On-line Recognition of Hatched and Filled Regions in Hand-drawings

Raphael Thierjung raphael.thierjung@unibw.de
Florian Brieler florian.brieler@unibw.de
Mark Minas mark.minas@unibw.de

Institute for Software Technology
Computer Science Department
Universität der Bundeswehr München
85577 Neubiberg, Germany

ABSTRACT
Many sketches, as well as many visual languages make use of shapes consisting of (among other things) hatched or filled regions, e.g., transitions in Petri nets. Sketch tools recognizing these shapes therefore must also recognize such patterns. In this paper algorithms are presented for the recognition of hatched and filled polygonal regions in hand-drawings. For recognition of the former, the Hough transform is applied. Dedicated algorithms then detect the typical patterns of the hatched regions in Hough space. For the recognition of filled regions, some features are computed for each stroke. The features were selected according to a user study that was initially conducted. Both hatched and filled regions are reliably detected and with little computational effort, as it is shown by a prototypical implementation.

Author Keywords
Recognition, Filled Regions, Hatched Regions, Hough transform, Features

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: Misc.

INTRODUCTION
Sketching, i.e., drawing diagrams by hand, is a popular topic, as it allows for very natural input of diagrams. Instead of requiring graphical widgets like lists and buttons, users can simply draw like with pen and paper. Hardware supporting input via a stylus or the finger is available, prices are dropping, and such devices become more common in offices and at home.

To fully utilize the advantages of this input metaphor, there is also special software required. A stylus should not be treated as a replacement to a simple pointing device like a mouse. For hand-writing, for example, software is quite matured now, and there are commercially available products which reliably understand hand-writing. The same holds for hand-drawing. Various approaches exist here, some specialized for certain types of hand-drawings, some being more generic. Specialized approaches are, for example, [5] for stick figures, and [4] for music scores. Examples for generic approaches are [1, 7, 2, 10, 11].

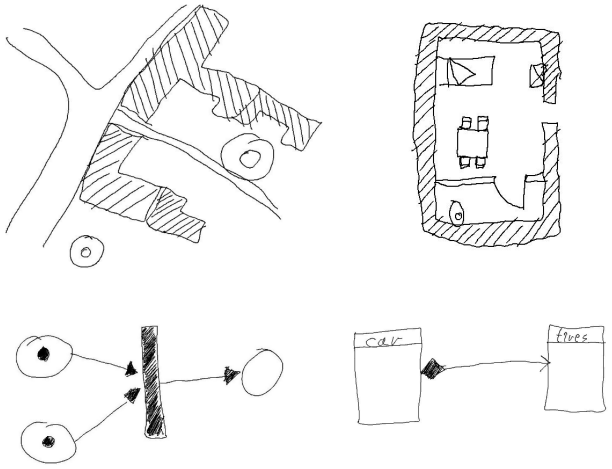

Figure 1. Examples of diagrams using hatched or filled regions: floor plan, architectural drawing, Petri net, class diagram.

Nevertheless, even many of the generic approaches do not support the recognition of hatched regions (also called *hatchings* in the following) or filled regions (also called *colored regions*). These are common for many types of diagrams, though. Figure 1 depicts some examples. In floor plans, hatched regions may be used to represent buildings, while in architectural drawings these patterns denote walls. In Petri nets, tokens and transitions are depicted by filled circles, in UML class diagrams, the composition arrow has a filled, diamond-shaped head.

In this paper algorithms are presented to detect hatched and filled regions in hand-drawn diagrams, where regions are described by polygons. The proposed approach assumes on-line recognition and can be included in sketching systems. Crosshatched regions can be recognized, too. Because hatched and filled regions are quite different, different algo-

IUI 2009 Workshop on Sketch Recognition
February 8, 2009, Sanibel Island, Florida
Chair: Tracy Hammond

rithms are utilized for the recognition of the two. For hatched regions, the Hough transform is applied, for filled regions a feature-based approach is used. Due to the diversity of hatched and filled regions, a user study has been conducted. This allowed us for observing how users actually draw such regions in practice, and to testify that our proposed algorithms are indeed useful.

This paper discusses only low-level recognition of hatched and filled regions. High-level aspects like the integration of the algorithms in a sketching framework are not considered. This includes the question of how to distinguish the outline of an object, e.g., a rectangle, from the strokes that make the hatching or filling. Also, ambiguities are not regarded, e.g., whether a stroke is a filling for a region, or another domain object like a spring?

