EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2012)
L. B. Kara and K. Singh (Editors)

Concepture: a regular language based framework for recognizing gestures with varying and repetitive patterns

N. Donmez[1] and K. Singh[1,2]

[1]Dept. of Computer Science, [2]Dynamic Graphics Project
University of Toronto

Abstract

We present Concepture, a framework based on regular language grammars for the authoring and recognition of sketched gestures with infinitely varying and repetitive patterns. Such gestures, while often seen in gesture based applications are currently hard-coded and not customizable. We endorse an example-based workflow, where users author gestures by sketching one or more example instances of the gesture. We de-construct these examples into perceptible stroke segments. Adjacent segment-pairs further capture local spatial relationships between segments and these segment-pairs form the alphabet of a regular language. We then initialize a grammar for our gesture by admitting strings that represent the user provided examples. Grammar refinement is user-friendly, in that we automatically generate new candidate gestures that are visually presented to the user for verification as instances of the gesture. We show Concepture to be effective in efficiently authoring a number of common, yet difficult to recognize gestures, and illustrate it using clip-art and image annotation applications.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Sketch Based Interfaces—Gesture Recognition

1. Introduction

Figure 1: *Lucy in the sky with diamonds: a scene depicted using gestures of variable repetitive structure.*

The use of symbolism in human visual communication is ubiquitous, whether in the form of signature facial expressions, hand signals or gestural sketches. It comes as no surprise thus, that gesture-based interfaces are becoming increasingly popular, fueled by the recent explosion of devices supporting touch and pen-based interaction [JGHYLD09]. Originally motivated by the problem of handwriting recognition, the science of symbolism and gesture recognition from sketched strokes is at least three decades old [Spr79]. The symbolism depicted in sketches conveying visual concepts frequently embody a sense of gestalt, that is, our capability to capture whole forms instead of just a collection of simple lines and curves [Edw02]. Figure 1, for example shows childlike depiction of an imagined scene comprising a sun, stars, clouds, a boat on a river and a little girl. The waves are recognizable independent of their size or the number of ripples, the sun and stars are similarly not bound to a specific shape or number of rays or points.

While these sketches are instantly recognizable to humans, current gesture recognition algorithms are likely to fail to capture the variability in these drawings. Frameworks for general gesture recognition such as the simple but powerful $1 Gesture Recognizer [WWY07] classify strokes by matching them against a finite set of example templates. The challenge in recognizing the symbols in Figure 1 is that their

Figure 2: *Repetitive gestures are often hard-coded into interfaces: roll back/fwd, scratch-out, resistor/spring in scientific drawing, continuation arrow.*

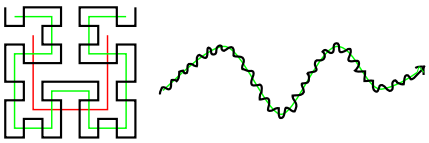


Figure 3: *Hierarchical gestures: a Hilbert curve and a layered sinusoid. We address the most detailed level.*

whole is bigger than the sum of their parts: not only do the parts or salient segments admit variations in shape, they are also repetitive, optional, and bear specific spatial relationships to each other. In current practice, such gestures cannot be authored by an artist or end-user sketching a few example instances of the gesture, but must be programmed to be recognized using logic specific to each type of gesture.

This paper thus presents the first framework where such repetitive gestures can be authored by providing one or more example instances of the gesture. Given these initial examples, we construct a number of candidate instances of the gesture for interactive user verification, to uniquely determine a Finite State Machine (FSM) that represents the gesture.

The utility of such gestures might not seem obvious from the simple motivating example in Figure 1. In more common use, Figure 2 shows example gestures that have been hard-coded into successful sketch-based interfaces [BBS08, IMT99]. Even more complex, pre-coded patterns have been used to retrieve architectural elements and material textures such as brick and shingles by sketching [CKX*08]. Current frameworks for gesture recognition typically operate by geometrically matching candidate strokes to a discrete set of examples that define a gesture, or at most some interpolated combination of them. These approaches are fundamentally ill-suited for the recognition of gestures in Figures 1 and 2, which requires an understanding of underlying patterns in the set of given examples.

Indeed, one needs to capture not only the given examples, but interpolate and extrapolate them based on a higher-level understanding of their shape. A body of research in shape perception [EM07] indicates that we mentally decompose complex shapes into segments based on various criteria such as sharp corners or turning points. We thus build our higher-level model of shape from stroke segments. Local context is captured by the spatial relationships between adjacent segments. At a global level, the stroke connecting a representative point of each segment can be treated as a hierarchical gesture as exemplified by the Hilbert curve in Figure 3. In practice though, one seldom sees gestures defined using fractal strokes, and a two level representation of shape, as implemented in this paper, is sufficient.

