arbitrary orientations to facilitate the annotation process. Accordingly, when interpreting a command, we consider the pigtail as the baseline for the command name (Figure 4(d)).

As shown in Figure 4, some operations may require several sketches to be drawn on more than one surface. For example, to create a cut of a given depth, the user first creates the shape of the cut, then marks the depth of the cut on another face, and then uses a pigtail to issue the cut command. As shown in Figure 6, this syntax makes it very easy to use real-world objects as references, without the need for further measurements. Another example is the creation of a groove on an object (Figure 5). In that case, the user first draws the profile of the groove on one surface and then the extent of the groove on an adjacent surface, then uses a pigtail to indicate the inner region, and finally writes a G (for "groove") on top of the pigtail.

Our system can accept closed paths drawn with several strokes on multiple surfaces as the main parameter of the syntax. During path construction, we consider each stroke as a separate spline without attempting to smooth sharp edges between strokes. This offers a flexible way to draw a wide variety of shapes. For example, a hexagon can be created by a set of six strokes while a

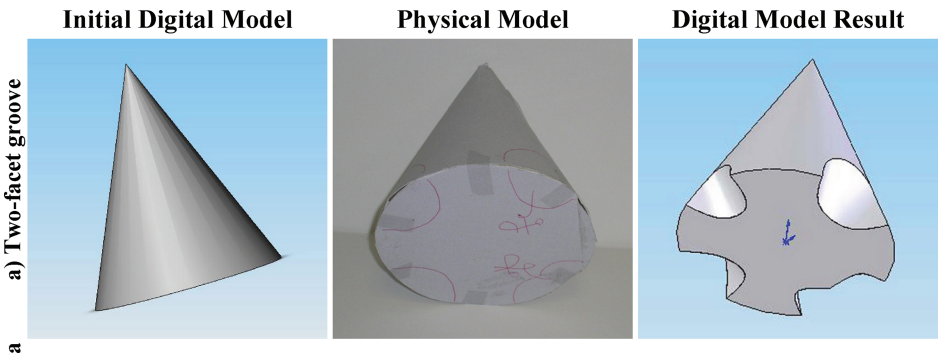

Fig. 5. The parameters are specified on two adjacent surfaces for a *groove* operation.

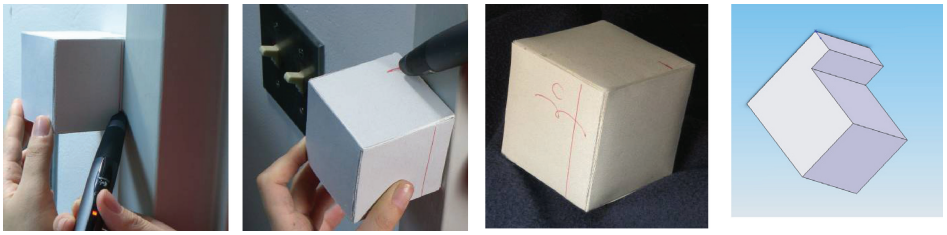

Fig. 6. Using external references to perform command: a cube is cut and extruded to fit a door frame. First we mark the thickness of the frame and then the width before executing.

smooth circle can be created using one stroke. This feature can be used to create complex, multifaceted grooves, as shown in Figure 7.

In some operations, a component of the model such as a face or an edge can serve as a main parameter. The desired component is indicated by the beginning of a pigtail. For example, to perform a shell operation (Figure 8), which creates a shell given a volume, the user selects a face using the pigtail and then writes the command on top of the pigtail. A similar sequence applies to the fillet operation, which smoothes a selected edge, in that the starting point of the pigtail is used to pick the edge of interest.

3.3 Augmented Tools

One of the main limitations of using the Anoto pattern as a tracking system is that it cannot track in free space, as the patterns are only mapped on the surface of a physical object. In other words, operations that use auxiliary parameters defined in free space around the model are not directly available. To address this problem, ModelCraft relies on the use of tools that have been augmented with a digital pattern.

The simplest tool is an augmented ruler used in conjunction with the extrude command. To extrude a shape from a surface, one draws the shape on the surface, then draws a mark on the ruler to indicate the extrusion length in the direction orthogonal to the surface, and finally, using the pigtail, issues the extrude command on the area to be extruded. As in the case of cutouts,

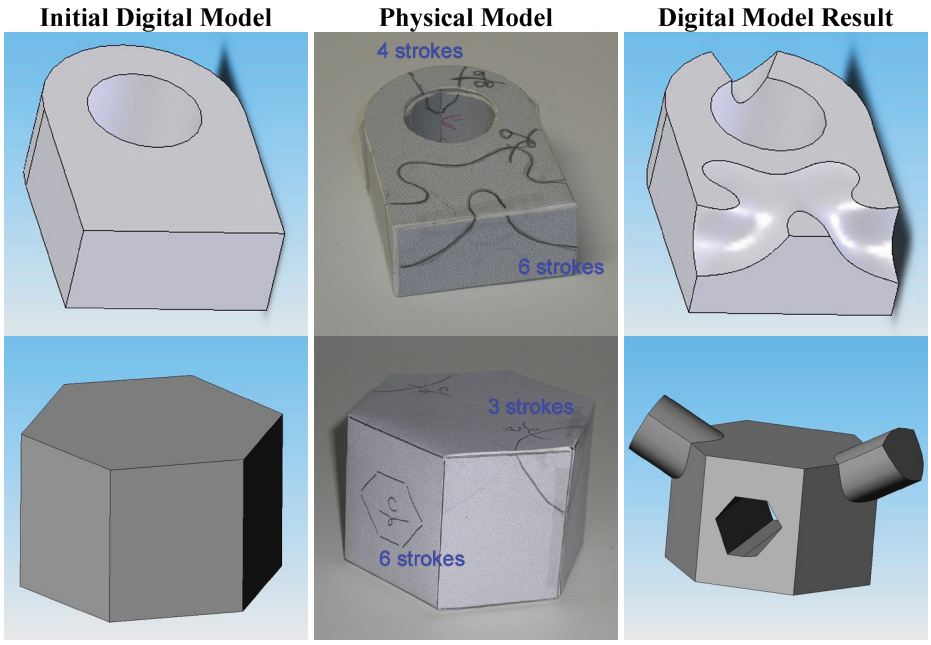

Fig. 7. The main parameter can be defined on multiple surfaces to cut, groove, or extend a portion of the model. The hexagon shape was drawn from several strokes to preserve its angles.

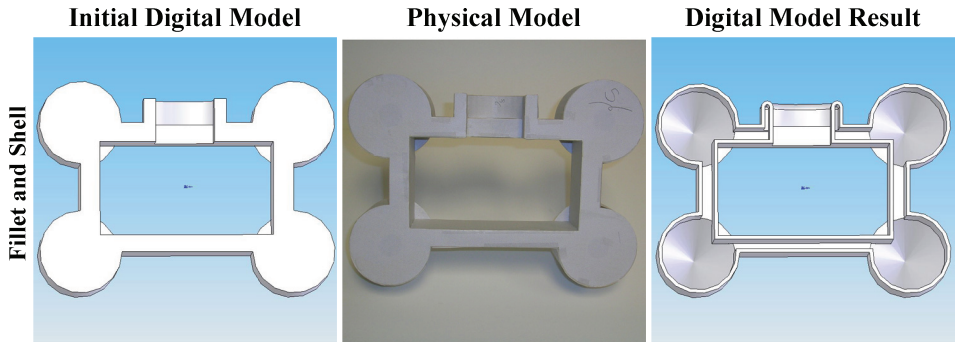

Fig. 8. Parameters are needed on only one surface for *shell* and *fillet* operations.

this command facilitates the process of using real-world objects as references (Figure 9).

Another useful tool in ModelCraft is an augmented protractor. While the ruler only allows for extrusion perpendicular to the surface, the protractor lets users specify the extrusion direction. Like the ruler, the protractor was created by printing out a protractor shape on a piece of Anoto paper mapped in our system as a special "protractor" area. The command syntax for the protractor is very similar to that of the ruler. First, users draw the shape they want to extrude on a surface. Then they indicate the direction and length of the extrusion using the protractor they want to consider by simply drawing the corresponding

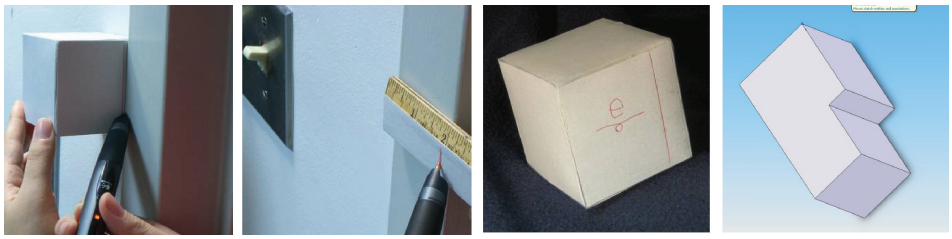

Fig. 9. Using external references to perform a command: one side of a cube is extruded to cover a door frame. First we mark the thickness of the frame and then use a ruler to mark the width of the frame before execution.