The remainder of this paper is as follows. The next section discusses the user study and its results, and the following sections explain the algorithms for recognition of hatched regions, resp. of filled regions. The section on **Evaluation** reports findings made with a prototypical implementation. The section on **Related Work** describes related work. The final section summarizes this paper and gives future work.

USER STUDY

An initial user study was conducted. It served served two purposes: (i) typical drawing styles of users were be identified. Based on this, fair assumptions could be postulated for the recognition of hatched and filled regions, and values could be determined for required thresholds. (ii) a large number of practical examples were collected for later assessment of the reliability and performance of a prototypical implementation of the approach.

For the user study there were 31 participants, mainly students from our university. Each subject was first given some time to get used to the hardware, a Fujitsu Siemens Stylistic 4120 tablet PC. After the subject was fine with that, three times he had to draw a rectangle, a triangle, an ellipse and an arbitrary, closed shape. First, the shapes were to be drawn with an outline, each being hatched. Second, the outline should be dismissed, so that the hatched regions made the shapes. Third, the outline was drawn again, and the shapes had to be completely filled. The 12 samples taken from one of the subjects are depicted in Fig. 2. In total, 372 test samples were obtained from this first series of tests.

Then, each user was requested to copy the diagram shown in Fig. 3, which consists of differently hatched and filled regions, some arrows and further lines, and some text. Here, 31 test samples were obtained which were later used to test whether the algorithms are not only capable of recognizing regions, but also of avoiding false positives. One of the subjects' copies is depicted in Fig. 4.

For all tests the subjects were explicitly told to draw the way they thought it would be right. No restrictions were made regarding order, style, or any other aspect. Each subject took about 15 minutes to complete the test.

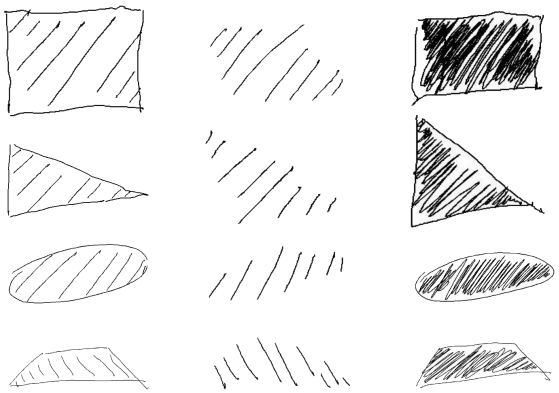

Figure 2. The 12 samples taken from one of the subjects during the first series of tests.

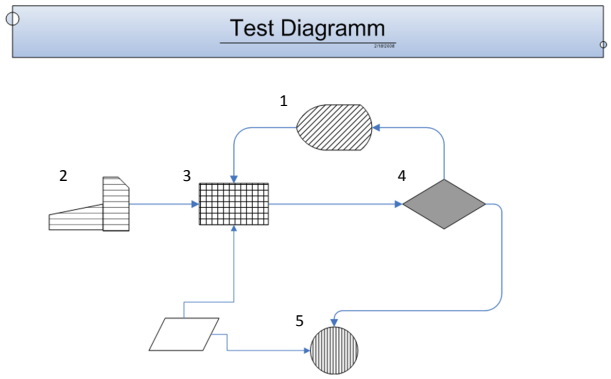

Figure 3. The master of the large test diagram for the second series of tests. The numbers are not part of the master, but used to reference the regions.

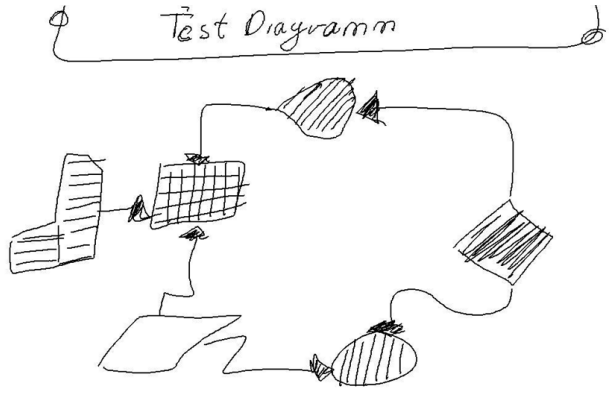

Figure 4. One subject's copy of the large test diagram for the second series of tests.