Our approach thus works on two levels. At the lower level, strokes are segmented via detection of corners and turning points and adjacent segment-pairs are identified as elements of a shape alphabet. At the higher level, a candidate gesture is represented as a string made from this alphabet. This abstraction allows us to apply regular language inference techniques to both the representation and recognition of gestures. Our authoring workflow is interactive. A user provides a number of example instances of the desired gesture. These are converted to strings from which a regular language is inferred by generating other strings as candidate gestures, which are displayed to the user for verification. The user accepts/rejects the candidates until the desired grammar representing the gesture is inferred. Once authored, our recognizer efficiently classifies candidate gestures by converting them to a string that is matched against the grammars of all defined gestures.

We provide an overview of related work (section 2), followed by details of our framework, named Concepture (Section 3,4). We illustrate our framework via clip-art drawing and image annotation applications (Section 5) and discuss limitations and extensions of our approach (Section 6).

2. Related Work

A recent survey of sketch recognition systems [JGHYLD09] categorizes this research into hard-coded recognizers, visual matching and textual description. Hard-coded gesture recognizers [Rub91, BBS08, IMT99] capture gestures such as a scratch-out or roll-back in Figure 2, but do not allow users to customize these gestures or add new ones. Within the domain of visual matching the simple and popular $1 Gesture Recognizer [WWY07] and its multi-stroke sequel [AW10], provide a reasonable solution by geometrically matching a given gesture to a user-given set of templates. In Concepture, we employ a variant of this recognizer at the lower level to visually match stroke segment-pairs and form the alphabet of gesture grammars.

While recognition engines based on scripting and textual description [HD05, Sti80, CDR05, SPRN02] are capable of encoding arbitrarily complex gestures, they require scripting skills to author new gestures. Even though our framework is based on regular language construction, this is transparent to users: They simply author gestures by providing one or more examples and verifying system proposed examples. Statistical and machine learning based approaches like [LBKS07]

and [SD05] also assume that the end user is oblivious to the details of the underlying model. Although both approaches are capable of handling small variations in sketching style, such as the stroke order in multi-stroke figures [LBKS07] or differences in the number of strokes that define the same figure [SD05], neither method is suitable for encoding the extent of variation that can be captured via a regular language. In contrast, Concepture can recognize a wave gesture with any number of ripples after seeing a single training example and answering two visual queries.

Concepture is also related to a few recognizers built in the context of 3D modeling [YSvdP05] and animation [TBvdP04], that allow shape variation and the hard-coded use of a gestural language. While we draw inspiration from this overall body of work and build upon a lightweight template-based gesture recognizer [WWY07], we believe Concepture is the first framework to allow easy authoring of gestures with variational and repetitive patterns.

3. Terminology and Framework

We use the term *gesture*, to define a class of sketches which share shape features that can be perceptually identified under a common name. Gestures consist of one or more strokes. Each stroke is defined as a sequence of points that have been sampled from a pointing device such as a mouse or stylus. Formally, a gesture instance is a sequence of $n \geq 1$ strokes $S^1, S^2, ..., S^n$ and each stroke S^i consists of a sequence of $m \geq 2$ points $S^i_1, S^i_2, ..., S^i_m$. Note that n and m are not fixed and may assume different values for different gestures or even different instances of the same gesture. Whenever it is clear from the context, we will drop the superscript i for brevity.

A *segment* is a perceptual part of a stroke delimited by annotation points. *Annotation points* are defined as corners and turning points. Two adjacent segments form a *segment-pair*. In our framework, gestures are converted into language strings where each letter in the alphabet is a segment-pair. Gesture recognition is thus reduced to a regular language string matching problem.

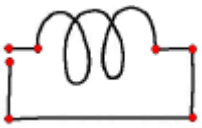
4. Authoring a Gesture

Users author a new gesture interactively by first drawing one or more examples then answering visual queries generated by Concepture. Each example is processed to find corners and turning points. These points are used for stroke segmentation. Two adjacent segments form a segment-pair, allowing successive segment-pairs to overlap. Each segment-pair is matched against a shape alphabet using a modified version of [WWY07]. This shape alphabet is incrementally built from an empty set: if a segment-pair does not match any of the shapes in the alphabet, a new shape is added to the alphabet. Each example is then represented as a string consisting of letters taken from the shape alphabet. These strings are input to the language inference algorithm, which proposes new examples for the gesture. The user either accepts or rejects each example. Based on the collected information, the inference algorithm builds a Finite State Machine that recognizes the language of the gesture.

An overview of the training procedure is given in Figure 4. In the rest of this section, we give details for each sub-task.

4.1. Stroke Segmentation

Before we start the segmentation we resample the strokes as in [WWY07]. We then annotate corners and turning points. Curvature indicators and corner processing is common in gesture recognition and sketch-based stroke processing [MS09]. While various methods for corner detection exist [WEH08, XL09], most of these approaches are fine tuned to detect only sharp corners where curvature is discontinuous. Since our goal is to segment strokes into meaningful individual units, such as each ring of a spring gesture (see figure on the right) we are interested not only in sharp corners but also points of very high curvature. As a result, we use an approach that is adapted to find curvature outliers in addition to any discontinuity. We estimate an indication of discrete curvature of a point S_j as:


[XL09]

$$\kappa(S_j) = \frac{\left(\sum_{t=j-k+1}^{j+k-1} \theta_t\right)}{\sum_{t=j-k+1}^{t=j+k} ||S_{t-1} - S_t||} \quad (1)$$

where θ_t is the angle between vectors $S_t - S_{t-1}$ and $S_{t+1} - S_t$. k is a parameter that averages out curvature noise, which we set to $k = 3$ in practice. Points with curvature greater than 2 standard deviations above the mean curvature of all points in the stroke are marked as corners. This method avoids the need for a fixed corner threshold and instead adapts to the stroke assuming that corners are a small fraction of the total number of points. We also set a minimum standard deviation of 0.05 to avoid picking up false positives on near circular arcs with low deviation of curvature. We further prune corners with proximity to other corners; corners less than 5 points from each other on a stroke are merged.

