EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2004)
John F. Hughes and Joaquim A. Jorge (Editors)

Cascading Recognizers for Ambiguous Calligraphic Interaction

João P. Pereira
Dep. de Eng.[a] Informática, ISEP/INESC
R. de São Tomé, Porto
jpp@dei.isep.ipp.pt

Joaquim A. Jorge
Dep. de Eng[a]. Informática, IST
Av. Rovisco Pais, Lisboa
jorgej@acm.org

Vasco A. Branco
Dep. de Comunicação e Arte, UA
Aveiro
vab@ca.ua.pt

Nelson F. Silva
Inst. Superior Técnico, IST
Av. Rovisco Pais, Lisboa
nelson.faria@netcabo.pt

Tiago D. Cardoso
Instituto Superior Técnico, IST
Av. Rovisco Pais, Lisboa
tiagocardoso@netcabo.pt

F. Nunes Ferreira
Dep. de Eng. Electrotécnica, FEUP
R. Dr. Roberto Frias, Porto
fnf@fe.up.pt

ABSTRACT

Throughout the last decade many approaches have been made to the problem of developing CAD systems that are usable in the early stages of product ideation. Although most of these approaches rely on some kind of drawing paradigm and on the paper-and-pencil metaphor, only a few of them deal with the ambiguity that is inherent to natural languages in general and to sketching in particular. Also the paper-and-pencil metaphor has not in most cases been fully accomplished, since many gesture-based interfaces resort to secondary buttons and modifier keys in order to make command strokes easier to differentiate from their geometry instantiating counterparts.

In this paper we describe the architecture of GIDeS++, a sketch-based 3D modeling system that approaches these problems in three different ways: by dealing with ambiguity and exploring it to the user's benefit; by reducing the command set and thus minimizing the cognitive load on the user; and by cascading different types of gesture recognizers, which allows interaction to resort only to the button located on the tip of an electronic stylus.

Calligraphic Interfaces, Non-Command Interfaces, One-Button Interfaces, Ambiguity, Expectation Lists, Sketch-Based Modeling.

1. Introduction and Related Work

The evolution of computer-aided design systems has been characterized, amongst other aspects, by the fact that its remarkable increment in functionality was obtained at the cost of an undesirable increase in the complexity of its use and of a consequent distance in relation to traditional paper and pencil. The abundance of existing commands and the strictness of the interaction tend to interfere with the designer's mind and disturb her/his creative processes, so it is not surprising that users continue to rely on paper and pencil in the initial stages of the design, resorting to the computer only in the final stages, when the shape of the object being created is already established and it is important to convert the sketch into a precise drawing.

Since then a lot of research has been done in order to develop CAD system interfaces that adopt some kind of drawing paradigm in an attempt to resemble the look and feel of paper and pencil.

On the one hand there are what we call pure drawing paradigms, in which the user sketches one (sometimes more) view of the object she/he plans to create and invokes a 3D solid reconstruction algorithm, which converts the drawing into a three-dimensional model. The resemblance with paper and pencil is high but there are a few drawbacks. As a matter of fact, most solid reconstruction techniques continue to rely on Huffman and Clowes labeling schemes [Huf71] [Clo71], which may only be applied to polyhedrons with trivalent junctions. Also the validity of the line drawing can't be fully verified since labeling is based on necessary, but not sufficient, conditions for the drawing to represent a 3D object [WG93]. Examples of CAD systems that rely on this type of paradigm are IdeS [BFC94], Digital Clay [SG98], and Stilton [TCP00].

On the other hand there are constructive (or incremental) drawing paradigms in which three-dimensional scenes are progressively built by combining and/or editing gesture-recognized instances of a small set of primitives. The paper and pencil metaphor is not as well accomplished as in the previous case, but there are not so many limitations in what concerns to the category and complexity of the solid's geometry. They can be smooth (e.g. an object of revolution), junctions must not be necessarily trivalent (e.g. the apex of a quadrangular pyramid) and the validity of the line drawing is always assured (otherwise no recognition takes place). Examples of CAD prototypes that resort to this kind of paradigm are SKETCH [ZHH96], Teddy [IMT99], the Translucent Sketchpad [EBSC99] [BES00], and Sketch-N-Make [BZ*98].

Another problem with some sketch-based CAD systems relates to the use of many buttons and modifier keys, in opposition to the pencil, which has no buttons at all. The identification of gestures based on the button the user is pressing (e.g. left button for creating geometry, and right button for instantiating commands) is easier from the system's point of view, but increases the cognitive load on the user.

Last but not least, sketching, as well as all other natural ways of conveying information, is intrinsically ambiguous. Such ambiguity can't therefore be avoided, and systems that try to unambiguously recognize a drawing tend to fail more frequently than others. Also for that kind of systems the recognition patterns must be mutually exclusive, which increases the number of different gestures that are subject to recognition, thus increasing once again the cognitive load on users. Mankoff describes many different ways of dealing with ambiguity in recognition-based interfaces [MAH00]. Examples of such interfaces are Burlap [MAH00], Pegasus [IMKT97], and Chateau [IH01].

The remainder of this paper describes the architecture of GIDeS++, an incremental drawing calligraphic interface that tries to address these problems by naturally handling ambiguous interactions. We show how the system successively tries to recognize commands, 2D primitives, 3D primitives, and Boolean operations without the need for more than just the button located on the tip of an electronic stylus. Finally we describe ongoing research directions and future work.

