# CSE 557 Project: Subimage Querying

Doug Zongker Nitin Sharma

March 17, 1997

Our project was investigations of algorithms for locating a weighted query image within a target image. The ultimate goal is to make this process fast enough to enable searching of an image database for a given subimage. We developed an application to test various methods for locating the query image.

## 1 Terminology

Let the query image be $Q$, with width $w_q$ and height $h_q$. This is the image we are searching for. The mask image $M$, of the same dimensions as the query and values in $[0, 1]$, is used to weight the pixels of the query. The target image be $T$, with width $w_t$ and height $h_t$. This is the image we are searching in.

When discussing time complexity of algorithms, we consider the dimensions of $Q$ to be $m$ and those of $T$ to be $n$. Thus $Q$ is $O(m^2)$ pixels and $T$ has $O(n^2)$.

This paper discusses matching of single channel images. The application uses these algorithms on color images by matching the red, green, and blue channels separately and adding the results.

## 2 Algorithms

### 2.1 Brute-force $L_1$

The first metric sweeps the query over the target, summing the weighted absolute value of the difference between the query and the target at each location. The location with the minimum sum is selected. This chooses $(u, v)$ so that

$$\sum_{(x,y)=(0,0)}^{(w_q,h_q)} M(x,y)\,|Q(x,y) - T(x+u, y+v)|$$

is minimized. This measure is the traditional $L_1$ norm for the two images. This algorithm takes $O(n^2m^2)$ time.

### 2.2 Brute-force $L_2$

This is the same as the previous algorithm, except that the square of the difference rather than the absolute value is used. This chooses $(u, v)$ to minimize

$$\sum_{(x,y)=(0,0)}^{(w_q,h_q)} M(x,y)\left(Q(x,y) - T(x+u, y+v)\right)^2$$

This measure is the square of the $L_2$ norm. This algorithm also takes $O(n^2m^2)$ time.

## 2.3 $L_2$ with bias and gain corrections

Matching two images taken under different light conditions requires eliminating the *bias* and *gain* effects. Every pixel might be scaled by a constant factor (called *gain*) and differ by a constant value (called *bias*).

This section describes the technique we used to eliminate bias and gain effects. Let $a$ be the gain parameter and the $b$ the bias parameter.

We wish to find $a$, $b$, which minimize

$$\sum_{(x,y)=(0,0)}^{(w_q,h_q)} M(x,y)\left(\{a.Q(x,y) + b\} - T(x+u, y+v)\right)^2$$

The parameters $a$ and $b$ can be found by taking the partial derivatives with respect to $a$ and $b$ respectively, and putting them equal to zero.

The optimal values of $a$ and $b$ are the solution to the following set of equations:

$$c_1 \cdot a + c_2 \cdot b = c_3$$

$$d_1 \cdot a + d_2 \cdot b = d_3$$

where

$$c_1 = \sum_{(x,y)=(0,0)}^{(w_q,h_q)} M(x,y) \cdot Q^2(x,y),$$

$$c_2 = d_1 = \sum_{(x,y)=(0,0)}^{(w_q,h_q)} M(x,y) \cdot Q(x,y),$$

$$d_1 = \sum_{(x,y)=(0,0)}^{(w_q,h_q)} M(x,y).$$

We find the bias and gain coefficients for each displacement of the target image, compute the (square of) $L_2$ norm with the corrections applied and then, find displacements (u,v) for which the $L_2$ norm is minimum.

There is also a penalty term for both the bias and gain parameters included in the $L_2$ calculation which is not shown above. This is to ensure that bias and gain terms don't become too large. In our implementation, we used a very simple penalty model: we treated the two images as non-matching if the gain and bias were more than a certain value (which we arbitrarily chose as 5 for gain, and 150 for the bias).

The bias and gain correction step takes $O(m^2)$ time, so the overall complexity of this algorithm is still $O(n^2m^2)$ time, but the constant involved in higher.

## 2.4 Correlation

Correlation is another metric for comparing degree of match between two images. It is defined as

$$WeightedCor(Q,T) = \sum_{(x,y)=(0,0)}^{(w_q,h_q)} M(x,y)\,(Q(x,y)\cdot T(x+u,y+v))$$

Note that correlation is just one of the components in the $L_2$ norm (the other components being the energy terms for the query and target images). We would want to *maximize* correlation because higher the correlation, lower will be the $L_2$ distance between the two images.

If we use this metric, the query image tended to match best with the brighter regions of the target image. This is as expected, because for a given image, its correlation will be the highest with a completely white image.

The time complexity for this metric is also $O(n^2m^2)$.

## 2.5 Correlation with bias and gain correction

We can also apply bias and gain corrections to the query image before computing the correlation. We use exactly the same technique for computing the correction parameters as for the $L_2$ metric.

