Chapter 8

Content-Based Image Retrieval

As large amounts of both internal and external memory become increasingly less expensive and processors become increasingly more powerful, image databases have gone from an expectation to a firm reality. Image databases exist for storing art collections, satellite images, medical images, and general collections of photographs. Uses vary according to application. Art collection users may wish to find work by a certain artist or to find out who painted a particular image they have seen. Medical database users may be medical students studying anatomy or doctors looking for sample instances of a given disease. General collections might be accessed by illustrators looking for just the right picture for an article or book. The domain of this application is enormous; one user might want to find images of horses, another might want sunsets, and a third might be looking for an abstract concept, such as love.

Image databases can be huge, containing hundreds of thousands or millions of images. In most cases they are only indexed by keywords that have to be decided upon and entered into the database system by a human categorizer. However, images can be retrieved according to their content, where content might refer to color distributions, texture, region shapes, or object classification. While the state of segmentation and recognition algorithms is still primitive, commercial and research systems have been built and are already in use; demo systems are often available on the World Wide Web. This chapter explores the methods by which humans can retrieve images without resorting to a keyword search.

8.1 Image Database Examples

Some image databases have been constructed just to show how a particular retrieval system works. The IBM QBIC Project Database is an example of this kind. QBIC is a research system that led to a commercial product developed and sold by IBM. It retrieves images based on visual content, including such properties as color percentages, color layout, and texture. Virage, Inc. developed a competing product, the Virage search engine, which can retrieve images based on color, composition, texture, and structure. These and other image search engines can be used to search databases that are provided by other institutions. For example, the Fine Arts Museums of San Francisco have provided QBIC access to their Imagebase, a collection of digitized paintings. A Renoir painting from this collection is shown in Figure 8.1a. Similar digital art collections are being created in many major cities of the

a) Renoir painting image

b) Amethyst image

Figure 8.1: Images from digital collections. (Pierre-Auguste Renoir painting image, Landscape at Beaulieu, c. 1893, courtesy of the Fine Arts Museums of San Francisco, Mildred Anna Williams Collection, 1944.9; amethyst image courtesy of the Smithsonian Institution, 1992.)

world.

In addition to art collections, there are general image collections whose individual images are available for licensing by private customers who may want them for marketing products or illustrating articles. One of the biggest is the Corbis Archive, which contains more than 17 million images, with nearly one million in digital format and growing. This archive tries to capture the full range of human expression and perception; it contains such categories as history, art, entertainment, science and industry, and animals. Corbis provides retrieval of its images by keyword and by browsing. Another company, PhotoDisc, also provides an online image database organized by categories and searchable through keywords.

In addition to artwork and photographs, there are scientific and medical image collections. The National Library of Medicine provides a database of Xrays, CT scans, MRI images and color cross-sections, taken at very small intervals along the bodies of a male and a female cadaver. There are over 14,000 images available for people who want to use them for medical research. The National Aeronautics and Space Administration (NASA) collects huge databases of images from its satellites and makes them available for public acquisition (for a fee). The United States Geological Survey (USGS) provides a web search capability for users who wish to find and order data sets including digital satellite and aerial images. Finally, the World Wide Web itself is a database that contains both text and large numbers of images. Search engines for finding images on the Web, based on keywords and, to a limited extent, on image content are being developed.

8.2 Image Database Queries

Given a database of images, we must have some way, other than searching through the whole set, to retrieve them. Companies that make databases of images available to their customers will general have a selection process for determining which images should be added to the collection and a categorization process for assigning general categories and other keywords to the selected images. Images appearing on the World Wide Web will usually have a caption from which keywords can be obtained automatically.

In a relational database system, entities can be retrieved based on the values of their textual attributes. Attributes used to retrieve images might include general category, names of objects present, names of people present, date of creation, and source. The images can be indexed according to these attributes, so that they can be rapidly retrieved when a query is issued. This type of textual query can be expressed in the SQL relational database language, which is available for all standard relational systems. For example, the query

```
SELECT * FROM IMAGEDB
WHERE CATEGORY = 'GEMS'  AND SOURCE = 'SMITHSONIAN'
```

would find and return all images from the set named IMAGEDB whose CATEGORY attribute was set to 'GEMS' and whose SOURCE attribute was set to 'SMITHSONIAN'. The object would be to retrieve images of the gem collection of the Smithsonian Institute. Figure 8.1b shows an amethyst image from this collection. The amethyst image would be retrieved along with many other gem images. In order to allow more selective retrievals, a descriptive set of keywords would have to be stored for each image. In a relational database, KEYWORD would be an attribute that could have multiple values for each image. So the amethyst image might have values of 'AMETHYST', 'CRYSTAL', and 'PURPLE' as its keywords, and could be retrieved according to all three or any of the three, depending on the desires of the user. For example, the SQL query

```
SELECT * FROM IMAGEDB
WHERE CATEGORY = 'GEMS'  AND SOURCE = 'SMITHSONIAN'
      AND (KEYWORD = 'AMETHYST' OR KEYWORD = 'CRYSTAL'
           OR KEYWORD = 'PURPLE')
```

retrieves all images from the set named IMAGEDB, whose CATEGORY is 'GEMS', whose SOURCE is 'SMITHSONIAN', and which has a KEYWORD value of 'AMETHYST', 'CRYSTAL', or 'PURPLE'. This will retrieve more than just the amethyst image; the user will be able to browse through and select images from the set returned.

a) Image with pigs

b) Image with no pigs

Figure 8.2: Pig images returned by keyword search.

There is a limit to what can be done with the keyword approach. Human coding of keywords is expensive and is bound to leave out some terms by which users will want to reference an image. In web databases, using HTML captions can help to automate, but also provides only limited indexing capability. Furthermore, some of the returned images may turn out to look very different than the user expects from the automatically derived keywords. Figure 8.2 shows two images returned from a web search with the keyword 'pigs'.

Given that keywords alone are insufficient, we will explore other methods for retrieving images that can be used instead of or in addition to keywords.

Exercise 1 Keyword Queries

Give an SQL query that will retrieve the image of Figure 8.2a, but will not retrieve the image of Figure 8.2b. Use whatever categories and keywords you think are appropriate.

8.3 Query-by-Example

Query-by-example is database terminology for a query that is formulated by filling in values and constraints in a table and can be converted by the system to SQL. The first QBE system was developed by IBM; today, Microsoft Access is a good example of this type of system. In standard relational databases, where the values of attributes are mainly text or numeric values, query-by-example merely provides a convenient interface for the user, without any additional power.

In image databases, the very idea of query-by-example is exciting. Instead of typing a query, the image database user should be able to show the system a sample image, or paint one interactively on the screen, or just sketch the outline of an object. The system should then be able to return similar images or images containing similar objects. This is the goal of all content-based image retrieval systems; each one has its own ways of specifying queries, determining similarity between a query and an image in the database, and of selecting the images to be returned.

