Chapter 1

Introduction

1.1 Motivation

In the last half of the 19th century people commonly went to a photographic studio for portraits. Photography was still in its infancy, resulting in black-and-white images and long exposure times. Because people generally do not like to sit still for minutes at a time, they usually ordered a dozen prints from one negative. These copies were spread around when the opportunity presented itself.
Nowadays many of these portraits can still be found, in photo albums and in public and private archives. The pictures, even when they were originally printed from one negative, may differ greatly. Some have been exposed to the sun for years. Some have the name of the person in the picture written on them. Some have been cut up or reframed.
Often it is no longer clear who the person in the picture is, and people spend considerable time searching for the names of those photographed. Because there is such a large of number of pictures around (collections known to the author number over 50,000 photos in the Netherlands alone) this is a very hard task for a human being.
The program written during this project, Photobase, attempts to answer questions like "Who is this man", by comparing an unknown person to a database of known persons and finding matching portraits. The goals of the program are the following:

- Search for direct copies of the input image in the database.
- Search for other photographs of the same person in the database.
- If a matching image is found, show information about these images. This information also applies to the input image.

Comparing images is a difficult problem. The comparison program must compensate for many factors: image quality, image orientation, image scale, etc.

The first step in comparing two images is removing as many of these factors as possible, which is a process referred to as normalization.

This report describes the initial implementation of the Photobase system. In the inital implementation much attention was given to normalization techniques and towards building a good user interface. Primitive comparison techniques were also implemented.

1.2 The degrees of freedom

Images must be scanned before they can be processed. The scanning process introduces various degrees of freedom, each of which must be compensated for before images can be successfully compared. Degrees of freedom caused by the scanning process include:

- **Rotation**
  Often an image is scanned in at a slight angle. It is possible to detect this angle and compensate for it by digitally rotating the image. Several rotation algorithms were implemented and tested.

- **Borders**
  Not every part of a scanned-in image is important. Especially the white background around the image is not significant for the image comparison process. Therefore, during the second normalization step this border is removed.

- **Lighting**
  Images can be scanned in at various lighting settings. In addition, the images themselves can be darker or lighter.

- **Scanner noise**
  The scanning process adds a small amount of noise to the image. Several methods to visualize and remove this noise were implemented and tested.

- **Resolution normalization**
  Images are often at a different resolution. Before comparison is possible the images must be rescaled. A scaling algorithm was developed that causes very little noise.

1.3 Input protocol

The implemented normalization techniques try to cope with many different degrees of freedom for the input image. Some restrictions remain, though:

- The input image must be in TIFF 6.0 format.

- When the input image is scanned a border must be left open around the entire image.
- The border around the image must be colored white or nearly white.
- The angle detection algorithm assumes that the image is contained in a rectangular box.
- It is assumed that no parts of the original image were cut off. (Sub-image searches have not been implemented yet.)

In the next chapters the image normalization techniques are presented.

Chapter 2

Angle detection

In order to automatically correct the rotation of an image its angle must be detected. An algorithm has been developed that can detect the orientation of a picture with some restrictions:

- It is assumed that the picture is contained in a rectangular box, and that there is a distinct border around the actual image.
- The orientation is within 45 degrees of being correct, correct being defined as the image having an angle of 0.0 degrees.
- In case of large aberrations in the angle, it is assumed that the image is higher than that it is wide.

The algorithm consists of several distinct steps. These steps are described below.

Step 1: find pixels that lie on the edge between the border and the image.

At regular distances, the algorithm tries to find pixels that together define one of the borders of the image. This is done by comparing pixels in the left-most column of the image with pixels that lie to the right of these; as soon as the difference in gray value exceeds a pre-defined threshold a pixel is considered to be on the edge, and its coordinates are stored in the coordinate array. A total of 20 coordinate pairs is selected this way.
Experimentation shows that 10% of the gray range is a good value for the threshold.
The coordinates of the highlighted pixels are stored in the coordinate array.

Step 2: remove redundant coordinate pairs.

At the extreme top and bottom of the image, the tested pixels may be completely outside the image. The coordinate pairs that describe these pixels hold no meaningful information and are removed from the coordinate array.

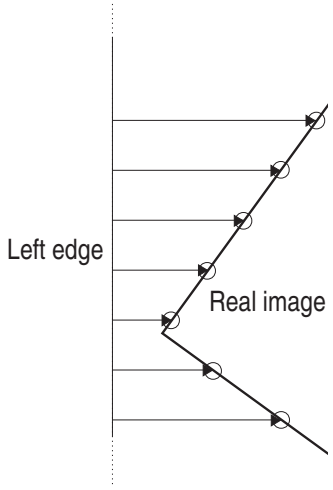

Figure 2.1: Example of angle detection

Step 3: select a set of coordinate pairs for further processing.

At this point the coordinate array contains information that describes either one or two lines, each of which corresponds with an edge of the image. In Figure 2.1 five of the pixels are part of the left edge of the image, while the other three are part of the bottom edge. It is necessary to find out which pixels are on which edge.
To this purpose, the algorithm counts the number of pixels that are mostly on a line from top to bottom and from bottom to top. The procedure responsible for this counting first measures the horizontal gap between the first two pixels; if the gap between the second and third pixels is approximately as wide than the three pixels are on one line. This process is repeated for successive pixels until a pixel is found that does not lie on the line, at which time the counting stops.
The number of pixels counted while going from top to bottom is referred to as the DownCount, while the number of pixels counted while going from bottom to top is referred to as the UpCount.
If the DownCount is bigger than the UpCount, the image is slanted to the right, and the down-going pixels describe the left border. If the DownCount is smaller than the UpCount, the image is slanted to the left, and the up-going pixels describe the left border. The set of pixels with the biggest count is selected for further processing in the next step.
This is done because the biggest set of pixels describes the longest side of the image (when seen from a projection on the Y-axis), and it is assumed that the images have an angle less than 45 degrees.

Step 4: use a least squares method to find a straight line through the selected pixels.

The selected coordinate pairs are placed in an array A and the formula $A^T Ax = A^T y$ is solved to find the values a and b that describe a line with the formula $y = ax + b$, which is the line that fits best through all selected coordinate pairs (x, y).

Step 5: calculate the angle of the line.

The angle of a line $y = ax + b$ is defined as $\tan^{-1} a$. This angle can be used to correct the orientation of the image. Figure 2.2 shows the angle together with an example image. This value can be used directly (without further calculation) in any of the rotation procedures.

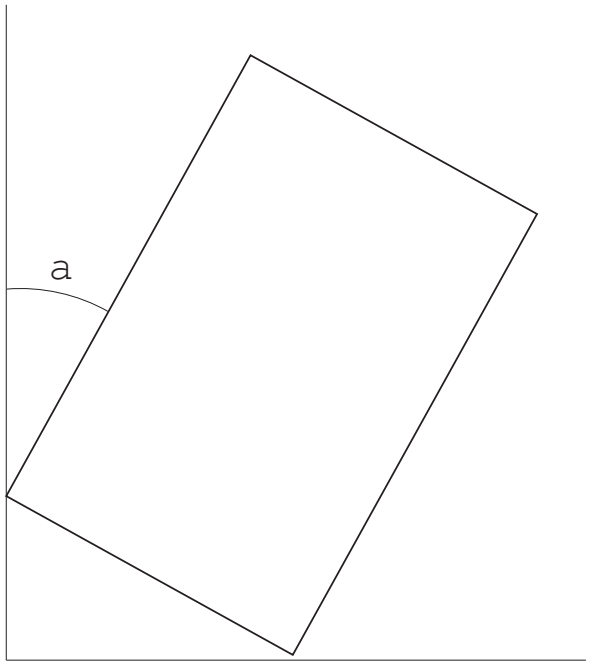

Figure 2.2: The detected angle

Note: $\tan^{-1} a$ has infinite solutions, but it is assumed that the rotation of the image is less than 45 degrees, thus dropping all but one.

Algorithms for correcting the rotation are shown in the following chapter.

Chapter 3

Rotation

It is possible that a scanned image is at a slight angle. The program must compensate for this by performing a rotation. Several rotation techniques were tried, and compared in quality and speed. This chapter describes the tested algorithms. Test results can be found in chapter 9.
There is no generally agreed upon method of measuring the rotation quality, therefore one had to be developed as well. Two factors decide the quality of a rotation:

- A rotation should introduce as little noise as possible. This can be measured by rotating an image over an arbitrary angle (say 15 degrees) and rotating the rotated image back over this angle. The doubly-rotated image should be very close (or if possible, equal) to the original image.

- A rotation should not take too long to complete. This is a subjective measure: whether a rotation is too slow depends on the patience of the user and the speed of his computer.

3.1 The inverse rotation with nearest neighbor

The first technique, also called the primitive rotation, is a simple affine transformation. The algorithm scans through the destination image. For every pixel it determines where it can be found in the source image, and retrieves and draws this pixel. Pixels that are retrieved from outside the source picture are given the intensity of the nearest pixel in the source picture.
Note that part of the source image is cut off during the rotation process. This is not a problem, as this algorithm is meant for straightening pictures that are already rotated. Those pictures have a wide border, and it is only the border that is cut off as can be seen in Figure: 3.1. Since one of the following normalization steps cuts off all the borders anyway no steps were taken to rectify this situation.

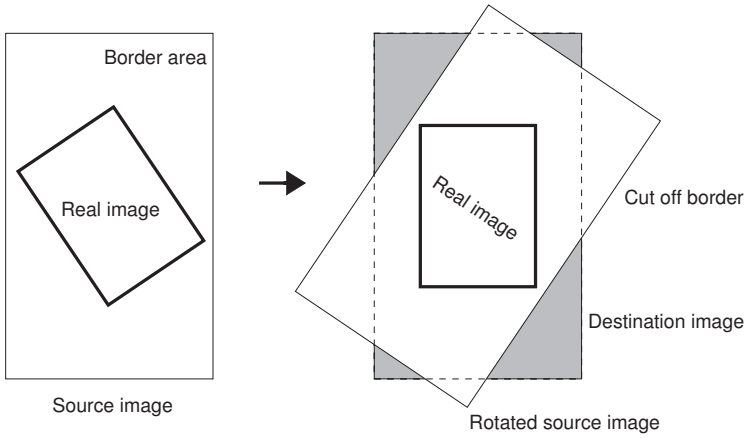

Figure 3.1: No part of the image is lost during rotation

Source coordinates are derived from destination coordinates with the following formulae:

$$SourceX = DestX * \cos(Angle) - DestY * \sin(Angle) \tag{3.1}$$

$$SourceY = DestX * \sin(Angle) + DestY * \cos(Angle) \tag{3.2}$$

Both SourceX and SourceY are cast to integer.