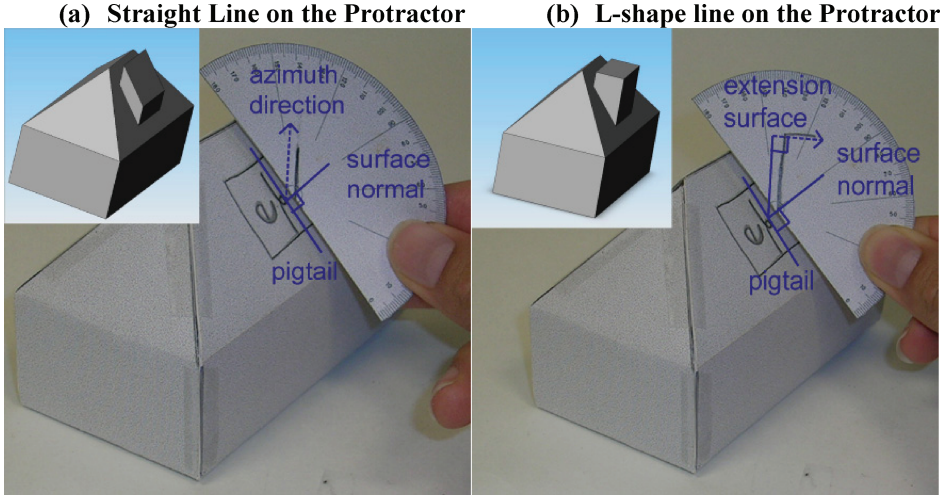

Fig. 10. An augmented protractor is used to specify the direction and extent of extrusion.

line on the protractor. Finally, users issue the extrude command on the target area using the pigtail as the baseline. In this case, the direction of the pigtail is important, as it indicates the orientation of the protractor on the surface. By convention, it is assumed that the base of the protractor is aligned to the direction of the pigtail (Figure 10). Using this syntax, it is possible to extrude a shape in any direction by indicating the azimuth in combination with the pigtail. The digital representation will be a simple extrusion like in Figure 10(a) with the top of the chimney parallel to the roof. It is also possible to create an extrusion with a face orthogonal to the extrusion direction by simply using an "L" shape instead of a straight line for the direction and length of the extrusion, as shown in Figure 10(b).

A similar syntax can be used for creating a sweep of a given shape along a planar curve using the *sweep sketchpad*. Using this tool, a user can transform the cone shown in Figure 11 into a teapot with the following operations: First, the user draws a cross-section of the handle on a cone (Figure 11(a)), then draws the path of the handle on the sketchpad (Figure 11(b)), and finally, using the pigtail (Figure 11(c)), issues the extrude command (Figure 11(d)) on the area

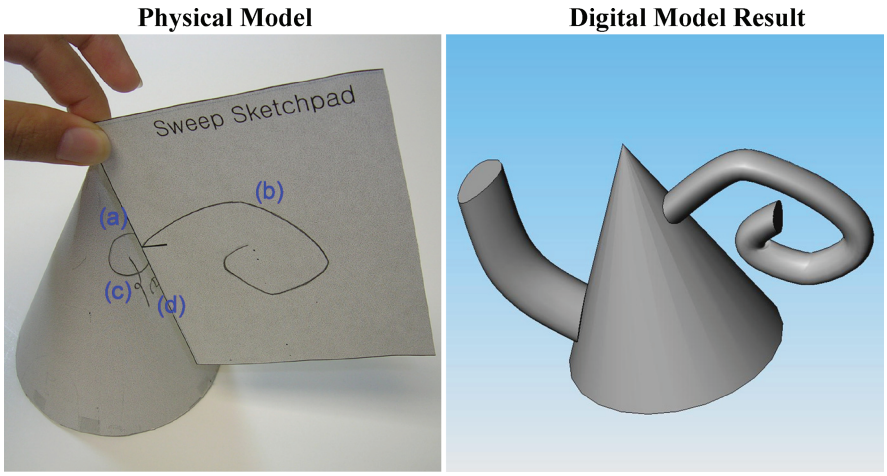

Fig. 11. The sweep sketchpad is used to define the path of a sweep-like extrusion.

to extrude. Like the protractor, the normal vector on the model surface and the pigtail is used to define the sweep sketchpad plane in which the sweep trajectory lies (Figure 11).

3.4 Multiple Model Assembly Syntax

Our interviews with architects suggested that operations across multiple objects will be very useful in early design phases, as architects often create new designs by stacking or joining available building blocks. Previous systems featuring tangible building blocks [Anderson et al. 2000; Suzuki and Kato 1995; Watanabe et al. 2004] exhibit a Lego™ -like connection system, which makes it easy to connect the blocks. Yet, these systems limit the shape of the building blocks and the directions of connections. Instead, we decided to emphasize connection flexibility, which is more important in the early phases of design. Our approach is based on the idea that people will first assemble the shape they wish to create, and then register the resulting arrangements by creating "stitching" marks between the different blocks. This idea was inspired by the PapierCraft system [Liao et al. 2005] and Stitching [Hinckley et al. 2004], both of which use a similar approach to allow people to create larger documents from smaller display surfaces using pen marks.

We first consider the simple case in which the two objects to be glued together share a common face and a common edge. In this particular case, the assembly can be registered using a simple stitching stroke across the common edge (Figure 12(a)). In this configuration, ModelCraft uses the coincident point (the point at which the stroke jumps from one object to the next) to register and set the relative position of the two objects (*point constraint*, fixes 3 degrees of translational freedom). The relative orientation of the two objects (3 additional degrees of rotational freedom) is then determined using the following conventions: First we assume that both edges at the intersection with the stitching mark will be collinear (*edge constraint*, fixes 2 degrees of freedom), finally we

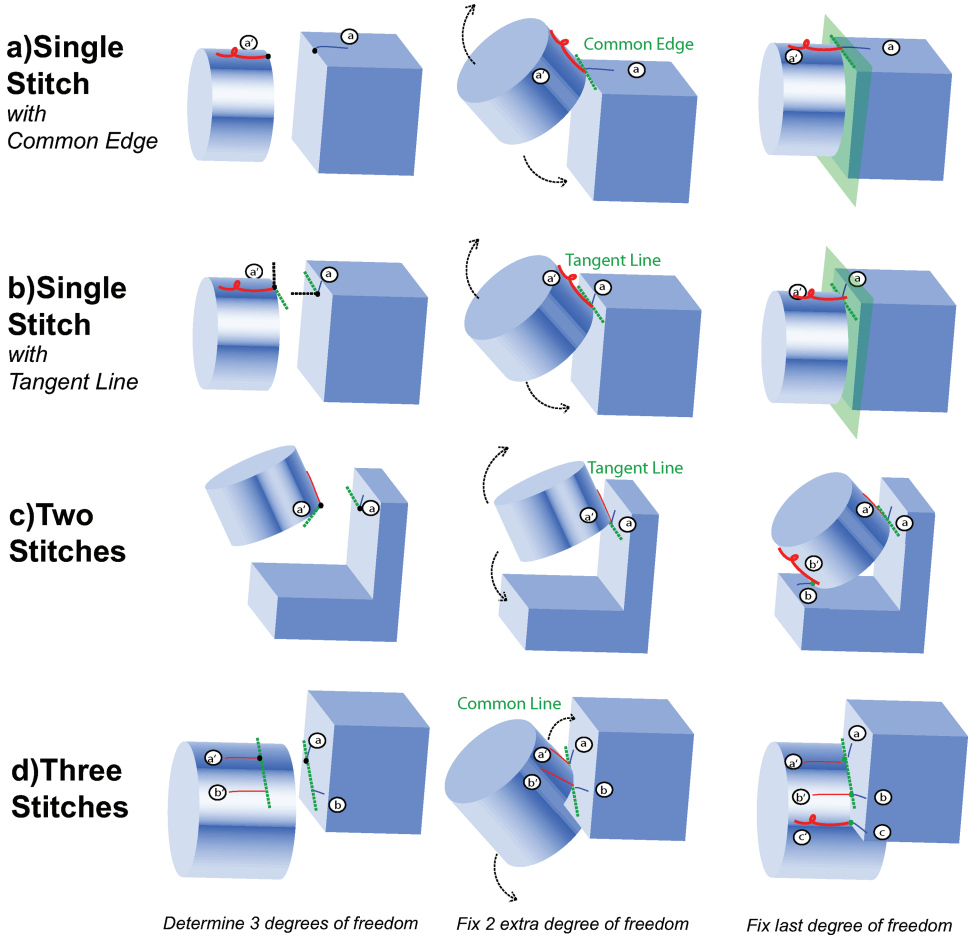

Fig. 12. Command syntax for stitch operations in multiple objects.

assume that the normals of the shared faces are collinear (*plane normal constraint*, fixes the last degree of freedom). This interaction is very convenient for simple assembly operations in which speed is valued over flexibility (Figure 13). It is also possible to use a similar syntax if one wishes to glue one object on another without a common edge (Figure 12(b)). In such a scenario, we create the equivalent of the edge constraint as follows: For the segment of the stitching mark not crossing an edge (grey ink in Figure 12(b)), we compute the cross product of the stitching mark segment and the surface normal, referred to here as the *tangent line*. This tangent line is then aligned with the edge of the other model (edge crossed by dark grey ink in Figure 12(b)).