Once the curve is segmented by corners, we search for turning points within each segment. To calculate the tangent at a point S_t, we adapt a second order approximation based on the Taylor Series expansion of the curve [LBS05]:

$$t = \frac{|d||e|}{|d|+|e|}\left(\frac{d}{|d|^2} + \frac{e}{|e|^2}\right) \quad (2)$$

where d and e are vectors that denote $S_t - S_{t-1}$ and $S_{t+1} - S_t$ respectively. If the tangent of a point is within a

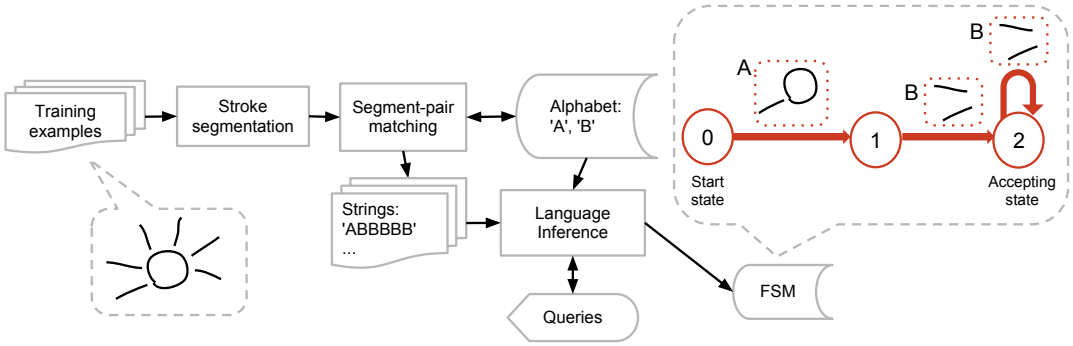


Figure 4: The training framework of Concepture. *The hand-drawn examples are first processed to find annotation points in order to segment the strokes. Stroke segmentation is followed by a segment-pair matching process that simultaneously constructs a shape alphabet for the gesture. The segment-pair matching process returns a string consisting of letters taken from the alphabet for each drawn example. Using these strings, the language inference engine generates membership queries for the gesture. Based on the user's answers to the queries, the program builds a Finite State Machine (FSM) that recognizes the language of the gesture.*

small threshold of the starting tangent of the segment and if the tangent of the curve has completed a full rotation, we annotate that point as a turning point. Addition of a new annotation point introduces a new segment starting at that point. The process is then repeated recursively for the new segment until no new turning point is found.

4.2. Segment-pair Matching

We encode the spatial relationship between adjacent segments in a stroke by representing our strokes as a sequence of overlapping segment-pairs. In our framework, segment-pairs are atomic primitives that are visually matched against each other using a variant of the $1 Gesture Recognizer (GR) [WWY07].

Briefly, the algorithm resamples the primitives using the same number of points and scales them to a unit measure at the origin. Unlike $1-GR, we scale the segment-pairs uniformly to unit total path-length as opposed to a unit square, making it possible to distinguish an "L" from a "V". An overall point-to-point distance minimizing orientation is found and this pairwise distance between the points serves as a measure of visual similarity between the two primitives. Using this value as a score, we decide whether the best match is significant enough to label the segment-pair with an existing member of the alphabet. If the score is less than a predefined threshold, we add the segment-pair as a new shape to the alphabet. This procedure is analogous to adding a new template in the framework of $1-GR. We handle multiple disjoint strokes similar to the $N [AW10] variant of $1-GR, however we require the drawing order to be consistent.

4.3. Language Inference

The segment-pair matching procedure returns a string for the gesture. We thus reduce the problem to one in Computational Linguistics, where a structured grammar has to be learned from a set of examples. This problem is known as Formal Language Inference, where each string is a sequence of letters taken from a fixed alphabet. Though arbitrary languages may be difficult or impossible to infer computationally, a subset called "Regular Languages" can be identified in polynomial time using a finite set of examples. Formally, these represent all languages that are accepted by a Finite State Machine (FSM) [dlH05].