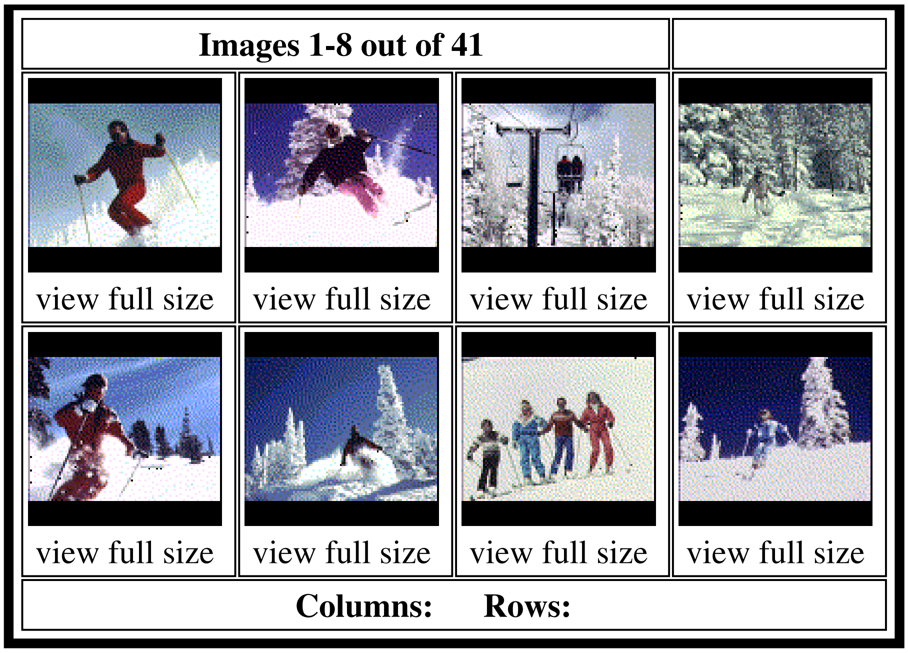


Figure 8.3: Results of a QBIC search based on color layout similarity; the query is the example image shown in the top left position (images courtesy of Egames).

To keep our discussion general, let us consider a *query* as an *example* image plus a set of *constraints*. The image may be a digital photograph, a user-painted rough example, a line-drawing sketch, or empty, in which case the retrieval set must only satisfy the constraints. The constraints may be keywords that should be present in some indexing system or may specify objects that should be in the image and even spatial relationships among them. In the most common case, the query is a digital image that is compared to images in the database according to an *image distance measure*. When the distance returned is zero, the image exactly matches the query. Values larger than zero indicate various degrees of similarity to the query. Image search engines usually return a set of images in order of their distance to the query. Figure 8.3 illustrates the results of a QBIC search based on its color layout distance measure. The images shown were the eight most similar images to the query image, which is shown in the upper left position as it is most similar to itself.

8.4 Image Distance Measures

The judgment of how similar a database image is to a query is dependent on which image distance measure or measures are used to judge similarity. There are four major classes of similarity measures:

1. color similarity
2. texture similarity
3. shape similarity
4. object and relationship similarity

8.4.1 Color Similarity Measures

Color similarity measures are often very simple. They compare the color content of one image with the color content of a second image or of a query specification. For example, QBIC allows users to specify a query in terms of color percentages. The user chooses up to five colors from a color table and indicates the desired percentage of each color. QBIC looks for images that are closest to having these color percentages. The particular placement of the colors within the image is not a factor in the search. Figure 8.4 shows a set of images returned for a query that specified 40% red, 30% yellow, and 10% black. While the colors returned are very similar in each of the returned images, the images have very different compositions.

A related technique is color histogram matching, as discussed in Chapter 6 and used for vegetable recognition in Chapter 16. Users can provide a sample image and ask the system to return images whose color histogram distance to the sample is low. Color histogram distances should include some measure of how similar two different colors are. For example, QBIC defines its color histogram distance as

$$d_{hist}(I, Q) = (h(I) - h(Q))^T A \; (h(I) - h(Q)) \tag{8.1}$$

where $h(I)$ and $h(Q)$ are the K-bin histograms of images I and Q, respectively, and A is a K $\times$ K similarity matrix. In this matrix colors that are extremely similar should have similarity values close to one, while colors that are very different should have similarity values close to or equal to zero.

Color layout is another possible distance measure. It is common for the user to begin with an empty grid representing the query and to choose colors for each of the grid squares from a table. In Figure 8.5, the user has selected two colors from the color matrix shown at the upper left and has painted them onto the 6 $\times$ 6 spatial layout grid shown at the upper right. The images shown are those that were judged most similar to the query according to a simple color layout distance measure. As was shown in Figure 8.3, it is also possible to start with an example image and have the system return other images that have spatial color distributions that are similar to it.

Color layout measures that use a grid require a grid square color distance measure $\hat{d}_{color}$ that compares each grid square of the query to the corresponding grid square of a potential

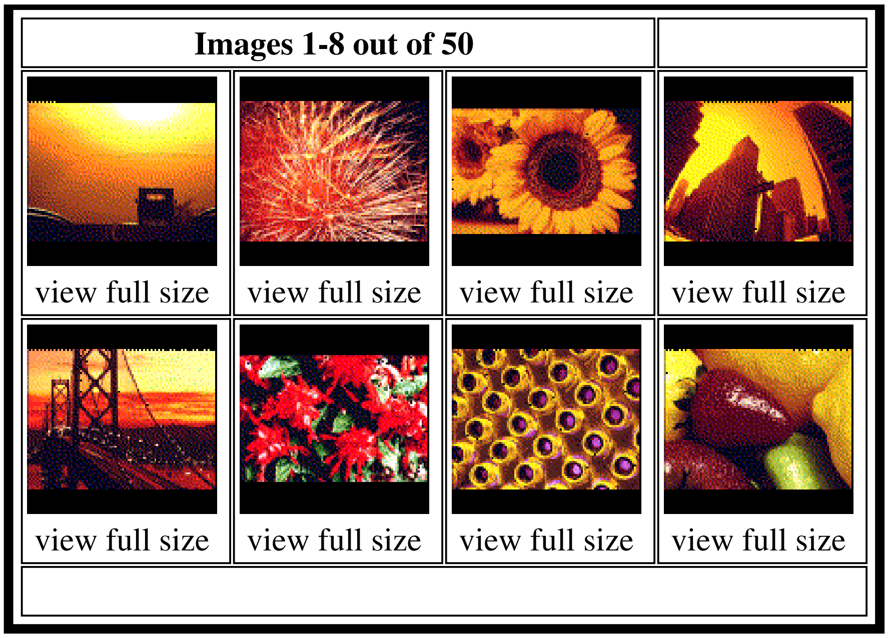


Figure 8.4: Results of a QBIC search based on color percentages; the query specified 40% red, 30% yellow, and 10% black (images courtesy of Egames).