Users can also assemble two objects that do not share a face (Figure 12 (c)). This can be accomplished via the use of either two or three stitching marks. When a user draws two stitching marks across two models, the first stroke is used to infer the two points and two edges to be used for aligning the objects. Once again, the points on the models where the stroke jumps from one object

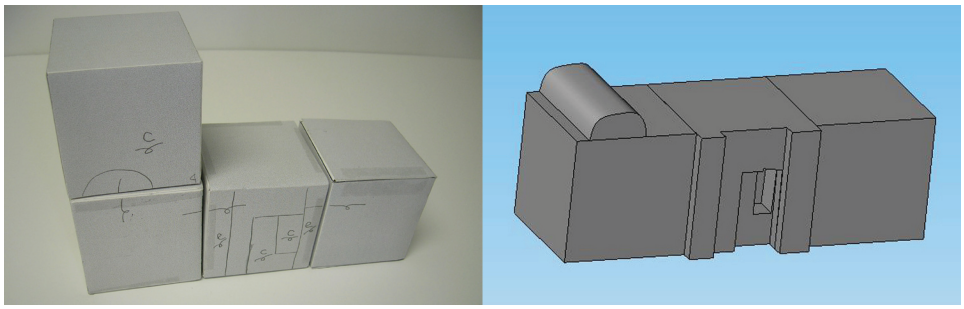

Fig. 13. Assembly syntax and form edit syntax to simulate the massing practice. Left: paper cube models with form edit and stitch operation to create a model in Figure 3. Right: result of execution.

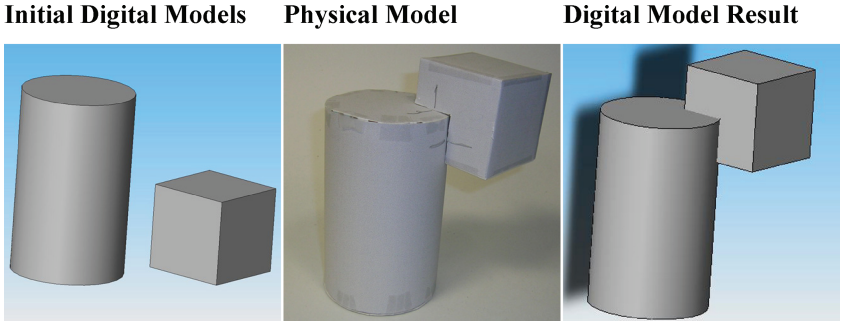

Fig. 14. Example of the assembly feature with models that intersect: cylinder and a cube are intersected using two stitching marks.

to the next are used as the two points. Furthermore, the tangent line of each model (a cross-product of the marking segment and the normal of the surface) is calculated for both models to be used for alignment. Since the first stitching mark doesn't end in a pigtail, ModelCraft doesn't align any predefined faces to finish the assembly. Instead, a second pair of stitching marks is used to finish the assembly. The coincident point from the second stroke is used to define the angle between the two models (i.e., angle between (a) and (a') in Figure 12(c)). Using this two-stitch assembly syntax, users can even overlap part of a model inside another model, thus allowing for the intersection of two physical models (Figure 14).

Finally, users can also use three-stitching syntax to assemble two objects in an arbitrary orientation (Figure 12(d)). In this case, users are required to create three stitching marks and the system uses the three points at which the stroke jumps from one object to the next to define the position and orientation of the two objects. This is the most precise and flexible alignment technique, since the system solely relies on the user's stroke to assemble the two objects. Hence, it can be used instead of the other syntax when alignment precision is important. Alignment precision will be discussed further in Section 4.6.

To show the potential of this assembly method, we demonstrate in Figure 13 how one could build the massing model provided to us by one of our participants (Figure 3). Starting with a set of simple blocks, users can draw several cut

Table I. List of Edit Commands and the Syntax Requirements

Parameter Commands	Recognized Shape Parameter		Location of the Auxiliary Parameter	Entity selected by a pigtail
	Ink on a single face	Ink on multiple faces		
Cut	Closed shape, Open shape	Closed shape	Any surface except where shape ink is laid	Shape
Extrude	Closed shape, Open shape	Closed shape	Augmented Tools (ruler, protractor, sweep sketchpad)	Shape
Groove		Closed shape		Shape
Shell				Face
Fillet				Edge
Assembly			Across pairs of models	Coincident point

The cut, extrude, and groove commands require a shape parameter on one or more faces. Cut and extrude require an auxiliary parameter. The shell and fillet commands are preceded only by a pigtail that indicates which part of the object is to be smoothed. The assembly syntax requires one to three pairs of auxiliary parameters for alignment.

operations to create the entrance area of the building on the center block. Then, users extend both sides of the model by using our stitch operation. Finally, a smaller semicircular block is used to create the detail on the roof and is glued onto the building.

Lastly, the entire set of edit commands is summarized in Table I.

3.5 Feedback and Error Management

In its original form, ModelCraft was a batch processing system, in which annotations and commands were captured by the pen to be processed upon pen synchronization [Song et al. 2006]. There were several reasons for this choice. First, as explained previously, it was important that interactions could take place away from a computer (Figure 6 and Figure 9). Second, by delaying execution, a batch approach might help keep users in the "flow" of their task by avoiding unnecessary interruptions.

The batch style of execution raises the question of how to correct for errors. In batch mode, our interface offers two main mechanisms to deal with errors. For marking errors in annotations and commands, we provide a simple scratch-out gesture to indicate that the underlying gestures should be removed, or that the underlying command should not be performed. For execution errors, it is important to remember that while our system might misrecognize gestures and command names, it accurately captures the parameters of the commands on the correct faces. Since this information is directly transferred to SolidWorks, it becomes a trivial matter to make corrections because all the relevant command parameters are already in place, and a correction involves changing the parameters.

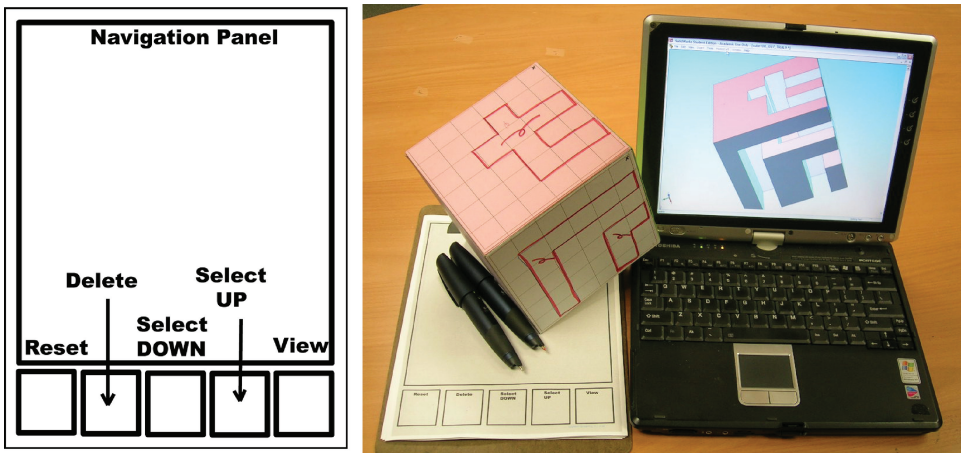

Fig. 15. Real-time interactions in ModelCraft. Left: paper tool palette for navigation, undo, feature select. Right: the CubeExplorer system consists of a paper cube, a computer display, and the paper tool palette.

This last feature also makes it possible to issue several "alternative" commands by simply drawing a new command over the last command, a common pattern in practice. Each command will be recognized as a different operator (or "feature" in SolidWorks terminology) and appear in the "feature tree" managed by SolidWorks. Once the strokes have been transferred to SolidWorks, the user can compare the results of different commands, pick the best of them, and delete alternative executions. Alternatively, commands can be applied in different model configurations to help document the design process.

3.6 Real-Time Pen Interactions

While gathering feedback about the batch version of the system, the possible appeal of providing immediate feedback became apparent. In particular, a professor in the architecture department at University of Maryland explained that he would like to use such a real-time system for one of his introductory classes. In a real-time version, strokes captured by the pen are transmitted via Bluetooth to a nearby computer, which processes them right away and renders the result for immediate inspection.

We explored this real-time digital pen interaction with the CubeExplorer [Song et al. 2007] system to teach different architectural space concepts to freshmen studying architecture. CubeExplorer was a simplified version of the ModelCraft system (limited to grid-snapping cut operations on a cube), but demonstrated the potential benefits of a digital "streaming" pen that can stream strokes in real time using a Bluetooth link. In CubeExplorer (Figure 15), user operations on the surfaces are instantly displayed on the screen of a nearby computer so that students could rapidly explore the 3D implications of their 2D marks.

Our work on CubeExplorer offered us several insights on how to implement real-time operations for ModelCraft. We discovered that it was important to

Table II. Feedback and Error Management in Batch and Real-Time Synchronization

Mode	Feedback			Error Management
	Audio	Visual	Tactile	
Batch Synchronization			Vibrates when the pen is near the edge	• Scratch out gesture • Pencil lead and eraser • Replace pattern patch if previous trace is troublesome
Real-time Synchronization	Audio beep TTS	Near-by screen		

provide a pen-based interface for both model navigation and quick access to frequently used commands. This reduces the mental load introduced by the interface, as users do not have to switch between a pen and a mouse to perform these actions. CubeExplorer includes a paper tool palette inspired by PaperPoint [Signer and Norrie 2007] and FlyPen. The CubeExplorer paper tool palette consists of six different regions. If a user draws the wrong shape for the cut operation, the user can restart the cut sequence by tapping on the *reset* region. Similarly, users can tap inside the *delete* region to undo the previous cut operation. After issuing more than five cut operations, the user may want to virtually undo a cut feature created in the beginning. To select a particular cut feature, users can tap inside the *select up* or *select down* regions to traverse the feature tree. To get a better view of the virtual model, users can also tap inside the *view* region to toggle the view between wireframe and solid views. Users can also use the *navigation panel* region with a pen to rotate the 3D virtual model on the screen. The direction of pen stroke on the navigation panel is used to move the virtual camera on the screen (Figure 15, left).

