Font Recognition Using Shape-Based Quad-tree and Kd-tree Decomposition

Alan Sexton, Alison Todman, and Kevin Woodward
School of Computer Science, The University of Birmingham, Birmingham B15 2TT, UK
A.P.Sexton@cs.bham.ac.uk, A.G.Todman@cs.bham.ac.uk

Abstract
The search for appropriate data representations and visual features for content-based image retrieval continues within the computer vision community, alongside the development of new matching and indexing techniques to facilitate fast search in large-scale image databases. In this study, we present a solution to the problem of typeface identification and character recognition in text-based images using this type of approach. Geometrical properties of a character are extracted from its binary image at different levels of spatial resolution, via a hierarchical abstraction of the image data. Two such abstractions are described here: a shape-based quad-tree, and a kd-tree. Unlike the traditional quad-tree representation in which an image is generally partitioned into 4 blocks of equal size at each level of decomposition, the block size in the centroid-based quad-tree is variable, being determined by the location of the centre of gravity of the regions represented in a sub-image. In a similar way, the kd-tree partitions the image data into two, again about an axis defined by the shape of the region represented in the image. Weighted and non-weighted feature vectors of the partitioning points are then used within a metric tree to index character images in a font database. We discuss factors that influence the performance of the resulting font retrieval system, both in terms of accuracy and speed.

1. Introduction
Recent research has considered font recognition mainly in its role within the field of OCR, where the ability to identify a typeface is seen as a possible pre-processing stage for character recognition in omni-font OCR systems [5,7,11,12]. However, the sophisticated graphical images used in today’s information technology frequently contain elements of text as an integral component of the picture. The ability to detect and identify this text, along with the typeface in which it was produced, in these circumstances is therefore also of current interest. Consequently, work in this area now includes studies investigating methods for the automatic location of text areas in images [4] and the possible development of text image editing facilities [6]. In addition to this, it is possible that the ability to recognise text in images could yield additional tools to aid classification and retrieval in database applications.

As the demand to create and manage very large image databases increases, a great deal of effort is currently being focused on the development of techniques for indexing and retrieving images based on their content [8]. Current methods for characterising the visual content of images generally involve the extraction of significant features from a pixel-based representation. Recognition or retrieval of the same, or similar images, then requires a pattern recognition or classification process to evaluate a measure of similarity between the query image and possible matches in the database, within this feature space. The aim of this work is to apply such an approach to the problem of font identification and character recognition in text-based images. In this case we consider the identification of a single character from its glyph, or image, as contained within its enclosing rectangle, by comparison with database images. The approach is based on a block-oriented image abstraction in which the geometrical properties of a character’s binary form determine the manner in which its image is decomposed. This yields an effective tree-based index that facilitates efficient identification of the typeface and, as a result, recognition of the character itself. The method is evaluated using a number of distance metrics, by which the most likely match is established.

2. Shape-Based Quad-tree Decomposition
The traditional quad-tree representation of image data is based on the recursive decomposition of an image into 4 equally-sized sub-images, with the level of decomposition being determined by the homogeneity of the region contained in each block. In contrast to this, the quad-tree defined here creates an abstraction of the form by dividing the data (by means of the quad-tree splitting strategy [2]) according to the shape of the character region contained in the binary image. In the quad-tree approximation, the data is partitioned about the centre of gravity at each level of decomposition, as shown in Figure 1. While this also yields 4 sub-images at each stage, these are generally variable in size. The coordinates of the centroids for the region of character pixels contained in each sub-image are stored in the corresponding level of a quad-tree, yielding a feature vector (whose size is determined by the level of decomposition) that is subsequently used to evaluate image similarity using a Euclidean metric.

The centre of gravity for a region of character pixels in a sub-image \(I(i,j)\) is given by

\[
\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}
\]

(1)

where the regular moments of order \(p+q\) of an image of MxN pixels are defined as

\[
m_{pq} = \sum_{i=0}^{M} \sum_{j=0}^{N} i^p j^q I(i,j)
\]

(2)
The centroid values are then normalised to 1 to maintain scale invariance and, thus, with 2 levels of decomposition, we obtain a 10-element feature vector,
\[ \tilde{f} = (x_0^0, y_0^0, x_0^1, x_0^2, y_0^1, x_1^1, x_1^2, y_1^1, x_1^3, y_1^2) \] (3)

Results obtained using this method are discussed in Section 6. However, it is clear that in terms of general shape discrimination, the approximation is potentially poor in circumstances where decomposition yields an empty sub-image (such as shown in Figure 2).