3.2 Rotation with bilinear interpolation

When source coordinates are calculated in the primitive rotation, information is lost during the conversion to integer in the final step of the calculation. However, it is possible to use this information to increase the quality of the rotation. This realization led to the second algorithm, called the rotation with bilinear interpolation.
The rotation with bilinear interpolation works the same way as the primitive rotation, but SourceX and SourceY are floating points values so that casting to integer is not necessary. The source coordinate pair points to a spot between four pixels. The color of the destination pixel is determined by the distance to these four pixels.
In the example in Figure: 3.2 the intensity of the destination pixel would be

$$0.73 * (0.52 * P_1 + 0.48 * P_2) + 0.27 * (0.52 * P_3 + 0.48 * P_4) \tag{3.3}$$

with P_n = the color of pixel n.
The general formulae for bilinear interpolation is

$$Result(x, y) = (P_4 - P_3)x + (P_1 - P_3)y + (P_3 - P_1 - P_4 + P_2)xy + P_2 \tag{3.4}$$

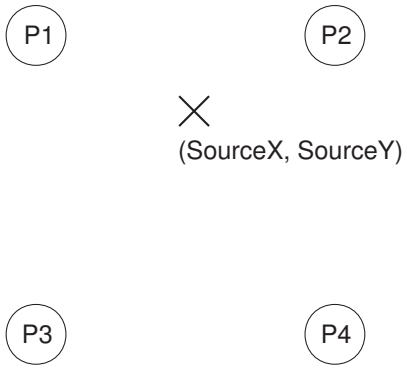

Figure 3.2: Example of rotation with bilinear interpolation

Tests show that this algorithm generates less noise than the primitive rotation, but takes longer to complete.

3.3 The shear-based rotation

An attempt was made to create a faster rotation algorithm based on shearing. This technique requires two steps. In the first step the source picture is sheared horizontally. In the second step the resulting picture is sheared vertically. The advantage of shearing over the first two algorithms is that much less time is spent calculating source coordinates, as entire lines (or columns) can be processed at once.
As with the primitive rotation, the color of each pixel in the destination picture depends on the color of one pixel in the source image. Pixels that are taken from outside the source image are given the value of the nearest pixel inside the source image.
Note that shearing is an approximation of rotation that only works well with small angles. This can be seen in the following manner:
A shear can be described as an affine transformation with the following matrix:

$$\begin{pmatrix} 1 & g \\ h & 1 \end{pmatrix} \tag{3.5}$$

- g = horizontal shear factor
- h = vertical shear factor

A rotation can be described by a similar matrix:

$$\begin{pmatrix} \cos(Angle) & \sin(Angle) \\ -\sin(Angle) & \cos(Angle) \end{pmatrix} \tag{3.6}$$

For very small angles the value of $cos(Angle)$ lies very close to 1, therefore for very small angles a shear is very close to a rotation.
Shear-based rotation is quicker than primitive rotation, but the rotated pictures are much noisier. For this reason it was decided to test shear-based rotation with bilinear interpolation as well.

3.4 Shear-based rotation with bilinear interpolation

This rotation works mostly the same as the shear-based rotation, but every pixel in the destination image is interpolated from two pixels in the source image (for both steps, leading to the expected four pixels).

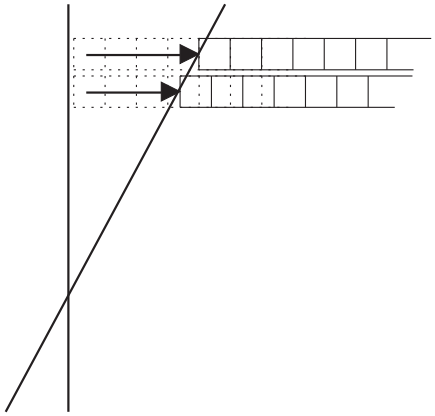

Figure 3.3: Example of shear-based rotation with bilinear interpolation

To create the intermediate picture, lines in the source image are shifted horizontally: every intermediate pixel depends on two source pixels. To create the destination image the columns of the intermediate picture are shifted vertically: every destination pixel depends on four source pixels.
Unfortunately this method is slower and generates more noise than the normal rotation with bilinear interpolation.

3.5 Rotation with bilinear- and cubic interpolation

Because rotation with bilinear interpolation performs best (from the tested algorithms) another variant on this algorithm was tested as well. In this variant the color of a destination pixel depends on the color of more than four source pixels.

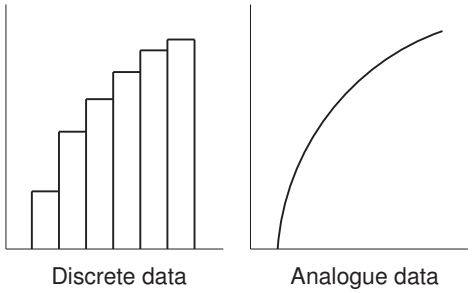

Figure 3.4: Discrete and analogue representation of the same data

To understand why this rotation works it is necessary to think of the digitized images in a different manner than as a collection of pixels. In the (one-dimensional) example in Figure: 3.4 there are two representations of a set of data. In the first graph this data is shown as discrete units. In the second graph the data is shown as it might appear in the real world: an analogue curve.
The first four rotation algorithms act as if the world consists of discrete data, but in reality they work with a digitized version of analogue data. The new algorithm uses a Lagrange method to interpolate data on a third-degree curve (rather than a line). This means that the color value of a pixel in the destination image depends on the colors of 16 pixels in the source image.
Interpolation between four pixels is done with the following formulae:

$$Result = P_1 * L_1 + P_2 * L_2 + P_3 * L_3 + P_4 * L_4 \tag{3.7}$$

- $L_n = \frac{(x-x_1)*\ldots*(x-x_{i-1})(x-x_{i+1})*\ldots*(x-x_n)}{(x_i-x_1)*\ldots*(x_i-x_{i-1})(x_i-x_{i+1})*\ldots*(x_i-x_n)}$
- P_n = the color value of pixel n
- x_n = the x coordinate of pixel n
- x = the required X coordinate

This formula is used four times to interpolate between the horizontal rows of pixels, then once to interpolate between the results of the first four interpolations.
In some cases this algorithm generates less noise than the normal rotation with bilinear interpolation, but in most cases more noise is generated. In all cases speed is significantly lower than any of the other algorithms.

Now that the image has the correct angle, the borders can be removed. This is described in the next chapter.

Chapter 4

Border detection

Borders are necessary for angle detection, but they are not part of the scanned-in image. Because they contain no real information they must be removed before images can be compared. A border can be detected because it starts at the side of an image and is of a uniform color. If one side of the image does not have a uniform color it can be assumed that this side has no distinguishable border which needs to be cut off. As an example, for the left border (see Figure: 4.1): The left-most column is assumed to be part of the border. Then, for the column to the right, it is determined for every pixel if the difference in color with the corresponding pixel in the leftmost column is small enough; if this is so this column is also part of the border. This process continues with columns that lie more and more to the right, until a column is found with pixels that have color values that differ enough from the corresponding pixels in the left-most column. Everything to the left of this column is considered part of the border and must be removed.
The threshold value (the minimum difference between pixels before a pixel is said to be in the image) is user settable. Experiments show that 10% of the intensity is a good value.
This process is repeated for the top, right, and bottom borders, thus cutting off every border of the image.
The process of removing the borders also takes care of removing any possible translation the picture may have. This happens because a translation is equal to wider or smaller upper and left borders.

When the image has the correct size, the differences in lighting have to be compensated for.

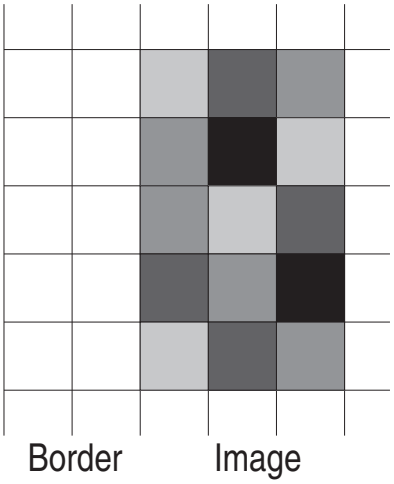

Figure 4.1: Example of an image with a border of uniform color

Chapter 5

Lighting normalization techniques

Normalization of lighting is a difficult task, because the lighting for the original image is unknown. One can not generally say that one technique is better than another. The problem is to find the best technique for the task at hand. What technique is best depends on the image being processed. Furthermore, more complex techniques need a lot of time to calculate the results. In this chapter the following two point processing techniques, techniques where a given gray level $i \in [0, L]$ is transformed to another gray level $v \in [0, L]$ by application of the equation

$$v = f(i), \tag{5.1}$$

are investigated.
Due to equation 5.1 the implemented algorithms are very fast in calculating the enhanced images.

Two alternatives for lighting normalization were implemented: contrast stretching and histogram equalization.

5.1 Contrast stretching

A common contrast stretching transformation for an 8-bit image can be expressed as

$$v = \begin{cases} \alpha i, & 0 \leq i < a \\ \beta(i-a) + v_a, & a \leq i < b \\ \gamma(i-b) + v_b, & b \leq i < 255. \end{cases} \tag{5.2}$$

Figure 5.1 shows a possible result of equation 5.2 .
The normal intention of contrast streching is an extension of the region between $[a, b]$. This is done in this project, too. Moreover, a special case in contrast stretching where $\alpha = \gamma = 0$ called clipping can be used. For instance, the

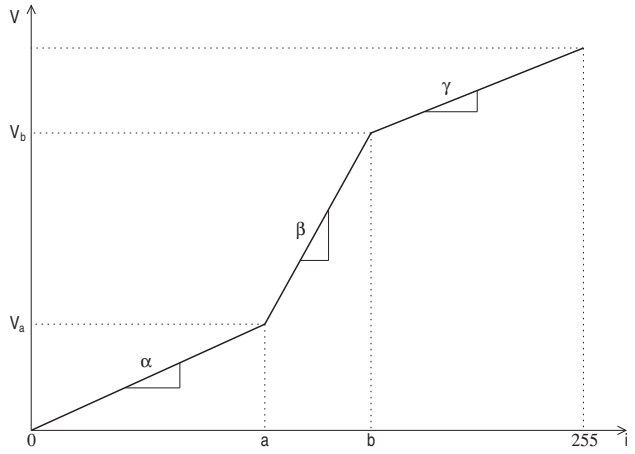

Figure 5.1: A contrast stretching transformation

histogram of Figure: 5.2 has no values between 0-10 and 175-255. The intensity values between 11-174 are stretched using the following formulae:

$$v = \frac{v_a + (v_b - v_a) * (i - a)}{b - a} \tag{5.3}$$

Figure 5.3 shows the result of a contrast stretch.

5.2 Histogram equalization

Histogram equalization has been found to be a powerful image enhancement technique if the image shows low contrast, which can be recognized as a narrow histogram. The histogram of an image represents the relative amount of occurrances of gray levels in the image.

Consider that n is the total number of pixels in an image and n_i represents the number of times the intensity i appears in the image. Then the probability (intensity count (%)) for the appearance of this intensity value in the image can be expressed as:

$$p_i = \frac{n_i}{n} \tag{5.4}$$

A graph of p_i versus i is called a histogram. Histogram equalization forms a uniform histogram by taking the probabilities into consideration.

To calculate the transformed intensities of a histogram equalized image, the following algorithm is used:

$$v_k = L * \sum_{i=0}^{k} \frac{n_i}{n} = L * \sum_{i=0}^{k} p_i \tag{5.5}$$

- n_i represents the number of times this gray level appears in the image
- n = the total number of pixels in the image
- $k = [0, 255]$ intensities
- L = the total amount of intensities

Figure 5.4 shows a result of histogram equalization.

A further step towards image comparison requires that they are scaled to equal resolutions. Chapter 6 gives a strategy to reach this goal.