## 2.6 Phase Correlation

Phase correlation is another technique for locating the best matches within the target image. It has its underpinnings in Fourier Analysis. The disadvantage of this method is that it does not work with weighting functions.

Suppose the query and target images at a displacement of $(u, v)$ relative to each other. That is,

$$Q(x,y) = T(x+u, y+v)$$

Let $Q$ and $T$ be the fourier transforms of the query and target images respectively. Then, by the translation property of Fourier Transforms, (also called *Shift Theorem*), on taking Fourier Transform of both sides:

$$Q = e^{-j(\omega_x u+\omega_y v)}T$$

If we compute the cross-power spectrum of the two images defined as:

$$\frac{Q\cdot T^*}{|Q\cdot T^*|} = e^{-j(\omega_x u+\omega_y v)}$$

the *Shift Theorem* guarantees that the phase of the cross-power spectrum is equivalent to the phase difference between the images.

If we take the inverse Fourier Transform of the phase of the cross power spectrum, we will get a function which is an impulse: that is, it is zero everywhere except at the displacement (u,v). So, the location the peaks in the cross-power spectrum tells us exactly where the best match occurs in the target image.

The advantage of using this technique is that unlike for L2 norm, this technique is insensitive to the changes in bias and gain, as bias and gain changes affect only a narrow frequency band in the fourier transform of the image.

### 2.6.1 Precalculating Target FFTs

We need to compute the fourier transform of the target image for every query image we search for. Query search could be made faster if we keep the fourier transforms of the target images handy. We precompute them as a part of the preprocessing step and store it in our database in addition to the original target images. However, this might lead to a blowup in the image database size. If we store them the naive way, we would need to store two values for each pixel in the original image, one for real and one for imaginary part, as the terms are complex quantities. Since both these are *floating point* values, this is an eightfold increase in storage.

However, we can store it in just two bytes, one byte for the *logarithm* of the magnitude and one byte for the phase of each term. We find the minimum and maximum values of these quantities, and map the these values to the range 0 to 255. This might involve quantization and hence some loss of precision, but we noticed very little deterioration in quality of the image reconstructed from the fourier transform stored in this way. (see figure 1)

This method leads to only a 100 % increase in the size of our image database. It could further be reduced to 50 % if exploit the fact the target image is a real function and its fourier transform is symmetric, and so we can do with storing only half the values. However, we did not implement this optimization.

## 2.7 $L_2$ via Fourier transform

$L_2$ computation for each displacement $(u, v)$ can be done more efficiently using the Fourier Transforms.

The square of the $L_2$ norm is:

$$= \sum_{(x,y)=(0,0)}^{(w_q,h_q)} M(x,y)\left(Q(x,y) - T(x+u, y+v)\right)^2$$

.

$$= \sum_{(x,y)=(0,0)}^{(w_q,h_q)} M(x,y)\left(Q^2(x,y) - 2Q(x,y)T(x+u,y+v) + T^2(x+u,y+v)\right)^2$$

.