In this case the image is partitioned as shown in Figure 3, giving a 3-level normalised feature vector,
\[ \tilde{f} = (x_0^0, y_0^1, x_1^1, x_2^1, x_1^2, x_2^2, x_1^3, y_1^2) \] (4)

Again this is a recursive decomposition, where \( x_0^0 \) is the coordinate of the vertical midpoint about which the data is partitioned at the root level, \( y_0^1, y_1^1 \) are the two vertical midpoints about which the data is split at the 1st level, and so on to the required depth.

4. Matching Using Weighted and Non-weighted Distance Metrics

In this study we explore the use of three Euclidean distance metrics to recover the best match between an input character glyph and images in a font database. The first metric is standard, and simply applies equal weighting to all elements of the feature vectors, regardless of the depth of image decomposition employed at the feature extraction stage. In general, the Euclidean distance between two normalised feature vectors \( p_i^j \) and \( q_i^j \) in an \( n \)-level decomposition where there are \( c(n) \) components at the \( n \)th level is measured as,
\[ d_E(p, q) = \sqrt{\frac{1}{n} \sum_{i=1}^{c(n)} (p_i^j - q_i^j)^2} \] (5)

and the normalised, non-weighted, distance is,
\[ d_N(p, q) = \frac{1}{\sqrt{\sum_{i=1}^{c(k)} \sum_{j=1}^{c(j)} (p_i^j - q_i^j)^2}} \] (6)

As we increase the level of image decomposition, we extract increasingly higher resolution features. Since the root level represents the character figure as a whole, it is less likely to be affected by noise or distortion than 1st or 2nd level features. For this reason further metrics were also considered that give decreasing precedence to successive levels of the features. The second metric considered applies equal weighting to each level of decomposition, i.e. in this case, each level of features is normalised between 0 and 1/k, where \( k \) is the depth of decomposition. So, for example, the root level feature has the same weight as all 4 1st level features. The equal-weighted distance is defined as,
\[ d_E(p, q) = \frac{\sum_{i=1}^{c(n)} (p_i^j - q_i^j)}{n \sum_{i=1}^{c(n)} c(j) c(f)} \] (7)

The final metric gives ever decreasing precedence to subsequent levels. In this case, the weighting applied is
\[ w_k = \frac{1}{2^k} \text{, where } \lim_{n \to \infty} \sum_{k=0}^{n} w_k = 1 \] (8)

Normalisation gives,
\[
\left(1 + \frac{1}{2^n - 1}\right) \sum_{i=0}^{\infty} w_i = 1
\]  
and, hence, the decreasing-weighted metric is defined as,

\[
d_{\text{dec}}(p, q) = \sqrt{\left(1 + \frac{1}{2^n - 1}\right) \sum_{i=0}^{\infty} w_i \sum_{j=0}^{\infty} \left| p_i - q_j \right|^2}
\]

5. Searching
While it is possible, given a metric, to search a database of images or character glyphs sequentially for the nearest matches, this quickly becomes impractical as the database size grows. For our experiments we implemented a main memory version of a GH Tree [10], which was used to store the feature vectors only. The GH-tree is possibly the simplest of the family of metric based index tree data structures. Each node contains a pair of reference points and a pair of subtrees. The pair of reference points partitions the metric space in two on the basis of relative closeness of points in the space to the reference points. The sub-trees then recursively index the corresponding sub-spaces. The result is a binary tree which, while unbalanced, was not sufficiently so as to cause difficulties and successfully reduced a searching problem of linear to one of logarithmic complexity.

One problem that does arise is due to the nature of metric space partitioning. A node in a GH-Tree partitions the space via a generalised hyperplane (hence the name of the structure) defined by the set of points that are equidistant from both reference points. Two feature vectors may be very close together as defined by the metric but just straddle the hyperplane. In this case, a search for one should find the other as a close match as well. However, a naïve search algorithm will descend only the subtree whose reference point is closest to the query vector and will miss the close neighbour stored in the other sub-tree. To counteract this effect, we used a branch and bound algorithm that searched both sub branches of a node if and only if the two distances from the query vector to each of the two reference points was within a certain bound.