2. Cascading Recognizer Architecture

The architecture of GIDeS++ includes four different classes of cascade recognizers – commands, 2D primitives, 3D primitives and Boolean operations. Two modules compose each one of these subsystems. The next sections describe the recognition process in GIDeS++ with some detail.

2.1 Command Recognizer

Figure 2 in the next page illustrates the basic architecture of GIDeS++. The first step on the recognition process consists of preprocessing the stream of points that make up strokes drawn by users. There are two purposes for this procedure: to eliminate jiggle, which is achieved by discarding every input point that lies within 3 pixels in relation to the previous input point; and to perform segmentation, that is to create a polyline that reasonably resembles the original stroke.

The preprocessed stroke is then fed into two different recognition units: a modified version of Rubine's gesture recognizer [Rub91] (module no. 1) and an auxiliary recognizer (module no. 2).

2.1.1 Rubine's Modified Recognizer

The enhancements made to Rubine's recognition process had four main purposes [PJBF00]:

- Support for multiple-stroke gesture recognition. The sequence by which strokes are drawn is irrelevant.
- Recognize strokes regardless of the direction they were drawn.
- Force recognition to depend on context aspects not necessarily related to gesture geometry.
- Support for handling ambiguity.

Rubine proposes a set made up of 11 static (geometric) features and 2 optional dynamic (temporal) features to identify strokes input by users. GIDeS++ takes into account the geometric features only, so that users sketching speed do not affect the recognition process. A second level of geometric features was added in order to deal with gestures made up of multiple strokes. For every possible pair of strokes that make up each gesture, a pair of values representing the distances (both horizontal and vertical) between the center points of stroke bounding boxes is computed (Figure 1). Whenever a stroke is identified as being part of a multi-stroke gesture a timer is triggered. If the user draws the next stroke before the timeout occurs and the stroke meets the spatial relationships described above, it is considered as being part of the same gesture. Otherwise the system assumes that a new gesture is being initiated. Taking into consideration both time and space relationships gives much generality and flexibility to the recognizer, allowing gestures that share the same strokes to coexist (for example the "3D" gesture shown in Figure 1 could coexist with a "3" gesture, a "D" gesture and even a "D3" gesture).

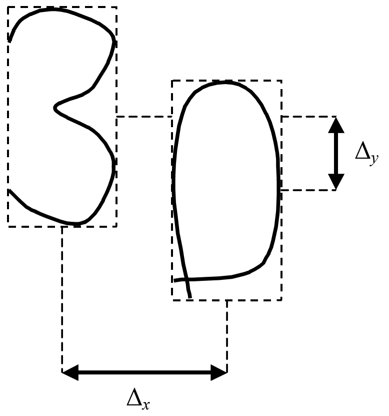


Figure 1: *Distances between pairs of strokes*

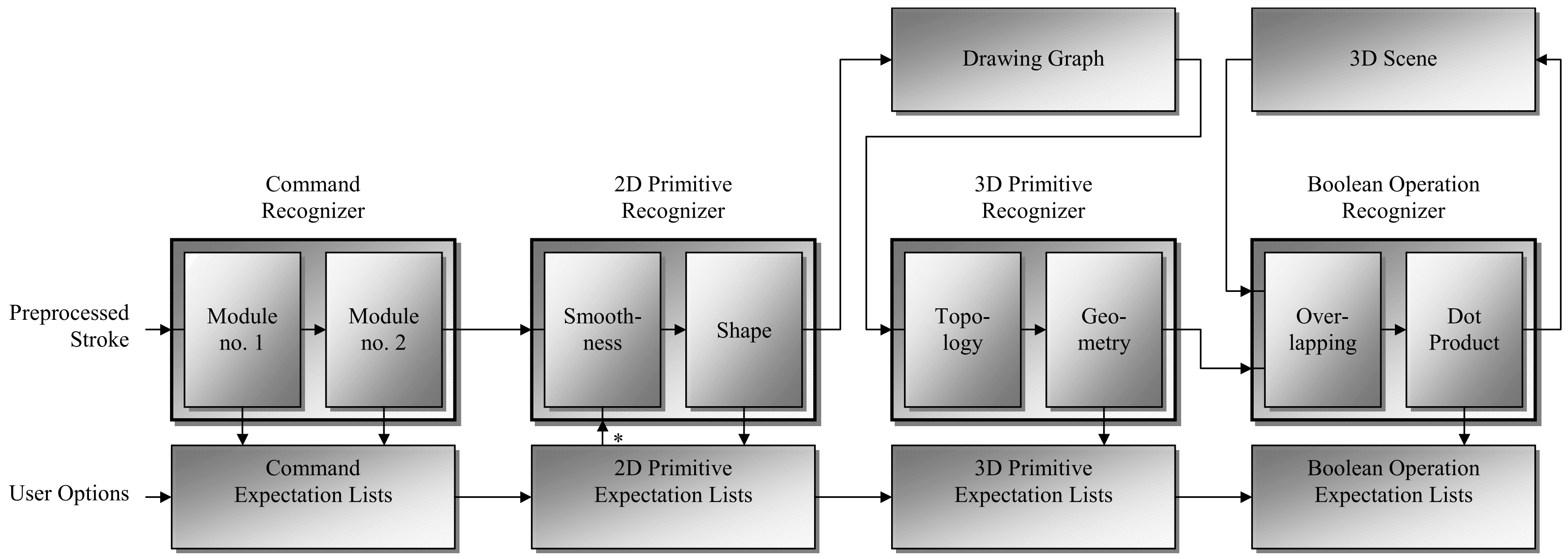
Slight modifications in the statistic equations proposed by Rubine were also made. The changes take into account that each gesture class is now eventually made up of more than one stroke.