Our work on CubeExplorer also revealed that it was important to limit the need to check the screen. Users would verify whether an operation had been successful. However, over-reliance on visual feedback could be tedious while issuing a series of modeling operations on a model. To alleviate this problem, CubeExplorer provides a simple audio cue indicating the success of the current state transition of the operation. We translated this simple interface to the diverse operations offered by ModelCraft by using Text-To-Speech (TTS) to confirm the recognized syntax of both shape and functional parameters.

One of the major changes brought by real-time operation is the availability of the "delete" button on the tool palette used to cancel the last operation. In this setting, it is better to use pencil lead instead of regular pen ink so that it is easy to erase the unwanted marks on the physical model. This configuration allows users to maintain their work area clean.

Feedback and error management implemented in both batch synchronization and real-time synchronization is summarized in Table II. More recent feedback and error management techniques for digital paper interfaces [Liao et al.] and recent a commercial product with visual feedback [LiveScribe] could be adapted in the context of ModelCraft. Currently, this is left as future work.

4. IMPLEMENTATION

As shown in Figure 16, the life cycle of a model in our system can be broken down into four phases: (1) unfolding the 3D model into a 2D layout; (2) printing

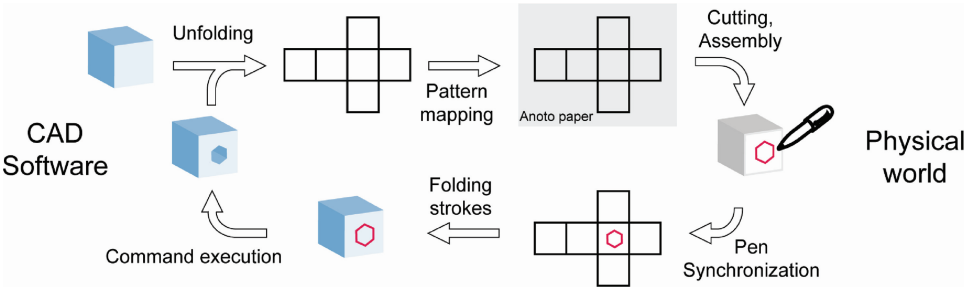

Fig. 16. Life cycle of a model using our system. Here we present the cycle for paper-based model construction, but a similar cycle will be used for applying water slide transfers onto 3D models.

the 2D layout as a paper prototype with a unique pattern on each side; (3) capturing the strokes made on the paper prototype in batch mode or real time and mapping the strokes onto the virtual 3D model; (4) executing the commands themselves. We now describe the implementation details of each phase.

4.1 Unfolding the Model

The original version of our unfolding algorithm [Song et al. 2006] was simple but limited. It relied on a triangulation of the model which did not include face information. As a result, even simple shapes could result in a complicated unfolding. We show an example of such behavior in Figure 17. As one can see on the top of the figure, the original algorithm creates many unwanted cuts in the middle of planar faces. These cuts are problematic as they create discontinuities in the tracking pattern, which cause unreliable tracking along the cut. The extended ModelCraft system can prevent faces from splitting as it does not merely use triangles as the basic geometric unit for unfolding. Rather, unfolding is performed on *groups* of triangles representative of the model faces (Figure 17 bottom).

4.1.1 *Unfolding into an Infinite Area.* There are many unfolding approaches for objects such as Mitani and Suzuki [2004] and Polthier [2003]. Our unfolding algorithm relies on a heuristic to create an efficient construction using the basic geometric unit (faces or triangles) of a model. We first consider the case in which we can unfold onto an infinite area of digital patterns while minimizing the number of patches.

The unfolding process follows a greedy algorithm (Figure 17), which tries to build as large a set of connected neighboring faces (a *patch*) as possible before starting a new patch.

When a user wants the 2D layout to preserve the connectivity at the expense of dividing a face, users do not have to choose a face as the basic unit (Figure 17, top). When a user wants the 2D layout to preserve the face information rather than preserving the connectivity, faces can be chosen as the basic geometry for unfolding (Figure 17, bottom).

4.1.2 *Packing Patches among Several Pages.* In practice, the pattern space is of finite size depending on the paper size of the printer. As one of our goals

```
UNFOLD(faces)
    patches.CLEAR()
    while (!faces.IS_EMPTY())
            p ← UNFOLD-INTO-A-PATCH(faces)
            patches.INSERT(p)
return patches

UNFOLD-INTO-A-PATCH(faces)
    NEW patch, copy-of-faces

     seedface ← faces.GET_LARGEST_AREA_FACE()
    faces.REMOVE(seedface)
    patch.INSERT(seedface.2D())
    while (!faces.IS_EMPTY()))  &&  copy-of-faces.EQUAL(faces) do
            copy-of-faces ← faces
            neighbors ← patch.GET_NEIGHBOR_FACES(faces)
            neighbors.SORT_LONGEST_COMMON_EDGE()
            foreach neighbor in neighbors do
                if (!patch.OVERLAP_2D(patch, neighbor.2D() )) then
                        faces.REMOVE(neighbor)
                        patch.ADD(neighbor.2D())
return patch
```

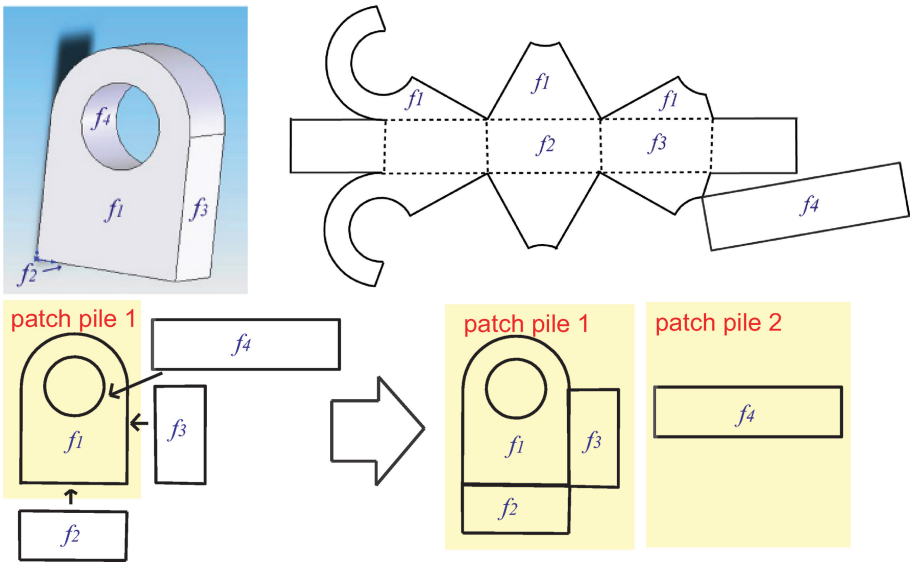

Fig. 17. Triangle- and face-based unfolding algorithm. Top: when the topology of the object is greater or equal to two, one face is split into several pieces. Bottom: additional face information during unfolding (bottom) preserves the continuous faces.

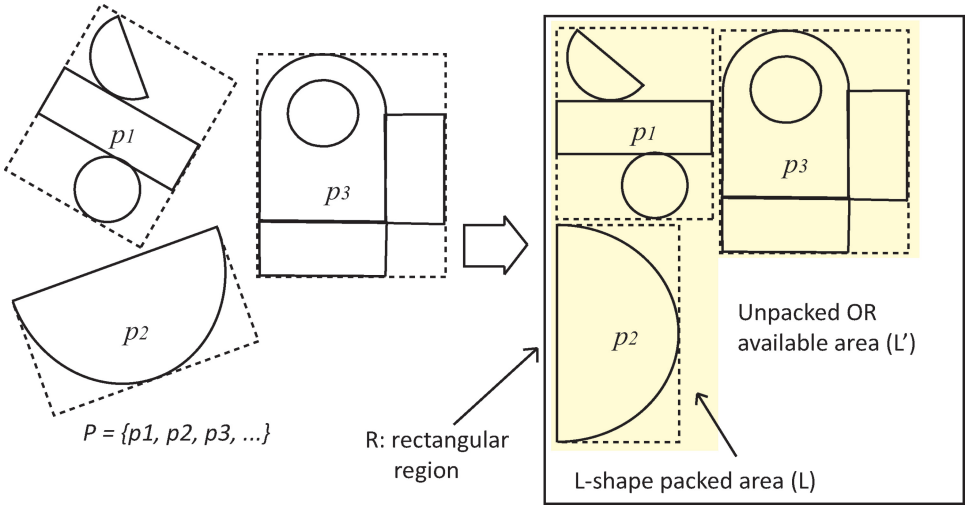

Fig. 18. Packing each page (R) with patches (pi).

was to develop a system where users can print out pattern using their typical printers to map onto large models, it was important to be able to deal with finite page size. Restricting the printed pattern to a finite size has two direct implications. First, patches can only grow up to the maximum pattern size and then the system must start a new patch. This problem is easily addressed by modifying the Unfold-Into-a-Patch() algorithm by terminating the iterative loop when the patch exceeds the finite pattern area. Second, the new version of the algorithm has to pack different patches onto a single pattern space when more than one patch can fit onto a page. In this case it is important to maximize the area covered with the patches, and to minimize the number of pattern pages required for a given model (Figure 18).