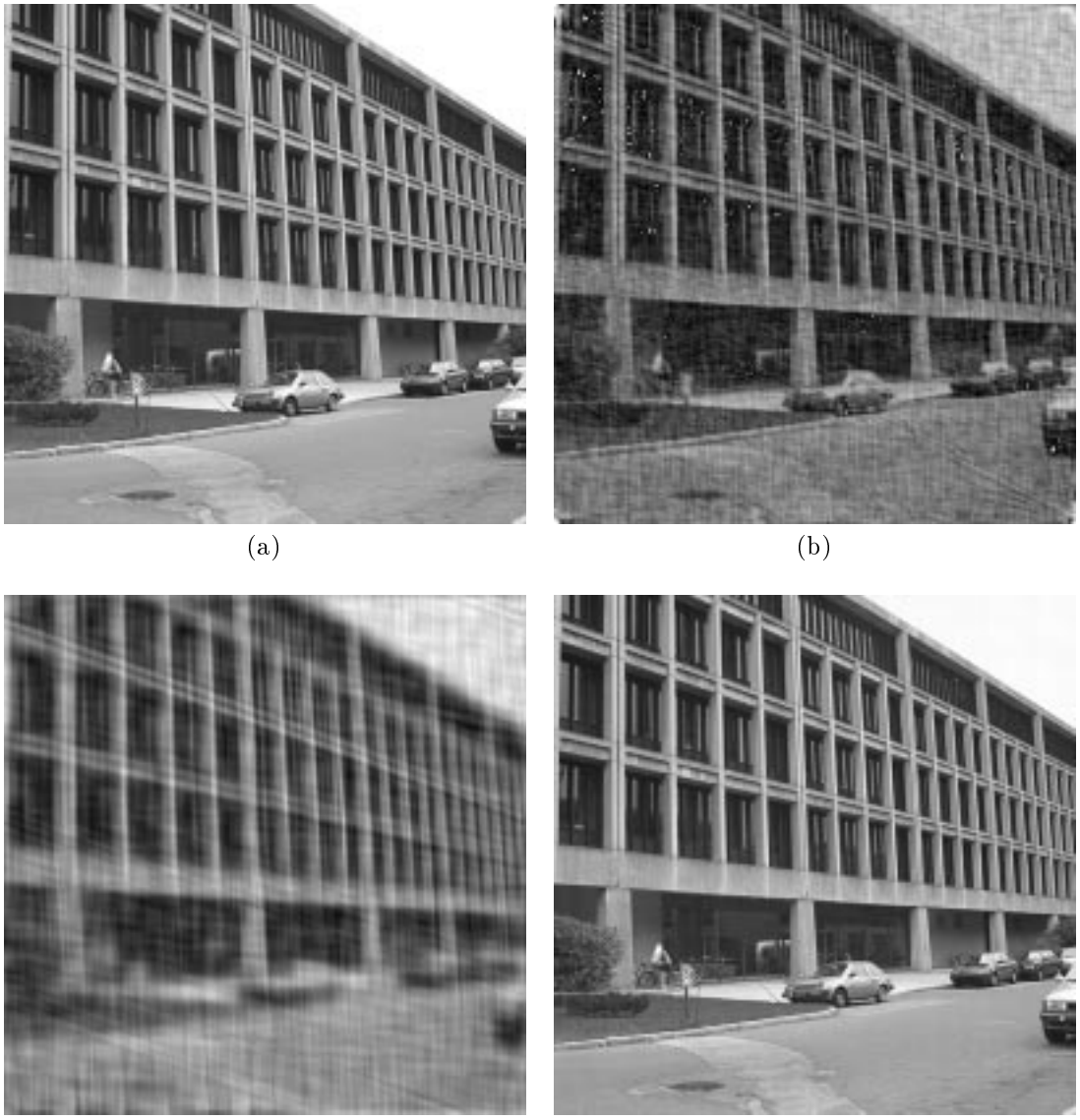
Taking Fourier transform of both sides:

$$\mathcal{F}(L_2) = k - 2MQ \cdot T^* + M \cdot (T^2)^*$$

where
$\mathrm{MQ} = \mathcal{F}\{M \cdot Q(x,y)\}$,
$M = \mathcal{F}\{M(x,y)\}$,
$T = \mathcal{F}\{T(x,y)\}$,
$T^2 = \mathcal{F}\{T^2(x,y)\}$, and
$k$ is a constant.

$$L_2 = \mathcal{F}^{-1}\{k - 2MQ \cdot T^* + M \cdot (T^2)^*\}$$

Fourier Transforms can be computed in $O(N \log N)$ time, where $N$ is the image size. So, the overall complexity of this method is $O((m^2 + n^2)\log n)$.



(a) (b)

(c) (d)

Figure 1: Results of quantizing FFTs to 16 bits/pixel and reconstructing. (a) original image; (b) quantizing real and imaginary parts to 8 bits each; (c) quantizing magnitude and phase to 8 bits each; (d) quantizing log magnitude and phase to 8 bits each.

### 2.8 Hierarchical $L_2$ (averaging)

A more efficient way of locating the best match is the *hierarchical* search. We construct coarser (and smaller) images of the query and target, and (conceptually) arrange them in form of a pyramid. The base layer of the pyramid is the original image. Each subsequent layer of the pyramid is a coarser (factor 2) approximation of its adjacent lower layer, ie. we take the average of every pixel and its right neighbour in the lower layer image to construct the next higher layer of the pyramid. Our implementation has a four-layer pyramid.

We begin by searching for the query image in the topmost layer of the pyramid and look for best 40 matches. This search is fast as the coarser image is of smaller size, but gives us only an approximate location of the probable best matches. We refine our search further by matching the query in the lower layer (higher resolution) image only around the neighbourhood of these best matches. For each match, we compute $L_2$ norms for its nine neighbours and select the best ones among them. We then repeat this step, pruning off the list of candidate best matches until we get to the base layer of the pyramid. Hopefully, we would have located the exact position of the best match in this process. It is possible to miss the best match quite earlier on in the search because the first few steps involve matching in the coarse image.

This approach leads to a remarkable speed-up in the running time, without compromising on the quality of matches (we missed the "real" best match in only one of the test cases.) The graphs compare the execution times of different algorithms we implemented.

We also tried a variation on the hierarchical searching idea. Instead of subsampling the image when we go up to higher levels in the pyramid, we compute the average of a pixel and its adjacent pixel. The searching algorithm remains the same.

The run-time of this algorithm is still the same but quality of matches and sensitivity to noise changes. We found that...

## 3 Performance Results

## 4 Conclusion