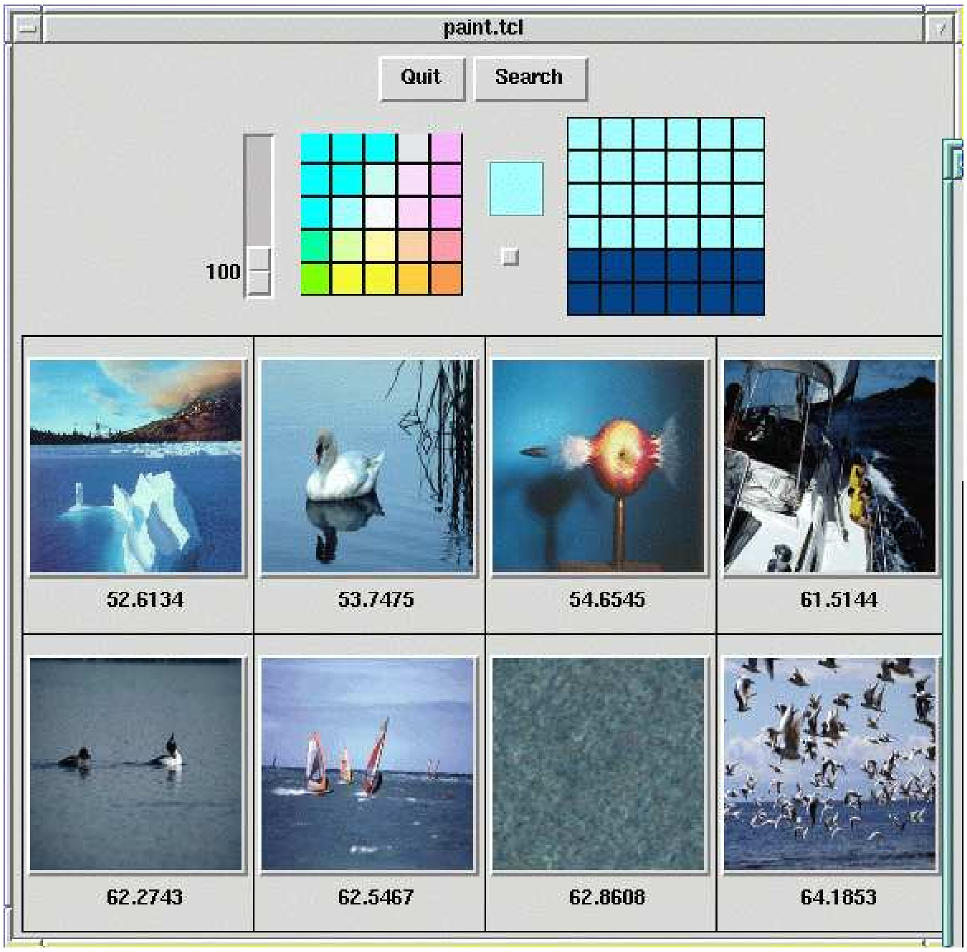


Figure 8.5: Results of an image database search in which the query is a painted grid. (Images from the MIT Media Lab VisTex database: http://vismod.www.media.mit.edu/vismod/imagery/VisionTexture/vistex.html).

matching image and combines the results into a single image distance

$$d_{gridded\_color}(I, Q) = \sum_g \hat{d}_{color}(C^I(g), C^Q(g)) \tag{8.2}$$

where $C^I(g)$ represents the color in grid square g of a database image I and $C^Q(g)$ represents the color in the corresponding grid square g of the query image Q. The representation of the color in a grid square can itself be very simple or more complicated. Some suitable representations are

1. the mean color in the grid square
2. the mean and standard deviation of the color
3. a multi-bin histogram of the color

The grid square distance $\hat{d}$ must be able to operate on the color representation to produce meaningful distances. For example, if mean color is represented as a triple (R, G, B), then the measure $\hat{d} = ||(R^Q, G^Q, B^Q) - (R^I, G^I, B^I)||^2$ would be an obvious choice, but not necessarily the best one. Instead of comparing (R, G, B) values, some systems partition color space into a set of 3D bins and keep a table of the numerical similarities between pairs of bins. This is the same technique that was used in QBIC's histogram distance as discussed above.

Exercise 2 Color Histogram Distances

Implement a $4 \times 4 \times 4$ color histogram distance measure that can imput two images and compare either the whole images or selected subimages of each. Use this basic measure to implement a gridded-color distance measure that allows the user to specify the dimensions of the grid and combines the distances between each pair of corresponding grid squares into a single distance as shown in equation 8.2. Try your gridded histogram distance measure on several pairs of color images and with grid dimensions 1×1, 4×4, and 8×8.

8.4.2 Texture Similarity Measures

Texture similarity is more complex than color similarity. An image that has similar texture to a query should have the same spatial arrangements of colors (or gray tones), but not necessarily the same colors (or gray tones). The texture measures described in Chapter 7 can be used to judge the similarity between two textures. Figure 8.6 illustrates texture-based image retrieval, using a distance based on the Laws texture energy measures. As can be seen from the query results, this distance is independent of the colors in the image. However, it is also possible to develop distance measures that look for both color and texture similarity.

Texture distance measures have two aspects:

1. the representation of texture and
2. the definition of similarity with respect to that representation.

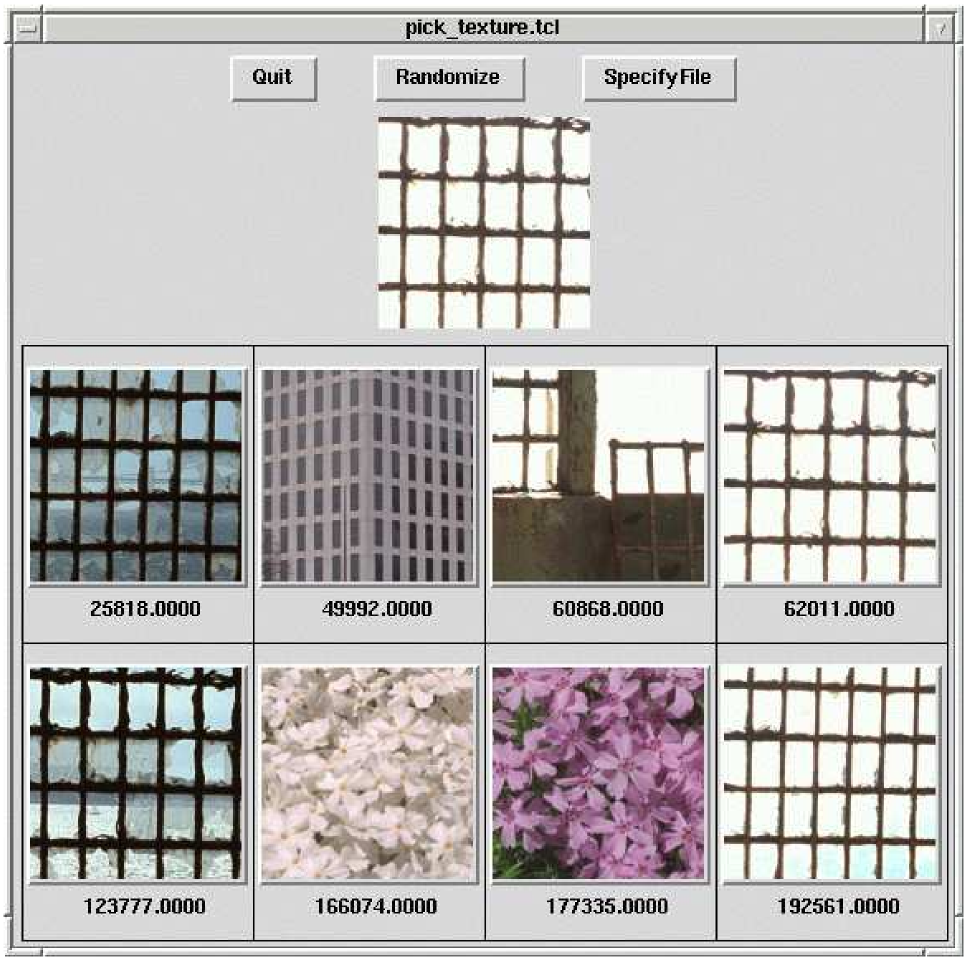


Figure 8.6: Results of an image database search based on texture similarity (Images from the MIT Media Lab VisTex database: http://vismod.www.media.mit.edu/vismod/imagery/VisionTexture/vistex.html).

The most commonly used texture representation is a *texture description vector*, which is a vector of numbers that summarizes the texture in a given image or image region. The vector of Haralick's five co-occurence-based texture features and that of Laws' nine texture energy features are examples of texture description vectors. While a texture description vector can be used to summarize the texture in an entire image, this is only a good method for describing single texture images. For more general images, texture description vectors are calculated at each pixel for a small (e.g. 15×15) neighborhood about that pixel. Then the pixels are grouped by a clustering algorithm that assigns a unique label to each different texture category it finds.