Let C be the number of gesture classes, S_c the number of strokes that make up the gesture class c, E_c the number of training examples of gesture class c, and F the number of features used to identify strokes.

Let f_{cesi} be the i^{th} feature of the s^{th} stroke of the e^{th} training example of gesture class c. The sample estimate of the mean feature vector for each stroke of each class is given by the equation:

$$\overline{f}_{csi} = \frac{1}{E_c} \sum_{e=0}^{E_c - 1} f_{cesi}$$

$$0 \le c < C,\ 0 \le s < S_c,\ 1 \le i \le F$$



* Feedback for system learning and adapting to user

Figure 2: *GIDeS++ Cascade Architecture*

The sample estimate of the covariance matrix for each stroke of each class is then given by the equation:

$$\Sigma_{csij} = \sum_{e=0}^{E_c-1} (f_{cesi} - \overline{f}_{csi})(f_{cesj} - \overline{f}_{csj})$$

$$0 \le c < C,\ 0 \le s < S_c,\ 1 \le i,j \le F$$

An estimate of the common covariance matrix is then evaluated as:

$$\Sigma_{ij} = \frac{\sum_{c=0}^{C-1}\sum_{s=0}^{S_c-1} \Sigma_{csij}}{\sum_{c=0}^{C-1}(E_c - 1)S_c}$$

$$1 \le i,j \le F$$

The weights used to evaluate each stroke of each class over its features are computed as follows:

$$w_{csj} = \sum_{i=1}^{F} (\Sigma^{-1})_{ij} \overline{f}_{csi}$$

$$w_{cs0} = -\frac{1}{2}\sum_{i=1}^{F} w_{csi} \overline{f}_{csi}$$

$$0 \le c < C,\ 0 \le s < S_c,\ 1 \le j \le F$$

To make the recognition process independent of the direction strokes are drawn, one or two steps are performed: the fist one attempts to classify the stroke in the original direction drawn by the user. If it fails to recognize the stroke, its internal representation is reversed (that is the direction is reversed) and a second classification attempt is accomplished.

Rubine's gesture classification process also had to undergo some changes so that it can deal with ambiguity.

Rubine proposes a linear equation that uses the previously computed weights and features to evaluate, for each gesture class c, a quantity called v_c. The classification of gesture g consists simply in finding the c that maximizes v_c. An estimate $P(c|g)$ of the probability that g was correctly classified is then evaluated, along with a quantity δ^2, called Mahalanobis distance [DH73]. The gesture will be rejected if $P(c|g) < 0.95$ or if $\delta^2 > F^2 / 2$.

In our case we begin to evaluate, for each stroke s of each class c the Mahalanobis distance, given by the equation:

$$\delta_{cs}^{\ 2} = \sum_{i=1}^{F}\sum_{j=1}^{F} (\Sigma^{-1})_{ij} (f_i - \overline{f}_{csi})(f_j - \overline{f}_{csj})$$

$$0 \le c < C,\ 0 \le s < S_c$$

Strokes for which $\delta_{cs}^{\ 2} > F^2 / 2$ are rejected. Then, for each one of the remaining candidates, we compute the quantity v_{cs} as follows:

$$v_{cs} = w_{cs0} + \sum_{i=1}^{F} w_{csi} f_i$$

$$0 \le c < C,\ 0 \le s < S_c$$

Based on these values we estimate the probability $P(cs|g)$ that user stroke g can be identified with stroke s of class c:

$$P(cs|g) = \frac{1}{\sum_{i=0}^{C-1}\sum_{j=0}^{S_i-1} e^{(v_{ij} - v_{cs})}}$$

$$0 \le c < C,\ 0 \le s < S_c$$

The acceptance or rejection of candidate strokes is then made in accordance to the following set of rules:

- If there is a candidate with probability $P \ge 0.95$, the candidate is accepted and all others are rejected (no ambiguity at all, as it happens with Rubine's algorithm).
- Otherwise, if there are two candidates each one with probability $P \ge 0.95 / 2$, both candidates are accepted (ambiguity) and all others rejected.
- Otherwise, if there are three candidates each one with probability $P \ge 0.95 / 3$, the three candidates are accepted (ambiguity once again) and all the others rejected.
- And so on.

The dependency of the recognition process on other aspects not related with the shape of strokes is achieved by means of a set of software functions, one for each gesture, which return the Boolean values *TRUE* or *FALSE* depending on whether those aspects are met or not.

2.1.2 Auxiliary Recognizer

The aim of the auxiliary command recognizer is to classify strokes that can't be classified by Rubine's improved recognizer. Instead of being specified by example, each gesture is coded into a dedicated software function responsible for its recognition, allowing the classification process to be more flexible and to deal with aspects such as variable geometric features, insensitivity (or not) to geometric transformations, etc.

Figure 3 illustrates the need for this auxiliary recognizer. The size and shape of the object selection gesture (lasso) vary according to the objects the user wants and wants not to select, making it impossible for Rubine's modified recognizer to classify it.

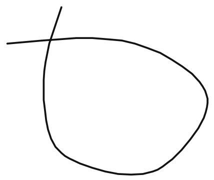


Figure 3: *Object selection gesture*

The auxiliary recognizer is also capable of dealing with multiple stroke gestures, ambiguity, and features stroke order and direction independence as well.