If the shape of each patch is approximated by a bounding box as shown in Figure 18, the packing problem reduces to finding the optimal packing of a set of rectangles (p_i: patches) on a larger rectangular region (R_{page}), which is known to be NP-complete. Many polynomial approximation methods exist for this rectangle packing problem [Jansen and Zhang 2004]. In our current implementation, we borrowed the strip packing algorithm described by Steinberg [1997] that proposes packing rectangles using an L-shape packed area. In doing so, this algorithm preserves the maximum available area in the target rectangular region. Following Steinberg's approach, our packing algorithm works as follows. Once all patches are created using the algorithm described in Section 4.1.1, we determine the minimum enclosing rectangular bounding box for each patch. Note that, as shown in Figure 18, this optimal bounding box might not be axis aligned, as the rectangular boxes are rotated to minimize the size of each bounding box. Next, the system traverses through the patch bounding boxes and finds an unoccupied section of a page in the queue of available pages. If one is found, the patch is placed so that the resulting L-shaped area occupies the minimum area. If the patch does not fit into any available area of the candidate

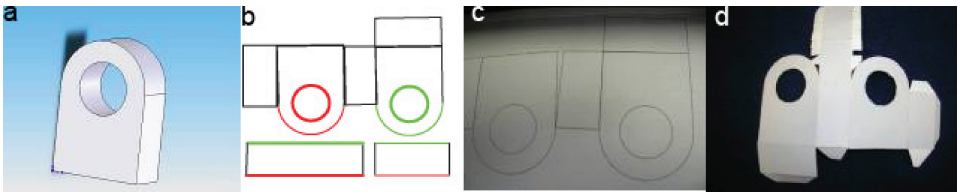

Fig. 19. The paper model building process using a laser cutter: (a) an electronic representation of the model; (b) a connectivity map, the 2D layout with the face ID and edge ID information; (c) the result after laser cutting the 2D layout; (d) the model after manual construction.

pages, it is placed on a new page which is added to the queue of documents. Since the algorithm checks only along the edges of the L-shape area to insert a new patch, the search time for the optimal location of the new patch on each page is constant. This heuristic approach relies on the assumption that the original patch area fits reasonably compactly inside the surrounding bounding box to save Anoto pattern space.

4.2 Printing the Model

If the unfolding algorithm generates several discontinuous patches for the 2D layout of a 3D model, manually constructing a physical model becomes difficult. Our plug-in offers an additional printout to simplify the construction of the resulting model. The *connectivity map* (Figure 19(b)) layout consists of the face ID, edge ID, and a set of alignment marks to properly align the patches together. This information is very useful even for building the simplest objects. For example, if a user opts to manually apply the traceable pattern and build the 3D model, the basic 2D layout will be printed on top of traceable pattern paper and then cut using a knife or scissors. Using the connectivity map, users can disambiguate and orient faces properly (e.g., identifying faces of a cube), align components precisely (e.g., the circular cap and rectangle of a cylinder should be aligned to preserve 3D geometry when folded), or make sense of the connectivity for complex models unfolded onto several patches.

To partially automate the model construction process, our system can output the unfolded layout (the edge layout and/or connectivity map) directly to a laser cutter to skip the manual cutting process (Figure 19(c)). We considered engraving the assembly information with the laser cutter to print the connectivity map as part of the laser cutting, but it proved somewhat difficult in practice to read the connecting edges and faces. When cutting water slide transfer paper to apply on top of a model printed with a 3D printer, the laser cutter was especially useful to cut 2D layouts from the water slide transfers. For creating a paper model, we decrease the laser power to engrave the cuts so that the tabs required for building the model can be manually created. The printing and construction process is depicted in Figure 19.

During printing, we use the PADD infrastructure [Guimbretiere 2003] to maintain the relationship between a given model's face and the unique page ID on which it has been printed. It is also used to record the calibration data

and geometric transformation used during the printing and unfolding process. This information is used during the synchronization process, to identify which digital model or face a stroke has been made upon.

4.3 Importing Captured Strokes

In the case of batch synchronization, the PADD infrastructure receives all the strokes captured by a pen when the pen is placed in its cradle. Strokes are recorded with a timestamp and the page ID on which they were made. We use the page ID information to recover the corresponding metadata from the PADD database, including the model ID, augmented tool ID (paper ruler, protractor, or sketchpad), and the calibration and geometric transformation of the model stored during the printing process. When the "Download" button is pressed (batch mode), our application fetches the metadata from the PADD database and transfers strokes from the PADD database onto the unfolded model. Then, each stroke point is mapped from page coordinates back into 3D coordinates by applying the inverse transformation that was originally used to map the face from 3D to the plane of the paper sheet.

If the user is using the Bluetooth-enabled pen for real-time interaction, each stroke is sent to our plug-in in real time. When the strokes are downloaded, page mapping and the transformation information are downloaded from the PADD database.

4.4 Executing Commands

All command strokes are made by the special "command" pen, so it is easy for our system to distinguish them from annotations. The sequence of strokes is parsed into individual commands using a set of heuristics to identify each command boundary, and to identify the parameters for each command. To do so, we first detect strokes that might look like a valid pigtail by looking for gestures with a relatively small loop and large outside tails. Once these are detected, we observe if there is a stroke recognizable as a character or word that has been drawn above the candidate pigtail within a predefined bounding box. If this is the case, the stroke is recognized as a valid pigtail, and the strokes drawn after the last command are used as parameters for the command execution. To further disambiguate input, we also check for natural command separators (such as creating an annotation or writing on one of the measurement tools) and check that the parameter set matches the command syntax. For example, the face ID or model ID associated with a pigtail delimiter and a command character should all be the same. In practice, this approach works well for batch mode synchronization.

In real-time mode, the recognized syntactic information is reported to the user as soon as it is available. While the batch-mode parsing engine considers subsequent strokes and timing information to parse strokes, our real-time parsing engine instead relies on the user's ability to redraw the shape parameter until it is recognized correctly. The system provides ample audio-visual feedback to indicate the current command state to the user in this configuration. Once the command syntax has been validated (e.g. command character detected),

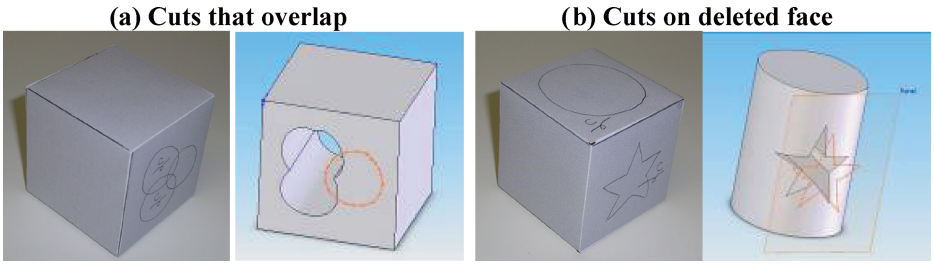

Fig. 20. Example of sketches on the reference plane of an object. Creating reference planes enables new operations to be independent from previous ones.

each parameter is processed according to the semantics of the command, and the command is executed inside SolidWorks.

4.5 Processing Shape Parameters

Various collections of strokes are parsed as shape parameters. Examples range from a single stroke on single face (Figure 20) to multiple strokes on multiple faces (Figure 7). When SolidWorks requires the shape parameter to form a closed sketch, ModelCraft automatically closes the sketch, either by moving the two end points closer or creating extra segments using the edges of the surface on which it is drawn (Figure 6).

Another important aspect to consider is how to map the captured shapes onto the 3D model. One simple way of mapping the shape parameters to the virtual model is to apply them directly onto the current state of the model. This greatly simplifies the implementation, but is problematic when parts of the surface on which the stroke is drawn have been removed in a previous operation. In Figure 20(a), successive cuts are overlaid on each other, resulting in parts of the parameter being drawn in free space from the perspective of the digital model. Figure 20(b) shows an even more extreme case in which the surface on which the star is drawn does not exist anymore in the digital version of the model. To address this problem, our system creates shape parameters on an independent reference plane tangent to the original surface. This guarantees that it is always possible to create a valid SolidWorks sketch using stroke data created on the surface of the original digital model, which may be different from the current state of the digital model.

Our system also needs to deal with complex curved surfaces, as shown in Figure 21. In such a case, we process input points after projecting them onto an imaginary plane. Although we would like to process the input points on top of original surface, the SolidWorks API limits our current implementation to creating 3D features from reference planes. In Figure 21, the surface of the original model is curved so a reference plane has to be approximated. In order to maximize the accuracy of the operation, the reference plane has to be defined so as to minimize the geometric deviation of the projected sketch from the original sketch.