Given that pixels can be assigned a texture description vector and labeled as belonging to a texture class, various texture distances can be defined. The simplest texture distance is the 'pick-and-click' distance. The user selects a texture by clicking on a pixel in a textured region of the query image or by selecting the texture from a predetermined set of choices. The selected texture is represented by its texture description vector, which is compared to the texture description vectors associated with the database. The distance measure is defined by

$$d_{pick\_and\_click}(I, Q) = min_{i \in I} ||T(i) - T(Q)||^2 \tag{8.3}$$

where $T(i)$ is the texture description vector at pixel i of image I and $T(Q)$ is the texture description vector at the selected pixel or for the selected texture class of the query. Although this looks like it might be computationally intensive, most of the computation can be avoided by representing a database image by a list of its texture categories as determined by the clustering procedure. For each database image, the query texture description vector needs only to be compared to the texture description vectors in its list. Indexing can provide even faster retrieval.

The pick-and-click distance requires the user to select a given texture; it cannot operate automatically on a query image. A more general texture measure is a generalization of the gridded measures discussed above from color to texture. A grid is placed over the query image and a texture description vector is calculated for each grid square. The same process is applied to the database image. The gridded texture distance is given by

$$d_{gridded\_texture}(I, Q) = \sum_g \hat{d}_{texture}(T^I(g), T^Q(g)) \tag{8.4}$$

where $\hat{d}_{texture}$ can be Euclidean distance or some other distance metric. Texture histogram distances can be defined in a similar manner to color histograms. For each texture category, the histogram specifies the number of pixels whose texture description vector falls into that category. One interesting and easy to compute texture histogram measure has been developed in terms of pairs of touching line segments. A line finder (see Chapter 10) is applied to the image to detect line segments. Pairs of line segments that touch or almost touch are found, and the angle between each such pair of segments is computed. These angles are the variables used to produce the texture histogram that describes the image.

8.4.3 Shape Similarity Measures

Color and texture are both global attributes of an image. Distance measures based on these quantities try to determine if a given image has a specified color or texture and whether

---

Exercise 3 Texture Distance Measures

Pick several different texture measures from Chapter 7 and implement them as image distance measures that compare the texture in a subimage of a query image to that in a subimage of a database image. Then write a program that implements a gridded-texture distance measure and can call on any of these as the individual measures used in each of the grid squares. Compare results on a set of images using each of the individual measures and trying several different grid sizes. Test on a database of images that each have several regions of different textures.

---

or not it occurs in the same approximate position as in the query image. Shape is not an image attribute; it does not make sense to ask what the shape of an image is. Instead, shape tends to refer to a specific region of an image. Shape goes one step further than color and texture in that it requires some kind of region identification process to precede the shape similarity measure. In many cases, this has to be done manually, but automated segmentation is possible in some domains. Segmentation is still a crucial problem to be solved, before shape-based retrieval can be made widely available. Segmentation will be discussed in Chapter 10; shape matching is covered here.

Two-dimensional shape recognition is an important aspect of image analysis. In Chapter 3, a number of properties of image regions were defined; these are what we call *global shape properties*, since they refer to the shape as a whole. Two shapes can be compared according to global properties by any of the statistical pattern recognition methods described in Chapter 4. Shape matching can also use structural techniques whereby a shape is described by its primitive components and their spatial relationships. Since the representation is a relational graph, graph-matching methods can be used for matching. Graph matching is powerful, because it is based on spatial relationships that are invariant to most 2D transformations. However, graph matching can be a very slow process; the computation time is exponential in the number of components. In the context of content-based image retrieval, we need methods that can quickly decide how similar an image shape is to a query shape. Often, we require shape matching methods to be invariant to translation and to size. Sometimes we also want rotational invariance, so that an object can be identified whether it is right-side-up or in some other orientation. However, this property is not always required in image retrieval. There are many scenes in which objects usually appear in the correct orientation. Buildings, trees, and trucks in outdoor scenes are common examples.

Shape measures abound in the computer vision literature. They range from crude global measures that help with, but do not perform, object recognition to very detailed measures that look for objects with very specific shapes. Shape histograms are examples of simple measures that can rule out shapes that could not possibly match, but that will return many false positives, just as color histograms do. Boundary techniques are more specific; they work with a representation of the boundary of a shape and look for shapes with similar boundaries. Sketch matching can be even more specific, looking not just for a single object boundary, but for a set of image segments involving one or more objects that match a query drawn or supplied by a user. We now discuss each of these categories.

Shape Histograms Given that histogram distances are fast and easy to compute and that they have been used for both color and texture matching, it is natural to extend them to shape matching. The main problem is to define the variable on which the histogram is defined. Consider the shape as a region of 1's in a binary image whose other pixels are all 0's. One kind of histogram matching is *projection matching* using horizontal and vertical projections of the shape. Suppose the shape has n rows and m columns. Each row and each column become a bin in the histogram. The count that is stored in a bin is the number of 1-pixels that appear in that row or column. This leads to a histogram of $n + m$ bins, which is useful only if the shape is always the same size. To make projection matching size invariant, the number of row bins and the number of column bins can be fixed. By defining the bins from the top left to the bottom right of the shape, translational invariance is achieved. Projection matching is not rotationally invariant, but may work with small rotations or other small geometric distortions. One way to make it rotationlly invariant is to compute the axes of the best-fitting ellipse (as discussed in Chapter 3) and rotate the shape until the major axis is vertical. Because we don't know which is the top of the shape, two possible rotations must be tried. Furthermore, if the major and minor axes are about the same length, four possible rotations must be considered. Projection matching has been successfully used in logo retrieval.

Another possibility is to construct the histogram over the tangent angle at each pixel on the boundary of the shape. This measure is automatically size and translationally invariant, but it is not rotationally invariant, because the tangent angles are computed from a fixed orientation of the shape. There are several different ways to solve this problem. One way is to rotate the shape according to its major axis as described above. Another simpler way is to rotate the histogram instead. If the histogram has K bins, there are K possible rotations. Incorrect rotations can be ruled out rapidly as soon as the histogram distance being computed becomes too large. Or, the histograms can be normalized by always choosing the bin with the largest count to be the first bin. Because of possible noise and distortion, several 'largest' bins should be tried.

Exercise 4 Shape Histograms

Write a program that implements a shape-histogram distance measure using the tangent angle at each pixel on the boundary of the shape. Make it rotationally invariant by rotating the histogram of the query image so that each bin gets a turn as the first bin and the result is the minimum distance returned by each of these rotations. Use your distance measure to compare shapes that you extract from real images either by thresholding or interactively.

Boundary Matching Boundary matching algorithms require the extraction and representation of the boundaries of the query shape and the image shape. The boundary can be represented as a sequence of pixels or may be approximated by a polygon. For a sequence of pixels, one classical kind of matching uses Fourier descriptors to compare two shapes. In continuous mathematics, the Fourier descriptors are the coefficients of the Fourier series expansion of the function that defines the boundary of the shape. In the discrete case, the shape is represented by a sequence of m points $< V_0, V_1, \ldots, V_{m-1} >$. From this sequence of points, a sequence of unit vectors

$$v_k = \frac{V_{k+1} - V_k}{|V_{k+1} - V_k|} \tag{8.5}$$

and a sequence of cumulative differences

$$
\begin{aligned}
l_k &= \sum_{i=1}^{k} |V_i - V_{i-1}|, \quad k > 0 \\
l_0 &= 0
\end{aligned}
\tag{8.6}
$$

can be computed. The Fourier descriptors $\{a_{-M}, \ldots, a_0, \ldots, a_M\}$ are then approximated by

$$
a_n = \frac{1}{L\left(\frac{n2\pi}{L}\right)^2} \sum_{k=1}^{m} (v_{k-1} - v_k) e^{-jn(2\pi/L)l_k}
\tag{8.7}
$$

These descriptors can be used to define a shape distance measure. Suppose Q is the query shape and I is the image shape to be compared to Q. Let $\{a_n^Q\}$ be the sequence of FDs for the query and $\{a_n^I\}$ be the sequence of FDs for the image. Then the Fourier distance measure is given by

$$
d_{Fourier}(I, Q) = \left[\sum_{n=-M}^{M} |a_n^I - a_n^Q|^2 \right]^{\frac{1}{2}}
\tag{8.8}
$$

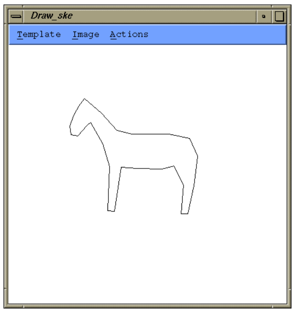
As described, this distance is only invariant to translation. If the other invariants are required, it can be used in conjunction with a numeric procedure that solves for the scale, rotation, and starting point that minimize $d_{Fourier}(I, Q)$.