2.1.3 Command Expectation List

The final step in the command recognition process is to present the user with a set of all commands that were recognized in relation to the sketched gesture. The user may then choose which one of the available commands shall be performed. She/he may also ignore the expectation list and proceed with the drawing process.

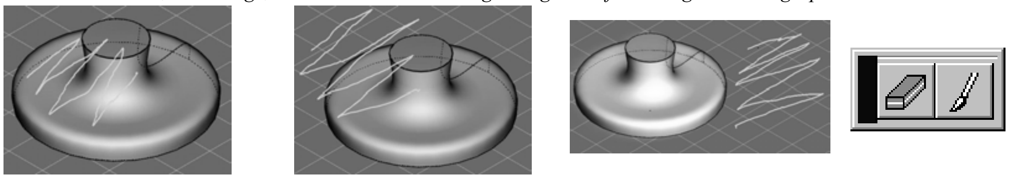


Figure 4: *Command Expectation List*

Figure 4 shows an example of a command expectation list. It also illustrates the way GIDeS++ deals with ambiguities and how expectation lists allowed us to overload command gestures, thus significantly reducing the number of needed command gestures and therefore minimize cognitive load on users [PJBF03].

In this case two commands – *delete* and *apply texture* – share the same "scratch" gesture. The difference is that the *delete* stroke must cross the object's boundary (Figure 4a), while the *texture* stroke must be entirely drawn over the object's surface, i. e. inside its two-dimensional projection (Figure 4b). The user may also opt to delete or conversely, to apply a texture to a previously selected object or set of objects. In that case GIDeS++ does not have enough contextual information to identify which command to apply. Therefore, the application generates a command expectation list and prompts the user to select the intended command (Figure 4c).

2.2 2D Primitive Recognizer

The next step on the recognition process consists of identifying 2D primitives. A gravity processing unit (not shown in Figure 2) compares the preprocessed stroke (a polyline) both with itself and with preexistent strokes stored in the drawing graph, and eventually modifies its component points in order to make them coincide with other existent points (linear gravity). It may as well align line segments with any of the main drawing directions (angular gravity).

The sample points of the preprocessed stroke are also used in generating a cubic spline. Both the polyline and the spline are then fed into the smoothness recognition module.

2.2.1 Smoothness Recognizer

The purpose of this unit is to find out whether the user intended to draw a rough stroke such as a polyline, or a smooth curve that will be subsequently converted into a cubic spline. Both the rough and the smooth strokes are compared to the original stroke and the one that closely matches it is elected to be the default option in the 2D primitive expectation list. Stroke comparison is performed by computing an error function consisting on the sum of the distances between each pair of corresponding points (interpolations are computed as needed). The stroke (polyline or spline) with the least error value is considered to be the one that closely matches the original stroke.

The smoothness recognizer has the ability to adapt itself to each user's unique drawing style, as described later in Section 2.2.5.

2.2.2 Shape Recognizer

The shape recognizer is responsible to determine whether the user intends to draw some specific primitive such as a circle or an ellipse, or a general one such as a smooth curve or a polyline. Three differently oriented ellipses are recognized, each one corresponding to the projection of a circle over each one of the three main planes in 3D space. Since the user can change the viewpoint over the scene, the ellipse recognition process is slightly more complex. In fact, what happens is that the geometric recognizer first projects the stroke onto each one of the three main planes and then tries to recognize a circle. The circle itself is identified by computing once again an error function that compares the stroke with a perfect circle. The centre of the circle corresponds to the center of the stroke's bounding box and the radius is the average distance between the centre and the stroke's sample points.

2.2.3 2D Primitive Expectation List

As soon as the recognition process itself comes to an end, a 2D primitive expectation list is generated. Figure 5 illustrates how this kind of expectation list also deals with ambiguity and explores it to the user's benefit. In this case the designer sketched a stroke that resembles an ellipse.

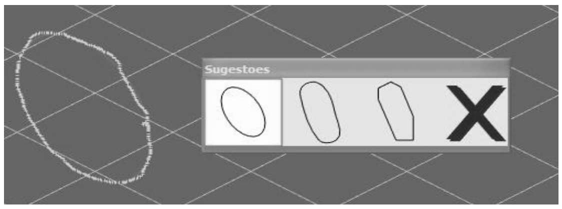


Figure 5: *2D Primitive Expectation List*

The smoothness recognizer classified the stroke as being smooth and the geometry recognizer classified it as being an ellipse. The resulting expectation list thus suggests an accurate ellipse as its default option. Nevertheless it is possible that the user wanted a generic smooth curve that happens to resemble an ellipse, instead of the ellipse itself. Or maybe the smoothness recognizer described above made a mistake and interpreted the stroke as smooth instead of rough. The remaining two options in the expectation list reflect these possibilities

2.2.4 Preprocessor and Gravity Control

The geometry recognizer has the ability to make changes to the parameters that control both the stroke preprocessor and the gravity processor units, in order to make them more tolerant to the imprecision inherent to hand drawn strokes. Figure 6 shows an example of a stroke originally recognized as a two-segment polyline (default option) that can also be interpreted as two different kinds of line segments. The conversion of the polyline into a line is a consequence of a change in the parameters that control the stroke preprocessor, and the different slope of the line segment that constitutes the third option in the expectation list is a result of a change in the gravity processor control parameters, which forced it to align with one of the main drawing directions.