Most subjects drew as expected (and as it can be seen on the figures). However, some participants did not. For example, one subject drew a rectangle, and instead of filling it completely with strokes to indicate that its colored, he drew a second rectangle within the first one, which was slightly smaller. He claimed that the second rectangle would indicate to him that the first rectangle indeed is colored. We dismissed such samples when we postulated the assumptions for the recognition of hatched and filled regions.

Based on the findings from the tests, for hatched regions the following assumptions are made:

- a hatched region consists of at least three nearly parallel strokes.
- each of these strokes is a roughly straight line without bends.
- all strokes forming a hatched region are nearly parallel, and have nearly an equal distance to their neighbors.
- neighboring strokes forming a hatched region overlap to some extent.

For filled regions there are the following assumptions:

- colored regions consist of very long strokes with many bends.
- the bounding box of such a stroke is quite small, compared to its length.

The proposed algorithms are based on these assumptions. In the section on **Evaluation** it is discussed how the approach performs when applied to the samples taken from the test series.

HATCHED REGIONS

Hatchings are usually found in many types of drawings and diagrams. Examples are shown is Fig. 1. This section describes how strokes are identified to be part of a hatched region, and how such regions are recognized. A stroke is regarded as a list of tuples (called *samples* in the following), where each tuple (x, y, t) consists of a coordinate pair x, y and timing information t. Note that timing information is actually not considered.

For recognition of hatched regions, a three step approach is taken:

1. filter strokes.
2. apply the Hough transform [8, 9].
3. detect characteristic shapes of points in Hough space.

For the first step those strokes should be filtered out which are no straight lines. Therefore, samples are discarded from a stroke so that the remaining samples each have a distance to their neighbors larger than some threshold (the values of this and the following thresholds have been found empirically). This way, the stroke is smoothed. Then, for each three remaining consecutive samples p_i, p_{i+1}, p_{i+2} of the stroke the angle $\angle p_i, p_{i+1}, p_{i+2}$ is computed. If one of these calculated angles differs from $180°$ by more than $15°$ (another threshold), this stroke is not regarded straight and is filtered out, so that it cannot contribute to a hatched region. The initial smoothing of the stroke helps to overcome noise in the samples of the stroke. For this purpose, there is also cut off a small part at both ends of the stroke. The user study revealed that lines for a hatched regions sometimes had small hooks at the ends, which would lead to a removal of an otherwise straight stroke (cf. the hatchings in Fig. 2). Alternatively to this procedure, other tests for linearity of a stroke can be applied equivalently, e.g., [12].

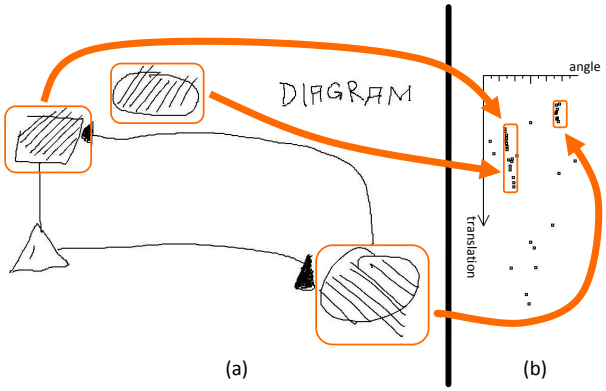

Figure 5. Example of a simple diagram with three hatched regions (a), and the corresponding points in Hough space (b). Lines in the drawing which are not straight are filtered out and are not represented in Hough space.

For the second step, only the first and last sample of each remaining stroke are kept. This is reasonable, as it is already known that the stroke forms a nearly straight line. Using the first and the last sample only, a straight line is obtained representing the stroke, which can then be transformed into Hough space. Fig. 5 exemplifies this. In Hough space, each line l is represented by a single point, where the abscissa denotes the angle which is enclosed between l and a horizontal line, and the ordinate denotes the translation, which is defined as the distance between l and the origin (note that this kind of transform actually differs from the original Hough transform, but follows a similar idea). As a consequence, all points in Hough space whose corresponding strokes form a hatched region (roughly) lie on a vertical line. The more different the angles of the strokes are, the more the points are translated to the left and to the right, and the more different the distances of neighboring strokes are, the more the distances of the points differ.