If the boundary is represented by a polygon, the lengths of the sides and the angles between them can be computed and used to represent the shape. A shape can be represented by a sequence of junction points $< X_i, Y_i, \alpha_i >$ where a pair of lines meet at coordinate location (X_i, Y_i) with angle magnitude α_i. Given a sequence $Q = Q_1, Q_2, \ldots, Q_n$ of junction points representing the boundary of a query object Q and a similar sequence $I = I_1, I_2, \ldots I_m$ representing the boundary of an image object I, the goal is to find a mapping from Q to I that maps line segments of the query to similar-length line segments of the image and requires that a pair of adjacent query line segments that meet at a particular angle α should map to a pair of adjacent image line segments that meet at a similar angle α'.

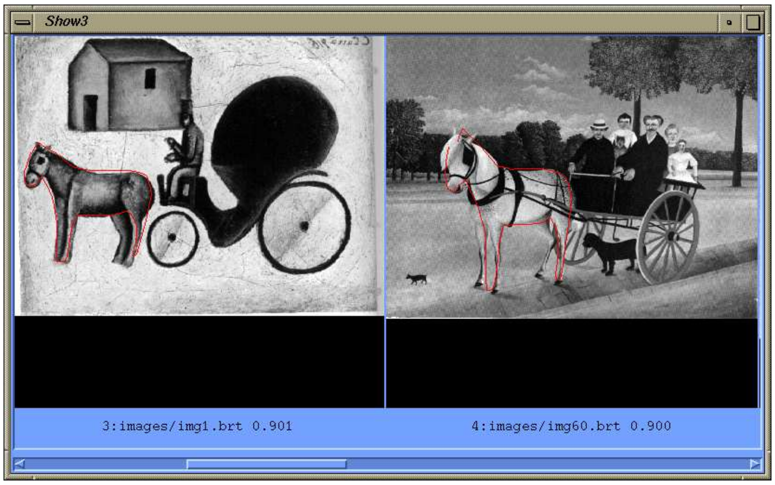
Another boundary matching technique is *elastic matching* in which the query shape is deformed to become as similar as possible to the image shape. The distance between the query shape and the image shape depends on two components: 1) the energy required to deform the query shape into a shape that best matches the image shape and 2) a measure of how well the deformed query actually matches the image. Figure 8.7 shows the retrieval of images of horses through elastic matching to a query in which a user drew a rough outline of the shape he wanted to retrieve.

Exercise 5 Boundary Matching

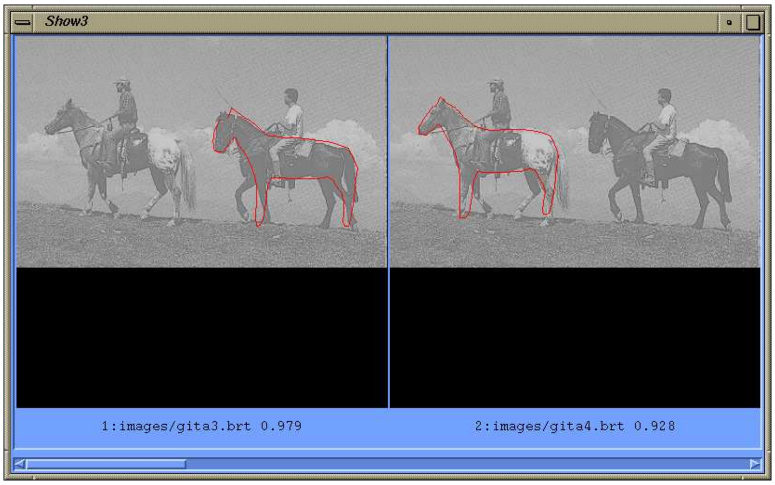
While there are many known algorithms for boundary shape matching, they have not been heavily used so far in content-based retrieval. Can you explain why?



a) The user's query shape



b) Two of the retrieved images.



c) Another retrieved image in which two horses were found.

Figure 8.7: Image retrieval by elastic matching (courtesy of Alberto Del Bimbo).

Sketch Matching Sketch matching systems allow the user to input a rough sketch of the major edges in an image and look for full color or gray-scale images that have matching edges. In the ART MUSEUM system, the database consists of full-color images of famous paintings. The color images are preprocessed as follows to obtain an intermediate form called an *abstract image*.

1. Apply an affine transform to reduce the image to a prespecified size, such as 64×64 pixels and a median filter to remove noise. The result is a normalized image.
2. Detect edges using a gradient-based edge-finding algorithm. The edge finding is performed in two steps: first global edges are found with a global threshold that is based on the mean and variance of the gradient; then local edges are selected from the global according to locally-computed thresholds. The result is called the refined edge image.
3. Perform thinning and shrinking on the refined edge image. The final result is called the abstract image. It is a relatively clean sketch of the edges of the original image.

When the user enters a rough sketch as a query, it is also converted to the normalized size, binarized, thinned, and shrunk. The result of this processing is called the linear sketch. Now the linear sketch must be matched to the abstract images. The matching algorithm is correlation-based. The two images are divided into grid squares. For each grid square of the query image, the local correlation with the corresponding grid square of the database image is computed. In order to be more robust, the local correlation is computed for several different shifts in the position of the grid square on the database image and the maximum correlation over all the shifts is the result for that query grid square. The final similarity measure is the sum of each of the local correlations. The distance measure is the inverse of this similarity measure. In terms of our previous notation, it can be expressed as

$$d_{sketch}(I,Q) = \frac{1}{\sum_g max_n[\hat{d}_{correlation}(shift_n(A^I(g)), L^Q(g))]} \tag{8.9}$$

where $A^I(g)$ refers to grid square g of the abstract image computed from database image I, $shift(A^I(g))$ refers to a shifted version of grid square g of the same abstract image, and $L^Q(g)$ refers to grid square g of the linear sketch resulting from query image Q.

Exercise 6 Sketch Matching

Design and implement a sketch-matching distance measure along the lines of the ART MUSEUM system. Use it to retrieve a set of known images according to the user's sketch.

8.4.4 Object Presence and Relational Similarity Measures

Although most of the distance measures offered by the first image search engines involve color, texture, and shape, these are not the quantities that most end users want to see. End users tend to ask for images containing certain entities, which can be particular objects, such as people or dogs or can be abstract concepts, such as happiness or poverty. The first systems to offer object recognition have looked for such objects as human faces, human bodies, and horses. This is an area that will require further research in object recognition in order to be useful in image retrieval.

Figure 8.8: Faces detected by a neural-net-based face finder (courtesy of Henry Rowley and Takeo Kanade).