In Computational Linguistics terms, the strings that belong to the language are called positive examples of the language. Except for stochastic or heuristic algorithms [CO94], regular language inference algorithms also require negative examples, that are made from the same letters of the alphabet albeit do not belong to the language being inferred. Asking a user to both imagine such negative examples and provide enough variety to uniquely determine the language is cumbersome cognitively and in terms of workflow. Instead, we adapt an algorithm [Ang81] that generates membership queries as our inference engine. This method uses positive examples to build an initial FSM of the language, and then progressively improves this FSM based on the answers to membership queries. If the initial set of positive examples constitutes a representative sample of the language, this algorithm is guaranteed to converge to the true language. Moreover, the number of membership queries needed by this algorithm is bounded by knN, where k is the size of the alphabet, n is the number of states of the minimal FSM accepting the language and N is one more than the size of positive examples.

Below, we provide a brief summary of the inference algorithm. We suggest interested readers see [Ang81] for a formal description and correctness proof.

4.3.1. The Inference Algorithm

Let L be a regular language defined on a finite alphabet U and let $A = (Q, q_0, F, \delta)$ be the canonical acceptor (i.e. the FSM with the smallest number of states) for L; where Q is the set of states, q_0 is the starting state, F is the set of accepting states and δ is the transition function that maps $Q \times U$ to Q. We also define λ to denote the empty string. λ is added to U if it is not already a member. The goal of the inference algorithm is to construct A given a representative set S of strings sampled from L and an oracle to answer membership queries.

Let P be the set containing the prefixes of S including λ and let $P' = P \cup \Delta$, where Δ is a special symbol not contained in U. The algorithm creates a set of states where each element of P' is represented by a distinct state. We also define a transition function f as follows: $f(\Delta, b) = \Delta$ for each $b \in U$ and $f(u,b) = ub$ for all $(u,b) \in P \times U$, creating a new state for each $ub \notin P$. Let T' denote this extended set of states. We also define $T = T' - \{\Delta\}$.

The motivation behind the algorithm is that T' contains a representative for every state in A, but we need to decide which states to merge in order to recover A. To achieve this, we begin with an initial partition of T' into accepting and non-accepting states and successively refine it. In this process, we generate a set of strings $V = \{v_o, v_1, ..., v_m\}$ such that no two states of A have the same behaviour on V. That is to say, for any two states q and q' in A, there exist a $v_i \in V$ such that $\delta(q, v_i) \in F$ and $\delta(q', v_i) \notin F$ or vice versa.

At each iteration i, we define $E_i(\Delta) = \emptyset$ and $E_i(u) = \{v_j : 0 \leq j \leq i$ and $uv_j \in L\}$ for $u \in T$. The i^{th} partition is constructed by putting all states with a common value of $E_i(u)$ in the same block. The first iteration starts with $v_0 = \lambda$. For each successive iteration, we set v_{i+1} as follows. We search for a pair of states $u, v \in P'$ and a symbol $b \in U$ such that $E_i(u) = E_i(v)$ but $E_i(f(u,b)) \neq E_i(f(v,b))$. If such a pair exists, we choose an arbitrary string $w \in E_i(f(u,b)) \oplus E_i(f(v,b))$, where $\oplus$ denotes the symmetric difference of two sets (i.e. $A \oplus B = (A \cup B) \setminus (A \cap B)$). We then set $v_{i+1} = bw$.

If no such pair is found, the algorithm constructs A using the final partition as follows. Each distinct set $E_m(u)$ for some $u \in T$ represents a state of A. The start state is taken to be $E_m(\lambda)$. The accepting states are those sets $E_m(u)$ such that $\lambda \in E_m(u)$. The set with $E_m(u) = \emptyset$ is taken to be the sink state, hence the transition on any input b from $E_m(u) = \emptyset$ is to itself. If $E_m(u) \neq \emptyset$, there $\exists u' \in P$ such that $E_m(u) = E_m(u')$, and the transition from $E_m(u)$ on input b is to $E_m(u'b)$ for all $b \in U$. Note that, depending on the language, the sink state may be unreachable after this construction. In such cases, it is omitted from A.

An example Suppose we would like to learn the "Sun" gesture depicted in Figure 4. The regular language for this gesture is denoted by ABB^*. Assume we are given the string AB as a representative set. We define:

$$P' = \{\Delta, \lambda, A, AB\} \quad (3)$$
$$T' = \{\Delta, \lambda, A, B, AA, AB, ABA, ABB\} \quad (4)$$

By definition, we set $v_0 = \lambda$ and generate queries of the form $u + v_0$ for each $u \in T$: $\lambda, A, B, AA, AB, ABA, ABB$. Since $AB + \lambda$ and $ABB + \lambda$ are in L, we set:

$$E_0(AB|ABB) = \{\lambda\} \quad (5)$$
$$E_0(\Delta|\lambda|A|B|AA|ABA) = \emptyset \quad (6)$$

We have $E_0(\lambda) = E_0(A)$ but $E_0(f(\lambda, B)) = E_0(B) \neq E_0(AB) = E_0(f(A,B))$, so we pick $w = \lambda$ and set $v_1 = B + \lambda = B$. At this iteration, we ask the queries $B, AB, BB, AAB, ABB, ABAB, ABBB$. Since $A + B$, $AB + B$ and $ABB + B$ are in L, we set:

$$E_1(AB|ABB) = \{\lambda, B\} \quad (7)$$
$$E_1(\Delta|\lambda|B|AA|ABA) = \emptyset \quad (8)$$
$$E_1(A) = \{B\} \quad (9)$$