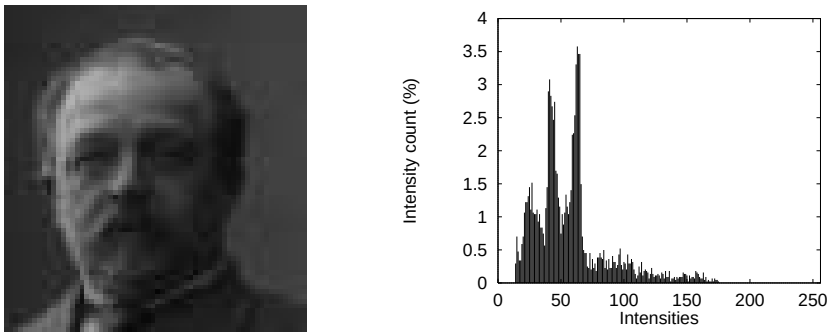

Figure 5.2: The original image and its histogram

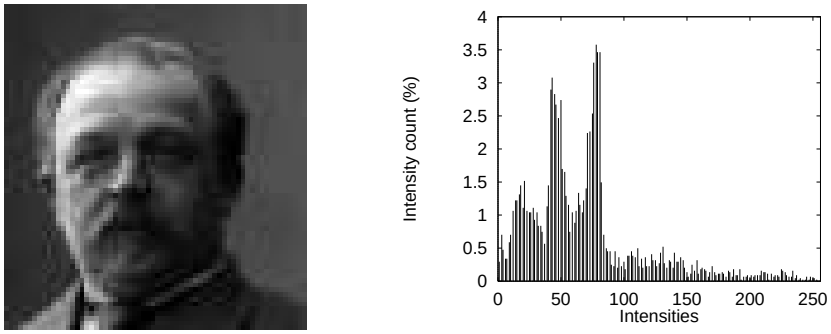

Figure 5.3: The image and its histogram after contrast stretching

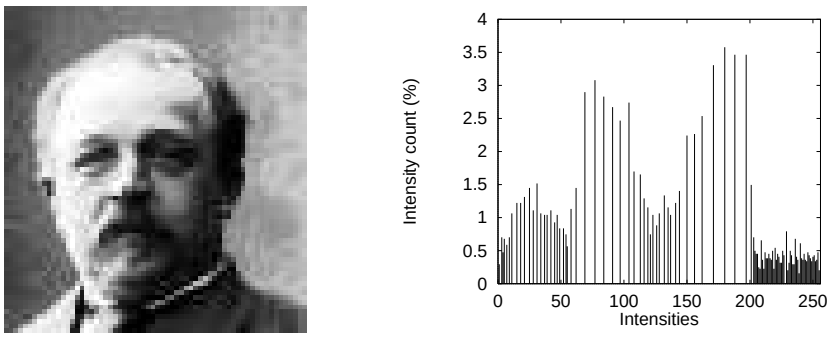

Figure 5.4: The image and its histogram after histogram equalization

Chapter 6

Scaling

The final normalization step that must be taken before images can be compared is scaling them to the same resolution. As with the other steps it is important that the scaling introduces as little noise as possible.
The implemented method for scaling works with bi-linear interpolation, and can (in theory) only shrink images. The routine scans through a destination image, and determines for every pixel in it on what source pixels it depends, and for how much.

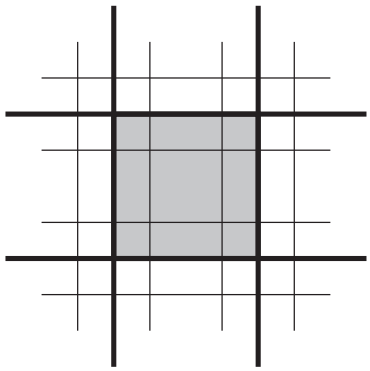

Figure 6.1: The intensity of a pixel in the destination image depends on the intensities of several pixels in the source image

In the example (Fig.: 6.1) The wide lines show pixels in the source image, while the fat lines show pixels in the destination image. The shaded destination pixel depends on nine source pixels, and its value is calculated with the following formula:

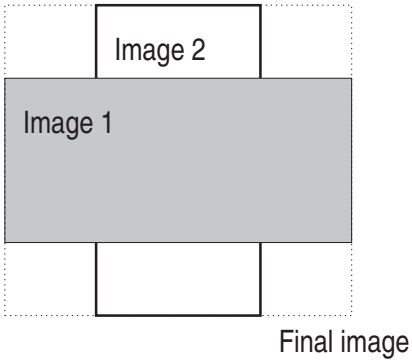

Figure 6.2: Images are scaled to the largest possible size

$$Result = \frac{P_1 + P_3 + P_7 + P_9}{16} + \frac{P_2 + P_4 + P_6 + P_8}{8} + \frac{P_5}{4} \tag{6.1}$$

The procedure can also enlarge images by pretending that every pixel is represented multiple times. This way the source image appears to be bigger than the destination image and the scaling is turned into a shrinking again, which is handled normally.

6.1 Strategy for scaling

When comparing images that have different resolutions, there are two possible methods:

- Scale to the smallest resolution.
- Scale to the biggest resolution.

While scaling to smaller resolution the values of several pixels are stored in one pixel, losing information which would not be lost while scaling to bigger resolution. Because of this the pictures are scaled to the resolution of the bigger picture.
Currently, scaling is applied regardless of the resolution of the images. If two images are of different sizes than they are scaled to the same size, even if it is already known that they have the same resolution and should be compared at their current sizes. When one image has bigger width, but the other has bigger height, the images are scaled to the largest alternative. This is demonstrated in Figure: 6.2.

Similar images that have the same resolution generally have (almost) the same size as well. This is demonstrated in Table: 6.1.

Image	Size
Image 1:	$251 * 412$
Image 2:	$251 * 413$
Image 3:	$251 * 413$
Image 4:	$250 * 412$

Table 6.1: Image sizes after normalization

This table shows the sizes of four similar normalized pictures. The originals differ greatly in size, angle, and lighting, but the normalized images are mostly the same, in size as well as angle and lighting.

Chapter 7

Noise

The process of scanning images is a transformation of nearly continuous intensities in the image to discrete gray levels. The consequence of the discretization is that when an image is scanned in several times without changing any parameter, the scanner does not produce the same result. This chapter attacks this problem and introduces two ways to visualize this noise. Additionally, two methods are presented to reduce this noise.

7.1 Discretization noise

As mentioned above noise is added by digitizing images. This noise is composed by each of the two steps in digitizing:

1. The process of digitizing can be described as putting a grid on the image. For the reason, that the images in the database are scanned with 300dpi * 300dpi resolution, the adjustment of the scanner changes and the grid does not start on exact the same position.

2. From each part of the grid, the scanner calculates an intensity value. This value lies between 0 and 255 using 8-bit images. This means, that it is not the exact gray value of the image. It is only an approximation.

The standard method to visualize noise in images is to split the image into its bit-planes. An 8-bit image can be taken apart by the following algorithm:

$$b_7 2^7 + b_6 2^6 + ... + b_1 2^1 + b_0 2^0 \tag{7.1}$$

where bit 7 is the most significant and bit 0 is the least significant bit. Now the 8 1-bit images can be created by showing only the nth bit-plane of the image. This technique is called bit-plane slicing.

Figure 7.1: An 8-bit digitized image

The output is created by using the formulae:

$$v = \begin{cases} L, & \text{if } k_n = 1 \\ 0, & \text{otherwise .} \end{cases} \tag{7.2}$$

The disadvantage using bit-plane slicing is the fact, that small changes in the graylevel are not well taken into consideration. If, for instance, a pixel with the intensity 127 (01111111) is adjacent to a pixel of the intensity 128 (10000000), every bit-plane contains a corresponding 0 to 1 (or 1 to 0) transition. For instance, if the most significant bits of the binary codes for 127 and 128 are different , bit-plane 7 will contain a zero-valued pixel next to a pixel of value 1, creating a 0 to 1 (or 1 to 0) transition at that point. A preferable algorithm to avoid the demonstrated effect is the Gray code. The Gray code can be computed using the following formulae:

$$\begin{aligned} g_i &= b_i \text{ OR } b_{i+1} \qquad 0 \le i \le m-2 \\ g_{n-1} &= b_{n-1} \end{aligned} \tag{7.3}$$

The application of this code gives a Gray code for 127 (11000000) and for 128 (100000000). Fig. 7.5 - Fig. 7.12 show the processed images using bit-plane slicing and Gray code. Due to the used formulas for bit-plane slicing and Gray code the result for bit-plane 7 is the same.

Regarding only the bit-planes created using bit-plane slicing, one could think that the two least significant bits do not store any information of the image since they show no structure. The impression changes if one takes the results of the Gray code into consideration. Even the least significant bit (Fig.:7.5) shows local structures.

7.2 Reduction techniques for sampling noise

Enhancement techniques which are performed on local neighborhoods of input pixels are called spatial operations. Two techniques which are slightly different using spatial masks are presented (see [Gonzalez 92]. These techniques can be used with variant spatial masks. In this thesis the $3 * 3$ mask (Fig.: 7.1) is chosen for description.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figure 7.1: A $3 * 3$ mask

7.2.1 Lowpass spatial filter

The lowpass spatial filter investigated here is a satisfying technique for image enhancement, if only small changes in the output images are desirable. The resulting image is a smoothed image. Using a low pass filter, all coefficents must be positive. The coefficients, which have the value 1 must be scaled by division with 9. Otherwise, the output result would not fit in the given amount of gray levels.

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

Figure 7.3: The used $3 * 3$ lowpass filter

7.2.2 Median filter

The disadvantage of the introduced lowpass filter is that it blurs edges and other sharp detail. A method to achieve noise reduction rather than blurring is the median filter.

The median filter replaces the gray level of every pixel with the median of the neighborhood of this pixel. Consider Figure 7.4 where the mask of the pixel with the value 1 is given. First all values have to be sorted. The median is the fifth value (four values are less than the median, four values are higher than the median).

$$median\{5,2,8,4,1,7,3,9,6\} = median\{1,2,3,4,5,6,7,8,9\} = 5 \qquad (7.4)$$

5	2	8
4	1	7
3	9	6

Figure 7.4 example for a $3 * 3$ mask for value 1

With the end of this chapter the image normalization is completed and the images can be compared. Several comparison methods are described in the next chapter.

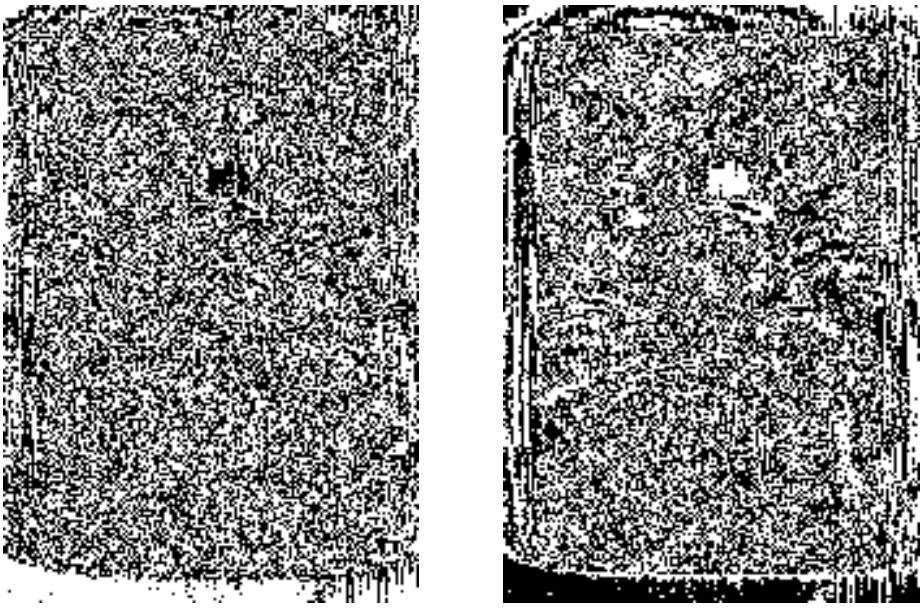

Figure 7.5: The binary (left column) and Gray code (right column) of bit plane 0 of the image in Fig. 7.1 .