Face Finding Face finding is important because it allows us to search for images containing people. It is difficult because faces are found in all sizes and locations in an image, can be in a frontal or other view, and come in a variety of colors. A system developed at Carnegie-Mellon University employs a multiresolution approach to solve the size problem. It converts color images to gray scale to avoid color differences, normalizes for lighting, and expands the gray-tone range through histogram equalization. It then uses a neural net classifier that was trained on 16,000 images of faces and non-faces to perform the recognition. The neural net receives an image of $20 \times 20 = 400$ intensity values as its input and classifies it as a face or non-face. While it is difficult to extract an exact algorithm from a neural net, a sensitivity analysis showed that the network relies most heavily on the eyes, then on the nose, and then on the mouth area of the 20×20 image. The method works well, finding most, but not all frontal views of faces, as shown in Figure 8.8. It is not generally extensible to other objects unless they have a very particular pattern that shows up in the same way that the eyes, nose, and mouth do in gray tone images.

Flesh Finding Another way of finding objects is to find regions in images that have the color and texture usually associated with that object. One of the first efforts in this area was focused on finding images of naked people, which has the useful application of filtering out pornography from query result sets. The method developed by Fleck and Forsyth has two main steps: 1) finding large regions of potential flesh and 2) grouping these regions to find potential human bodies.

The flesh filter operates at the pixel level. The initial RGB image is transformed into log-opponent space as follows:

$$I = L(G) \tag{8.10}$$
$$R_g = L(R) - L(G) \tag{8.11}$$
$$B_y = L(B) - \frac{L(G) + L(R)}{2} \tag{8.12}$$

where $L(x)$ is defined by

$$L(x) = 105 \; log_{10}(x + 1 + n) \tag{8.13}$$

and n is a random noise value from the range [0,1]. The I component is used to produce a texture amplitude map as follows:

$$texture = med_2(|I - med_1(I)|) \tag{8.14}$$

where med_1 and med_2 are two separate median filters of different sizes (med_2 is 1.5 times the size of med_1). The texture amplitude map is used to find regions of low texture, since skin in images tends to have a very smooth texture.

Hue and saturation are used to select the regions whose color matches that of skin. The R_g and B_y images are also median filtered before being used in the calculations. The conversion from log opponent space to hue and saturation is given by

$$hue = atan(R_g, B_y) \tag{8.15}$$
$$saturation = \sqrt{R_g^2 + B_y^2} \tag{8.16}$$

If a pixel falls into either of the following two ranges, it is marked as a skin pixel.

1. $texture < 5, \; 110 < hue < 150, \; 20 < saturation < 60$
2. $texture < 5, \; 130 < hue < 170, \; 30 < saturation < 130$

Note that all the constants used above are from the original work and can be modified for different data sets or user preference.

The skin map is a binary array where 1-pixels are skin pixels and 0-pixels are non-skin pixels. This array can be processed by a morphological closing operation to produce a cleaner result. Once the images with skin regions have been found, they can be checked for 1) having enough flesh to be considered pornography (30% of the image was used) and 2) having regions in appropriate spatial relations to be considered human body parts.

Exercise 7 Flesh and Face Finding

Implement a flesh finder to find regions of flesh color. Select regions of a specified size and larger and try to find evidence of facial features: in particular, the eyes, nose, and mouth, in that order of priority. Based on the features you find, assign to each region the probability of being a face.

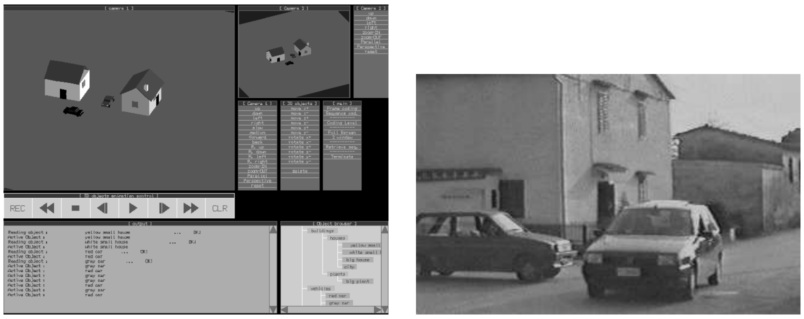
Original Image Segmentation Symbolic Representation

Figure 8.9: Objects and spatial relationships that can be extracted from images and used for retrieval. (Original image licensed from Corel Stock Photos.)

Spatial Relationships Once objects can be recognized, their spatial relationships can also be determined, and queries can be formulated that require a certain set of named objects in predetermined spatial relationships. This is the final step in the image retrieval hierarchy. In recent work at Berkeley and in similar work at Santa Barbara, researchers have sucessfully used both color and texture to segment images into regions that often correspond to objects or scene backgrounds. Such objects as tigers and zebras that stand out well and have a particular color/texture pattern can be found in this way. Backgrounds such as jungle or sky or beach can also be isolated. Figure 8.9 gives an example of this type of segmentation process. The original color image is shown at left, and its segmentation into regions is shown at center. A symbolic representation of the image in which the regions of interest are depicted as ellipses is shown at right. This representation can be used to construct a relational graph whose nodes are the classifications of the regions and whose edges represent spatial relationships. Now relational matching techniques can be used to create relational distance measures for image retrieval. Although the system shown here doesn't go that far, Del Bimbo has developed a retrieval system that could use this representation as its input. This system allows users to construct queries by placing selected icons in spatial relationships on the query screen and returns images having the corresponding objects in those relationships. Figure 8.10 shows an example of retrieval by this spatial query system.

Exercise 8 Retrieval by Objects and Relationships

Obtain or write a program that segments a color image into regions based on color and, if possible, texture. Run the program on a set of training images in which each class of object, such as tiger, sky, jungle, is present in several images. Train a classification algorithm on these known regions according to their color and texture properties. Write a program that uses the segmenter and classifier to produce a set of labeled regions for an input image and then computes the spatial relationships *above*, *below*, *left-of*, *right-of*, and *adjacent-to* between pairs of regions. Then write an interactive front end that allows users to input a graph structure in which the nodes are objects from the training set and the edges are the required relationships. The program should return all database images that satisfy the user's query.



Query Window Retrieved Image

Figure 8.10: Results of a spatial-relationship query. (Figure courtesy of Alberto Del Bimbo with permission of IEEE. Reprinted from "Symbolic Description and Visual Querying of Image Sequences using Spatio-Temporal Logic," by A. Del Bimbo, E. Vicario, D. Zingoni, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 4, Aug. 1995. © 1995 IEEE)

8.5 Database Organization

Large databases of images, like any other large databases, are too big to search the whole database for images that satisfy a given query. Instead, the images must be organized and indexed so that only a fraction of them are even considered for any one query. There are standard methods for indexing numeric and textual data that are used in most relational database systems. Methods for indexing spatial data also exist and have been used, in particular, for geographic information systems. Methods for indexing images for content-based image retrieval are being developed for current research systems.

8.5.1 Standard Indexes

In most relational databases, the user can specify an attribute on which an index is to be built. Usually this attribute is an important key associated with each data record. For example, if the database contains records of the employees who work for a certain company, then the social security number would be a good attribute to use to index the data. Since everyone has a unique social security number, this attribute is called a *primary key*. If the data is often accessed by some other attribute, such as last name of employee, a separate index can be built for it.

In a relational database, an *index* is a data structure with which the system can look up a given attribute value and rapidly find the set of all records in the database that have that value for that attribute. There are two common types of indexes used in relational database systems: hash indexes and B-tree indexes. Hash indexes allow rapid determination of the set of data records that have exactly the attribute value specified in the query. B-tree or B^+-tree indexes allow a speedy search for records whose attribute values lie in the range specified by a query.

Hash Indexes A hash index applies the theory of the *hash table* to access a large set of records in a database. The assumption is that there is a potentially large set of possible key values and only a fraction of these are ever present in the database at one time. Suppose that the database consists of a file of N records. Each record contains several different fields including one field for the key values. The access mechanism for the hash index is a *hash function* that maps any key value into an address in the file that contains (or sometimes points to) a database record containing the particular key value. If the key values are numeric, then one simple hash function is $f(x) = x \ mod \ N$, which reduces to dividing x by N and using the remainder of the division as the record number of the record to be accessed. Figure 8.11 shows a hash index for a database with numeric keys. The hash table has ten positions numbered 0 through 9 (a real hash table would be much bigger). The hash function being used is $f(x) = x \ mod \ 10$. The query shown is asking for all records whose key value is equal to 45, which hashes to positon 5 in the hash table.