Now we have $E_1(\Delta) = E_1(\lambda)$ but $E_1(f(\Delta, A)) \neq E_1(f(\lambda, A))$. Thus we pick $w = B$ and set $v_2 = A + B = AB$. Then the queries are $AB, AAB, BAB, AAAB, ABAB, ABAAB, ABBBAB$. Since we have $\lambda + AB \in L$:

$$E_2(AB|ABB) = \{\lambda, B\} \quad (10)$$
$$E_2(\Delta|B|AA|ABA) = \emptyset \quad (11)$$
$$E_2(A) = \{B\} \quad (12)$$
$$E_2(\lambda) = \{AB\} \quad (13)$$

At this point there is no pair $u, v \in P'$ such that $E_2(u) = E_2(v)$ but $E_2(f(u,b)) \neq E_2(f(v,b))$ so we stop the algorithm and construct the canonical acceptor A. The sets $\{\{\lambda, B\}, \emptyset, \{B\}, \{AB\}\}$ above denote the states of A. The transitions are given as follows:

Source state		on input A	on input B
$\{\lambda, B\}$	$\rightarrow$	$\emptyset$	$\{\lambda, B\}$
$\emptyset$	$\rightarrow$	$\emptyset$	$\emptyset$
$\{B\}$	$\rightarrow$	$\emptyset$	$\{\lambda, B\}$
$\{AB\}$	$\rightarrow$	$\{B\}$	$\emptyset$

This acceptor is easily seen to be isomorphic to the FSM given in Figure 4 save that the sink state is omitted from the figure for brevity.

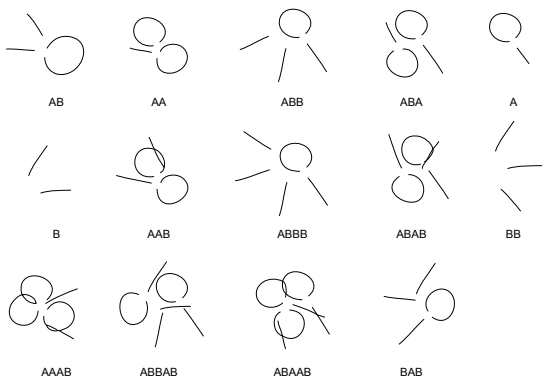


Figure 5: Visual queries generated for the sun gesture. *The gesture on the top left corner (i.e. AB) is the example drawn by the user. The positive queries are ABB and ABBB. The rest are negative queries.*

In practice, we only ask a query if it has not been seen before as a positive example or a previous query. We also assume $\lambda \notin L$. The complete set of the queries for the sun gesture is given in Figure 5. In the next section, we describe how these queries are visualized.

4.3.2. Visual Query Generation

Naturally, the inference algorithm generates queries from strings of the alphabet defined by the positive examples. These strings have to be converted to strokes before being presented to the user. Segment-pairs corresponding to the letters of the alphabet for this process are drawn from the positive examples. The first letter is placed in the middle of the canvas. Subsequent letters are placed by aligning the first segment of the current letter with the latter segment of the previous letter. Similar to segment-pair matching, this is achieved by calculating the orientation that gives the best point-to-point distance between the two segments.

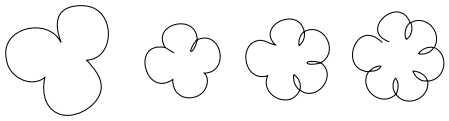


Figure 6: Examples of automatically generated queries for the cloud gesture. *The leftmost figure is the example drawn by the user. The successive figures are positive queries.*

Although this method works well for linear gestures such as a scratch-out gesture, it is problematic for closed gestures such as a cloud. In order to overcome this problem, we modify the above algorithm for closed gestures. First, we decide whether the gesture is closed or not by analyzing the positive examples drawn by the user. If the end points of the examples are always within a threshold, the gesture is marked as closed. For closed gestures we compute the angle difference between 2π and the turning angle from start to end of the string created as an open gesture. This angle of excess is accounted for at the segment boundaries producing a gesture with the right orientation of its segments. The start and end points are then snapped together, producing visually better queries in the case of closed gestures (Figure 6).

4.4. Gesture Recognition

When recognizing gestures, the candidate gesture is segmented as explained above and segment-pairs are matched to the letters in the alphabet. Unlike training, no new letters are added to the alphabet during recognition: each segment-pair is labeled with the highest scoring match. The algorithm then compares the string computed for the candidate against the regular language of each existing gesture and classifies the candidate if a match is found.

The matching score produced by visual matching approaches [WWY07] allow gestural ambiguities to be handled by returning multiple gestures with similar scores. Our approach as described is more deterministic, in that the segment-pair is mapped to the best matching shape from the alphabet producing a single string representing the candidate gesture. This string is then tested for acceptance by the various grammars representing the authored gestures.