With this criterion in mind, the received input points are used to approximate a reference plane. A 3D point is picked among these input points and a normal

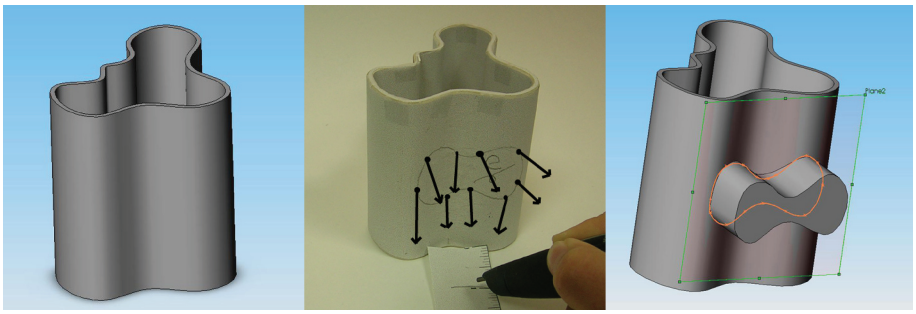

Fig. 21. Reference plane is created to simulate the nonflat surface to execute extrusion. Note that the original 3D sketch points are projected onto the reference plane defined by our system.

vector is calculated using these input points. We calculate a normal vector on the original surface for every point of the sketch (Figure 21) and average them to determine the reference plane normal. Next, we select the point that is farthest away from the model. Had we chosen any other point to create a reference plane, the reference plane will intersect the model, making some operations (cut, extrude, groove) difficult to execute. We then project the sketch data onto the reference plane to create a shape used for the command to be executed. For a "cut" operation, we simply extrude the cut in the opposite direction of the reference normal. For an "extrude" operation, using either the paper ruler or paper protractor, we extrude the shape first toward the direction of the reference normal up to the specified distance and then toward the solid in the opposite direction. Although this extrusion is defined above the highest ridge of the surface in our current implementation, other alternatives such as using the starting point of the pigtail for extrusion are also possible.

4.6 Interpreting Assembly Parameters

Our assembly syntax was designed with two goals in mind. First, the syntax should provide sufficient and appropriate reference geometry for SolidWorks to create the alignment that requires six degrees of freedom with as minimal amount of user input as possible. To achieve the first goal, our plug-in calculates possible referential geometry such as normal vector, cross-vector, coincident edge, and coincident plane for each syntax stroke. Figure 22 shows reference planes created for two models using the first stroke vector and the second stroke point.

Our second goal was to be sure that we could accommodate the limited precision of the data captured by the pen, so that alignment success rate is high. Due to the accuracy of the tracking, the two triangles (one on each object) created by the three alignment points for the three-stitch syntax are not identical and direct point alignment will fail in most cases. Snapping, which is a common feature in 3D applications, can be used to offset the misalignment error. However, increasing the tolerance level in SolidWorks is not a permanent solution, as the size of the triangles to define alignment varies. To alleviate this problem we proceed as follows: (1) align the first two points (the points on models at which

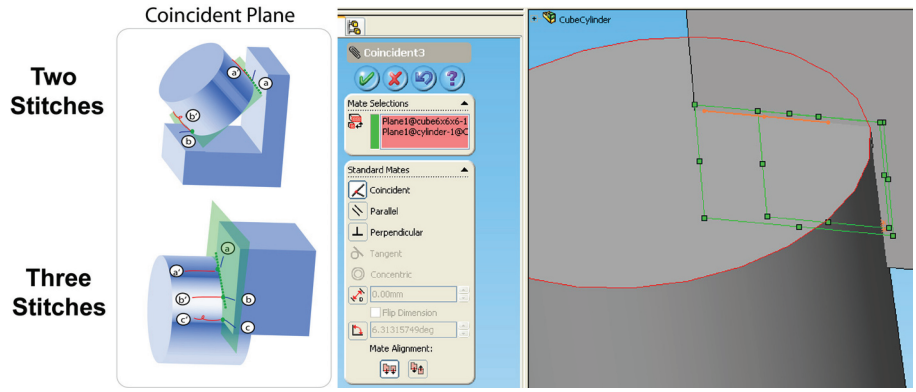

Fig. 22. Coincident plane for assembly: reference planes are created on both assembly models to create coincident plane relation.

the first stitching mark jumps from one object to the next); (2) align the two axes created by point 1 and point 2 on each object. This relaxes the distance constraint between the two pairs of points; (3) align the planes generated by each triad of points on each object. This relaxes the need to have perfectly identical triangles. Other stitch syntaxes relax the problem in similar ways (Figure 22). Note that our system only considers the case in which two objects are stitched together, but could be easily extended to three or more objects (for example, to create a bridge), by introducing a global connectivity graph data structure as was done by Anderson et al. [2000]. We are considering implementing such a system in the future version of the system.

5. DISCUSSION

To verify the proof of concept, we gathered feedback about ModelCraft from potential users during several phases of our implementation (Section 5.1). During and after these evaluations, we identified several technical limitations of our system that interfere with fluid interaction. We discuss the difficulties associated in editing models (Section 5.2), printing, constructing models (Section 5.3), and tracking performance and limitations (Section 5.4).

5.1 Evaluation

Our system was built over several iterations of user evaluations and feedbacks. Early on, we conducted six semistructured interviews, including a demonstration and a hands-on test. Our participant population covered a wide range of architectural and mechanical engineering backgrounds. In particular, it included a student in an architecture school working as a drafter, two young architects, a senior architect, a senior partner in an architecture firm, and a faculty member at the University of Maryland, School of Architecture. Despite the shortcomings of our early prototype (such as the use of handwriting recognition without training), seasoned architects reacted very positively to our system. Several architects pointed out that our system would be perfect for bridging the gap between the physical practice and virtual modeling practice, and in particular,

massing a building. During massing, new models are built based on marks or shapes that were suggested in the previous iterative cycle. This type of practice is well suited for the ModelCraft interactions. Architects further pointed out that annotations on paper models could be useful for capturing feedback from some of their clients who might be intimidated by digital models. They also commented that the support for "free space" sketching by using information sketched on extra papers will be useful. This feedback motivated the idea of paper protractor and paper sketchpad in the extended version of ModelCraft. They also mentioned that operations involving multiple objects are very useful in early design because architects often create new designs by stacking or joining available building blocks. This participant also pointed out that she often "deconstructed" her models in order to reconfigure them. This feedback inspired the assembly feature in the extended version of ModelCraft.

The response to the system from the younger participants (one a student drafter and one a CAD modeler) was more muted, since their work did not require extensive use of tangible 3D models. Yet the architecture student pointed out that the system would be very useful for teaching and could support current practices taught at school. The CAD modeler, while skilled in building models, did not use them at work.

The professor remarked that our system would allow students to explore prototyping and develop 3D thinking skills, because visualizing the 3D results of subtractive operations drawn on a face of a cube is a common task in architecture training. ModelCraft may also create a natural bridge between the traditional approach to architecture (based mostly on paper-based sketching) and the use of modern applications such as SketchUp [Google 2006]. As pointed out previously, we explored this idea by developing the CubeExplorer project and conducting an in-depth study comparing CubeExplorer [Song et al. 2007] to other conventional architectural education tools. Our results showed that CubeExplorer not only simplifies the training process but also provides simultaneous context for physical and virtual interactions. Yet it was difficult to measure whether our tool is better than conventional tools for improving creativity or performance. While we implemented CubeExplorer within the current version of ModelCraft, it was difficult to generalize our findings to the ModelCraft system. Hence, we are planning to conduct a new user study in the near future.

5.2 Editing Models

The design of our command language followed a different path than that of Teddy and Sketch. While those systems adopted a gesture-based approach well suited for sketching, we used a structured syntax based on a simple extendable command structure and a pigtail as a separator between parameter strokes and command selection [Hinckley et al. 2005; Liao et al. 2005]. One of the strengths of our approach is that it can be easily extended to a wider set of complex commands by using longer command names, while keeping an informal feel. Using techniques described in the PapierCraft system [Liao et al. 2005], we could also transfer a shape captured on transfer paper onto a given surface and extrude it. It would also be easy to extend the system to accept postcommand parameters

like numerical arguments. The current command set implementation attempts to keep syntax structure simple while permitting diverse combinations of editing operations, summarized in Table I.

Another important difference between ModelCraft and other 3D tracking systems is the scope of tracking. ModelCraft only allows users to draw on the surfaces of models as opposed to the free space. However, even if more complex tracking systems were used, people find it difficult to draw precisely in free space [Schkolne et al. 2001]. In ModelCraft, we present a partial solution to this problem by providing augmented tools. Users can specify parameters in the free space through the use of simple tools such as a ruler, a protractor, and a sweep sketchpad.

In regard to providing two modes (annotation and editing), we delegate specialized functions to different pens. This separation can be implemented differently by introducing multiple modes per pen using a mechanism such as a physical switch. However, as it is common practice to assign different tasks on different sketching devices, our implementation was well received by the designers.

5.2.1 *Command Recognition.* Character recognition and pigtail recognition together determined the total number of successfully recognized editing commands. Several problems might affect the recognition rate: First, the pen provides samples at a temporal resolution of (50Hz $\sim$ 75Hz), which is too low for character recognition but too high for shape recognition. We address this problem by oversampling strokes to provide closer samples before performing character recognition, and downsampling strokes before performing shape recognition. Second, the orientations of the command letter or word also have an effect on the command recognition rate if the axis of writing is unidentified. Our informal tests showed that using the pigtail as a baseline of character recognition was quite successful. Overall our tests show that our pigtail recognition rate is about 99%, and with our small dictionary of commands, we achieved a command recognition rate of about 92%. Further empirical evaluation will be needed to confirm these numbers and the effect of command letters or other syntax component deformation by writing on a nonplanar surface.