As a consequence, the third step must detect vertical accumulations of points in Hough space. Two challenges are involved in this issue. First, lines in the drawing which are straight but which do not belong to a hatched region should not be included in a vertical accumulation. Second, two separated hatched regions may appear as one vertical accumulation in Hough space, because points in Hough space do not represent segments, but straight lines. Fig. 5 shows an exam-

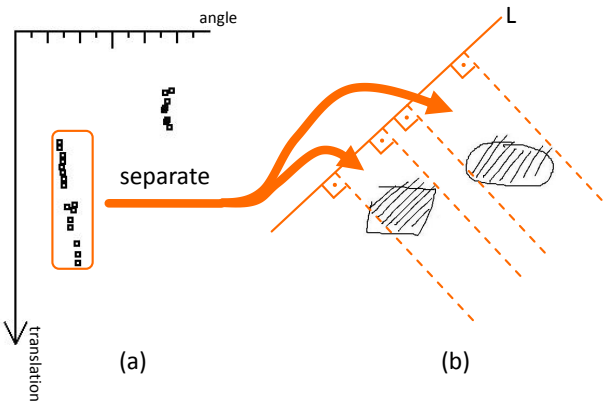

Figure 6. (a) vertical accumulations of points in Hough space (accumulations with less than three points are not shown). (b) separating the two hatched regions by their ranges on a line L.

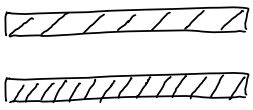

Figure 7. Two examples of hatching of a thin object. The upper hatching is not recognized, the lower is.

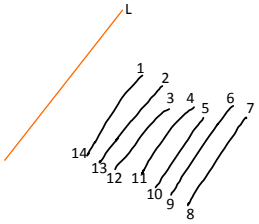

Figure 8. The polygon described by a hatching.

ple where the two upper hatched regions appear as a single accumulation in Hough space.

To find the vertical accumulations, a horizontal sweep line is used going downwards in Hough space. Based on the coordinates of the points in Hough space, they are collected in lists of potential hatched regions. During this process, each point is inspected. It can either be appended to an already existing list (i.e., to an already existing hatched region), or a new list is created with this point as first entry. A point p is appended to an existing list if its distance to the last point q in that list is considerably small (based on the average distance of the points in that list), and if it roughly lies below q, i.e., $|p.x - q.x|$ is smaller than some threshold. Otherwise, a new list is created. After all points are inspected, lists with less than three entries are discarded, as it is assumed that a hatched region consists of three lines at least. For the drawing in Fig. 5 the result of this process is depicted in Fig. 6 (a). As mentioned before, two of the hatched regions are transformed into one accumulation. To solve this issue, a straight line L is constructed which is roughly parallel to the lines represented by the points in the accumulation. Then, for each of these lines, its projection on L is taken, which is a range on L. Overlapping ranges are merged if one of the two ranges is covered by the other by 50% at least. This prevents merging of accumulations if there is an outlier which barely overlaps with other ranges. After all lines are processed, those lines whose projections were merged belong to the same region. Again, such regions are discarded with less than three strokes. This way, the two hatched regions in the example are separated from each other, as it is shown in Fig. 6 (b). The dashed lines mark the borders of the merged ranges from the projections of the lines on L. The outlines of both hatched regions (the rectangle and the ellipse-like shape) are not regarded, as they are not straight.

Because strokes contributing to the same hatching are required to overlap at least by 50%, the algorithm can fail to recognize hatching of a long thin object. Figure 7 gives two examples. The upper hatching is not recognized, becazse the strokes do not overlap at least by 50%, while the lower one is. If the domain for sketch recognition exhibits many such objects, the threshold of 50% could be set to a lower value in order to be not so rigid.

To compute the polygon that actually is hatched, such end points of the segments are considered which contribute to the hatching. The segments can be naturally ordered by their distance from L. The end points are then appended to a list according to the order of the segments, first on the one side of the region, then in reversed order on the other side. An example is depicted in Fig. 8.

The limitations of this procedure become obvious at the example depicted in Fig. 9. Both regions will clearly appear as one accumulation in Hough space. Because the larger region somewhat encloses the smaller one, they cannot be separated by our approach, but are considered as one large hatched region instead. Accordingly, the polygon describing the region cannot be computed as easily as in the example before. What happens is that the polygon becomes very jagged. Smoothing may help here to get rid of the jags.

To detect crosshatched regions like the one in the center of Fig. 4, both single-hatched regions are recognized before, independently of each other. Then the two regions can be checked for overlapping. If so, the two average angles of both hatches are compared. If they are nearly orthogonal (at least $60°$), the two hatchings are regarded as one crosshatching.