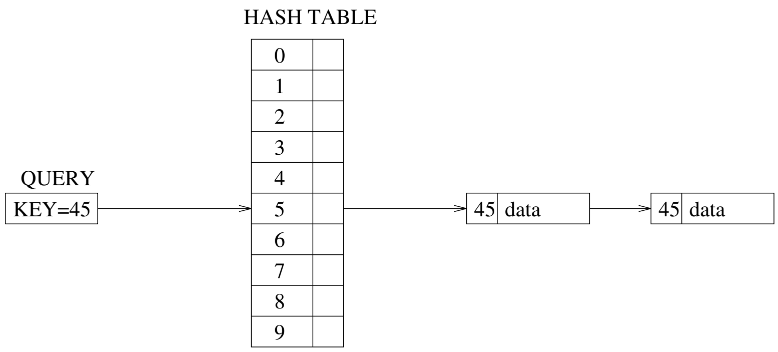


Figure 8.11: A sample hash index.

If each key value hashes to a different location in the table, then the access time for any given key is constant. Generally, this doesn't happen. Instead, several different keys may hash to the same location. This phenomenum is called a *collision* and solutions to it can be found in any data structures text. The solution shown in the figure is to keep a linked list of all records whose keys hash to the same location. The end result, in any case, is that the access process may involve a small amount of search, rather than being a simple direct access, but for good hash functions and tables that are not too full, the access time is still approximately constant. Because of the nature of the hashing process, it is most suitable for an exact match constraint of the form $KEY = VALUE$ and is less useful for range queries.

B$^+$-Tree Indexes B-trees and B$^+$-trees are balanced multi-way search trees that can be used for indexing and are suitable for range queries. B-trees have values of keys and data at both internal and leaf nodes, while B$^+$ trees have data only at leaf nodes. We will concentrate on B$^+$ trees, since the data in a database should be separate from the index.

A search tree of order p is a tree in which each node contains $\leq p - 1$ key values and p pointers. A B$^+$-tree is a search tree that has one format for internal nodes and a second format for leaf nodes. Each internal node of a B$^+$-tree has the following constraints:

1. It has the format $< P_1, K_1, P_2, K_2, \ldots, P_{q-1}, K_{q-1}, P_q >$ where each P_i is a pointer to another node and each K_i is a key value. Intuitively, P_{i-1} points to a subtree whose nodes contain keys with values less than or equal to K_i, while P_i points to a subtree whose nodes contain keys with values greater than K_i.

2. If it is a nonroot node, it has at least $\lceil (p/2) \rceil$ subtree pointers.

3. If it is a root node, it has at least 2 subtree pointers.

Each leaf node of a B$^+$-tree satisfies the following:

1. It has the form $< K_1, Pr_1, K_2, Pr_2, \ldots, K_{q-1}, Pr_{q-1} >, P_{next} >$ where K_i is a key, Pr_i is a data pointer, and P_{next} points to the next leaf node.

2. Pr_i points to a record whose search key is K_i or to a block of such records if the search field of the index is not a key of the file.

3. Each leaf node has $\lfloor (p/2) \rfloor$ values.

4. All leaf nodes are at the same level of the tree.

Internal nodes may have a different order p than leaf nodes; the goal is usually to fit each type of node into the physical block of storage that is the unit amount transferred from disk to internal memory.

To find a given key value or range of values in a B$^+$-tree, the retrieval system starts at the root of the tree. It reads that node into memory and performs a binary search of the keys in that node. If it finds two adjacent keys values in the node between which the given key value lies, the pointer between them points to the subtree that will contain the given key value or the smallest value in the given key range. If the given key value is less than the first key value in the node, the pointer to the left of this key value references the appropriate subtree. Similarly, if the given key value is greater than the last key value in the node, the pointer to the right of this key value references the correct subtree. Once the appropriate subtree has been identified, it becomes the root of the tree to be searched. The retrieval system performs this operation recursively until it reaches a leaf node.

In the leaf node, a binary search is again performed to find the given key or starting key value K_i. The associated pointer P_i then points to the data record containing this key value. If only one key value is sought, the associated record can now be returned. If this is a range search, the P_{next} pointers in the data records can be used to find the remaining data records until the end of the specified key range is encountered.

Figure 8.12 gives an example B$^+$-tree that indexes database records with numeric keys and image data. The internal nodes are shown with solid rectangles, and the leaf nodes are shown with dashed rectangles. The root node points to three different subtrees: those with key values less than or equal to 100, those with key values between 100 and 200, and those with key values greater than 200. The subtree with key values between 100 and 200 is shown. Its root points to two leaf nodes: one with key values less than 110 and one with key values between 110 and 150. The leaf nodes contain some actual key values and associated

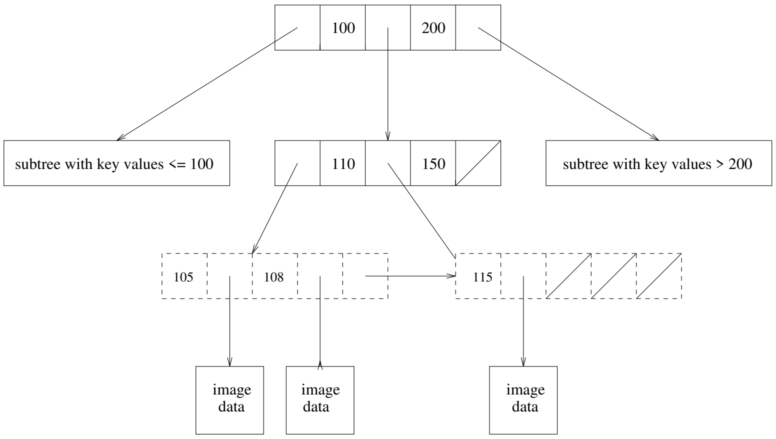


Figure 8.12: A sample B^+-tree index.

image data files.

B^+ trees are flexible and efficient and are used heavily in relational database systems. They can be used in image database systems to index single numeric or text fields associated with an image. They were not intended to be used to index multi-dimensional data.

8.5.2 Spatial Indexing

Spatial information systems contain data that is multi-dimensional. A number of structures have been proposed for spatial indexing. Quadtrees are hierarchical structures of degree four that break up the search space for 2D data into four sub-quadrants at each level of the tree. Quad trees can be used to represent regions in binary images. K-d trees are an extension of binary search trees, which allow search for k-dimensional data. R-trees are an extension of B-trees to higher dimensions and are suitable for a variety of spatial information system applications. In an R-tree, a data object is indexed by an n-dimensional minimum bounding rectangle (MBR), which bounds the space occupied by the object. Each actual data object is referenced by a unique identifier (ID). The leaf nodes of the R-tree contain the data object IDs. The internal nodes contain entries of the form (MBR,CHILD) where CHILD is a pointer to a lower node in the R-tree and MBR covers all the rectangles in the lower node's entries. Figure 8.13 shows a sample R-tree index for a collection of 2D objects. The distribution of the rectangles depends on the order in which the tree was constructed and the exact R-tree construction algorithm used. Variants such as R^+-trees and R^*-trees also exist.