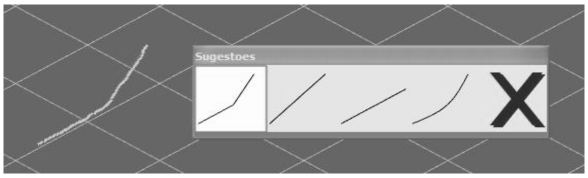


Figure 6: *Changes in Stroke Preprocessor and Gravity Control Parameters*

2.2.5 Adapting to User Behavior

The smoothness recognizer described in Section 2.2.1 is responsible for identifying whether the stroke input by the user should be interpreted as a rough or a smooth line. An error function relating the original stroke both with the polyline and spline interpolations is evaluated.Let ε_p and ε_s respectively, be the error summations of the polyline and spline approximations over the entire length of the stroke. Then we compute two estimates of the likelihood of choosing the polyline (p_p) or spline (p_s) approximations, respectively. Note that $0 \le p_p, p_s \le 1$ and $p_p + p_s = 1$.

$$p_p = \frac{\varepsilon_s}{\varepsilon_p + \varepsilon_s}$$

$$p_s = \frac{\varepsilon_p}{\varepsilon_p + \varepsilon_s}$$

The simplest way to classify the stroke is to compare p_s (or p_p) against a preset threshold (typically 0.5; Figure 7a). If p_s is above the threshold ($p_s > 0.5$), the stroke is a smooth line; otherwise ($p_s \le 0.5$) the system chooses the polyline approximation. Preliminary user tests showed this was not entirely acceptable, so we decided to develop an adaptive approach, using a dynamic threshold (p_{th}), which varies as a function of the number of previous mistakes (Figures 7b and 7c). Mistakes are identified based on user choice over the expectation list:

$$p_{th} = \frac{1}{\frac{n_s + 1}{n_p + 1} + 1}$$

where n_s and n_p are the number of times the stroke was erroneously classified as a spline (n_s) or a polyline (n_p), respectively. We now compare p_s to p_{th}, to decide whether to select a spline approximation ($p_s > p_{th}$) or a polyline otherwise ($p_s \le p_{th}$).

A simple analysis of this procedure shows that the formula is simple (which is good), also balanced ($n_s = n_p \Rightarrow p_{th} = 0.5$), and converges rapidly with the number of errors. This can be seen by the partial derivatives:

$$\frac{\partial p_{th}}{\partial n_s} = -\frac{n_p + 1}{\left[(n_s + 1) + (n_p + 1)\right]^2}$$

$$\frac{\partial p_{th}}{\partial n_p} = \frac{n_s + 1}{\left[(n_s + 1) + (n_p + 1)\right]^2}$$

which are inversely proportional to the number of errors squared.

2.3 3D Primitive Recognizer

The next step on the recognition process consists of identifying 3D primitives.

2.3.1 Drawing Graph

Whenever the user inputs a stroke that is recognized as a drawing element, the corresponding 2D primitive is inserted into the system's drawing graph, triggering the 3D recognition process.

2.3.2 Subgraph Evaluator

First we evaluate the subgraph consisting of the recently inserted stroke along with all the eventually preexistent strokes that are connected to it. The gravity processing unit plays an important role here since it helps users in connecting strokes, thus avoiding the need for timeouts in identifying which strokes relate to the very same gesture.

