

Bringing Sketch Recognition into Your Hands

Gabriele Nataneli
Department of Computer Science
University of California Los Angeles
Email: nataneli@cs.ucla.edu

Petros Faloutsos
Department of Computer Science
University of California Los Angeles
Email: pfall@cs.ucla.edu

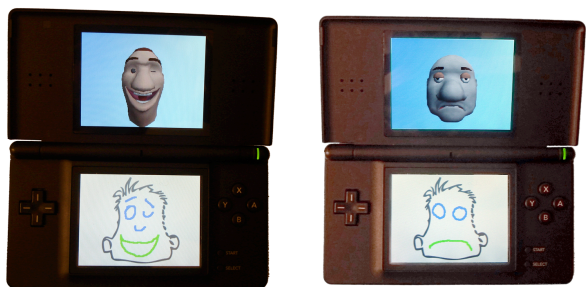


Fig. 1. Our sketch recognition pipeline running on a Nintendo DS.



Fig. 2. Our sketch recognition pipeline running on the the iPhone.

Abstract—

Sketching is an important enabling technology for modern animation interfaces. We present an approach for the analysis of sketches that is very flexible and works consistently across a variety of different software and hardware platforms. Importantly, our method suits well the restricting requirements of mobile devices. The proposed framework builds a semantic representation of informal drawings and uses this model along with other properties of the sketch to drive an output module. Our method is reliable and can run in real-time even on devices with very limited hardware resources. We fully implemented the proposed recognition pipeline on a Nintendo DS, an iPhone, and a regular PC. We showcase an application of this technology for driving facial expressions.

Index Terms—recognition, sketching, embedded systems, computer graphics

I. INTRODUCTION

Building a pen-based computer interface has been a great ambition of computer science for over forty years. However,

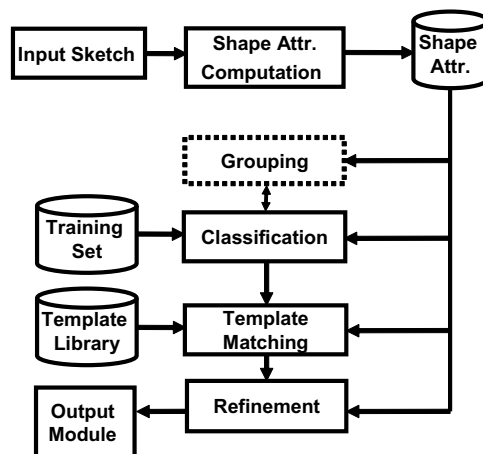


Fig. 3. High level diagram of the recognition pipeline.

only recently this area of research has experienced a substantial resurgence, due to the wide-spread availability of devices that use sketching as their primary form of input. Applications range from simple interfaces, such as the touch menus of the iPhone to very advanced applications for computer-aided design. Furthermore sketch-based interfaces are also emerging as a promising ground for innovative applications and games.

Sketch-based interfaces are designed to control a computer application, with the same fluidity of writing on paper. The main challenge is that the input is unconstrained and there may be an entire range of valid ways to trigger a specific action. This is in contrast with traditional interfaces in which we can establish a direct mapping between input events and actions. In fact, sketches are inherently ambiguous and their interpretation often leads to some fundamentally under-constrained or even ill-posed problems.

Our work focuses on the analysis of *informal drawings*. An informal drawing is a freely-drawn sketch, whose semantics are defined by the general shape of strokes and their overall arrangement in the sketch. Our approach is designed for applications that can be driven with these kinds of sketches. We do not attempt to do precise symbol recognition, which requires a substantially different methodology.

The main purpose of our framework is to analyze a given sketch and build an internal representation of its semantics. This representation is a quantitative description of the sketch that is used to drive an output module. In our case, any

application that is capable of utilizing this description is a valid output module. The process of recognition is only concerned with the aforementioned analysis of the sketch and is the main focus of this paper. However, the main purpose of our work is to devise an approach that can work well on devices with minimal hardware capabilities, while still allowing good artistic control for practitioners designing rich interactive applications. We also insist on the importance of separating the recognition stage from output stage; this notion is very important when designers seek to create an interface that works consistently across a large range of devices with vastly different hardware specifications. We showcase the results of our framework working with several different output modules, such as the Poser animation system, a custom graphics engine on the Nintendo DS and the iPhone (Figure 1). Figure 3 shows a high level diagram of our framework. The recognition pipeline is organized in three distinct stages: *stroke classification*, *template matching*, and *refinement*. *Stroke grouping* is an optional stage that interacts with the classifier and extends the range of sketches that our framework can handle reliably. Our analysis proceeds by first probing only higher level features of the sketch and then gradually moving to a more detailed inspection of the drawing. The classifier establishes a correspondence between individual components of the sketch and components of the output module. A component of the sketch may be an individual stroke or a group of strokes replaced by a single representative stroke as explained in Section III-D. The classifier is based on a machine learning approach and requires a *training set*. The template matcher compares the sketch against a library of templates that encode the semantics of the application domain quantitatively. For every component of the sketch the template matcher picks the template that best matches the features of that component. The refinement module adjusts the representation of the sketch built so far by considering nuances of the drawing. A key advantage of our framework is to encapsulate the analysis of the sketch, so that sketch recognition is independent of the applications is used for. As a result, the *output module* can be any tool that can interpret the quantitative description of the sketch generated by our recognition pipeline.