In terms of parameter recognition, we introduced the reference plane for shape parameters to make operations such as cut, extrusion, and sweep-extrusion independent of one another. Even if the surface that the sketch lies on is removed, sketch-based operations can easily be created on the inferred reference planes, which allow unlimited iterations of operations. However, operations such as shell or fillet that rely on edge selection or face selection succeed only if it is unambiguous for the system to identify which edge the pigtail is highlighting after several previous edit operations For example, if an edge was created after two disparate cuts, it is difficult to specify the edge on the physical model with a pigtail because there is no corresponding edge on the physical model. In summary, our edit operations suffer from discrepancies between the physical and digital models if the editing sequence alters the original object to an extent such that the user can no longer infer changed parts of the digital model from the original physical model.

Our assembly command allows the user to align models at a variety of angles and configurations. However, there are certain extreme cases that are our current implementation cannot handle. If a cone contacts a surface with only one contact point, none of current stitching commands allow users to specify the relationship between the cone spike and the surface normal. Such cases can be resolved by allowing the user to specify additional constraint using one of our augmented tools such as a ruler or protractor. Such extension is left for future work.

5.3 Printing Models

Our current prototype was designed with paper-based models. For simple models, this approach works extremely well. Cutting and scoring the models by hand is easy and accurate. Using a laser cutter greatly simplifies and increases the accuracy of this process. For more complex models, it is often easier to affix the pattern to existing models once they have been built using a 3D printer. Using the connectivity map and the alignment cues provided by our system, the 2D layout of complex shapes can be applied rapidly and only adds a small overhead to the production process.

As we created larger 3D models using our face-based unfolding algorithm, we discovered that mapping the pattern to an existing model using patterned paper introduces gaps. If one is to wrap a piece of paper of thickness t around a cylinder of diameter d, the length of paper needed will not be $2\pi d$ but $2\pi(d+t)$. Enlarging or shrinking the volume of the model and printing out the 2D layout can alleviate the problem but is not a general solution when the model is both concave and convex. Another solution is to use materials such as water slide transfers to print out patterns, which are very thin and significantly alleviate this problem.

5.3.1 *Automatic Printing of the Tracking Pattern.* The preferred solution for pattern mapping is to have the 3D printer print the pattern at the same time that the 3D object itself is printed. Some 3D printers (ZCorp Z510) can print at a resolution of up to 600 dpi in the plane of the printing bed and 540 dpi vertically [ZCorp 2005]. While this is in the same range as laser printers that are able to reproduce the Anoto pattern, our tests showed that a pattern printed with the ZCorp Z510 printer was not recognized by the digital pen. To understand why, we show segments of patterns printed on a laser printer and on a Z510 (Figure 23). As seen in Figure 23, left, the dots produced by our laser printer are of somewhat irregular shape but the use of pure black ink provides a highly contrasted image. The dots printed on the Z510 (Figure 23, right) are diffused and do not use true black ink but a combination of C, M, and Y inks to simulate black. As a result, they are likely invisible to the infrared pen sensor. We believe that this problem can be readily addressed by introducing a true CMYK printing process and using finer-grained printing material. Another solution would be to use a different tracking system such as Data Glyph, which is designed for 300 dpi printing (on par with the minimum layer thickness of .089mm (286 layers per inch) of the ZCorp process).

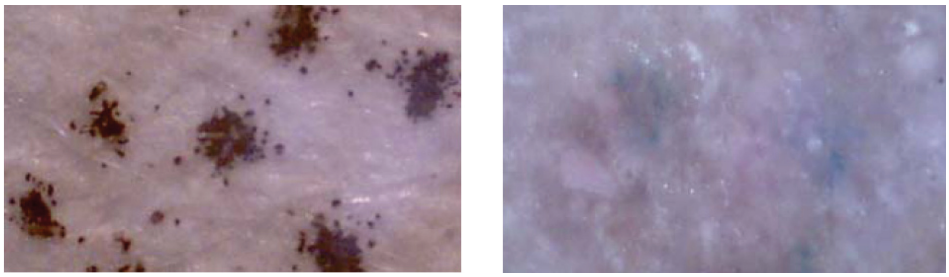

Fig. 23. Printing the Anoto pattern (all prints from a 600pdi rendering). Left: Anoto pattern printed using a 2400 dpi laser printer in black and white mode. Right: a pattern printed using a ZCorp Z510 printer (600 dpi). All pictures were taken at about ×200 magnification.

5.4 Tracking Performance and Limitations

One of our goals during this project was to better understand the limitations of a tracking method based on a pattern printed on the model surface. We now discuss our observations derived from working with our prototype.

5.4.1 *Accuracy.* The Anoto tracking system reports points with 678 dpi accuracy, but taking into account the errors introduced by pen orientation and the printing process, the system's maximum error is around 1mm (27 dpi). Of course, the overall accuracy of the system also depends on the accuracy at which the paper is cut and folded (around 1 mm in our current manual process). Using the laser cutter improves accuracy.

5.4.2 *Optical Tracking of Passive Patterns.* When the pen camera overhangs the edges of a face because users are trying to draw inside a groove or on an indented face, the system might lose tracking. When the tip of the pen is about 3mm from the border, it vibrates because the camera does not see enough pattern space. The tracking also fails at the edges of a face because of our pattern mapping scheme combined with how the Anoto tracking is done. The location of the pen tip is offset from the deciphered pattern space location. However, our unfolding algorithm doesn't guarantee that adjacent faces are mapped with continuous pattern. When the tip of the pen and the captured pattern are not on continuous pattern space, current Anoto tracking fails. If the Anoto firmware releases the location of the deciphered pattern instead of the calculated pen tip location, the tracking will improve.

In our tests, the pen was able to track a pattern at the bottom of a 4.8mm × 4.8mm groove or mark a 6.4mm-diameter circle using a 1.6mm-thick template. Because the pen was developed for tracking on flat surfaces, the system cannot track strokes on cylinders (or cones) whose radius of curvature is smaller than 12mm. It is not clear how significant these limitations will be in practice and future work will be necessary to evaluate their impact.

Another limitation of using the Anoto pattern as our tracking system is that it cannot track in free space. As demonstrated earlier, an instrumentation of the traditional tools used by wood workers (such as rulers, tracing paper,

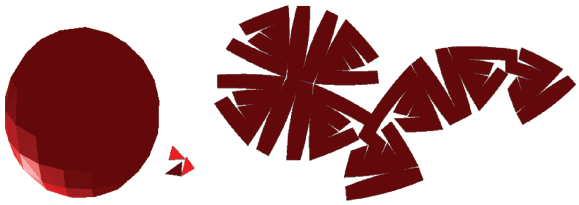

Fig. 24. An example of a nondevelopable surface creating many discontinuities in the pattern space.

protractors, and sketch pads) may help to address this problem. For example, we used our instrumented ruler to indicate the height of an extrusion.

Finally, the current version of our digital pen does not provide orientation information for the object that is being tracked. So far, this limitation proved to be mainly relevant for handwriting recognition and our use of the pigtail as a reference mark addressed the problem successfully.

6. FUTURE WORK

The system presented in this article was designed as an exploration tool allowing us to investigate the feasibility of our approach and provide us with a hands-on demonstration for potential users. Our next step will be to start deploying the system presented here to gather users' feedback in a realistic working environment either in architecture, industrial design, or in a teaching environment.

This will be made easier if our system could deal smoothly with nondevelopable surfaces. Nondevelopable surfaces are problematic for our system because unfolding of such surfaces leads to many discontinuities in the pattern space (Figure 24) and creates gaps in tracking. Our tests suggest that the pen's field of view is about 5mm wide and that the current pen firmware decodes the pattern correctly only if there is only one continuous pattern in its field of view. A closer look at the design of the Anoto pattern [Lynggard and Pettersson 2005] reveals that this is merely a limitation of the current implementation. In principle, one could uniquely resolve a position if any 2.4mm $\times$ 2.4mm patch is visible. We believe that if the firmware were modified to detect the edge of each continuous pattern region (maybe by recognizing printed edges) and each face of the model was wider than 2.4mm, the pen would be able to uniquely identify its position even around discontinuities in the pattern. Another solution to this problem would be to adopt a different approach to tracking altogether. Instead of mapping a 2D pattern onto our models, we could tile them with small (2–3mm) optical tags which can be tracked by the pen. For example, one could use the system proposed by Sekendur [1998], or the Data Glyph system [Petrie and Hecht 1999], or even the Anoto position pattern itself. All of these provide the large number of unique identifiers that are necessary. In all cases, the requirement of the minimum patch size can be accomplished using subdivision-based techniques such as the one used in the Skin system [Markosian et al. 1999] and extended by Igarashi and Hughes [2003].

We would also like to examine in more detail how our system could be adapted to 3D printing systems. In particular, we would like to explore the feasibility of a 3D version of the Anoto pattern. This would not only simplify the printing process and alleviate the pattern discontinuity problem but also allow for annotations on newly exposed, cut, or fractured surfaces of objects, and also may enable lightweight interactive morphing and sculpting techniques.