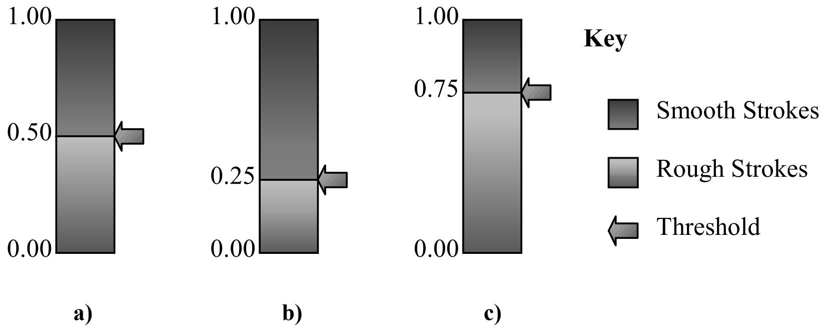


Figure 7: *Thresholding*

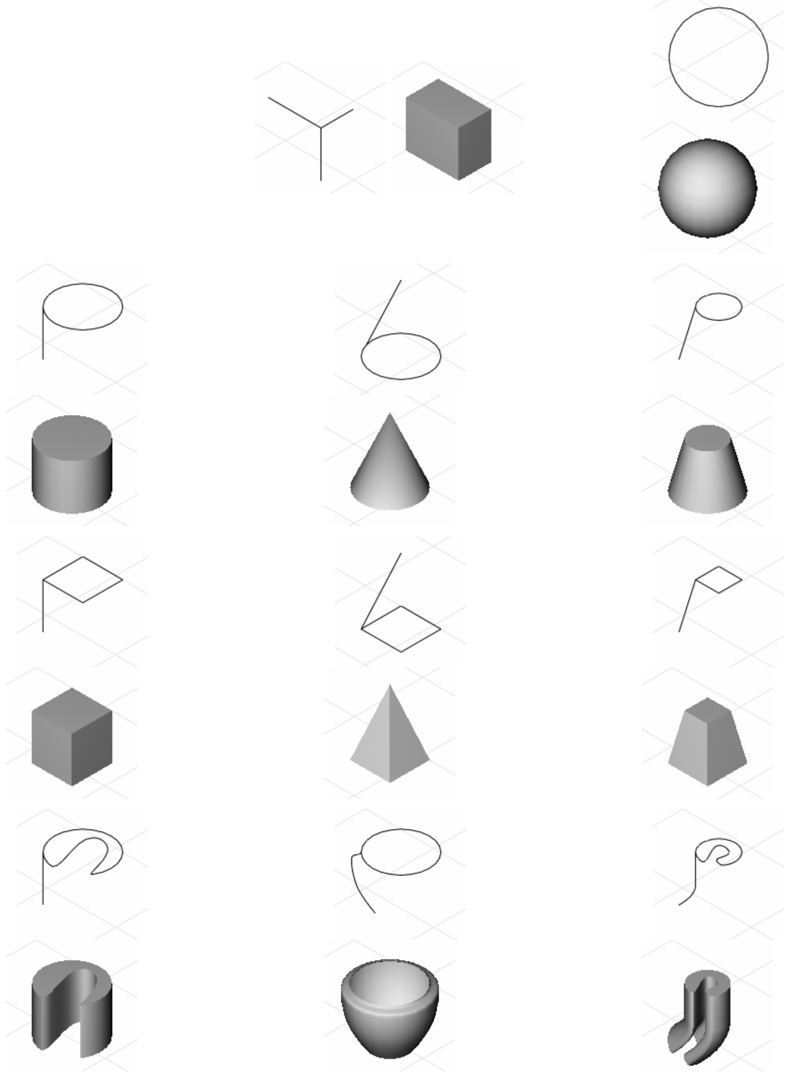


Figure 8: *3D Primitives and Corresponding Instantiating Gestures*

2.3.3 *Topology Recognizer*

The subgraph is then subject to a topological classification process that tries to identify which one of the following categories the gesture falls into (Figure 8):

- Trivalent junction (box gesture).
- Closed line (sphere gesture).
- Closed line connected to an open line consisting of one or more line segments (surface of revolution and duct gestures), or of one single line segment (remaining gestures).

2.3.4 *Geometry Recognizer*

If the topological recognition process succeeds, the gesture is then subject to a geometry analysis procedure in order to find the primitive (or primitives, if there is an ambiguity) that eventually matches the gesture.

For example for the cylinder, cone and truncated cone to be recognized, the above-mentioned closed line must be an ellipse. In the case of the cylinder, the line segment connected to the ellipse must be perpendicular (in 3D) to the ellipse's plane. Otherwise, the segment's open end must (in the case of the cone) or must not (in the case of the truncated cone) be close to the axis perpendicular to the ellipse's plane that goes through its center.

2.3.5 *3D Primitive Expectation List*

Whenever both the topological and geometric recognition processes succeed, a 3D primitive expectation list is generated. Figure 9 shows an example of such a list.

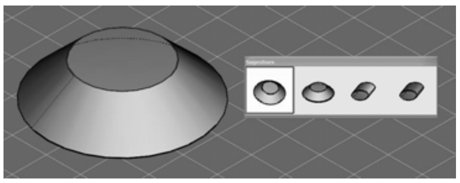


Figure 9: *3D Primitive Expectation List*

Again notice how the interface explores ambiguity in order to reduce the required repertoire of recognized gestures. In this case the same idiom can instantiate four distinct objects, namely a truncated cone, a surface of revolution (the most obvious choices) and two less evident kinds of cylindrical sections with different orientations. The user may as usual ignore the suggestions and proceed with the drawing.

2.4 Boolean Operation Recognizer

Whenever users sketch a primitive over an existent solid in the scene, the newly created object is properly placed and attached to that solid. Under these circumstances GIDeS++ attempts to find the appropriate Boolean operation (union or subtraction) based on gesture orientation.

Figure 10 illustrates this procedure. The gesture orientation (in the example the orientation provided by the line segment together with the ellipse attached to it) is compared with the object's surface normal. The system chooses to perform union or subtraction depending on whether the sign of the dot product is positive or negative.

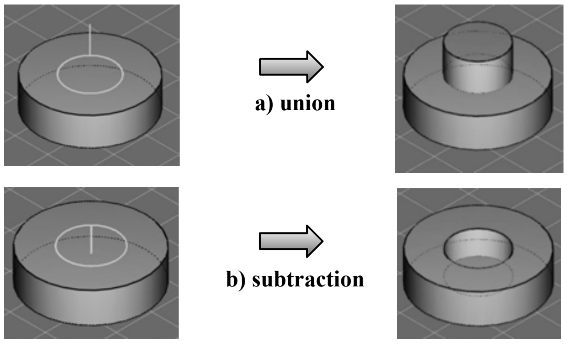


Figure 10: *Recognizing Boolean Operations*

Some primitive gestures such as the sphere provide no orientation information. It may also happen that the above-mentioned dot product equals zero (*i. e.* the gesture orientation and the surface's normal are perpendicular to each other). These circumstances prevent GIDeS++ from identifying the desired Boolean operation. An expectation list is then generated, allowing the user to choose between union and subtraction as shown in Figure 11.