Our algorithm is easily adapted to handle gestural ambiguity by leveraging the segment-pair visual matching scores. For each segment-pair of the candidate gesture, instead of simply picking the best match, we can form sets using the n best matches. Combinations of these sets produce a number of strings that represent the candidate gesture. Each string also has a matching score averaged from the matching scores of its segment-pairs. Strings that are accepted by a gesture grammar can then be returned along with their average matching score, making the overall approach more robust to noise and ambiguity. While an exponential number of query strings may be generated, the system can remain interactive, since the FSM simulation time is negligible compared to that of segment-pair matching. As a comparison, while the total elapsed time to recognize 64 gestures (see Results for details) is 877 milliseconds, only 1 millisecond of this time is devoted to FSM simulation.

5. Results

Figure 7 shows several examples of gestures we have authored. Table 1 lists the number of examples and queries needed to successfully learn these gestures. We emphasize here that Concepture can learn all of these gestures from as little as a single example drawn by the user. Even though the subsequent number of verification queries generated by the system can be as high as 22 (eg. the scratch gesture), the

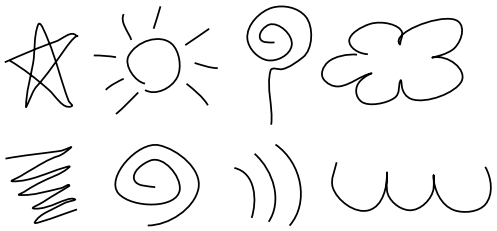


Figure 7: *Several gestures authored and recognized by our system. Top: star, sun, lollipop, cloud. Bottom: scratch, spiral, sound, wave.*

Shape	# Examples	# Queries	Time (sec)
sun	1	13	26
star	1	8	22
spiral	1	2	18
sound	1	2	14
cloud	1	4	21
lollipop	1	13	32
scratch	1	22	38
wave	1	2	16

Table 1: *The number of examples and queries that are needed to train our framework to recognize the shapes given in Figure 7. The time refers to the wall clock time of the entire training process (user+system) in seconds.*

workflow of simply accepting or rejecting each query as an instance of the gesture is both fast and easy for users, with a typical gesture being authored in under a minute. Informal user feedback indicates that such a training workflow is greatly preferable and more efficient than one where a user has to draw a number of, both positive and negative, instances of the gesture. Detailed statistics about the training process for each gesture including the final FSMs produced are presented in the Appendix.

Though a formal evaluation is the subject of future work, our preliminary test, in which a single user is asked to draw a randomized list containing each gesture a total of eight times, resulted in an average of 93% all-or-nothing accuracy for the gestures of Figure 7. Table 2 shows that while similar gestures are more likely to be confused with each other, the discriminatory power of Concepture is still promising.

Concepture-authored gestures can be used in a variety of applications. Systems such as ILoveSketch [BBS08] that employ various pre-programmed gestures like scratch-out and roll-back (shown in Figure 2) can both expand their vocabulary of such gestures and render them customizable by the end-user. Perhaps the most powerful aspect of gestures with repetitive patterns is their ability to simultaneously convey both a gesture and one or more numeric parameters via the number of repetitions (a volume level, for example, based on the number of sound wavefronts in Figure 7).

Figure 8: Clip-art application. *Learned gestures can be used to retrieve clip-art images to quickly illustrate a scene.*

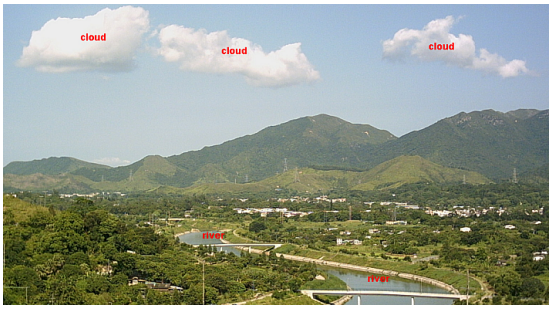


Figure 9: Image annotation application. *Learned gestures can capture regions of an image and their repetitive structure.*

We illustrate this with a drawing application based on clip-art retrieval and composition. The user trains the system by authoring gestures for several glyphs (sun, cloud, star, tree etc. in Figure 8). The user can then sketch freely. When a gesture is recognized, we search a repository for clip-art annotated to be of the same type and segment-pair structure as the user's sketch. Matched clip-art is then spatially transformed to fit and replace the sketch. Figure 8 depicts a snapshot of a scene composed with this application. Tasks such as image segmentation and labeling (see Figure 9) can also be enriched by using Concepture, so that not only are regions of an image segmented and labeled but their repetitive structure encoded by the over-sketched gestures (see accompanying video).