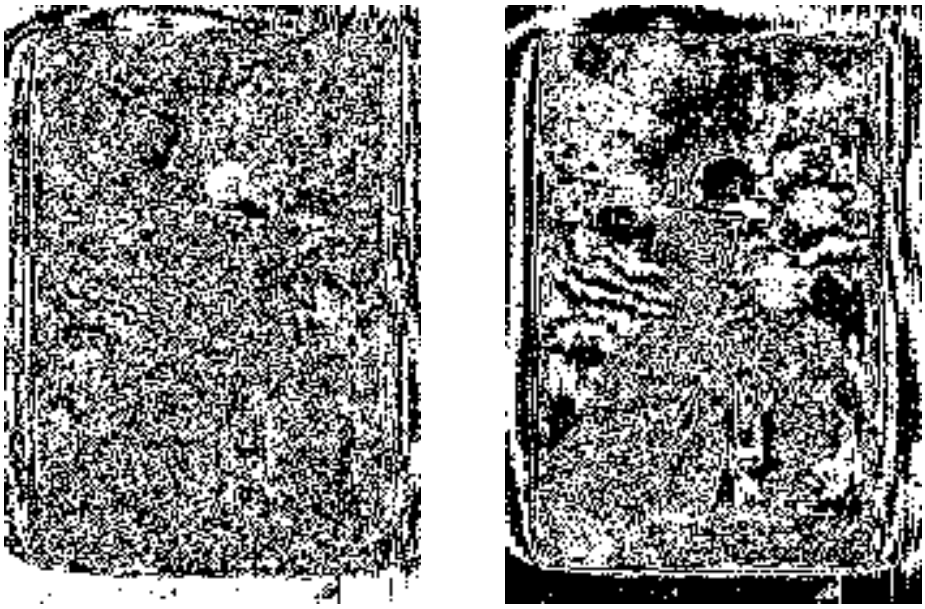

Figure 7.6: The binary (left column) and Gray code (right column) of bit plane 1 of the image in Fig. 7.1 .

Figure 7.7: The binary (left column) and Gray code (right column) of bit plane 2 of the image in Fig. 7.1 .

Figure 7.8: The binary (left column) and Gray code (right column) of bit plane 3 of the image in Fig. 7.1 .

Figure 7.9: The binary (left column) and Gray code (right column) of bit plane 4 of the image in Fig. 7.1 .

Figure 7.10: The binary (left column) and Gray code (right column) of bit plane 5 of the image in Fig. 7.1 .

Figure 7.11: The binary (left column) and Gray code (right column) of bit plane 6 of the image in Fig. 7.1 .

Figure 7.12: Bit plane 7

Chapter 8

Comparison techniques

During the development of the normalization algorithms it became necessary to develop routines that could compare pictures and report on the pixel-by-pixel difference. Another use for a comparison routine is to determine wether two pictures are different or similar. This chapter describes several comparison routines that were implemented. Test results from these algorithms can be found in chapter 9.

8.1 Rotation comparison

The rotation comparison compares every pixel in one image to the corresponding pixel in another image. Rotation comparison was developed to test the amount of noise generated by the various rotation algorithms. In order to do this, it was necessary to make a small change to the rotation algorithms, so that they left blank those parts of an image that were rotated in from outside the original image.
The standard comparison algorithm does not take pairs of pixels into account when one or both of the pixels are blank. This way only pixels that were actually rotated are tested.
The noise is calculated as the sum of all absolute values of all the differences of every pixel pair, ie:

$$noise = \sum |P_1 - P_2| \tag{8.1}$$

with $P_n =$ the color values of corresponding pixels in both images, $P_n \neq 0$.
The output of the algorithm consists of several values:

- The number of pixels that were skipped during comparison (by virtue of one or both being blank).
- The number of pixels that were taken into account during comparison.

- The sum of all absolute values of all differences taken together, called the total difference.
- The total difference divided by the number of pixels that was taken into account, called the average difference.

Also generated is a difference image. Pixels in this image have a color value equal to the absolute value of the difference of the corresponding pixels in the input images.
The difference image makes it easy to spot the distribution of large difference values. During the tests it became apparent from these distributions that often a double rotation (ie. back and forth over the same angle) seemed to leave the image at a slight offset from the original. To further test this phenomenon a routine was developed that calculates a so-called difference vector.

8.2 The difference vector

The difference vector represents the distance and direction in which one source image must be shifted to find the smallest average difference with the other source image. A function was developed to calculate the difference vector.
The algorithm compares pixels in one image with several pixels in the other image. These pixels are distributed as follows:

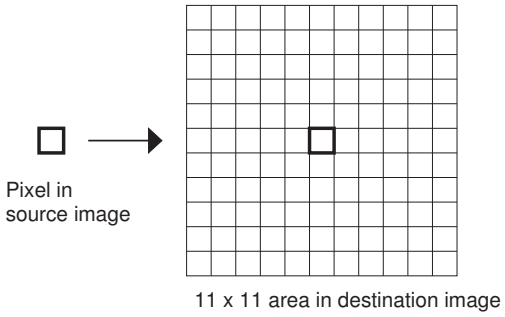

Figure 8.1: One pixel in the source image is compared to several in the destination image

The routine calculates the absolute value of the difference of every selected pixel in the second image with the pixel in the first image. These differences are added to an $11 * 11$ array of total differences. The whole process is repeated for a user-selectable number of pixels from the first image.
The array element with the lowest value represents the best likely difference vector. The standard comparison routine can automatically translate an image over this vector during the comparison process.

Likely candidates have been tested; these tests show that none of the rotations has a systematic translation error.

8.3 Absolute comparison

The comparison algorithms can also be used to identify wether two images are the same or different. If they are to succeed in this they must have two important properties:

- Similar images must be detected as being similar.
- Different images must be detected as being different.

This may seem obvious, but it is easy to create algorithms that satisfy only one of these rules! The first algorithm that was tested for these properties is based on the rotation comparison. Absolute comparison differs from rotation comparison in that absolute comparison does not skip blank pixels. Other than this the routines are completely the same.
Apart from the absolute comparison algorithm two others were tested. These are the nonabsolute comparison and the best neighbor comparison.

8.4 Nonabsolute comparison

Nonabsolute comparison is the same as absolute comparison, but the absolute value is taken over the total difference rather than every different pixel pair.

8.5 Best neighbor comparison

Best neighbor comparison searches, for every pixel P_1 in an input image, an area around the corresponding pixel in order to find the pixel P_2 so that $|P_1 - P_2|$ is as small as possible. The area that is searched is user selectable.
Best neighbor comparison is much less sensitive to small differences in image position, angle, or size, as long as the area that is searched is big enough to contain slightly misaligned corresponding pixels.

Several images have been normalized and compared with all of these methods. Results can be found in chapter 9.

Chapter 10

Evaluation

10.1 Rotation

Method:	Average difference	Speed of operation
primitive:	11.44 (4.49%)	20 secs.
bilinear:	4.73 (1.85%)	44 secs.
shear-based:	28.22 (11.07%)	12 secs.
bilinear shear-based:	28.25 (11.08%)	44 secs.
multiple neighbors:	8.60 (3.37%)	480 secs.

Table 10.1: Average noise generated by rotation

- Rotation with bilinear interpolation is in all cases (except one) better than the other rotations techniques.
- Rotation with bilinear- and cubic interpolation can in some cases be better than just rotation with bilinear interpolation.
- Rotation with bilinear- and cubic interpolation is exceptionally slow.

For these reasons the normalization procedure uses the rotation with bilinear interpolation.

10.2 Border removal

Image	Size
Image 1:	$251 * 412$
Image 2:	$251 * 413$
Image 3:	$251 * 413$
Image 4:	$250 * 412$

Table 10.2: Image sizes after normalization

- After borders are removed, images at the same resolution have almost the same size.

For this reason, images can be compared directly using the $3 * 3$ best neighbor method after the normalization procedure has finished (the best neighbor method compensates for small differences in image size; with a $3 * 3$ area it compensates for one pixel in all directions).

10.3 Lighting noise

Table 9.24 compares the average difference of the four images without processing and after processing. The difference between the unprocessed images is calculated as 0.45 in intensity. This result corresponds to the visualization using bit-plane slicing (chapter 7).
Reducing this noise output images using $3 * 3$ median filtering and $3 * 3$ lowpass spatial filtering were investigated. The reduction of the average difference to 0.38 in intensity in median filtering is not a significant improvement. The reduction to 0.24 in lowpass filtering is better, but it has the disadvantage of loosing information because of smoothing.
Because of these reasons none of the filters is recommended for implementation in this project.

10.4 Scaling

- Scaling to larger size adds less noise than scaling to smaller size.
- Scaling to an integer multiple of the current size adds no noise to the image.

Because of this the normalization scales to the largest possible size, when necessary. Several likely resolution scalings (100dpi to 400dpi, 200dpi to 400dpi) can be done without adding noise.

New size	Average difference
70%	5.78 (2.27%)
100%	0.00 (0.00%)
130%	3.80 (1.49%)
200%	0.00 (0.00%)

Table 10.3: Average noise generated by scaling

10.5 Comparison

Method	Similar images	Different images
Absolute	7.43 (2.91%)	66.99 (26.27%)
Nonabsolute	0.70 (0.27%)	1.13 (0.44%)
$3 * 3$ best neighbor	2.22 (0.87%)	47.14 (18.49%)
$5 * 5$ best neighbor	1.52 (0.60%)	42.81 (16.79%)

Table 10.4: Average differences found by the comparison methods

- Nonabsolute comparison does not recognize different images as being different.
- The other methods are all capable of recognizing different images as being different and similar images as being similar.

$3 * 3$ best neighbor comparison is recommended for image comparison after normalization for the following reasons:

- There is a big enough difference in average difference between comparisons done with similar images and comparisons done with different images.
- This method compensates for the 1 pixel difference in image size which may be left after normalization.

10.6 Discretization noise

Comparing the results of histogram equalization (Tables 9.8 + 9.11) with contrast stretching(Table 9.9 + 9.12), histogram equalization results in a difference smaller than 1% in the gray level range from -1 to +2, whereas contrast stretching has a difference smaller than 1% in a smaller range.

According to this result and the high dependance from contrast stretching on

the appearance of the histogram (see chapter 5), histogram equalization is recommended for this image normalization implementation.

10.7 The normalization procedure

The normalization procedure consists of the following steps:

Detect angle: Angle detection must take place before the angle can be corrected.

If necessary, apply rotation with bilinear interpolation: This rotation method adds the least noise of all tested methods. Rotation must take place before the borders can be removed.

Remove borders: Borders are generally much lighter than the actual image. They must be removed before histogram equalization can be applied.

Equalize the histogram: Histogram equalization removes most differences in lighting conditions.

Scale to new resolution: If necessary, the image can be scaled to a new resolution. The new resolution depends on the images that will be compared with the image that is normalized; for this reason scaling is not part of the current normalization procedure.