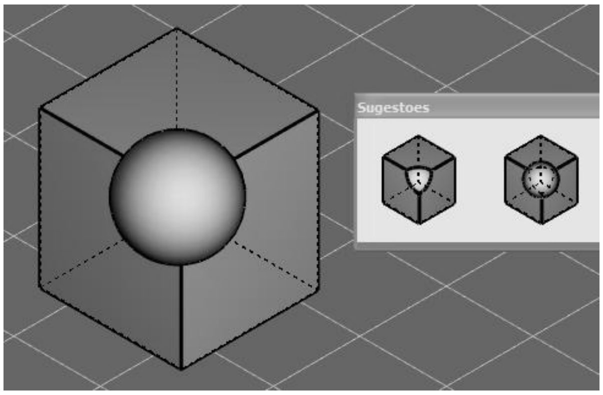


Figure 11: *Boolean Operation Expectation List*

3. Conclusions and Future Work

GIDeS++ (as well as its predecessor, GIDeS) was subject to usability evaluation sessions in which participated designers, mostly from the mould industry, and architects. The results achieved were very encouraging and seem to validate the calligraphic model of interaction, as well as the adopted drawing paradigm and the corresponding architecture. Nevertheless we are currently working on a hybrid paradigm that combines the advantages of incremental drawing with the straightforwardness of 3D solid reconstruction techniques such as the one proposed by Naya for creating *normalons* and *quasi-normalons* [NJC*02] (Figure 12).

In addition to the 2D drawing graph we will have a 3D graph for those circumstances in which it is possible to infer the coordinates of the stroke in the 3D space. Examples of these situations are when the user is drawing with the help of an auxiliary plane (the equation of the plane is known), or when she/he is sketching on the surface of an existing solid (the description of the surface is known as well). In those cases the drawing is directly stored in the 3D graph and fed into the 3D input of the primitive recognizer. The recognition process becomes simpler, since there is no need for taking the currently defined projection into account.

Whenever it is not possible to determine the space coordinates of the stroke, it is stored in the 2D graph and an attempt is made to reconstruct the resulting subgraph's 3D geometry. If it succeeds, the subgraph is passed to the 3D graph and then to the primitive recognizer, which builds the corresponding *normalon* or *quasi-normalon*. Otherwise everything happens the same way it is happening now, that is the subgraph is fed into the 2D input of the primitive recognizer, which tries to identify it as one of the basic solids illustrated in Figure 8.

4. Acknowledgements

The work described in this paper has been supported in part by the European Commission Grant #IST-2000-28169 (SmartSketches project) and by the Portuguese Science Foundation under grant POSI/34672/SRI/2000.

References

[BES00] BIMBER O., ENCARNAÇÃO L. M., STORK A.: A multi-layered architecture for sketch-based interaction within virtual environments. *Computers & Graphics*, Vol. 24, No. 6, pp. 851 – 867, Elsevier, Dec. 2000.

[BZ*98] BLOOMENTHAL M., ZELEZNIK R., FISH R., HOLDEN L., FORSBERG A., RIESENFELD R., CUTTS M., DRAKE S., FUCHS H., COHEN E.: Sketch-N-Make: Automated Machining of CAD Sketches. *Proceedings of ASME Design Engineering Technical Conferences*, Atlanta, Georgia, September 1998.

[BFC94] BRANCO V., FERREIRA F. N., COSTA A.: Sketching 3D models with 2D interaction devices. *EUROGRAPHICS '94 Conference Proceedings*, Daehlen M, Kjelldahl L (Eds.), Oslo, Blackwell Pub., 489-502, 1994.

[Clo71] CLOWES M. B.: On Seeing Things. *Artificial Inteligence*, 2(1):79-112, 1971.

[DH73] DUDA R., HART P.: Pattern Classification and Scene Analysis. *Wiley Interscience*, 1973.

[EBSC99] ENCARNAÇÃO L. M., BIMBER O., SCHMALSTIEG D., CHANDLER S. D.: A Translucent Sketchpad for the Virtual Table Exploring Motion-based Gesture Recognition. *Computer Graphics Forum*, Vol. 18, No. 3, pp. C-277 – C-285, 1999.

[Huf71] HUFFMAN D. A.: Impossible objects as nonsense sentences. B. Meltzer, D. Michie (Eds.), *Machine Intelligence*, 295-323, Edinburgh University Press, 1971.

[IMKT97] IGARASHI T., MATSUOKA S., KAWACHIYA S., TANAKA H.: Interactive Beautification: A Technique for Rapid Geometric Design. *Proceedings of the ACM Symposium on User Interface Software Technology* (UIST), 1997.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: A Sketching Interface for 3D Freeform Design. *SIGGRAPH '99 Conference Proceedings*, ACM, 1999.

[IH01] IGARASHI T., HUGHES J. F.: A Suggestive Interface for 3D Drawing. *14th Annual Symposium on User Interface Software and Technology* (UIST '01), Orlando, Florida, 173-181, Nov. 2001

[MAH00] MANKOFF J., ABOWD G. D., HUDSON S. E.: OOPS: a toolkit supporting mediation techniques for resolving ambiguity in recognition-based interfaces. *Computers & Graphics*, Vol. 24, No. 6, pp. 819 – 834, Elsevier, Dec. 2000.

[NJC*02] NAYA F., JORGE J., CONESA J., CONTERO M., GOMIS J. M.: Direct Modeling: from Sketches to 3D Models. *Proceedings of the First Ibero-American Symposium in Computer Graphics*, Guimarães, Portugal, 109-117, July 2002.

[PJBF00] PEREIRA J. P., JORGE J. A., BRANCO V., FERREIRA F. N.: GIDeS: Uma Abordagem Caligráfica à Edição 3D. *Actas do 9.º Encontro Português de Computação Gráfica*, pp. 101 – 108, Feb. 2000.