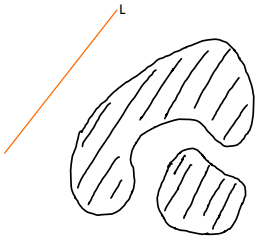

Figure 9. Two hatchings which cannot be separated.

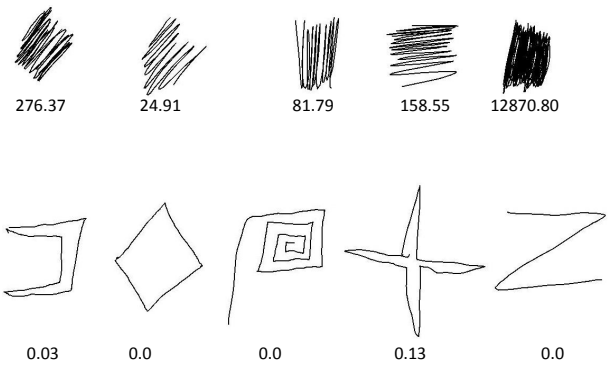

Figure 10. Ten examples of strokes and the measure M for each stroke.

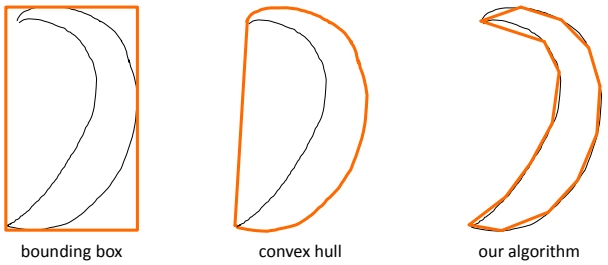

Figure 11. Bounding box, convex hull, and polygon P for a stroke representing a crescent-like shape. The former two both introduce a large error in the area of the shape.

FILLED REGIONS

Based on the assumptions established for filled regions we decided for a simple feature-based approach to recognize these regions. As candidates for a filled region we consider only those strokes which are *not* considered for hatched regions, i.e., which are not straight (see previous section). For each of these strokes three features are computed:

- the total length l of the stroke, measured as cumulated distance between each pair of consecutive samples of the stroke.
- the number of sharp bends c of the stroke, i.e., bends greater than $340°$ or smaller than $20°$. To compute this value, the stroke is first smoothed by dropping samples like in the first step for hatched regions. Then, a bend is computed as the angle enclosed by three consecutive samples of the stroke.
- the total area A covered by the stroke. The bounding box can be taken, but we decided for a more precise polygon enclosing the stroke, which is described below.

From these three features a measure M is computed which indicates whether the stroke represents a filled region. The measure is calculated by

$$M = \frac{c^2 \cdot l}{A}$$

Practical experiments show that strokes meant to represent a filled region gain values much greater than 1, while those strokes not representing a filled region gain values less than 1, so this threshold can be used to decide for filled regions. Fig. 10 shows ten examples of strokes, and the measure for each stroke. The difference between the upper row (colored regions) and the lower row (other strokes) can clearly be seen. Three of the strokes in the lower row gain a measure of 0.0 because they have not even a single bend, i.e., $c = 0$.

For calculating the area covered by a stroke the smallest polygon P which includes all samples of the stroke is computed. There is a trade-off made between precision and number of segments of P. For many shapes the convex hull is not very precise and may add a large amount of area which is actually not covered by the shape. The bounding box performs even worse, so we refrained from these two (cf. Fig. 11). To compute P a sweep-line algorithm is applied four times, one time each from up, down, left, and right. Each time a subset of all samples is computed from the stroke. Finally, these four sets are merged into one to gain P.

Fig. 12 (a) shows the horizontal sweep-line moving down, as it encounters the first sample of a very small example. An *elimination zone* is created as depicted in the figure. It extends from sample 1 downwards to infinity, its width is an ϵ-neighborhood around the sample. All other samples within the elimination zone will be discarded, and cannot be encountered by the sweep-line later. Fig. 12 (b) shows the sweep-line after it has encountered the first three samples. All samples within the elimination zones are already discarded. Neighboring samples which were encountered are connected by a preliminary line (*current outline*) in the order of their x-coordinates. In the figure, the next sample which will be encountered is 4. The current outline will then be 3-4-1-2. This process continues until all samples are encountered or discarded. The result is a list of samples, ordered from left to right, which contains all samples never covered by an elimination zone. A larger ϵ means less samples in the result, which will be more coarse-grained. A smaller ϵ means more samples, and a finer result. We decided for quite a small value, as the density of samples is, for a real stroke, very high.