We illustrate this process by referring to the sketches in Figure 1. First, the classifier labels the strokes as: mouth, nose, eyes, and eyebrows. Then, the template matcher uses the template library to discriminate between a smile and a frown or an open eye and a closed eye. Lastly, the refinement stage will probe the mouth stroke to determine the intensity of the smile or frown. Each stage of the pipeline is art directable and can be adapted to work with completely different application domains as explained in Section V.

We demonstrate our work by presenting a concrete application that runs in real-time on both a high-end PC, a low-end PC, and mobile platforms with limited resources (Section IV-A). In this paper we analyze the proposed pipeline in detail and also discuss the technical challenges that we had to overcome in order to adapt our technique to fully run on mobile platforms with an emphasis on the Nintendo DS, which

is our most limited target platform. The main highlight of our work is to show that, by employing a robust sketch recognition engine and maintaining a full separation between the analysis of the sketch and the output module, we obtain an interface that is highly adaptable and works consistently across a variety of platforms.

The contributions of this paper are:

- We present a framework for sketch recognition that leverages machine learning, template matching, and other techniques to achieve the level of efficiency and flexibility that is required to target applications that must work consistently on wide range of devices. Furthermore, our work demonstrates that is feasible to run these techniques on mobile devices (Section IV).
- We propose a series of techniques that elevate sketch-recognition to the level robustness needed by real-world applications.
- We demonstrate that the entire recognition pipeline is highly efficient and can even work in real-time on a Nintendo DS, which has very limited hardware resources.

II. RELATED WORK

One way to obtain usable sketch-based interfaces for computer graphics is to establish a direct mapping between strokes and geometry [1]. These approaches tie the analysis and handling of the sketch with the final output closely together. However, these methods are unsatisfactory when the interface must scale to a range of devices with varying computational capabilities. With this paradigm, a high-end device capable of manipulating detailed models and accepting precise input will offer a rich user experience, but the same interface running on a low-end device will offer an experience that is rather poor in comparison and significantly inconsistent with the other. Our work instead separates the analysis of the sketch from the target application by generating a reusable quantitative description of the sketch. Therefore, when we provide a framework that can perform the same analysis on all devices, regardless of how well these devices can manipulate an output model, then we can enable applications that can work more consistently across different platforms.

For this reason we focus our method on the problem of sketch recognition rather than insisting on a specific target application. This contrast is also evident by comparing our work to the papers by Chang et al. [2] and Lau et al. [3], that focus specifically on posing facial expressions—an application similar to the one we present in Section IV. We also point out that the computational requirements of these methods make them largely unfeasible on mobile devices.

Closely related to our work are the papers by Yang et al. [4] and Sharon et al. [5] that exploit template matching and stroke classification for the interpretation of sketches. The work by Yang et al, however, focuses on modeling which is a different problem than ours, while the paper by Sharon et al. is restricted on the problem of classification (labeling) and does not allow for the more general analysis of drawings that is possible with our full recognition pipeline. Furthermore, these

approaches are not designed with mobile applications in mind. There are several interfaces for the recognition of sketches within specific application domains, such as chemistry, electric circuits, or music. In contrast, our work does not focus specifically on the vernacular of a particular domain, but it focuses on the recognition of informal drawings, which are more appropriate for applications in computer graphics and animation.

Substantial effort is also devoted to the study of more general recognition frameworks and the difficult problem of multi-domain sketch understanding [6]. We do not attempt to achieve the same level of generality for sketch-recognition as these papers. Nevertheless, this area of research was an important basis for our design of a framework that can recognize informal drawings robustly and with sufficient generality.

Several papers study specific sub-tasks related to the problem of sketch recognition, such as parsing [7] and grouping [8]. We do not consider directly the problem of parsing in our work, but we exploit the concept of grouping to enable our approach to work with complex sketches.

Another notable domain of research studies languages for describing sketch-recognizers, such as LADDER. These tools, however, are appropriate for a different class of interactive techniques that rely on the recognition of symbols as opposed to informal drawings (Section I) and require more technical expertise; thus they are more difficult to direct artistically.

III. APPROACH

A brief overview of our approach is given in section I. In this section we describe each stage of our recognition pipeline in detail with reference to figure 3.

A. Shape Attributes

Shape attributes (or shape descriptors) are quantitative descriptors that are used to build a sound representation of the sketch. They have been used extensively in the past as a means to improve the performance of sketch recognition [9] and address the problem of fast shape retrieval from large databases. In general shape attributes are functions that analyze a stroke or a collection of strokes and return a scalar that measures a desired property. Shape attributes cannot be used to recover the original shape, but they are useful for discrimination. In our case, we use shape attributes to:

- Obtain a statistically well-behaved representation of strokes to improve the reliability of the classifier.
- Improve the performance of the template matcher to discriminate between shapes.
- Probe perceptually meaningful features of strokes to refine our analysis of the sketch.