8.5.3 Indexing for Content-Based Image Retrieval with Multiple Distance Measures

The above methods can be used to index images for retrieval via simple distance measures that are based on a single attribute or a small number of attributes. They are not suitable for

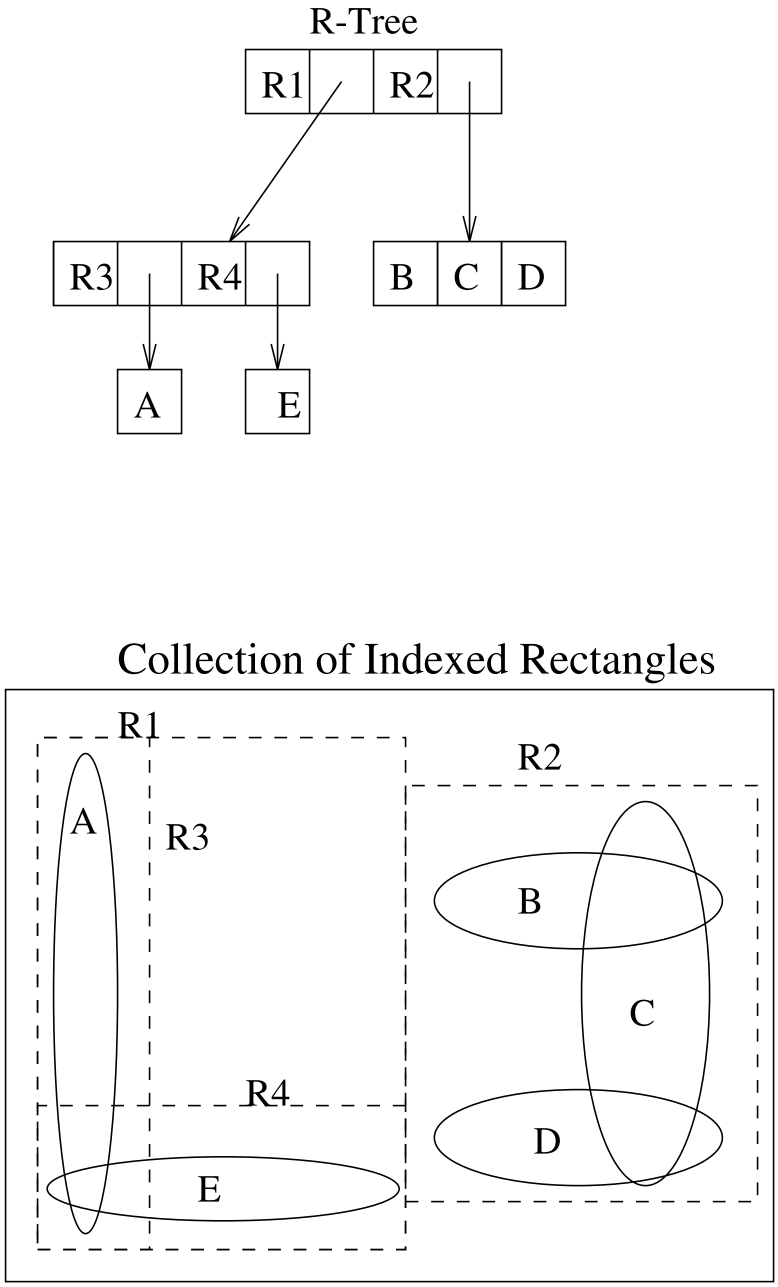


Figure 8.13: A sample R-tree index for 2D data. The ellipses represent the data objects and the indexing rectangles are shown with dashed lines. Rectangle R1 has been broken down further into rectangles R3 and R4, each of which contain a single data object. Rectangle R2 has not been broken down further and contains three data objects: B, C, and D.

a larger, general system that provides the user with the choice of a number of base distance measures and methods to combine them. Such a system requires a more flexible form of organization and indexing. If the base measures are metrics, then the triangle inequality property can be used to provide a nonstandard indexing method. The triangle inequality says that if Q is a query image, I is a database image, and K is a specially-selected *key image*, then

$$d(I, Q) \geq |d(I, K) - d(Q, K)|$$

for any image distance measure d. Thus by comparing the database and query images to a third key image, a lower bound on the distance between the query image and the database image can be obtained.

Consider first the case of a single distance metric d. A set of key images can be selected from the database; intuitively, they should represent the different classes of scenes in the database. The query image Q is compared to each of the keys, $K_1, K_2, \ldots, K_M$, obtaining a set of distances $d(Q, K_1), d(Q, K_2), \ldots, d(Q, K_M)$. Suppose that the user has specified that all images with distance less than T from the query Q should be returned. Then for each key K_i, all images I such that

$$|d(I, K_i) - d(Q, K_i)| > T$$

can be immediately ruled out, since $d(I, Q)$ is already known to be too large. A data structure called the *triangle-trie* has been designed to take advantage of this approach and to rule out most of the images in a database from direct comparison to the query. The technique has also been extended to handle dynamically defined distance measures that are linear or Boolean combinations of the base distance measures.

Exercise 9 Indexing

Suppose a set of images is to be indexed according to Laws' texture energy measures. Explain how you could use R-trees as your indexing mechanism for this system.

8.6 References

1. R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu, "Proximity matching using fixed-queries trees," *Combinatorial Pattern Matching*, pp 198–212, Springer-Verlag, June (1994).

2. A. P. Berman, "A new data structure for fast approximate matching," Technical Report 94-03-02, Department of Computer Science and Engineering, University of Washington, (1994).

3. A. P. Berman and L. G. Shapiro, "A Flexible Image Database System for Content-Based Retrieval," *Computer Vision and Image Understanding*, Vol. 75, Nos. 1-2, pp. 175-195 (1999).

4. C. Carson, S. Belongie, H. Greenspan, and J. Malik, "Region-Based Image Querying," in Proc. IEEE Workshop on Content-Based Access of Image and Video Libraries, (1997).

5. A. Del Bimbo, E. Vicario, D. Zingoni, "Sequence retrieval by contents through spatio temporal indexing," *IEEE Simposium on Visual Languges* pp. 88-92, (1993).

6. A. Del Bimbo, P. Pala, S. Santini, "Visual image retrieval by elastic deformation of object sketches," *IEEE Symposium on Visual Languages*, pp. 216-223, (1994).

7. M. M. Fleck, D. A. Forsyth, and C. Pregler, "Finding Naked People," *Proceddings of the European Conference on Computer Vision*, Springer-Verlag, 1996, pp. 593-602.

8. Forsyth, D. A., J. Malik, M. M. Fleck, H. Greenspan, T. Leung, S. Belongie, C. Carson, and C. Bregler, "Finding Pictures of Objects in Large Collections of Images," *Proceedings of the 2nd International Workshop on Object Representation in Computer Vision*, April, 1996.

9. T. Kato, T. Kurita, N. Otsu, K. Hirata, "A sketch retrieval method for full color image database," *11th International Conference on Pattern Recognition*, pp. 530-533, (1992).

10. W. Y. Ma and B. S. Manjunath, "NETRA: A Toolbox for Navigating Large Image Databases," in Proc. IEEE Workshop on Content-Based Access of Image and Video Libraries, 1997.

11. T. P. Minka and R. W. Picard, "Interactive Learning with a Society of Models," *Proceedings of CVPR-96*, pp. 447-452, (1996).

12. W. Niblack *et al.*, "The QBIC Project: Querying Images by Content using Color, Texture, and Shape," in *SPIE Proc. Storage and Retrieval for Image and Video Databases*, 1993.

13. R. W. Picard and T. P. Minka, "Vision texture for annotation," *Journal of Multimedia Systems*, Vol. 3, pp. 3-14, (1995).

14. H. Rowley, S. Baluja, and T. Kanade, "Human Face Detection in Visual Scenes," Carnegie-Mellon University, 1996.

15. H. Samet, *The Design and Analysis of Spatial Data Structure*, Addison-Wesley, Reading, MA, 1990.