6. Conclusion

Our framework is a first attempt to automatically recognize gestures that conceptually capture an infinite family of shapes using regular language inference. Positive aspects of our system include an encapsulation of local spatial relationships between stroke segments with our shape grammar and

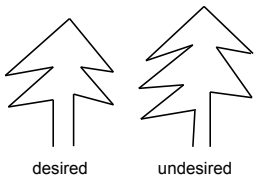
	sun	star	spiral	sound	cloud	lollipop	scratch	wave	none
sun	8	0	0	0	0	0	0	0	0
star	0	8	0	0	0	0	0	0	0
spiral	0	0	7	0	0	0	0	0	1
sound	0	0	0	8	0	0	0	0	0
cloud	0	0	0	0	7	0	0	1	0
lollipop	0	0	1	0	0	6	0	0	1
scratch	0	0	0	0	0	0	8	0	0
wave	0	0	0	0	0	0	0	8	0

Table 2: Confusion matrix of the test gestures. *Each gesture is drawn a total of 8 times. Each row corresponds to the drawn gesture, while columns depict the predictions made by Concepture. The last column represents the cases Concepture did not make a prediction due to a low score.*

a simple workflow where users are oblivious of the underlying language inference, providing only positive examples and then verifying system generated candidates. We illustrate our framework with clip-art drawing and image annotation applications.

While we are able to successfully recognize a variety of commonly used gestures, our current implementation has several limitations. While our approach can conceptually capture hierarchical structure, we currently only segment and analyse strokes at the most detailed level. In the future, we hope to handle hierarchical structure, by recursively treating the stroke connecting the centers of segments at one level as a more abstract gesture. This approach would also restore higher-level spatial relations between segments lost during the string generation phase, making it possible to distinguish between clouds and waves without relying on the closed vs. open distinction.

Our framework, like similar algorithms based on template matching of stroke geometry [WWY07], also suffers from a dependence on stroke ordering and correct stroke segmentation. Stroke ordering can be alleviated by treating permutations of strokes as multiple examples, or performing matches in image space [KS04]. Moreover, there is some evidence [SD05] supporting the notion that users have a natural tendency to draw strokes in a consistent order even if this order varies across subjects. A correct stroke segmentation, however, is fundamental to our approach as it defines the alphabet and subsequent conversion of a gesture to a regular language string. We believe that there is scope for improvement in both our criteria for choosing annotation points and the algorithms to identify them.



Our approach is limited to regular languages. While there are gestures that can only be represented by languages that are higherin the Chomsky hierarchy, typically, such languages can not be learned in polynomial time. On the other hand, our framework can approximate some of these gestures if false positives can be tolerated: e.g. a Christmas tree with a different number of branches on each side as illustrated above.

Despite these limitations, we believe our framework takes a step forward in gesture recognition research by facilitating the simple authoring and recognition of repetitive gestures, using a regular language representation.

References

[Ang81] ANGLUIN D.: A note on the number of queries needed to identify regular languages. *Information and Control 51* (1981), 76–87. 4, 5

[AW10] ANTHONY L., WOBBROCK J. O.: A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics Interface 2010* (2010), GI '10, pp. 245–252. 2, 4

[BBS08] BAE S.-H., BALAKRISHNAN R., SINGH K.: Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models. In *ACM Symposium on User Interface Software and Technology* (2008). 2, 7

[CDR05] COSTAGLIOLA G., DEUFEMIA V., RISI M.: Sketch grammars: A formalism for describing and recognizing diagrammatic sketch languages. *Document Analysis and Recognition, International Conference on 0* (2005), 1226–1231. 2

[CKX*08] CHEN X., KANG S. B., XU Y.-Q., DORSEY J., SHUM H.-Y.: Sketching reality: Realistic interpretation of architectural designs. *ACM Trans. Graph. 27*, 2 (2008), 1–15. 2

[CO94] CARRASCO R. C., ONCINA J.: Learning stochastic regular grammars by means of a state merging method. In *ICGI '94: Proceedings of the Second International Colloquium on Grammatical Inference and Applications* (1994), pp. 139–152. 4

[dlH05] DE LA HIGUERA C.: A bibliographical study of grammatical inference. *Pattern Recogn. 38*, 9 (2005), 1332–1348. 4

[Edw02] EDWARDS B.: *The New Drawing on the Right Side of the Brain*. Tarcher, 2002. 1

[EM07] E C., M S.: Geometric determinants of shape segmentation: Tests using segment identification. *Vision Research 47* (2007), 2825–2840. 2

[HD05] HAMMOND T., DAVIS R.: LADDER, a sketching language for user interface developers. *Elsevier, Computers and Graphics 29* (2005), 518–532. 2

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), SIGGRAPH '99, pp. 409–416. 2

[JGHYLD09] JOHNSON G., GROSS M. D., HONG J., YI-LUEN DO E.: Computational support for sketching in design: A review. *Found. Trends Hum.-Comput. Interact. 2* (January 2009), 1–93. 1, 2

[KS04] KARA L. B., STAHOVICH T. F.: Hierarchical parsing and recognition of hand-sketched diagrams. In *Proceedings of the 17th annual ACM symposium on User interface software and technology* (2004), UIST '04, pp. 13–22. 8

[LBKS07] LEE W., BURAK KARA L., STAHOVICH T. F.: An efficient graph-based recognizer for hand-drawn symbols. *Computers and Graphics 31*, 4 (2007), 554–567. 2, 3

[LBS05] LANGER T., BELYAEV A. G., SEIDEL H.-P.: Asymptotic analysis of discrete normals and curvatures of polylines. pp. 229–232. 3