To obtain these properties, we designed shape attributes based on concepts of statistical pattern classification as explained in Section III-B, and the general literature on human perception of shape [10].

Shape attributes in our framework can be categorized based on two criteria:

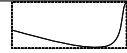



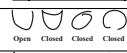

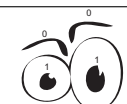
Name	Example	Indicates
Bounding Box		Size
Centroid		Position
Ordering		Arrangement
Orientation		Orientation
Topology		Open or Closed?
Concavity		Upward or Downward?
Depth		Overlap

TABLE I
THE LOCAL SHAPE ATTRIBUTES USED IN OUR FRAMEWORK.

- 1) The level of granularity they offer in describing detailed aspects of strokes.
- 2) Whether they describe strokes in isolation or with respect to each other.

We picked shape attributes so that each stage of analysis is working at the most appropriate level of abstraction and to take advantage of both local and global features of the drawing. Table I illustrates the local shape attributes that we use in our framework. For a detailed description and algorithmic details of the shape attributes used for classification and grouping please refer to prior work by the authors [8]. The *topology* and *concavity* shape attributes are detailed in the Appendix and used for template matching (Section III-E). A local shape attribute measures properties of individual strokes and their relationships. A global shape attribute probes overall features of the drawing which are not related directly to how each individual stroke is drawn.

For global shape attributes we use the *overall bounding box* of the entire sketch and the *global count* of strokes in the sketch, which helps to disambiguate certain special cases. When appropriate, shape attributes are normalized with respect to the overall bounding box to make them independent of scaling.

B. Classification

The purpose of the classifier is to establish a correspondence between components of the sketch and related components of the output module. We formulate the classification problem as assigning a user-defined label to each component of the sketch. Since our work emphasizes interactive applications and the robustness of the recognition process, we restrict at first the classifier to assign labels to individual strokes. In order

to further our control over the classification, we make two additional assumptions:

- 1) The sketch must not contain strokes that do not have any valid user-defined label. We call these outliers.
- 2) The sketch does not contain multiple strokes that belong to the same class. We call these duplicate strokes.

We refer to sketches that satisfy these two assumptions as *clean sketches*. Our classification mechanism is designed to offer the best level of classification robustness for clean sketches. As explained in Section III-D, the reliability of the classifier for clean sketches allows us to extend our classification mechanism to deal with sketches that violate the stated assumptions.

Rule-based mechanisms and statistical methods are the two dominant approaches for classification that are commonly used for sketch recognition. While over the years several grammars and languages were proposed to ease the development of robust rule-based symbol recognition systems [11] these methods do not provide the level of artistic control that we desire. Yet an even greater problem is that rule-based approaches rely on compound strokes and classify symbols based on their connectivity patterns. In our work we wanted instead to classify individual strokes based on their individual shape and arrangement in the overall drawing. For these reasons we decided to use a statistical approach.

The method we use for classification is a supervised machine learning technique based on Support Vector Machines (SVMs). The classifier is trained by drawing a sufficient (Section III-C) number of example sketches with manually labeled strokes. The example sketches with their labels constitute the training set for the SVM. The classifier then uses the statistical information of the training set to assign labels to the strokes of new sketches that are not found in the training set. In order to achieve the desired level of robustness, we rely on the statistical notion of data representation. In general a good training set for discrimination must satisfy two properties:

- Data points that belong to the same class should exhibit a small scatter.
- The *means* of two different classes of data points should be well-separated.

The common approach is to project the original data points to a lower-dimensional space that maximizes these properties. In statistics one can find this space automatically with methods such as the Fisher Linear Discriminant. However, for our purpose linear methods may not be able to effectively represent strokes and nonlinear methods are computationally expensive. Furthermore, we need more control over the recognition process and we can achieve better overall robustness by exploiting theories of visual perception [10]. Therefore, we devised a good representation space manually by means of shape attributes. We started with a set of over a hundred manually labeled sketches and implemented several shape attributes following the suggestions of the general literature on visual perception. Our original shape attributes mirrored closely the notions described in the classic work by Arnheim

Bounding Box Width
Bounding Box Height
Bounding Box Aspect Ratio
Centroid X
Centroid Y
Horizontal Ordering
Vertical Ordering
Overall Stroke Count
Depth

TABLE II
THE STABLE SELECTION OF SHAPE ATTRIBUTES USED FOR THE CLASSIFICATION.

[10]. By studying the class scatter and the inter-class *means*, we narrowed down the set of shape attributes so that the aforementioned properties are maximized. We repeated the same process with a comparable number of sketches belonging to different application domains to ensure that our selection is sufficiently general. Table 2 shows the shape attributes that we chose for classification. Using these shape attributes, we trained the classifier for three distinct types of drawings: facial expressions, houses, and cars. We studied the reliability of the classifier using cross-correlation tests and we obtained an average accuracy of 93%. The cross-correlation results show that our classifier is very reliable and at the same time is not overfitting the data, thus it has a good ability to generalize. The original training sets had about a hundred training examples each. We then reduced the number of training examples in each set and repeated the cross-correlation tests. We found experimentally that we need about 10 training examples per class in order to achieve the desired level of robustness.