To combine the four outlines identified by the four runs of the sweep-lines, all identified samples are first added to a set. Second, an arbitrary sample a is removed from the set. Then, that sample b in the set is removed which has the closest distance to a. Then the next sample c is removed which has the closest distance to b, and so on, until the set is empty. The order in which the samples were removed is taken as a polygon which represents the region that is colored.

This algorithm generates good fits in general. However, for certain shapes some error is introduced, just like the bounding box and the convex hull do, but not as severe. Fig. 13 depicts an example. The solid black line represents a perfect fit for a set of samples (not shown). Because the sweep-lines approach the shapes from the four sides, they cannot appropriately detect inner recesses; holes cannot be detected at all. Nevertheless, the results are still more accurate than the mentioned alternatives.

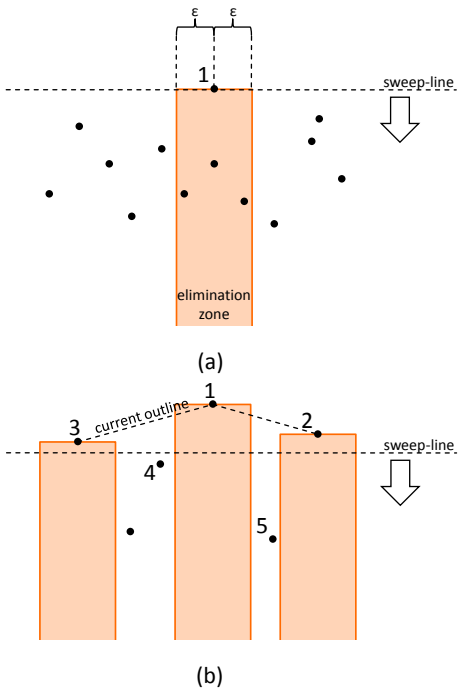

Figure 12. (a) the sweep-line encounters the first sample of a small example. (b) the sweep-line has finished processing the third sample. Two samples (4, 5) are still missing.

Motivated by the initial user study we assumed that a region is colored by one stroke only. It can happen that there are several, overlapping strokes coloring the same region. High-level recognition has to deal with this circumstance by merging the strokes to gain one, colored region. However, as mentioned in the first section, high-level aspects are not considered in this paper.

EVALUATION

To assess the performance and recognition rate of our algorithms we implemented a prototype in Java. As test samples we took the diagrams from our user study. All performance tests were running on a PC operated by Windows XP on an Intel dual core CPU with 2.4 GHz, with 2 GB of main memory.

For the first series of tests 372 regions were obtained, 31 for each of the 12 different samples (rectangle, triangle, ellipse, arbitrary shape / hatching with outline, hatching without outline, colored). For each of the twelve sets of 31 samples the average time to analyze the region was measured. Then the input was linearly increased by copying the regions and placing them next to each other. A considerable amount of free space in between the regions was left to avoid side effects. It total, the load was increased up to ten copies. Each test was performed ten times to avoid outliers. Because the differences between hatching with and without an outline are marginal, the average of both is taken for each region. The resulting eight plots are shown in Fig. 14. As it can be seen, the time required to process the regions grows linearly. Furthermore, for the colored regions there seems to be some

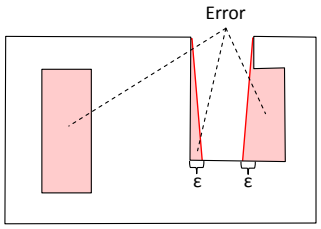

Figure 13. Inner recesses and a hole which cannot be detected by our algorithm.

region no.	recognized regions	rate
1	27	87%
2	29	94%
3	23	74%
4	31	100%
5	31	100%

Table 1. Average recognition rates for the five different regions in the sketches from the second series.

constant initial time required (less than 10ms), which we think is due to our implementation. Nevertheless, the average time to identify a region, no matter what type, clearly ranges below 5ms.