6. Results
Currently, the implementation of the method takes as input a single character glyph and returns a list of typefaces (and the character itself), ordered by the degree of match indicated by the distance metric. Font databases containing 1300 unique uppercase glyphs taken from 50 different fonts were created. The database images were rendered at 100 pts / 72dpi using the FreeType† engine. Tests were then performed on 100 randomly selected characters (all represented in the database) whose images were scaled to 300% and blurred to reduce visual quality. 2 levels of decomposition were used for the quad-tree approximation, yielding a 10-element feature vector. These were compared with results obtained using a 15-element kd-tree vector. Finally, 4 bounding values (0.0, 0.01, 0.02, and 0.05) were used to control the branch and bound search mechanism in each case.

In the experiments we noted perfect search hits only if a correct typeface match was returned as the first entry in the list. Otherwise a success was registered if a correct match was found within the first 5 entries and a failure otherwise. Tables 1 and 2, and the graphs in Figures 4 and 5 summarise the results of these experiments, performed using both the quad-tree and the kd-tree abstractions with the three metrics described above.

Here it can be seen that, while the feature extraction facilitates good results, a major improvement in performance is achieved through effective searching using the branch and bound technique. Overall, without

<table>
<thead>
<tr>
<th>Structure type</th>
<th>Bounding value</th>
</tr>
</thead>
<tbody>
<tr>
<td>quad-tree (equal)</td>
<td>85 99 100 99</td>
</tr>
<tr>
<td>quad-tree (decreasing)</td>
<td>81 95 96 97</td>
</tr>
<tr>
<td>kd-tree(non-weighted)</td>
<td>91 99 99 98</td>
</tr>
<tr>
<td>kd-tree (equal)</td>
<td>87 99 99 98</td>
</tr>
<tr>
<td>kd-tree (decreasing)</td>
<td>84 96 98 97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Structure type</th>
<th>Bounding value</th>
</tr>
</thead>
<tbody>
<tr>
<td>quad-tree (equal)</td>
<td>16 1 0 0</td>
</tr>
<tr>
<td>quad-tree (decreasing)</td>
<td>16 2 1 0</td>
</tr>
<tr>
<td>kd-tree(non-weighted)</td>
<td>9 1 0 0</td>
</tr>
<tr>
<td>kd-tree (equal)</td>
<td>13 0 0 0</td>
</tr>
<tr>
<td>kd-tree (decreasing)</td>
<td>16 3 1 1</td>
</tr>
</tbody>
</table>

Figure 4. Perfect Hits

Figure 5. Comparative failure rate
bounding, the mean success rate was 86%, with the non-weighted kd-tree metric performing best at 91%. While setting the bound at 0.01 increases the search time, the mean success is significantly increased to 97.6%, with all metrics yielding comparable results. Increasing the bound beyond this point increases search time substantially without an equivalent improvement in accuracy.

7. Future Work
The work presented here represents a preliminary study of a content-based retrieval approach to the recognition of characters in text images and identification of the typeface in which they were produced. At this stage the work has focused on relatively large scale character images, such as may be commonly found on the Internet. As such, the results are very promising. However, font recognition is also potentially useful in OCR, where scanned images created using small fonts may be of lower quality than those used here. For this reason, further work includes investigation of the process within this domain, where substantial noise and distortion need to be considered.

In relation to the method itself, the depth of decomposition of the image is currently decided arbitrarily. In these circumstances it is possible that situations could arise in which substantially different images yield identical feature vectors at a given level. Although unlikely in our situation, Figure 6 illustrates this point. Further work is therefore required to devise either a method whereby the required granularity is established automatically, or one in which other factors (such as the total mass of the region) influence the location of the partitioning point (e.g. introducing a gravity field).

![Figure 6. These images will have the same root and 1st level centroids, i.e. at the 1st level, the quadtree feature vectors will be identical. We need a further level of decomposition to discriminate between the two.](image)

Block-oriented image decomposition is not new to image processing, (although others are only now beginning to recognise the variety of structures available to aid the development of large-scale image databases [3]). Similarly, geometric features, such as centroids and moments are commonly used in shape description [9]. However, here we describe a combined approach to feature extraction using quad- and kd-trees that facilitates the implementation of an effective indexing system for an image database. Furthermore we highlight the general importance of the search mechanism in achieving good performance in content-based image retrieval. This is likely to be the case regardless of the application domain.

References

† FreeType is a free and portable TrueType font rendering engine