C. Training

The classifier is trained interactively by drawing a sufficient number of clean sketches (refer to Section III-B) and manually labeling each stroke. It is important that the training sketches are varied, so that the classifier can easily learn important properties of the application domain. This follows from the fact that a statistically well-behaved training set should exhibit a large separation between class *means*. For each stroke s_i that is manually labeled, we generate a training vector t_i of the following form:

$$t_i = [l_i, A_1(s_i), A_2(s_i), \dots, A_m(s_i)]$$

where l_i is a value that uniquely identifies each class label and $A_j(s_i)$ is a scalar produced by a specific shape attribute from Table 2 applied to stroke s_i . Note that each training vector incorporates three kinds of information relating to the individual shape of each stroke, the relation between strokes, and global attributes of the sketch as discussed in Section III-A. The selection of class labels by the user is also important to address the problem of ambiguity resolution. For instance, the classifier will not reliably distinguish ambiguous class labels for a face, such as a smile and a smirk, but is reliable in distinguishing classes for the mouth and the nose. Instead, subtle variations like these are handled by the template matcher

and the refinement stages, discussed in Section III-E and Section III-F.

In Section III-B, we pointed out that in general ten training vectors per class are sufficient to obtain reliable classification. Thus, our method requires a fairly small training set for most applications. Furthermore, in most cases the training set is prepared only once for a given application domain and end-users should not be concerned with it. On the other hand, in our implementation a training set with about 120 training vectors can be processed in less than a second on an entry level laptop, so designers can also envision applications in which end users can interactively train the classifier and customize the behavior of the recognition pipeline.

D. Grouping

Our classifier makes some restricting assumptions on the input sketch as explained in Section III-B. These assumptions enable us to obtain a very reliable and robust classifier, but they also limit the range of sketches that the user can draw. Grouping strokes beforehand allows us to relax these assumptions and greatly extends the range of sketches that our approach can handle reliably. We replace each group of strokes with a representative stroke that captures important features of that group. As a result, the subsequent stages of the recognition pipeline can still operate as if we used an individual stroke per class label.

We repeat the assumptions of the classifier here for convenience:

- 1) Each class label is associated to a single stroke.
- 2) The sketch does not contain outliers.
- 3) The sketch does not contain duplicate strokes.

Our approach for grouping follows the method introduced by the authors [8], which we summarize in this section for convenience. The key observation of this technique is that we can always reduce a sketch that violates assumptions (2) and (3) to one that satisfies them by defining a proper grouping of strokes. Assumption (1) is accommodated by replacing each group with a single stroke that captures important properties of the shape of that group. The SVM classifier presented in Section III-B is used to find the most appropriate grouping within the space of possible groupings for a given sketch. This process is reliable and robust, because we ensured that our classifier is well-behaved for clean sketches. One fundamental difficulty with this and other similar methods is that the space of possible groupings is prohibitively large and it needs to be pruned beforehand to make the problem tractable. The pruning heuristics suggested by the aforementioned paper proved to be appropriate for the applications presented in our work as well. Figure 4 shows an example of a difficult sketch that can be correctly classified after grouping. In this example, the training set had distinct classes for mouth, nose, L/R eye, L/R pupils, and L/R eyebrow. One peculiarity of our method for grouping is that the training set differs from the actual sketches that we used to test the classifier. For a sample of 100 non-clean sketches, the grouping stage produces more than 90 % correct groupings and, when it fails, it rarely results in non-sensical

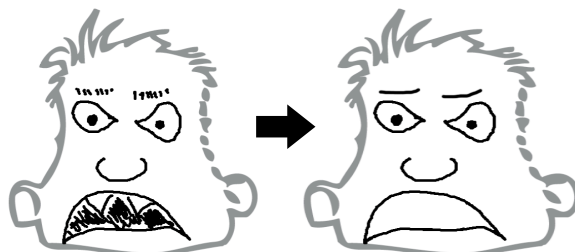


Fig. 4. A difficult sketch that is correctly handled by our framework using grouping. Left: the original sketch. Right: the representative strokes after grouping.

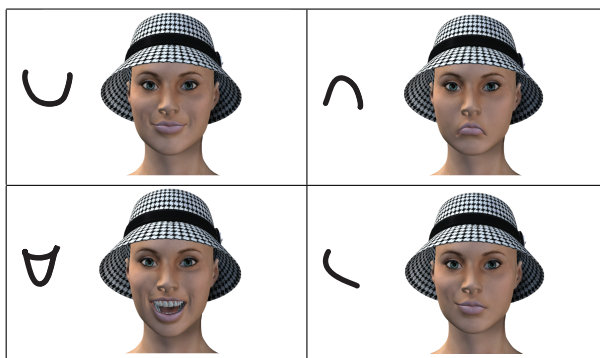


Fig. 5. A few sample templates for mouth expressions. Each template is composed of a template shape and a set of parameters that can be interpreted by the output module

groupings. For a more detailed analysis of grouping, please refer to [8].