When an image is scanned in twice with different angle, borders, position, lighting, and resolution the normalization procedure (together with $3 * 3$ best neighbor comparison) demonstrates that both scans are similar with less than 1% difference in intensity, and less than 1 pixel in size. Thus it is possible to use normalization to find copies of images in a database.

Chapter 11

Ingres/Windows 4GL

11.1 Overview

This chapter briefly describes the Ingres/Windows 4GL toolkit [Ingres 92] and emphasizes the advantages and restrictions of this tool.
Ingres/Windows4GL (4GL = Fourth Generation Language) is an application editor to create and edit Windows4GL applications. It is intended to support experienced developers to create window-based applications.

11.2 Advantages

The packages includes tools to support every step of application development, from the initial design and creation of prototypes to implementation, testing, configuration management and deployment. After comparison with the Motif toolkit [X Window 90] and TCL/TK [TCL/Tk 94] Ingres/Windows4GL was chosen as the interface development tool for the Photobase project for the following reasons:

- The generated and already known information of the photographs will be stored in a database. Ingres/Windows4GL is the only program which offers easy database access. (Ingres is a well known company in database applications). TCL/TK and MOTIF do not have built-in database functions.
- TCL/TK does not support images in 8-bit graylevel format. It can only load black and white (1-bit) images.
- User interface code written in Ingres/Windows4GL is much more compact and readable than similar code written for the Motif toolkit.
- For future programmers on this project the basics to add procedures to Ingres/Windows4GL can be understood after a relatively short time of acquaintance.

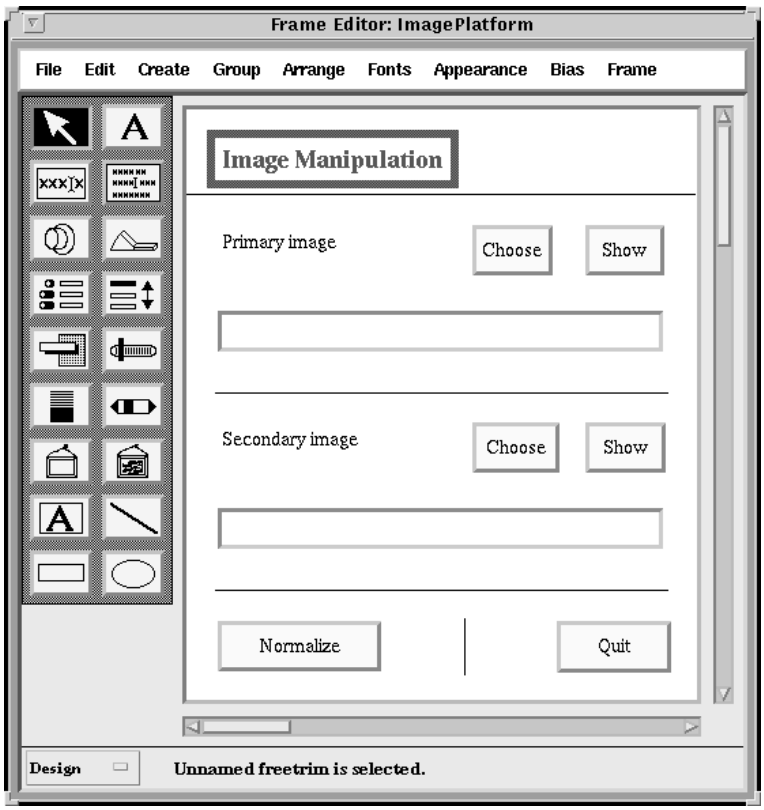

Figure 11.1: The working platform

- Ingres/Windows4GL offers some support for working with multiple programmers on one project.
- Ingres/Windows4GL can be used on Unix, VMS and DOS platforms.

11.3 Restrictions

Regarding the state of the art in modern user interfaces, Ingres/Windows4GL has the following limitations:

- A help menu on the right side of the menu bar is not possible.
- The complete generation of the implemented code including the code of the windows in an author readable format, for instance as the appendix of a thesis, is not possible.
- Opening and redrawing windows is very slow, even on a fast computer.
- Ingres/Windows4GL does not use procedures calls, but so called userevents. Userevents are executed in an asynchronous, out-of-order fashion, which can be limiting in many situations.

11.4 Linking with C

Ingres has the ability to use a link library of external functions at run-time. A special utility, make3gllib, is included to create these object files. The interface between Ingres and the external procedures is not very smooth:

- Every called C function must specifically be registered with Ingres.
- Ingres can only link the external functions when it is started. If a new external function library is generated Ingres must be quitted and restarted.
- Only one external function library can be used at any time. This means that a project with multiple authors must somehow use a shared directory for C source. They must manage (and limit) access to this directory themselves.
- Ingres/Windows4GL can call C, but C cannot call Ingres/Windows4GL.

After introducing the user interface development tool chapter 13 shows the result of its application.

Chapter 12

Implementation of the user interface

This section describes the user interface of the Photobase program. It consists of several windows, each of which are described in detail.

12.1 The Image Processing Platform

After starting the Photobase program, the main window opens (Fig.: 13.1). From here it is possible to select a primary or secondary image, and operate on them. Operations that require only one input image, such as histogram equalization, act on the primary image. The secondary image is used for operations that need two input images, such as comparison.
The primary or secondary image can be chosen directly by clicking on the corresponding **Choose** button. A file requester (Fig.: 13.2) opens and the required image can be selected. The name of the image appears in the text widget and the attached **Load** button is no longer ghosted. It is now possible to show this image, or process it directly.

To normalize an image, it must be designated as primary image, after which the **Normalize** button must be pressed. This starts the image normalization process:

- The angle of the image is detected and if necessary corrected.
- The borders are cut off.
- The image is histogram equalized.
- The image is rescaled if necessary.

The processed image will be shown in a separate window.
Quitting the application can easily be done by pressing the **Quit** button.

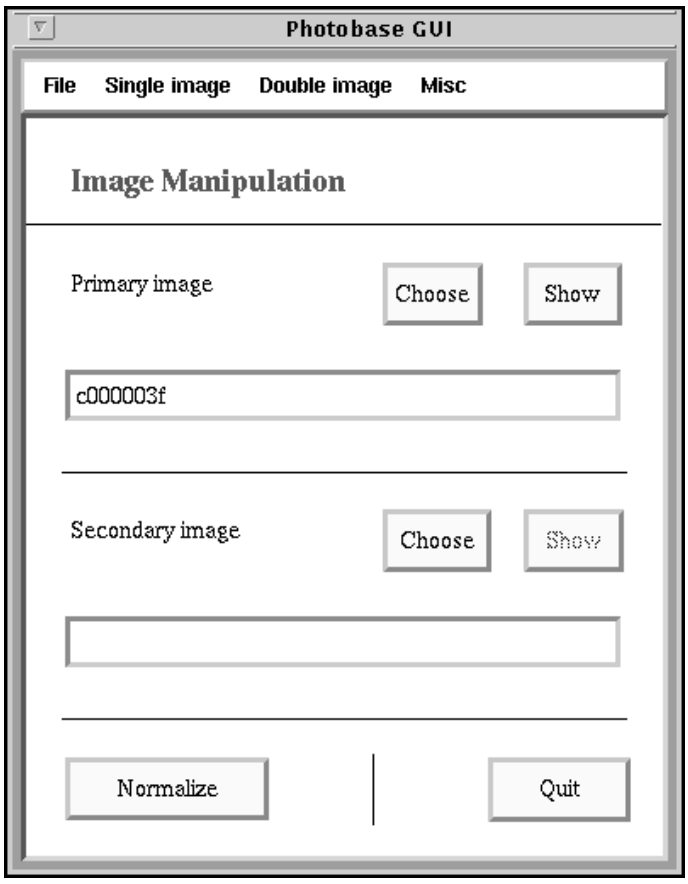

Figure 12.1: The Photobase window

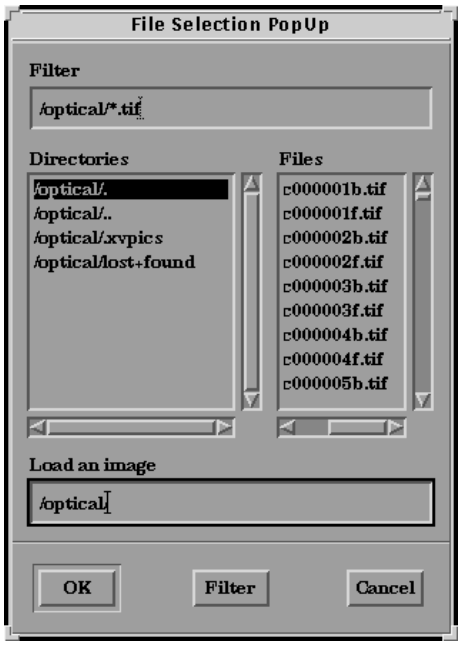

Figure 12.2: The file requester

The following options can be found in the menu bar of the Image Platform window: (If no comment is added to the button, a detailed description appears in other chapters.) The menu options offer acces to all the techniques tested for the normalization procedure and enables the user to execute the normalization steps one by one.

- **menu "File"**
  Option
  - Load image
    (= Pops up a file requester and loads and shows the selected image immediately in the image window. The same result can be reached by pushing **Choose** and **Load**.)
  - Preferences
    (= Pops up the Preferences window.)
  - Credits
    (= Presents information on version and author)
  - Quit
    (= Closes the application)

  The following options all perform some kind of operation on the primary image.

- **menu "Single image"**
  Option
  - Detect angle
  - Rotate
  - Set method
    - Primitive
    - Bilinear
    - Shearbase
    - Shearbased bilinear
    - $4x4$ bilinear
  - Set angle
    - -15
    - -10
    - -5
    - +5
    - +10
    - +15
    - User defined
  - Scale
    - to picture window
    - User defined
  - Cut borders
  - Histogram
    (= Opens the histogram window.)
  - Histogram equalization
  - Contrast stretching
  - Median filter
    - 3 * 3
    - 5 * 5
    - 7 * 7
  - Low pass filter
    - 3 * 3
    - 5 * 5
    - 7 * 7
  - Bit operation
    - Gray code
      - Bit plane 0 (LSB)

        - Bit plane 1
        - Bit plane 2
        - Bit plane 3
        - Bit plane 4
        - Bit plane 5
        - Bit plane 6
        - Bit plane 7 (MSB)
      - Bit plane slicing
        - Bit plane 0 (LSB)
        - Bit plane 1
        - Bit plane 2
        - Bit plane 3
        - Bit plane 4
        - Bit plane 5
        - Bit plane 6
        - Bit plane 7 (MSB)
    - Threshold
      - Automatic
      - Manual

  The following options perform an operation on both the primary and secondary image.

- **menu "Double image"**
  Option
  - Difference image
    (= creates a difference image)
  - Difference
    - Absolute
    - Non absolute
    - Best neighbor
  - Super impose
  - Calculate diff. vector

- **menu "misc"**
  Option
  - History
    (= Opens the history window.)

12.2 The Image window

Images are shown on screen in Image windows. Each image has it's own window, and one image can be visible multiple times. When the window is resized the image is automatically scaled as well, if necessary.
The visible part of the image can be altered by moving the horizontal and vertical scroll bars.

- **menu "File"**
  Option
  - Save
    (= Saves the image as a TIFF 6.0 file.)
  - Close
    (= Closes the window and deletes the temporary sun rasterfile of this image. The original is not deleted.)

