# Teaching Image Processing: A Two-Step Process

**Article** *in* Computer Applications in Engineering Education · October 2008

DOI: 10.1002/cae.20149

2 authors:

Clarence Yapp
Harvard Medical School
**111** PUBLICATIONS **3,581** CITATIONS
SEE PROFILE

Alex Bin See
Ngee Ann Polytechnic
**13** PUBLICATIONS **91** CITATIONS
SEE PROFILE

# Teaching Image Processing: A Two-Step Process

CLARENCE HAN-WEI YAPP,[1] ALEX KOK BIN SEE[2]

[1]*School of Engineering, Monash University Malaysia, No. 2 Jalan Kolej, Bandar Sunway, 46150, PJ, Selangor Darul Ehsan, Malaysia*

[2]*School of Engineering, Division of Electrical Engineering, Ngee Ann Polytechnic, 535 Clementi Road, Singapore 599 489*

Received 18 January 2007; accepted 7 March 2007

**ABSTRACT:** An interactive program for teaching digital image processing techniques is presented in this article. Instead of heavy programming tasks and mathematical functions, students are led step by step through the exercises and then allowed to experiment. This article evaluates the proposed program and compares it with existing techniques. ©2008 Wiley Periodicals, Inc. Comput Appl Eng Educ 16: 211–222, 2008; Published online in Wiley InterScience (www.interscience.wiley.com); DOI 10.1002/cae.20149

**Keywords:** image processing; computer assisted learning; education; LabVIEW; vision

## INTRODUCTION

Image processing is becoming a popular and important topic in engineering and computer science [1]. Today, image processing can be found in the fields of robotics, algorithms, machine vision, and mathematics. However, despite it being a topic of great importance, it is a well-known fact that it is a hard topic to teach in lectures alone. This dilemma is described in numerous sources. For example, Lee et al. [2] say that teaching image processing requires the use of images, which are hard to describe verbally. Furthermore, since most algorithms are mathematically orientated, students are compelled to look at the underlying equations for an explanation to its workings. Such techniques include the Fourier Transform, convolution, Wavelet Transform, Gaussian blurring, and so on. Bebis et al. [1] state that it was normal for computer vision to be taught in a classroom environment during lectures and using textbooks. Apparently, lecture notes were issued containing poorly photocopied pictures with degraded quality. Since image processing is obviously to do with images, it is important that visual quality be kept at the highest reasonable level. The use of an overhead projector was also discussed as being disruptive to the flow of learning when the lecturer changed the slides. Bebis et al. [1] were quick to point out that this method does not encourage critical thinking nor motivate students to think further in the topic. Such materials should allow students to explore rather than absorb materials [3]. Similarly, a ‘‘hands-on’’ approach was considered by Wayne [4]. As McAndrew and Venables [5] state, a student’s first encounter to learning image processing should be ‘‘stimulating,’’ ‘‘instructive,’’ fun, and ‘‘gender neutral,’’ while being

Correspondence to A. K. B. See (alex.see@ieee.org).

taught in a way that matches the skill of the learner. If the intended audience consists of high school students, a university level style of teaching may not be suitable and cause the students to quickly become lost or disinterested [4]. To illustrate this, Robinson [6] has designed a program while assuming that their students can program in at least one language. Unfortunately, this cannot be assumed in all cases, thus a problem may arise when students are not majoring in computer science but still want to take up vision as an elective. These students may not have the necessary background to allow them to write programs while trying to comprehend the theory behind many of the complex algorithms at the same time [7].

Finally, another problem that has been highlighted is the fact that, though there may be many potentially pedagogical programs in the market, they are usually expensive, inaccessible, and not designed specifically for the purpose of education [8]. One only needs to look at the popular Photoshop as an example. Although it does an excellent job at showing the effects of a wide range of imaging filters, it does not show students how it arrived at the result. Furthermore, Photoshop is very expensive and may compel universities to invest in only a limited number of licenses. Thus, this could pose a limit on the number of students who can participate in lab classes.