E. Template Matching

We use a library of templates to produce a quantitative description of sketch semantics. Each template is composed of a representative stroke and a corresponding set of parameters that are usable by the output module. The significance of this representation is that the template library is easily understood in visual terms and therefore it lends itself well for art direction. We further scope the template library by defining a different set of templates for each class label specified in the training set. Figure 5 shows a few templates for one of the applications that we designed using our recognition pipeline. The role of the template matcher is to match input strokes to the most similar template in the library and feed the corresponding parameters to the next stage of the recognition pipeline.

For each classified stroke (or the representative stroke of a group) we query the templates that correspond to its class and search for the template that is most similar. We perform template matching in two stages as to make the process fast and robust. We call these stages respectively: *hard matching* and *soft matching*. Hard matching uses special discrete shape attributes to exclude obvious mismatches and narrow down the size of the search space. This is important to avoid user frustration. In fact, we found that users prefer the interface to ignore a stroke completely, rather than produce results that

are obviously wrong. A good choice of shape attributes for hard matching requires some knowledge of the application domain, but for the sake of generality we found that the following shape attributes are visually relevant in most cases: *topology* and *concavity* (refer to Table 1 and the Appendix). Soft matching uses a metric to rank the remaining templates based on similarity and pick the best match. We experimented with a few established shape matching metrics to measure similarity, such as the modified Hausdorff distance, the Frechet distance, and other metrics based on turning angle [12]. In our experiments, all these metrics provide an excellent success rate above 90%, but these metrics suffer from fairly high computational requirements and are not practical for mobile platforms. Furthermore, the Frechet metric, which is the best performer, is too slow to be practical for applications even on high-end machines. Instead, for applications running on limited hardware an appropriate metric is constructed by computing the norm difference of multiple shape attributes. Simple shape metrics are viable because we resolve most ambiguities upstream in prior stages of the pipeline.

In order to obtain the highest level of robustness, it is important that template shapes exhibit substantial variation. This way we favor the ability of the template matcher to discriminate shapes and reduce the inherent ambiguities of the template library. On the other hand, we account for nuances in the input sketch at the refinement stage discussed in Section III-F.

F. Refinement

The refinement stage accounts for nuances in the input sketch by altering the parameters generated by the template matcher. We defer the refinement stage to the end of the pipeline, because this way we can perform the main portions of the recognition discriminatively and robustly in the classification and template matching stages. Furthermore, because of this important design choice, we do not have to specify subtle features of the sketch as part of the training set and template library. Consequently, we can reduce ambiguities and favor the recognition process by relying on a data set that has good statistical features. Yet we can still provide a good level of control to the user by means of a dedicated refinement stage.

We refine each stroke (or the representative stroke of a group) individually. We define the refinement of each stroke by three elements: 1) refinement function. 2) lower bound. 3) upper bound. The refinement function uses the lower and upper bounds to normalize the value of a specified shape attribute to the range $[0, 1]$. For each template in the template library we define a configuration table that specifies the values of these three elements. The bounds can be defined both as absolute numbers or relatively to other well-defined elements of sketch, typically a class label. As an example for a facial expression, we can refine how much to open the mouth, by parameterizing the height of the mouth stroke with respect to the bottom of the nose and the top of the chin. Our approach to refinement is straightforward, art directable, and has minimal computational requirements.

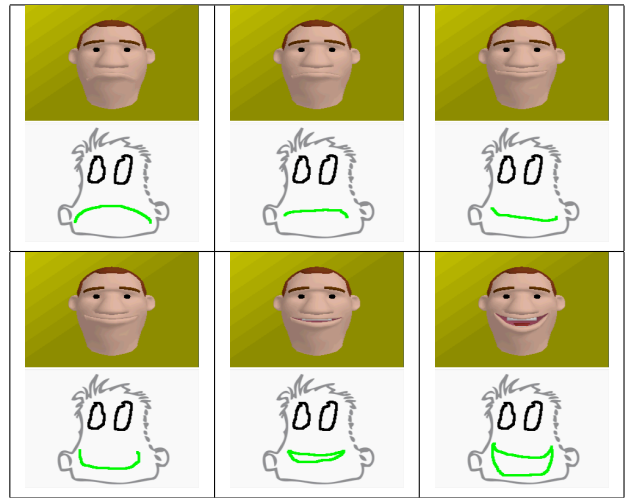


Fig. 6. A range of mouth expressions that we obtained with refinement on the Nintendo DS. The images were captured from an emulator for convenience.