- **menu "Select"**
  Option
  - as primary
    (= Selects the image as primary. Its name is shown in the primary text widget in the main window. This selection is important for additional processing of the image.
  - as secondary
    (= Selects the image as secondary. Its name is shown in the secondary text widget in the main window.

- **menu "Scale"**
  Option
  - No scale
    (= The picture is shown in its generated or scanned size, without any scaling.)
  - scale to height
    (= Set as default. The image is scaled to the height of the image widget.)
  - scale to width
    (= The image is scaled to the width of the image widget.)

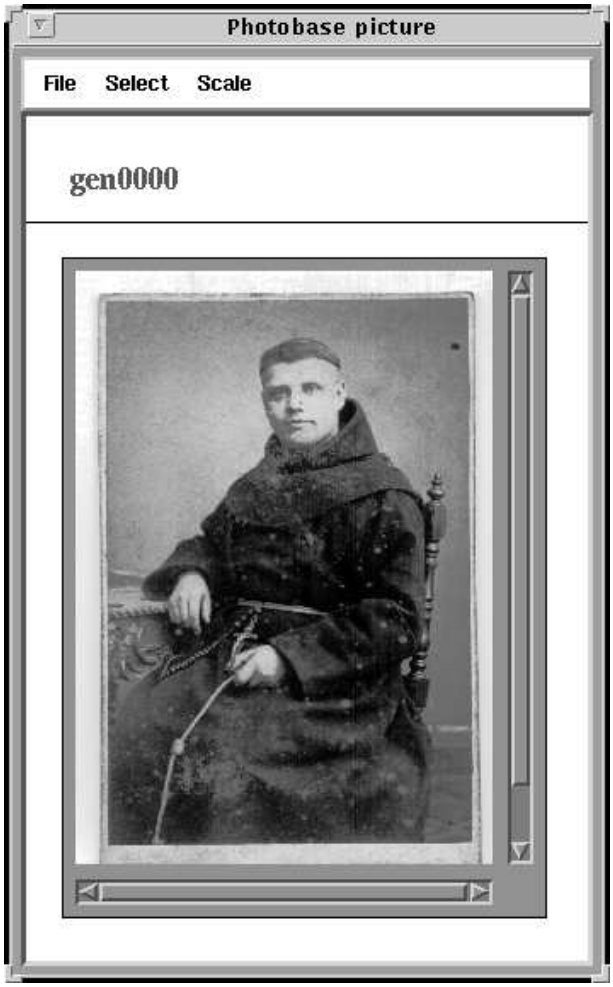

Figure 12.3: The image window

12.3 The Preferences window

It is possible to change the default path for loading and saving images from the preferences window. A new path is not used until the okay button is clicked.

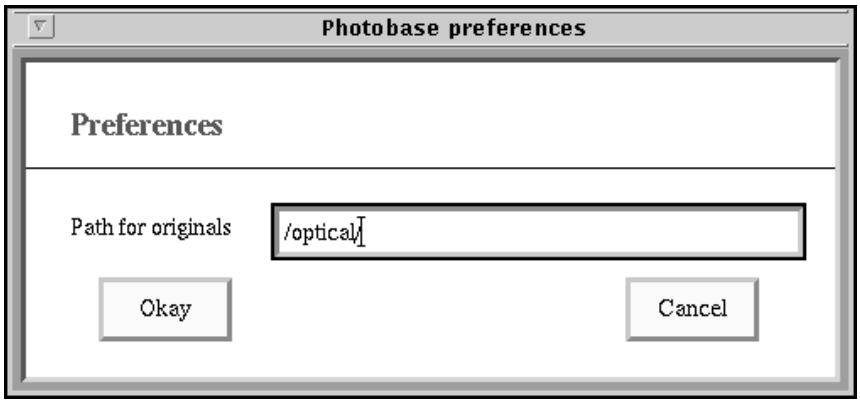

Figure 12.4: The preferences window

12.4 The History window

The history window shows a list of previously loaded or generated images. An image can be selected from the history window by clicking its name, and selecting the **Show** button. Through this window the user can keep track of images even after they are no longer visible on the screen.

Figure 12.5: The history window

12.5 The Histogram window

The histogram of the image contains the information about the distribution of the grayvalues or intensities of an image, respectively. There are 256 different intensities.
The histogram window can be selected from the **"single" menu**, option **histogram**. This selection opens the histogram window and shows the histogram of the image which is selected as primary on the main window.

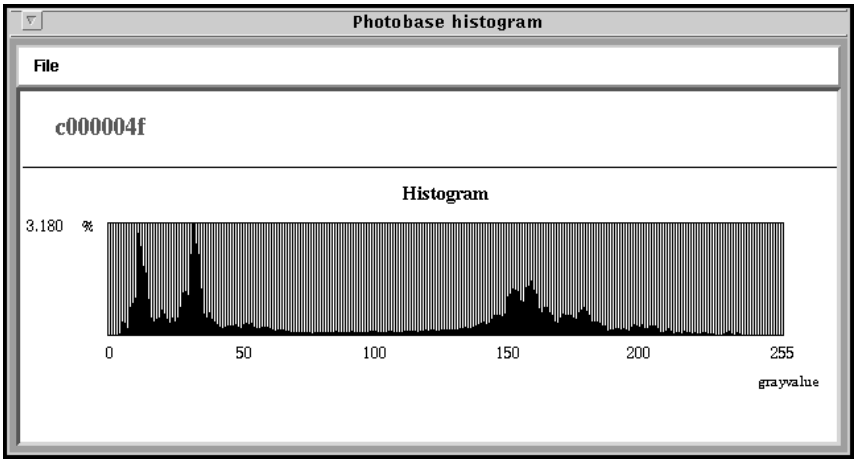

Figure 12.6: The histogram window

The next chapter deals with future enhancement of the project.

Chapter 13

Future enhancement of Photobase

13.1 Example of adding an image manipulation

This chapter gives an example of adding an image manipulation to Photobase. The manipulation that will be added is a super impose function. Super imposing is a technique where corresponding pixels of two pictures are added together and divided by two to generate a pixel in the destination image. This can be useful to determine visually how much two images overlap.

Step 1: add the super impose function to the C source

After writing the super impose function it must be added to the C source. Generally, manipulations that require one source image are put in single_proc.c while manipulations that require two source images are put in double_proc.c. Note that this is just a matter of organization; there is no real need to add new functions to these files. Because SuperImpose() requires two source images it is placed in double_proc.c. A header for SuperImpose is added to double_proc.h.

Step 2: call SuperImpose() from Process()

Process() (in picture.c) is used to dispatch Ingres calls to the rest of the C code. All image manipulations are activated through a call to Process(). Process() takes care of loading the required images, scaling them to the same size if necessary, and saving the results. Process() determines what image manipulation to call through a number given by Ingres. SuperImpose() must be given such a number. Numbers in the range 0..99 are reserved for operations that manipulate one source image, while numbers in the range 100..200 are reserved for operations that require two source images. Because SuperImpose() uses two source images it is given the (so far unused) number 105. A call to SuperImpose() is added in the large switch statement:

```
...
case 105:
     ErrCode = SuperImpose ();
     break;
...
```

Step 3: add a menu entry to the image platform frame

To make the manipulation accessible to the user a menu entry must be added to the image platform frame. There are two menus with image manipulation, 'single image' and 'double image'. SuperImpose() is added to 'double image'. The Windows 4GL code for the operation looks like this:

```
on click menu.double.superimpose =
     BEGIN
          curframe.senduserevent (eventname = 'process',
          messageinteger = 105);
     END;
```

The userevent Process calls the C routine Process(), places the resulting image on screen and handles possible error conditions. The number 105 must be the same number that was used in Process().

13.2 Source for SuperImpose()

Most initialization is done automatically, but some things must be done by SuperImpose(). The following source shows what these things are.

```
int SuperImpose (void)
{    LONG x, y, Width, Height;
     UBYTE ByteVal1, ByteVal2;

     /* First, the output image must be initialized */

     Width = idata[PRIMARY].width;
     Height = idata[PRIMARY].length;

     idata[OUTPUT].width = Width;
     idata[OUTPUT].length = Height;
     if (!init_pic(OUTPUT)) return ERR_OUT_OF_MEMORY;

     /* If an error occurs an error code is returned.  */
     /* Error codes are defined in defines.h           */

     for (y=0; y<Height; y++) {
          for (x=0; x<Width; x++) {
```

```
          /* Read image data from input images */

               ByteVal1 = idata[PRIMARY].image[y][x];
               ByteVal2 = idata[SECONDARY].image[y][x];

               /* Write resulting data to output image */

               idata[OUTPUT].image[y][x] =
                          (ByteVal1+ByteVal2) >> 1;
     }
     if (y % UPDATEGAP == 0)
         fprintf (stderr, "Calculating line %d (%d%%)\n"
                                       , y, (100*y)/Height);
     }

     /* No error occurred, so return ERR_NONE */

     return ERR_NONE;
}
```

Operations that do not return an image must return ERR_DONT_SAVE, which causes Ingres to skip the image loading code.

13.3 Handing values from Ingres to C

In some cases it is necessary to set values in the C code prior to calling Process(). This can be done with a call to C_SetField(). C_SetField() takes four arguments: an identifier, an integer, a float, and a character string. The identifier determines which variable in the C source is given a value. This variable can be integer, float, or a character string, and the argument of the same type is used to set it. Currently all values that can be set by C_SetField() are members of the dfield datastructure. This structure is globally accessible in picture.c, single_proc.c and double_proc.c. As an example of the use of C_SetField, this is the Ingres code for Threshold():

```
on click menu.single.threshold.value =
    BEGIN
        ThresholdPercentage = menu.single.threshold.value;
        Result = C_SetField (3, 0, ThresholdPercentage, ' ');
        curframe.senduserevent (eventname = 'process',
                                        messageinteger = 8);
    END;
```

13.4 Suggestions for further research

The current normalization process stop after the angle of the image was corrected, the border was cut off, and the histogram was equalized. However, there are other steps that could be taken as part of a normalization process. An example of such a step is removing the outer frame that surrounds the actual photo. This outer frame can mostly be removed the same way that borders are currently removed: by checking for differences in intensity. It is recommended that this step is taken after the borders are cut off, but before the histogram is equalized. This is because histogram equalization can have a profound impact on the intensities of the individual pixels. If the intensities of the pixels in the borders change too much the method of removing borders described here (which relies on the border having a uniform color) will no longer work. Extra attention must be given to removing the lower part of the outer frame, as it is usually embellished with texts and the logo of the photographer.
A possible refinement of the border removal algorithm is the following: the current algorithm is susceptible to noise in the border, even a single stray pixel of differing intensity (to the rest of the border) can disrupt the correct operation of the border removal. It may be possible to find such pixels (and groups of such pixels) and remove them before the borders themselves are removed.
Another refinement could be made to the comparison algorithms. The current algorithm scans the entire image for differences, but a future algorithm could take into account that more interesting information is generally found closer to the center of the image. A comparison algorithm that assigns a weight to the differences in intensity based on the difference to the center of the image (or some other point) may be more succesful in identifying copies of images. See Figure: 14.1 for an example of a weight distribution.

Figure 13.1: A possible weight distribution

Finally, the current normalization procedures all operate on 8-bit data. Several normalization routines can be made to operate on 24-bit data by splitting the 24-bit image into its red, green, and blue components (each of which are 8 bits in size) and normalizing these component images separately. This will not work for angle detection, border removal, and possibly histogram equalization. In each of these cases the routines must be altered to directly operate on 24-bit data.