[MS09] MCCRAE J., SINGH K.: Sketch-based interfaces and modeling (sbim): Sketching piecewise clothoid curves. *Comput. Graph. 33* (August 2009), 452–461. 3

[Rub91] RUBINE D.: Specifying gestures by example. *SIGGRAPH Comput. Graph. 25*, 4 (1991). 2

[SD05] SEZGIN T. M., DAVIS R.: Hmm-based efficient sketch recognition. In *Proceedings of the 10th international conference on Intelligent user interfaces* (2005), IUI '05, ACM, pp. 281–283. 3, 8

[Spr79] SPROULL R. F.: *Principles of interactive computer graphics (2nd ed.)*. McGraw-Hill, Inc., New York, NY, USA, 1979. 1

[SPRN02] SHILMAN M., PASULA H., RUSSELL S., NEWTON R.: Statistical visual language models for ink parsing. In *In AAAI Spring Symposium on Sketch Understanding* (2002), pp. 126–132. 2

[Sti80] STINY G.: Introduction to shape and shape grammars. *Environment and Planning B 7*, 3 (1980), 343–351. 2

[TBvdP04] THORNE M., BURKE D., VAN DE PANNE M.: Motion doodles: An interface for sketching character motion. In *ACM SIGGRAPH* (2004). 3

[WEH08] WOLIN A., EOFF B., HAMMOND T.: Shortstraw: A simple and effective corner finder for polylines. In *Proceedings of the 5th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (2008), SBIM '08, pp. 33–40. 3

[WWY07] WOBBROCK J. O., WILSON A. D., YANG L.: Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. *ACM Symposium on User Interface Software and Technology* (2007). 1, 2, 3, 4, 6, 8

[XL09] XIONG Y., LAVIOLA JR. J. J.: Revisiting shortstraw: improving corner finding in sketch-based interfaces. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (2009), SBIM '09, pp. 101–108. 3

[YSvdP05] YANG C., SHARON D., VAN DE PANNE M.: Sketch-based modeling of parameterized objects. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches* (2005). 3

APPENDIX

Here we provide detailed statistics of the training process of test gestures. For each gesture we list:

- Name of the gesture
- The example entered by the user
- The queries generated by the algorithm
- The final FSM learned by Concepture

Please note that we represent a segment-pair by the label that is automatically assigned by our algorithm. If two gestures share a label, this means Concepture has chosen to represent a segment-pair with a label that has already been defined when learning a previous gesture. The queries that are accepted by the user as positive examples of the gesture are underlined. Keywords **start**, **state**, **accept** and **sink** mark corresponding FSM states and the transitions are described as:

source state → (input) target state;

Name: **sun**
Entered: AB
Queries: AA, ABB, ABA, A, B, AAB, ABBB, ABAB, BB AAAB, ABBAB, ABAAB, BAB
sink 3 → (*A*)3; (*B*)3;
state 1 → (*A*)3; (*B*)2;
accept 2 → (*A*)3; (*B*)2;
start 0 → (*A*)1; (*B*)3;

Name: **star**
Entered: CCCC
Queries: CCCCC, CC, C, CCC, CCCCCC, CCCCCCC, CCCCCCCC, CCCCCCCCC
state 3 → (*C*)1;
accept 2 → (*C*)5;
state 1 → (*C*)5;
start 0 → (*C*)3;
state 5 → (*C*)2;
sink 4 → (*C*)4;

Name: **spiral**
Entered: D
Queries: DD, DDD
sink 2 → (*D*)2;
start 1 → (*D*)0;
accept 0 → (*D*)0;

Name: **sound**
Entered: E
Queries: EE, EEE
sink 2 → (*E*)2;
accept 1 → (*E*)1;
start 0 → (*E*)1;

Name: **cloud**
Entered: FF
Queries: FFF, F, FFFF, FFFFF
accept 3 → (*F*)3;
sink 2 → (*F*)2;
state 1 → (*F*)3;
start 0 → (*F*)1;

Name: **lollipop**
Entered: DG
Queries: D, DGD, G, DGG, DD, DGDG, GG, DGGG, DDG, DGDDG, GDG, DGGDG, DDDG
accept 3 → (*D*)2; (*G*)2;
sink 2 → (*D*)2; (*G*)2;
state 1 → (*D*)0; (*G*)2;
start 0 → (*D*)0; (*G*)3;

Name: **scratch**
Entered: CHC
Queries: CHCH, CHH, CH, CC, C, H, CHCC, CHCHC, CHHC, CCC, HC, CHCCC, CHCHH, CHHH, CCH, HH, CHCCH, CHCHCH, CHHCH, CCCH, HCH, CHCCCH
state 3 → (*C*)4; (*H*)2;
accept 2 → (*C*)0; (*H*)4;
start 1 → (*C*)3; (*H*)4;
accept 0 → (*C*)4; (*H*)2;
sink 4 → (*C*)4; (*H*)4;

Name: **wave**
Entered: I
Queries: II, III
accept 2 → (*I*)2;
sink 1 → (*I*)1;
start 0 → (*I*)2;