G. Output Module

One key feature of the proposed recognition pipeline is that it keeps the recognition process completely separate from the output module. As a result the sketch recognition process can be used to drive a variety of different output modules that may operate rather differently from each other. The only requirement is to implement a thin interface layer that translates the quantitative description generated by our system to the one required by the chosen output module. To demonstrate the flexibility of this framework, we used our pipeline with several different output modules without modifications, except for a rather minimal external interface layer between our recognition pipeline and the output module:

- 1) The Curious Labs Poser commercial animation package.
- 2) A custom blendshape based animation engine on the Nintendo DS.
- 3) A custom scene-graph engine on the Nintendo DS.
- 4) A custom blend shape based animation engine on the iPhone.
- 5) A vector based engine for posing cartoons generated in Adobe Illustrator.

An additional advantage of our method is that a single sketch can be fed simultaneously to different visualization packages or reused at different times in a production environment and with different target graphic models. As a result, artists can use a sketch interface to consistently express their ideas independently of the art assets and software used for final production.

IV. RESULTS

We have implemented two distinct versions of the sketch recognition pipeline. We wrote the first version in the Python interpreted language and ran it on two different machines: 1) a laptop PC with a single core Pentium 4 CPU at 2.13 Ghz and 1 GB of RAM. 2) a desktop PC with a single core Pentium 4 at CPU 2.4 Ghz and 2 GB of RAM. The second version was

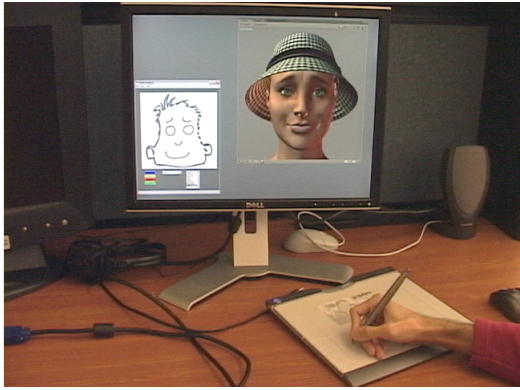


Fig. 7. A live shot of using our framework with Poser.

written in C++ and targeted to mobile platforms. This version was optimized to run entirely on a Nintendo DS handheld gaming system and it was ported to the iPhone with minimal effort. The two versions implement the same functionality with the exception that the PC version can perform stroke grouping and uses more refined algorithms for soft matching as explained in Section III-E. Furthermore, despite these minor differences, the recognition output is consistent. The entire process of recognition is completed in less than a second with real-time output even on the the most limited platform.

We demonstrate our method primarily with an application for posing facial expressions. We created the training set and the template library using a custom graphical tool. The training set contains 12 classes that represent the main components of the face with distinct classes for right and left components. The full training set contains 123 training vectors and was generated by drawing 41 different clean sketches (each sketch contains multiple strokes). The most complex template library that we used contains 23 distinct templates.

For the PC version we used the Poser animation package as the output module, which uses blend shapes for posing facial expressions. The communication between the two is provided by a short Poser script that translates our description of the sketch to the parameters required by the software. We successfully drove four distinct face models within Poser with our sketch-based interface, one of them being a high quality third-party face model. Figure 7 shows a user using our application, and Figure 8 shows the rendered Poser models and the corresponding sketches.

We also interfaced the PC version with a tool that can interpolate vector graphics to animate 2D cartoon facial expressions. Our sketch-based interface drove the 2D cartoons without modification and with the same data sets for the training set and the template library. Again, the interface between the two applications was provided by a short script. Figure 9 shows an expression generated with this output module. This result shows the advantage of separating the analysis of the sketch from the geometry of the target model.

The version for the Nintendo DS was developed from scratch and highly optimized to work on such limited hard-



Fig. 8. These images were generated by running our framework within Poser.

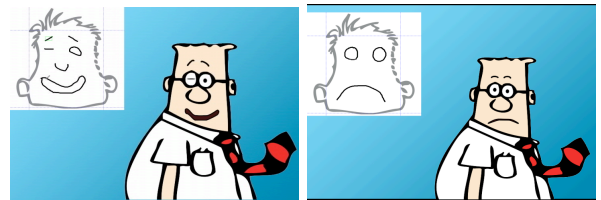


Fig. 9. Our recognition pipeline used to drive a 2D cartoon.

ware. However, it reads the same training sets and template libraries that we use on the PC version. We developed a 3D engine for this platform that can animate deformable models using blend shapes, similarly to the Poser engine for facial expressions. Nevertheless, we cannot use the same Poser 3D models on the Nintendo DS, due to their high polygon count. We use simpler 3D cartoon models on the DS, but the recognition pipeline operates in the exact same way. However, we can use Poser models directly on the iPhone, due to its more capable hardware. To use our interface with a second face model, we only had to modify the 3D

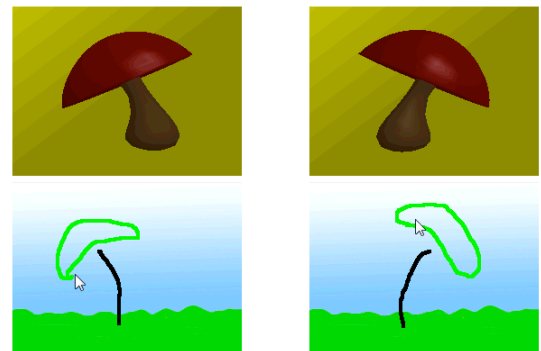


Fig. 10. A prototype application for posing mushrooms. We emphasize that our framework was adapted to such radically different application without having to write any code.

assets. Figure 1 and 6 show several examples generated with this implementation. Figure 10 shows a completely different prototype application we produced without writing any code. The included video shows our interface used in real-time on several distinct platforms and with different output modules.