13.5 Other applications for the normalization procedure

The current normalization procedure has some limitations that prohibit use in certain situations. However, several other uses can be found for the procedures described in this report.

- **Paintings:** Paintings meet the requirements set for the normalization procedure. A database that can identify unknown paintings can be created if the normalization procedures are changed to operate on 24-bit data, as is described above.
- **Stamps:** The normalization procedures seem cut out for identifying stamps. Because stamps generally have irregular borders it may be necessary to weaken the identifying conditions for lines in the angle detection routine. Also, stamps exist that are not rectangular.
- **Telephone cards:** Telephone cards are quickly becoming a collectors item; an on-line database of telephone cards could be an invaluable aid in card auctions.
- **ID cards:** A database with ID cards connected to a camera could be equipped with the normalization procedure to allow a completely optical identification of cards. Other problems that must be solved before this becomes a possibility: ID cards move in 3D space and can be rotated over the X- and Y-axis (which must be compensated for), and the card must be identified in a screen that can contain many other details.

Bibliography

[Brown 92] L. G. Brown, "A Survey of Image Registration Techniques", ACM Computing Surveys, vol. 24, no. 4, pp. 325-376, 1992

[Fang 93] N. Fang and M.-C. Cheng, "An automatic crossover point selection technique for image enhancement using fuzzy sets," Pattern Recognition Letters, vol. 14, no. 5, pp. 397-406, 1993

[Gonzalez 92] R. C. Gonzalez and R. C. Woods, Digital Image Processing, Addison-Wesley, USA, ISBN 0-201-50803-6, 1992

[Hill 90] Francis S. Hill, Computer Graphics, Macmillan Publishing Company, USA, ISBN 0-02-946185-5, 1990

[Jain 89] A. K. Jain, Fundamentals of Digital Image Processing, Prentice-Hall, Englewood Cliffs, NJ, ISBN 0-13-332578-4, 1989

[Ingres 92] Ingres/Windows4GL, -Application Editor User's Guide-, -Programming Guide-, -Language Reference Manual-, Ingres Corporation, California 94501, 1992

[Koelma 94] D. Koelma and A. Smeulders, "A visual programming interface for an image processing environment," Pattern Recognition Letters, vol. 15, no. 11, pp. 1099-1109, 1994

[Leu 92] J.-G. Leu, "Image Contrast Enhancement Based on the Intensities of Edge Pixels," CVGIP: Graph. Models Image Proc., vol. 54, no. 6, pp. 497-506, 1992

[Lindenbaum 94] M. Lindenbaum, M. Fischer and A. Bruckstein, "On Gabor's Contribution to Image Enhancement," Pattern Recognition, vol. 27, no. 1, pp. 1-8, 1994

[Ousterhout 94] J. K. Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley, Masssuchettes, ISBN 0-201-63337-X, 1994

[Pratt 78] W. K. Pratt, Digital image processing, John Wiley & Sons, Inc., USA, ISBN 0-471-01888-0, 1978

[TIFF 92] TIFF Revision 6.0, Aldus Corporation, Seattle, WA 981004-2871, 1992

[Yan 93] Hong Yan, Skew Correction of Document Images Using Interline Cross-Correlation, CVGIP: Graphical Models and Image Processing, Volume 55, no. 6, pp 538-543, 1993

[X Window 90] The X Window System Vol. 0 - 7, O'Reilly & Asscciates Inc., Sebastopol, CA 95472, ISBN 0-937175-13-7, 1990

Appendix A

The TIFF file format

To describe image data that come frome scanners, one has to face the problem which file format is the best to store the processed images. The decision to store the digitized images in the TIFF 6.0 file format has several below mentioned reasons.

- The TIFF 6.0 header includes the image size and the original scanning resolution.
- TIFF is a public domain standard. A library which reads and writes TIFF files can be downloaded from `ftp.sgi.com (graphics/tiff/v3.3beta.src.tar.Z)`.
- A complete description of the TIFF [TIFF 92] file format and example applications are offered.
- TIFF offers the possibility to store graylevel as well as colour images.
- TIFF can be used with different scanners, an advantage, because this project operates with the Epson and the Hewlett-Packard II cx scanner.
- For future extension, TIFF includes several compression schemes for saving disk space.

Appendix B

The Sun Raster format

The Ingres system can only read pictures that are stored in the Sun Raster format. The format consists of three chunks, with the following specifications:

The header

The header consists of 8 32-bit words, with the following meaning:

word 1: The identifier 0x59A66A95.

word 2: The width of the image in pixels, rounded to the nearest even number.

word 3: the height of the file in pixels.

word 4: the depth of the file, expressed in bits per pixel.

word 5: the length of the image chunk in bytes, including possible padding bytes.

word 6: specifies the type of palette chunk in this image, must be 1 for Ingres.

word 7: specifies the type of image chunk in this image, must be 1 for Ingres.

word 8: the length of the palette chunk, in bytes.

The palette

This chunk is best described with the corresponding C structure:

```
struct Palette {
     UBYTE Red   [NumEntries];
     UBYTE Green [NumEntries];
     UBYTE Blue  [NumEntries];
};
```

UBYTE is an 8-bit unsigned number. NumEntries is the number of palette entries in this image. If there are less than 256 colors than the palette specifies the colors for the first NumEntries pixel values. NumEntries is calculated from the length of the palette chunk by dividing it by three.

The image data

Every byte in this chunk describes the color of a single pixel. The first byte describes the palette number of the upper left pixel, the second byte the palette number of the pixel to the right of it, and so on, until a line is filled. If the image width is odd there is a padding byte before the byte that describes the first pixel of the second line.

Appendix C

Photobase style

The user interface

An effort was made to keep the Photobase user interface visually consistent. The following rules were used while developing this interface:

- The interface looks best on the Aurora (Sun) workstation. On other machines the colors may differ, and window layout changes slightly in different environments. If new frames are added they must be normalized on the Aurora.
- In the top-left corner of every frame there is title which describes what is found in the frame. The title is printed in 18 pts. Times Roman characters. The color of the title is bright red.
- Below the title there is a thin black line which stretches across the entire width of the frame.
- All other text is in 14 pts. Times Roman, colored black.
- Buttons have a pale yellow background.
- Scrolling lists and images have a medium blue background.
- Viewports have as few lines and borders as possible. The reason is that generally the images already have several frames within each other, causing visual confusion.
- Everything is aligned to the grid, both top-left and bottom- right corners.

Other than this, nothing was changed from the Ingres default settings.

Programming style

The code also follows certain style guides, though less rigidly so than the user interface. The following style rules apply to the Ingres code:

- Names of frames, C procedures, and global variables start with "F_", "C_", and "V_" respectively.
- Names of buttons and other 'clickable' interface parts start with "gad_".
- Code for menu entries is kept locally with the menu entry. Code for buttons is either kept locally or globally with the frame. Userevent code is kept globally.

Chapter 1

Introduction

1.1 Motivation

In the last half of the 19th century people commonly went to a photographic studio for portraits. Photography was still in its infancy, resulting in black-and-white images and long exposure times. Because people generally do not like to sit still for minutes at a time, they usually ordered a dozen prints from one negative. These copies were spread around when the opportunity presented itself.
Nowadays many of these portraits can still be found, in photo albums and in public and private archives. The pictures, even when they were originally printed from one negative, may differ greatly. Some have been exposed to the sun for years. Some have the name of the person in the picture written on them. Some have been cut up or reframed.
Often it is no longer clear who the person in the picture is, and people spend considerable time searching for the names of those photographed. Because there is such a large of number of pictures around (collections known to the author number over 50,000 photos in the Netherlands alone) this is a very hard task for a human being.
The program written during this project, Photobase, attempts to answer questions like "Who is this man", by comparing an unknown person to a database of known persons and finding matching portraits. The goals of the program are the following:

- Search for direct copies of the input image in the database.
- Search for other photographs of the same person in the database.
- If a matching image is found, show information about these images. This information also applies to the input image.

Comparing images is a difficult problem. The comparison program must compensate for many factors: image quality, image orientation, image scale, etc.

The first step in comparing two images is removing as many of these factors as possible, which is a process referred to as normalization.

This report describes the initial implementation of the Photobase system. In the inital implementation much attention was given to normalization techniques and towards building a good user interface. Primitive comparison techniques were also implemented.

1.2 The degrees of freedom

Images must be scanned before they can be processed. The scanning process introduces various degrees of freedom, each of which must be compensated for before images can be successfully compared. Degrees of freedom caused by the scanning process include:

- **Rotation**
  Often an image is scanned in at a slight angle. It is possible to detect this angle and compensate for it by digitally rotating the image. Several rotation algorithms were implemented and tested.

- **Borders**
  Not every part of a scanned-in image is important. Especially the white background around the image is not significant for the image comparison process. Therefore, during the second normalization step this border is removed.

- **Lighting**
  Images can be scanned in at various lighting settings. In addition, the images themselves can be darker or lighter.

- **Scanner noise**
  The scanning process adds a small amount of noise to the image. Several methods to visualize and remove this noise were implemented and tested.

- **Resolution normalization**
  Images are often at a different resolution. Before comparison is possible the images must be rescaled. A scaling algorithm was developed that causes very little noise.

1.3 Input protocol

The implemented normalization techniques try to cope with many different degrees of freedom for the input image. Some restrictions remain, though:

- The input image must be in TIFF 6.0 format.

- When the input image is scanned a border must be left open around the entire image.
- The border around the image must be colored white or nearly white.
- The angle detection algorithm assumes that the image is contained in a rectangular box.
- It is assumed that no parts of the original image were cut off. (Sub-image searches have not been implemented yet.)

In the next chapters the image normalization techniques are presented.

Chapter 9

Results

9.1 Rotation

It is possible to measure the noise generated by a rotation by rotating over a certain (randomly chosen) angle and rotating back over that same angle, and comparing the resulting image with the original. The comparison technique used is described in chapter 8 under 'Rotation comparison'. The angle used for this test is randomly chosen as 15 degrees.
Table 9.1 shows the result of this procedure executed for four different images and all methods. The numbers in the table represent the average difference per pixel in gray level. Table 9.2 shows these same numbers as a percentage.

Method:	Image 1	Image 2	Image 3	Image 4
primitive:	11.25	12.79	10.80	10.90
bilinear:	4.90	5.47	4.62	3.93
shear-based:	24.90	31.07	24.05	32.21
bilinear shear-based:	24.95	30.88	24.70	32.47
multiple neighbors:	11.85	12.54	6.87	3.14

Table 9.1: Average difference in gray level after rotation

Method:	Image 1	Image 2	Image 3	Image 4
primitive:	4.41%	5.02%	4.23%	4.27%
bilinear:	1.92%	2.15%	1.81%	1.54%
shear-based:	9.76%	12.18%	9.43%	12.63%
bilinear shear-based:	9.78%	12.11%	9.69%	12.73%
multiple neighbors:	4.65%	4.91%	2.69%	1.23%

Table 9.2: Average difference expressed as a percentage after rotation

It is important to note the great difference in execution time between the first four methods and the last one. Rotation with multiple neighbors is much slower than the other methods, as can be seen in Table 9.3. The machine used for measuring has a 25MHz 68030 and operated under a stable taskload. The images have approximately 40000 pixels.

Method:	Image 1	Image 2	Image 3	Image 4
primitive:	20	20	19	20
bilinear:	45	43	44	45
shear-based:	12	11	11	12
bilinear shear-based:	44	43	43	45
multiple neighbors:	487	469	472	490

Table 9.3: Execution times for rotations in seconds

From Table 9.1 and Table 9.3 it becomes clear that rotation with bilinear interpolation is the best available technique. For this reason this method was tested further.