For evaluating the recognition rate the 31 sketches obtained in the second series of tests were used (cf. Fig. 3 and 4). The results are depicted in Tab. 1. For regions 4 and 5 a recognition rate of 100% was obtained, which shows that our assumptions are valid for these regions. However, for a larger test series it is unlikely that such a high recognition rate can still be obtained. For region 3, the recognition rate is not as good, as there is a crosshatched region here. This means that both single-hatched regions have to be recognized. To summarize, the overall recognition rate for all types of regions is 90.1%, the overall rate for hatchings is 88.7%, and the rate for colored regions is 100%.

In an informal test case several more *free-form* regions were verified. Some of them are depicted in Fig. 15. The goal was to evaluate whether hatched and colored regions can still be recognized properly if the outline of the regions is more complex. All of the examples shown in the figure were actually recognized. Regions that were not recognized failed to meet the assumed preconditions. The results from this informal testing suggest that the symbol outline does not affect the recognition if these preconditions are satisfied.

RELATED WORK

Most approaches to sketching do not cover hatched or filled regions. Most of the previous work in this field goes back to computer-aided design (CAD).

Dori et al. [3] directly identify hatched regions in the drawing by creating boxes around each stroke. If another stroke is found within a box, both are assumed to contribute to the same region. The approach was originally designed for precise CAD drawings. To decrease the amount of strokes, only

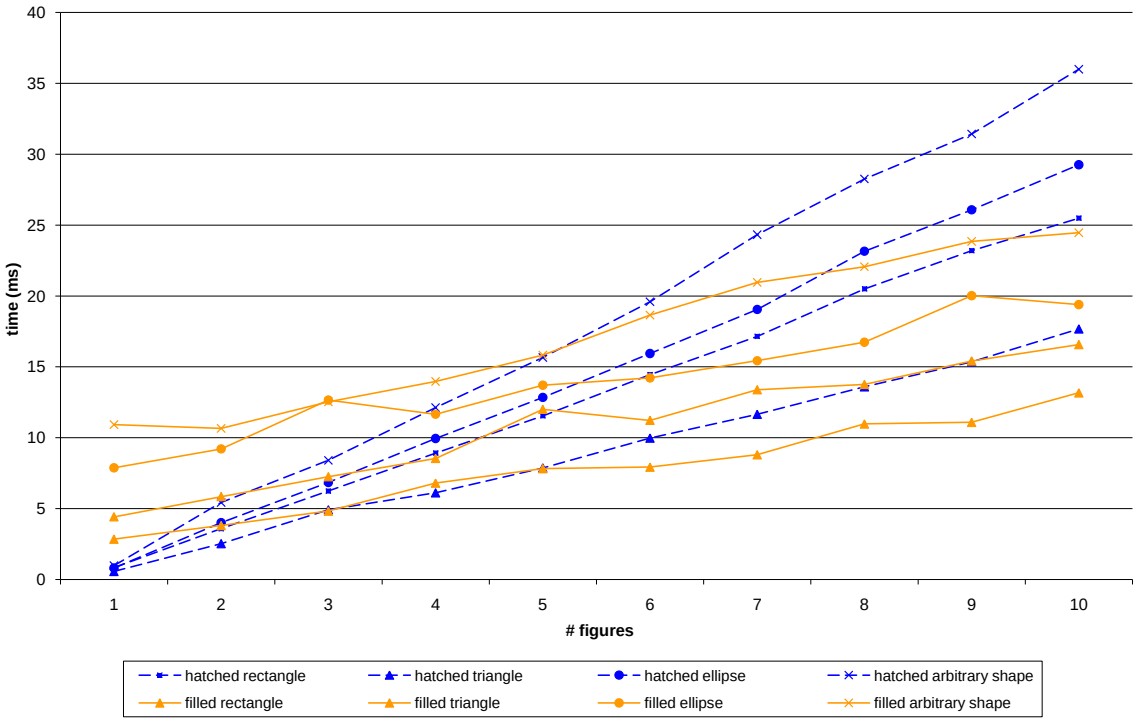

Figure 14. Average time required to recognize different kinds of regions.

Figure 15. Examples of regions with a more complex outline. Each region is recognized properly.

those strokes are regarded which enclose an angle between 30° and 60° with a horizontal line. We found this approach to be both costly and inflexible for sketched regions.