A. A Major Challenge: Running the framework in real-time on the Nintendo DS and the iPhone

In this section we discuss the technical challenges that we had to overcome in order to run the full recognition pipeline on the Nintendo DS. The Nintendo DS was the the platform with the most severe hardware limitations in our experiments; thus we believe that the following discussion makes a good case for the feasibility of our approach on various mobile device. The highly optimized version that we developed for the Nintendo DS was later ported to the iPhone without major difficulties. The Nintendo DS has 4 MB of user memory and two main processors: an ARM7 at 33 MHz and and ARM9 at 67 MHZ. Our application is running entirely on the ARM9 CPU. The clock rate of this CPU is clearly orders of magnitude less than what we normally find on commodity computers, including Palm computers and Tablet PCs, which are the most common target for sketch-based interfaces.

A critical component of our framework is the Support Vector Machine we use for classification. Since the overall design of our recognition pipeline allows us to use a fairly small training set to get robust yet varied output, the computational requirements and the memory footprint for running the SVM are adequate to our needs. The full classification is performed in 0.2 seconds in average and requires an estimated 300 KB of memory at its peak. The classification is performed using a precomputed training model that the Support Vector Machine extracts from the actual training set. The actual computation of the training model is much more expensive computationally, but we can still run it on the Nintendo DS in an average of 51 seconds. This means that designers may devise applications in which users are allowed to alter the training set interactively.

For the template matcher, each distinctly labeled stroke needs to be compared against the full set of templates for its class. This gives an average run-time of $O(n * m)$ where n is the number of strokes in the input sketch and m is the average number of templates per class. For the first stage of the template matcher, we use discrete shape attributes, which are computed as the user draws each stroke. Thus, the run time for performing a hard match for each stroke is constant and the overall run-time for hard matching is extremely small. Soft-matching is performed using the Euclidean norm. This norm metric is not as effective as the more expensive shape metrics we use in our high-end implementation (Section III-E), but we were able to offset this deficiency by simply removing problem templates from the template library. The quality of the results is still very good, since the models on the Nintendo DS are less expressive in the first place.

The run-time of the refinement stage is linear in the number of classified strokes and again it does not present any substantial computational problem, since the required shape attributes

are computed as strokes are drawn. The refinement stage also counter-acts the fact that the template matcher is not able to perform accurate shape matching on the DS, by still providing a satisfactory response to nuances.

On the Nintendo DS one major problem is how to fit our data structures in the limited amount of available memory. We overcome this problem with highly optimized procedures for writing and reading data streams to the file system. This way we limit memory usage by caching data structures in and out of the file system as needed. This optimization is not needed on the iPhone, but when used it improves load times by a factor of a hundred in some cases.

V. DISCUSSION

The framework described in this paper focuses on the recognition of informal drawings and addresses many important problems of sketch-recognition. Our method also diverges from existing approaches in several key aspects. One key challenge of sketch-based interfaces is robustness. A robust interface is one that behaves in a predictable fashion and only rarely misinterprets the user input. For sketch-recognition a satisfactory level of robustness is particularly difficult to achieve, given the intrinsic ambiguities that affect unconstrained sketches. We tackle this problem by pushing the analysis of the most ambiguous aspects of the sketch to the latter part of pipeline at the refinement stage. To further this notion, we designed the entire pipeline so that each stage processes the sketch at a progressively lower level of granularity. The critical part of the recognition is performed by the classifier and the template matcher. We ensure the robustness of the classifier by choosing carefully the representation space of the sketch by means of shape attributes. This gives us a good statistical description of informal drawings. Classical studies in psychology [10] support our choice and we use quantitative methods to validate our approach against several user drawn sketches. We further improve the template matcher by removing unacceptable mismatches upfront discriminatively with hard matching. The refinement stage accounts for nuances in the sketch that would be otherwise very hard to detect robustly with statistical methods.

Another issue of great importance for us is to offer a framework that is flexible and can be easily adapted to different application domains. The success of frameworks like Adobe Flash show that it is very important for interface designers to have a tool that allows quick experimentation without time-consuming development work. As demonstrated in Section IV, our framework can be adapted to different application domains by merely modifying the training set and the template library, neither of which require any coding. For our applications, we had to implement a thin interface layer to translate the quantitative description of the sketch generated by our interface to the parameters of the output module. However, this small burden can be avoided altogether by establishing a standard for the parameterization of sketches.

Some of the most interesting platforms for deploying sketch-based interfaces are handheld systems with limited

hardware. Therefore, we tried to limit the resources needed by our recognition pipeline. As a result, we were able to implement the full recognition pipeline on the Nintendo DS and an iPhone with only minimal modifications in functionality with respect to the PC counterpart.