Extra tests performed on the rotation with bilinear interpolation

- The noise added to an image steadily increases while the image is being rotated back and forth multiple times (over an angle of 15 degrees):

Number of rotations	Average difference
1	3.33 (1.31%)
2	5.35 (2.10%)
3	6.84 (2.68%)
4	8.07 (3.16%)

Table 9.4: Noise increases during multiple rotations

- Noise is similar for all angles:

Angle	Avg. diff.
1 degree	3.29 (1.29%)
2 degrees	3.19 (1.25%)
3 degrees	3.17 (1.24%)
4 degrees	3.24 (1.27%)

Table 9.5: Noise is similar for all angles

- Rotation over 0 degrees leaves the image untouched.

9.2 Lighting noise

To simulate the normal noise of lighting in images, they are scanned with different levels of lighting. The Epson scanner makes seven different settings of lighting available (Fig. 9.1-Fig.9.7).
A measure for the difference of the created images is the result of a pixel by pixel comparison. Table 9.7 and Table 9.10 shows the results in absolute and percentage values.
Afterwards, contrast stretching and histogram equalization (Fig.9.8-Fig.9.14) were used to produce equalized images.

9.3 Scaling

It is possible to measure the noise generated by scaling, by scaling an image to a new size and scaling it back to the old size, and comparing the results with the original. The comparison technique used is the absolute comparison, described in chapter 8. Table 9.6 shows the average difference in gray level and the same number as a percentage for several pictures.

New size	Image 1	Image 2	Image 3	Image 4
70%	6.21 (2.44%)	6.57 (2.58%)	5.08 (1.99%)	5.24 (2.05%)
100%	0.00 (0.00%)	0.00 (0.00%)	0.00 (0.00%)	0.00 (0.00%)
130%	3.90 (1.53%)	4.29 (1.68%)	3.48 (1.36%)	3.54 (1.39%)
200%	0.00 (0.00%)	0.00 (0.00%)	0.00 (0.00%)	0.00 (0.00%)

Table 9.6: Noise generated by scaling

Images$\mathcal{A}-\mathcal{D}$	(-2)	(-1)	(0)	(+1)	(+2)	(+3)
(-3)	18.537	47.17	58.31	74.21	97.76	129.15
(-2)	-	28.64	39.77	55.68	79.23	110.61
(-1)	-	-	11.14	27.04	51.33	75.01
(0)	-	-	-	15.89	39.45	70.84
(+1)	-	-	-	-	23.55	54.94
(+2)	-	-	-	-	-	31.39

Table 9.7: Scanning with different lighting (absolute values)

Images $\mathcal{A}-\mathcal{D}$	(-2)	(-1)	(0)	(+1)	(+2)	(+3)
(-3)	51.41	65.09	65.28	65.56	66.38	72.64
(-2)	-	15.10	15.15	15.27	16.06	22.23
(-1)	-	-	1.49	1.57	2.32	8.66
(0)	-	-	-	1.49	2.22	8.55
(+1)	-	-	-	-	2.20	8.46
(+2)	-	-	-	-	-	7.60

Table 9.8: Histogram equalization (absolute values)

Images $\mathcal{A}-\mathcal{D}$	(-2)	(-1)	(0)	(+1)	(+2)	(+3)
(-3)	26.92	56.16	55.15	55.85	71.19	100.97
(-2)	-	29.25	28.24	28.94	44.28	74.06
(-1)	-	-	1.26	0.83	15.11	44.83
(0)	-	-	-	1.01	16.05	45.83
(+1)	-	-	-	-	15.39	45.13
(+2)	-	-	-	-	-	31.29

Table 9.9: Contrast stretching (absolute values)

Images $\mathcal{A} - \mathcal{D}$	(-2)	(-1)	(0)	(+1)	(+2)	(+3)
(-3)	7.24%	18.43%	22.78%	28.99%	38.19%	50.45%
(-2)	-	11.19%	15.54%	21.75%	30.95%	43.21%
(-1)	-	-	4.35%	10.56%	20.05%	29.30%
(0)	-	-	-	6.21%	15.14%	27.67%
(+1)	-	-	-	-	9.2%	21.46%
(+2)	-	-	-	-	-	12.26%

Table 9.10: Scanning with different lighting (percent)

Images $\mathcal{A} - \mathcal{D}$	(-2)	(-1)	(0)	(+1)	(+2)	(+3)
(-3)	20.08%	25.43%	25.50%	25.61%	25.93%	28.37%
(-2)	-	5.90%	5.92%	5.97%	6.28%	8.69%
(-1)	-	-	0.58%	0.62%	0.91%	3.38%
(0)	-	-	-	0.58%	0.86%	3.34%
(+1)	-	-	-	-	0.86%	3.31%
(+2)	-	-	-	-	-	2.97%

Table 9.11: Histogram equalization (percent)

Images $\mathcal{A} - \mathcal{D}$	(-2)	(-1)	(0)	(+1)	(+2)	(+3)
(-3)	10.51%	21.94%	21.54%	21.81%	27.81%	39.44%
(-2)	-	11.43%	11.03%	11.30%	17.29%	28.93%
(-1)	-	-	0.49%	0.33%	5.90%	17.51%
(0)	-	-	-	0.39%	6.27%	17.90%
(+1)	-	-	-	-	6.01%	17.63%
(+2)	-	-	-	-	-	12.22%

Table 9.12: Contrast stretching (percent)

Figure 9.1: Lighting level -3

Figure 9.2: Lighting level -2

Figure 9.3: Lighting level -1

Figure 9.4: Lighting level 0

Figure 9.5: Lighting level +1

Figure 9.6: Lighting level +2

Figure 9.7: Lighting level +3

Figure 9.8: Lighting level -3 after histogram equalization

Figure 9.9: Lighting level -2 after histogram equalization

Figure 9.10: Lighting level -1 after histogram equalization

Figure 9.11: Lighting level 0 after histogram equalization

Figure 9.12: Lighting level 1 after histogram equalization

Figure 9.13: Lighting level +2 after histogram equalization

Figure 9.14: Lighting level +3 after histogram equalization

9.4 Comparison

Two tests were done; in the first test, similar images (that is, images that look very much the same to a human) were normalized and compared with several methods, and in the second test, different images (images that look very different to a human) were normalized and compared with the same methods. The numbers in the tables show the average difference in gray value per pixel for several images, and the same number expressed as a percentage.

Figure 9.15: Test image 1 before and after normalization

	Image 2	Image 3	Image 4
Image 1	52.96 (20.77%)	49.24 (19.31%)	81.86 (32.10%)
Image 2	-	44.73 (17.54%)	88.94 (34.88%)
Image 3	-	-	84.23 (33.03%)

Table 9.13: Comparison of different images using the absolute method

Figure 9.16: Test image 2 before and after normalization

Figure 9.17: Test image 3 before and after normalization

Figure 9.18: Test image 4 before and after normalization

	Image 2	Image 3	Image 4
Image 1	0.81 (0.32%)	2.08 (0.82%)	0.82 (0.32%)
Image 2	-	0.12 (0.05%)	1.53 (0.60%)
Image 3	-	-	1.42 (0.56%)

Table 9.14: Comparison of different images using the non-absolute method

	Image 2	Image 3	Image 4
Image 1	36.65 (14.37%)	36.11 (14.16%)	56.00 (21.96%)
Image 2	-	31.21 (12.24%)	61.82 (24.24%)
Image 3	-	-	61.07 (23.95%)

Table 9.15: Comparison of different images using $3*3$ best neighbor comparison

	Image 2	Image 3	Image 4
Image 1	30.12 (11.81%)	32.02 (12.56%)	52.80 (20.71%)
Image 2	-	27.31 (10.71%)	57.77 (22.65%)
Image 3	-	-	56.82 (22.28%)

Table 9.16: Comparison of different images using $5*5$ best neighbor comparison

Figure 9.19: Test image 5 before and after normalization

Figure 9.20: Test image 6 before and after normalization

Figure 9.21: Test image 7 before and after normalization

Figure 9.22: Test image 8 before and after normalization

	Image 6	Image 7	Image 8
Image 5	7.76 (3.04%)	7.26 (2.85%)	6.95 (2.73%)
Image 6	-	6.91 (2.71%)	7.78 (3.05%)
Image 7	-	-	7.91 (3.10%)

Table 9.17: Comparison of similar images using the absolute method

	Image 6	Image 7	Image 8
Image 5	0.98 (0.38%)	0.73 (0.29%)	0.27 (0.11%)
Image 6	-	0.29 (0.11%)	0.84 (0.33%)
Image 7	-	-	1.07 (0.42%)

Table 9.18: Comparison of similar images using the non-absolute method

	Image 6	Image 7	Image 8
Image 5	2.30 (0.90%)	2.08 (0.82%)	2.22 (0.87%)
Image 6	-	1.92 (0.75%)	2.49 (0.98%)
Image 7	-	-	2.33 (0.91%)

Table 9.19: Comparison of similar images using $3 * 3$ best neighbor comparison

	Image 6	Image 7	Image 8
Image 5	1.67 (0.65%)	1.44 (0.56%)	1.49 (0.58%)
Image 6	-	1.36 (0.53%)	1.66 (0.65%)
Image 7	-	-	1.49 (0.58%)

Table 9.20: Comparison of similar images using $5 * 5$ best neigbor comparison

9.5 Discretization noise

Remembering the visualization of discretization noise in chapter 7, there is a way to prove this optical result mathematically, by calculating difference images. For this reason, four original images (Fig.: 9.23) were scanned with the same lighting and no changes in their position on the scanner. The resulting images were compared pixel-by-pixel. The result of image A and the average difference is shown in Table 9.21.
Table 9.22 and Table 9.23 show the results after $3 * 3$ median filtering and $3 * 3$ lowpass filtering.

Figure 9.23: The four example images

Image $\mathcal{A}$	scan 2	scan 3	scan 4	scan 5
scan 1	0.38	0.37	0.37	0.37
scan 2	-	0.39	0.44	0.43
scan 3	-	-	0.42	0.41
scan 4	-	-	-	0.37

Average = 0.39

Table 9.21: Scanner noise (Image A, Lighting level 0)

Conclusions from these findings follow in the next chapter.

Image $\mathcal{A}$	scan 2	scan 3	scan 4	scan 5
scan 1	0.34	0.32	0.32	0.31
scan 2	-	0.35	0.39	0.38
scan 3	-	-	0.37	0.35
scan 4	-	-	-	0.31

Average = 0.35

Table 9.22: Scanner noise after $3x3$ median filtering (Image A, Lighting level 0)

Image $\mathcal{A}$	scan 2	scan 3	scan 4	scan 5
scan 1	0.20	0.20	0.20	0.21
scan 2	-	0.21	0.23	0.21
scan 3	-	-	0.21	0.21
scan 4	-	-	-	0.19

Average = 0.21

Table 9.23: Scanner noise after $3x3$ lowpass filter (Image A, Lighting level 0)

Images$\mathcal{A} - \mathcal{D}$	Average
without processing	0.45 (0.18%)
median filter (3*3)	0.38 (0.15%)
lowpass filter (3*3)	0.24 (0.10%)
histogram equalization	0.84 (0.33%)
contrast stretching	0.89 (0.35%)

Table 9.24: Application of the processing techniques