The approach given by Hachimura [6] works on Gestalt principles. According to the laws of proximity, similarity, continuity and closure, hatchings are recognized. We made our own implementation which works as follows. Four features are computed for each stroke which correspond to the four laws. Strokes where the Euclidean distance between their feature vectors is below a threshold are considered to be part of the same region. Finally, we abstained from this approach, as we found it more costly than the one proposed in this paper, comparing prototypical implementations. It may be the case that an optimized implementation could perform better. One way or another, the recognition rates are good.

For colored regions we are not aware of previous work which can be applied to sketches. Of course most drawing programs are equipped with a function to automatically cut a region which is shaded or colored according to some pattern. However, regions colored by hand look substantially different to those in CAD, so we could not use these approaches.

CONCLUSIONS AND FUTURE WORK

This paper discusses an approach to the recognition of hand-drawn hatchings and colored regions. Recognition of the former is based on Hough transform, where characteristic accumulations of points in Hough space are identified using a sweep-line algorithm. Recognition of the latter is based on three features of a stroke (total length, covered area, number of sharp bends). Using a simple classification step, colored regions are reliably distinguished from other strokes. The applicability of the proposed approach is shown by a user study. For 31 participants, an overall recognition rate of about 90% was obtained, while the average time to recognize a pattern undercuts 10ms.

As future work we like to get rid of some of the thresholds, either by changing our algorithms such that less thresholds are required at all, or by computing dynamic thresholds based on the input data, e.g., by taking average values. For the thresholds that cannot be discarded, a thorough evaluation is necessary in order to reveal their impact and to improve recognition results. Additionally, the limitations of our approach shown mentioned throughout the paper need to be investigated more deeply, and better solutions have to be found. Furthermore, we plan to integrate this work into our sketching system [1].

REFERENCES

1. F. Brieler and M. Minas. Recognition and processing of hand-drawn diagrams using syntactic and semantic analysis. In *Proceedings of the working conference on Advanced visual interfaces (AVI '08)*, pages 181–188, New York, NY, USA, 2008. ACM.

2. G. Costagliola, V. v, and M. Risi. A multi-layer parsing strategy for on-line recognition of hand-drawn diagrams. In *Proceedings of the Visual Languages and Human-Centric Computing (VLHCC '06)*, pages 103–110, Washington, DC, USA, 2006. IEEE Computer Society.

3. D. Dori and L. Wenyin. Automated CAD conversion with the machine drawing understanding system: concepts, algorithms, and performance. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 29(4):411–416, July 1999.

4. A. Forsberg, M. Dieterich, and R. Zeleznik. The music notepad. In *Proceedings of the 11th annual ACM symposium on User interface software and technology (UIST '98)*, pages 203–210, New York, NY, USA, 1998.

5. M. P. J. Fromherz and J. V. Mahoney. Interpreting sloppy stick figures with constraint-based subgraph matching. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP '01)*, pages 655–669, London, UK, 2001. Springer-Verlag.

6. K. Hachimura. Decomposition of hand-printed patterns. *Pattern Recognition, 1992. Vol.III. Conference C: Image, Speech and Signal Analysis, Proceedings., 11th IAPR International Conference on*, pages 417–420, 1992.

7. T. Hammond and R. Davis. LADDER, a sketching language for user interface developers. *Computers & Graphics*, 29(4):518–532, 2005.

8. J. Llados, J. Lopez-Krahe, and E. Marti. Hand drawn document understanding using the straight line hough transform and graph matching. In *Proceedings of the 13th International Conference on Pattern Recognition (ICPR '96)*, pages 497–501, Washington, DC, USA, 1996. IEEE Computer Society.

9. J. Lladós, E. Martí, and J. López-Krahe. A hough-based method for hatched pattern detection in maps and diagrams. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition (ICDAR '99)*, pages 479–482, Washington, DC, USA, 1999. IEEE Computer Society.

10. B. Paulson and T. Hammond. PaleoSketch: accurate primitive sketch recognition and beautification. In *Proceedings of the 13th international conference on Intelligent user interfaces (IUI '08)*, pages 1–10, New York, NY, USA, 2008. ACM.

11. B. Plimmer and I. Freeman. A toolkit approach to sketched diagram recognition. In *Proceedings of the 21st British HCI Group Annual Conference (HCI '07)*, pages 205–213, September 2007.

12. T. M. Sezgin, T. Stahovich, and R. Davis. Sketch based interfaces: early processing for sketch understanding. In *Proceedings of the 2001 workshop on Perceptive user interfaces (PUI '01)*, pages 1–8, New York, NY, USA, 2001. ACM.