VI. CONCLUSION

We have presented a new framework for sketch recognition that is efficient, robust, flexible and can be easily adapted to platforms of limited computational power, such as mobile devices. Furthermore, well-defined stages in our framework allow a user to intuitively adopt it for different application domains. Our framework is suitable for educational and entertainment applications for which hand drawn sketches can be an effective interactive interface. Commodity mobile devices, such as the iPhone and Nintendo DS, can now support a wide range of practical applications that could benefit from our framework.

ACKNOWLEDGMENT

This work was partially supported by the NSF grant CCF-0429983. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NSF. We would also like to thank Intel Corp., Microsoft Corp. for their generous support through equipment and software grants.

APPENDIX

In this appendix we detail the *topology* and *concavity* shape attributes, which are used for template matching (Section III-E).

Topology: The topology attribute labels a stroke as either open or closed. We determine the topology using the *span angle* α_{span} which measures the angle subtended by the centroid and regular sample points on the stroke.

$$\text{topology} = \begin{cases} \text{closed} & \text{if } \alpha_{\text{span}} \geq \alpha_{\text{threshold}}, \\ \text{open} & \text{if } \alpha_{\text{span}} < \alpha_{\text{threshold}}. \end{cases}$$

The value of $\alpha_{\text{threshold}}$ should be a around 360 degrees, signifying that the stroke "wraps" around the centroid when its closed. We found in our experiments that a value of 340 degrees accommodates well the expectation of most users.

Concavity: The concavity attribute captures whether a stroke arches up or down. Thus we say that the concavity can be either upward or downward. We use the centroid as a reference for computing this attribute. If most of the points around the middle of the stroke are below the centroid we define the concavity to be upward and downward vice versa. More formally, given a stroke S let C be the centroid of S and p_i be some point in S . The concavity is then computed as follows

$$\begin{aligned} v_i &= C - p_i, \\ w_i &= e^{-\text{influence} \cdot \|v_i\|}, \\ \text{concavity} &= \sum_{i=1}^n w_i \text{sign}(v_{iy}), \end{aligned}$$

where v_i is a vector from the centroid to any given point of the stroke and $\text{sign}(x)$ returns the signum of x . If the y coordinate of v_i is positive then the point p_i is below the centroid, otherwise p_i is above the centroid. The contribution of each point is accounted using a weighted sum. We choose weights in a way that prioritizes points that lie closer to the centroid and decays exponentially as the distance with the centroid increases. We found in our experiments that setting the influence parameter to 10 produces results that are visually sound.

The concavity is always well defined for strokes that are topologically open, but may be ambiguous in this form for strokes that are topologically closed. Hence, we approximate closed strokes with their medial axis.

REFERENCES

- [1] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, "FiberMesh: Designing freeform surfaces with 3D curves," *ACM TOG '07*, vol. 26, no. 3, p. article no. 41, 2007.
- [2] E. Chang and O. C. Jenkins, "Sketching articulation and pose for facial animation," in *Data-Driven 3D Facial Animation*, Z. Deng and U. Neumann, Eds. Springer, 2007, ch. 8, pp. 132–144. [Online]. Available: <http://www.springer.com/978-1-84628-906-4>
- [3] M. Lau, J. Chai, Y.-Q. Xu, and H.-Y. Shum, "Face poser: interactive modeling of 3d facial expressions using model priors," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 161–170.
- [4] C. Yang, D. Sharon, and M. van de Panne, "Sketch-based modeling of parameterized objects," in *SBIM '05*, 2005, pp. 63–72.
- [5] D. Sharon and M. van de Panne, "Constellation models for sketch recognition," in *SBIM '06*, 2006, pp. 19–26.
- [6] C. J. Alvarado, "Multi-domain sketch understanding," Ph.D. dissertation, Cambridge, MA, USA, 2004, supervisor-Randall Davis.
- [7] L. B. Kara and T. F. Stahovich, "Hierarchical parsing and recognition of hand-sketched diagrams," in *UIST '04*. New York, NY, USA: ACM, 2004, pp. 13–22.
- [8] G. Nataneli and P. Faloutsos, "Robust classification of strokes with SVM and grouping," in *ISVC '07*. Springer-Verlag, 2007, pp. 76–87.
- [9] O. Veselova and R. Davis, "Perceptually based learning of shape descriptions," in *AAAI '04*, San Jose, California, 2004, pp. 482–487.
- [10] R. Arnheim, *Art and Visual Perception: A Psychology of the Creative Eye*. University of California Press, 1974.
- [11] G. Costagliola, V. Deufemia, and M. Risi, "Sketch grammars: a formalism for describing and recognizing diagrammatic sketch languages," *ICDAR '05*, pp. 1226–1230 Vol. 2, 29 Aug.-1 Sept. 2005.
- [12] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell, "An efficiently computable metric for comparing polygonal shapes," in *SODA '90*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1990, pp. 129–137.