Finally, we would like to explore how our real-time system could be combined with a tangible user workbench such as the Urp system [Underkoffler and Ishii 1999] to explore how the ability to change the models on-the-fly might influence the use of such systems.

7. CONCLUSION

We presented a system that lets users capture annotations and editing commands on physical 3D models and design tools. Then, captured annotations are transferred onto the corresponding digital models. Our system is inexpensive and easily scalable in terms of objects, pens, and interaction volume. Users can perform subtractive (cut, groove, fillet, shell operations) or additive (extrude, assembly) edits on the model using our system. They can also create complex shapes by stitching simpler shapes together, which reflects the current practices of model builders. Depending on designer needs, the system can be used in two modes. Batch processing mode can be used to work in the field away from a computing infrastructure. Real-time processing can be used when immediate feedback is needed such as in teaching.

During a formal user study [Song et al. 2007] and many interviews, we gathered views on how our system allows users to deploy resources of both physical and digital media for the task at hand. We believe that once a fully automated pattern mapping process is realized, our approach will provide an efficient tool for the early phases of designing 3D models in both architecture and product design.

ACKNOWLEDGMENTS

We would like to thank the architectural and interior firm of BeeryRio for their support during the interview process (with special thanks to R. Keleher) and I. Savakova of DMJM H&N. C. Lockenhoff, A. Bender, and R. Schmidt provided many useful comments to help improve this document. We would also like to thank B. Bederson, B. Bhattacharjee, and B. Pugh for their support. We would like to acknowledge ZCorp (Figure 1, right), LGM (Figure 3, right), and Foster and Partners (Figure 1, left) for providing us with useful figures.

REFERENCES

AGRAWALA, M., BEERS, A. C., AND LEVOY, M. 1995. 3D painting on scanned surfaces. In *Proceedings of the I3D'95*. 145–150.

ANDERSON, D., YEDIDIA, J. S., FRANKEL, J. L., MARKS, J., AGARWALA, A., BEARDSLEY, P., LEIGH, J. H. D., RYALL, K., AND SULLIVAN, E. 2000. Tangible interaction + graphical interpretation: A new approach to 3D modeling. In *Proccedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*. 393–402.

ANOTO. 2002. *Development Guide for Service Enabled by Anoto Functionality*. Anoto Lund.

CHING, F. D. K. 1996. *Architecture: Form, Space, and Order* 2nd Ed. Wiley.
GOOGLE. 2006. SketchUp. http://sketchup.google.com/
GRASSET, R. L., BOISSIEUX, J. D., GASCUEL, AND SCHMALSTIEG, D. 2005. Ineractive mediated reality. In *Proceedings of the 6*th *Australasian Conference on User Interfaces*. 21–29.
GUIMBRETIERE, F. 2003. Paper augmented digital documents. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'03)*. 51–60.
HANRAHAN, P. AND HAEBERLI, P. 1990. Direct WYSIWYG painting and texturing on 3D shapes. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*. 215–223.
HART, S. 2005. Building a state-of-the-art home: Part II. *Architectural Record Innovation*, 24–29.
HINCKLEY, K., BAUDISCH, P., RAMOS, G., AND GUIMBRETEIRE, F. 2005. Design and analysis of delimiters for selection-action pen gesture phrases in sriboli. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'05)*. 451–460.
HINCKLEY, K., PAUSCH, R., GOBLE, J. C., AND KASSELL, N. F. 1994. Passive real-world interface props for neurosurgical visualization. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'94)*. 452–458.
HINCKLEY, K., RAMOS, G., GUIMBRETIERE, F., BAUDISCH, P., AND SMITH, M. 2004. Stitching: Pen gestures that span multiple displays. In *Proceedings of the AVI'04*. 23–31.
IGARASHI, T. AND HUGHES, J. F. 2003. Smooth meshes for sketch-based freeform modeling. In *Proceedings of the I3D'03*. 139–142.
IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. In *Proccedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*. 234–241.
ISHII, H. AND ULLMER, B. 1997. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'97)*. 234–241.
JANSEN, K. AND ZHANG, G. 2004. On rectangle packing: Maximizing benefits. In *Proccedings of the 15*th *Annual ACM-SIAM Symposium on Discrete Algorithms*. 204–213.
JUNG, T., GROSS, M. D., AND DO, E. Y.-L. 2002. Sketching annotations in a 3D Web environment. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'02) Extended Abstracts*. 618–619.
LIAO, C., GUIMBRETIERE, F., AND HINCKLEY, K. 2005. PapierCraft: A command system for interactive paper. In *Proccedings of the ACM Symposium on User Interface Software and Technology (UIST'05)*. 241–244.
LIAO, C., GUIMBRETIERE, F., AND LOECKENHOFF, C. E. 2006. Pen-Top feedback for paper-based interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'06)*. 201–210.
LIU, X. 2004. Editing digital models using physical materials. Master thesis, University of Toronto.
LIVESCRIBE. LiveScribe homepage. http://www.livescribe.com/
LOGITECH. 2005. IO digital pen. http://www.logitech.com
LYNGGARD, S. AND PETTERSSON, M. P. 2005. Devices method and computer program for position determination. U.S. Patent Office, Anoto AB: USA.
MARKOSIAN, L., COHEN, J. M., CRULLI, T., AND HUGHES, J. 1999. Skin: A constructive approach to modeling free-form shapes. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*. 393–400.
MASRY, M., KANG, D., AND LIPSON, H. 2005. A pen-based freehand sketching interface for progressive construction of 3D objects. *Comput. Graph. 29*, 563–575.
MITANI, J. AND SUZUKI, H. 2004. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Trans. Graph. 23*, 3, 259–263.
PETRIE, G. W. AND HECHT, D. L. 1999. Parallel propagating embedded binary sequences for characterizing objects in N-dimensional address space. U.S. Patent Office, Xerox Corporation.
PIPER, B., RATTI, C., AND ISHII, H. 2002. Illuminating clay: A 3-D tangible interface for landscape analysis. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'02)*. 355–362.
POLTHIER, K. 2003. Imaging maths—Unfolding polyhedra. *Plus Mag. 27*.

SCHKOLNE, S., PRUETT, M., AND SCHRODER, P. 2001. Surface drawing: Creating organic 3D shapes with the hand and tangible tools. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'01)*. 261–268.

SEKENDUR, O. F. 1998. Absolute optical polistion determination. U.S. Patent Office.

SHARLIN, E., ITOH, Y., WATSON, B., KITAMURA, Y., SUTPHEN, S., AND LIU, L. 2002. Cognitive cubes: A tangible user interface for cognitive assessment. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'02)*. 347–354.

SHENG, J., BALAKRISHNAN, R., AND SINGH, K. 2006. An interface for virtual 3D sculpting via physical proxy. In *Proceedings of the GRAPHITE'06*. 213–220.

SIGNER, B. AND NORRIE, M. C. 2007. PaperPoint: A paper-based presentation and interactive paper prototyping tool. In *Proceedings of the Conference on Tangible and Embedded Interaction*. 57–64.

SOLID CONCEPTS INC. 2004. SolidView. http://www.solidview.com/

SOLIDWORKS. 2005. SolidWorks homepage. http://www.solidworks.com/

SONG, H., GUIMBRETEIRE, F., AMBROSE, M., AND LOSTRITTO, C. 2007. CubeExplorer: An evaluation of interaction techniques in architectural education. In *Proceedings of the INTERACT Socially Responsible Interaction Conference*. 43–56.

SONG, H., GUIMBRETIERE, F., LIPSON, H., AND CHANGHU. 2006. ModelCraft: Capturing freehand annotations and edits on physical 3D models. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'06)*. 13–22.

STEINBERG, A. 1997. A strip-packing algorithm with absolute performance bound 2. *SIAM J. Comput. 26*, 2, 401–409.

SUZUKI, H. AND KATO, H. 1995. Interaction-level support for collaborative learning: AlgoBlock—an open programming language. In *Proceedings of the 1st International Conference on Computer Support for Collaborative Learning (CSCL)*. 349–355.

ULLMER, B. AND ISHII, H. 1997. The metaDESK: Models and prototypes for tangible user interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'97)*. 223–232.

UNDERKOFFLER, J. AND ISHII, H. 1999. Urp: A luminous-tangible workbench for urban planning and design. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'99)*. 386–393.

UNDERKOFFLER, H. AND ISHII, H. 1998. Illuminating light: An optical design tool with a luminous-tangible interface. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'98)*. 542–549.

WATANABE, R., ITOH, Y., ASAI, M., KITAMURA, Y., KISHINO, F., AND KIKUCHI, H. 2004. The soul of ActiveCube: Implementing a flexible, multimodal, three-dimensional spatial tangible interface. In *Proceedings of the ACE'04*. 173–180.

ZCORP. 2005. ZCorp 3D printing system. http://www.zcorp.com/en/Products/3D-Printers/Spectrum-Z510/spage.aspx

ZELEZNIK, R., MILLER, T., HOLDEN, L., AND LAVIOLA, J. J. 2004. Fluid inking: An inclusive approach to integrating inking and gestures. Tech. rep. CS-04-11, Department of Computer Science, Brown University.

ZELEZNIK, R. C. AND HERNDON, K. P. 1996. SKETCH: An interface for sketching 3D scenes. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*. 163–170.

Received October 2007; revised August 2008; accepted April 2009 by Wendy MacKay