[PJBF03] PEREIRA J. P., JORGE J. A., BRANCO V., FERREIRA F. N.: Calligraphic Interfaces: Mixed Metaphors for Design. *Proceedings of the 10th International Workshop on the Design, Specification, and Verification of Interactive Systems*, Funchal, Madeira Island, Portugal, June 2003.

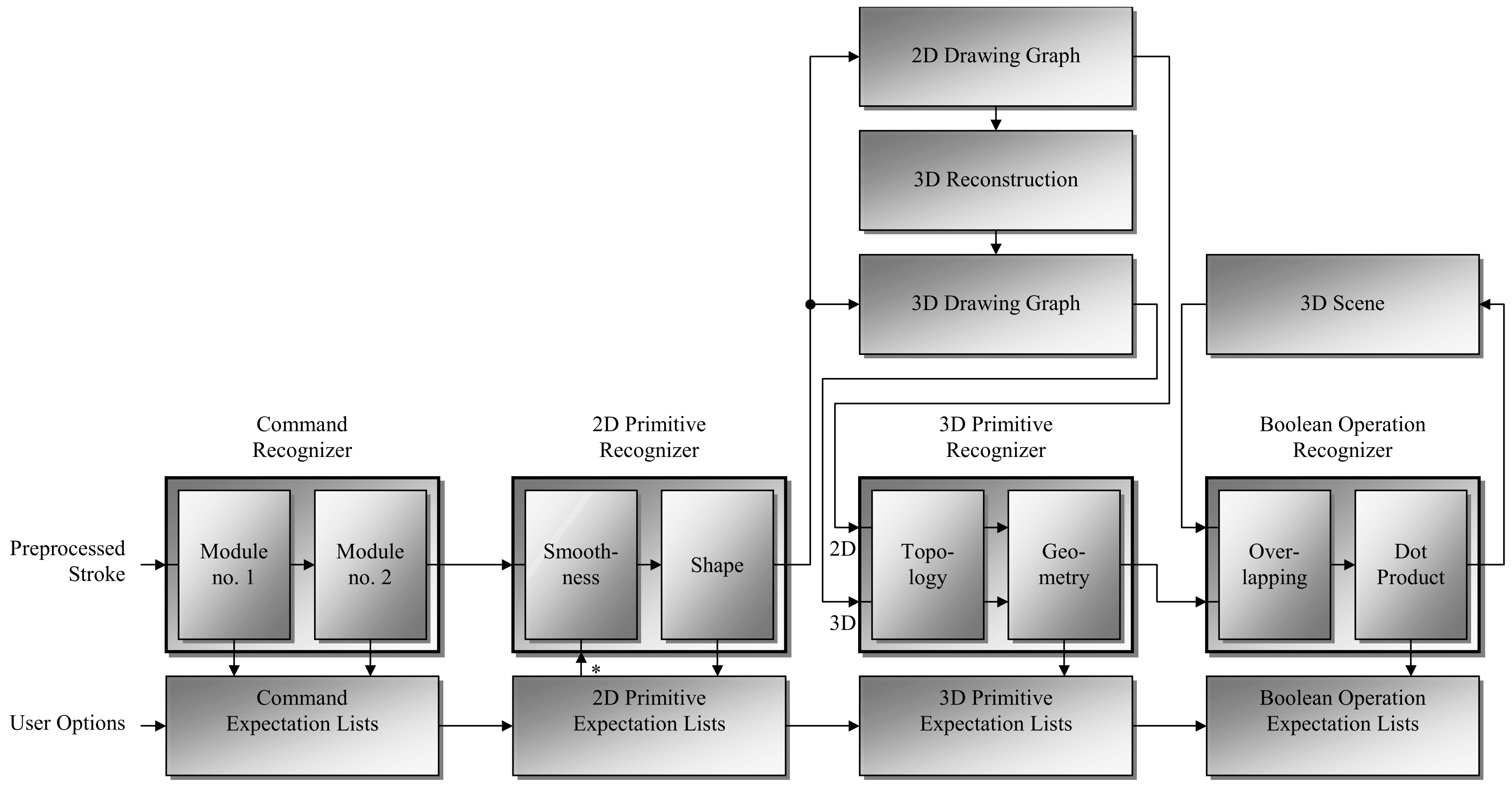
[Rub91] RUBINE D.: Specifying Gestures by Example. *SIGGRAPH '91Conference Proceedings*, ACM, Vol. 25, No. 4, pp. 329 – 337, 1991.

[SG98] SCHWEIKARDT E., GROSS M. D.: Digital Clay: Deriving Digital Models from Freehand Sketches. *Proceedings of ACADIA '98*, T. Seebohm and S. V. Wyk (Eds.), Quebec City, Canada, 202-211, Oct. 1998.

[TCP00] TURNER A., CHAPMAN D., PENN A.: Sketching space. *Computers & Graphics*, Vol. 24, No. 6, pp. 869 – 879, Elsevier, Dec. 2000.

[WG93] WANG W., GRINSTEIN G. G.: A Survey of 3D Solid Reconstruction from 2D Projection Line Drawings. *Computer Graphics Forum*, 12(2):137-158, 1993.

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: SKETCH: An Interface for Sketching 3D Scenes. *SIGGRAPH '96 Conference Proceedings*, ACM, Vol. 30, No. 4, pp. 163 – 170, 1996.



* Feedback for system learning and adapting to user

Figure 12: *GIDeS++ Future Architecture*