The rest of this article is as follows. An overview of the main methods currently employed to teach image processing is in Background Section. Then, in Design Section, some of the programs developed for this project will be explained. Evaluation/Feedback Section discusses the results to an evaluation survey carried out to obtain some feedback from students. Finally, Conclusion Section presents the conclusion.

## BACKGROUND

This section covers the analysis of some of the different methods and programs aimed at teaching image processing that exist today.

One method is to ask students to program their own versions of the algorithm after having learnt them. Sage and Unser [9] have proposed that students program in Java. Java's chief advantage is that it does not require expensive hardware while still able to run quickly and that it is robust with good error handling features. The rationale for asking students to program their own filters is so that they are directly involved in the development of their algorithms. They claim that it is very encouraging for students to see their code be transformed into images. This is in contrast to passive learning in lectures and visual demonstrations. As Sage and Unser [9] point out, the students may not be familiar with Java at all. Thus, their solution is to present fragments of code where students are required to fill in the blanks or perform modifications.

The second method that will be covered in this article is the use of stand-alone applications. These applications do not require students to program their own filters. They are very interactive with a simple point and click interface and attempt to teach filters with minimal use of mathematics. Thus, it is possible that students from majors other than computer science and mathematics can also learn about image processing. There are many resources that support this method. For example, the goal of Rajashekar et al. [10] was to create visual examples that would cater to a wider audience with different academic backgrounds (geology, psychology, astronomy, and computer science). They also state that they used simple mathematics in their explanations. Although Sage and Unser [9] stated that the best way to understand an algorithm is to code and test it, Ageenko and Russa [11] clearly argue against this. Simply because a student has memorized the code of a filter does not ensure that he or she understands how and why the filter works and when it should be used. In fact, forcing students to program the filter may be obstructive to their learning. Supposing a student shows great interest in machine vision but has weak programming skills. This student would find it difficult to finish their labs. In this way, programming sessions may not be testing their image processing understanding, but testing their programming skills instead [1].

One problem that has not been addressed here is that students are unable to see the actual underlying mathematics that occurs on each individual pixel. For example, to brighten an image, students should be numerically shown that adding a constant to each pixel brightens an image. Alternatively, darkening an image subtracts a constant from each pixel. This feature can be seen in Ref. [12], a program called Digital Image Processing Simulator (DIPS) that focuses on the "under-the-hood" processes on each pixel. In this case, the filter is only applied to randomly created $7 \times 7$ array patterns. This is a reasonable size as students will not be bored since they only have to perform each calculation 49 times (once for each pixel). An image of this size, however, holds little visual content and it would be difficult to observe the effects of the filter such as blurring or sharpening. There simply is not enough visual information compared to a full size image, say $1,024 \times 768$. On the other hand, asking students to perform that many calculations would quickly cause them to lose interest. Thus, the work proposed here will combine features of both techniques just discussed. It will not only show the actual calculations performed at the pixel level with arrays, it will also demonstrate the overall effect by performing the same filter on an actual image such as a face.

The methodology just discussed may not be sufficient by itself, however, as Morozov et al. [13] point out, today's learners are of the "videogame generation." High interactivity and rich media are necessary in sustaining interest in the topic. For example, Morozov et al. [13] use stylish pre-rendered animations and background plates. This is to further give an illusion that the learner is in a laboratory rather than staring at a flat gray background plate.

National Instruments' LabVIEW, a visual programming language, was selected to program the application. The benefits of a visual programming environment were already stated [14]. One of LabVIEW's strengths is that it runs very quickly due to its ability to generate optimized code that can run as fast as C programs. This is especially important as image processing requires heavy amounts of calculations. With LabVIEW, students can experiment with filters and get an almost immediate response, similar to real time [6].

## DESIGN

The image processing techniques that were covered in this project were:

- Basic operators (add, subtract, multiply, divide) on images.
- Fourier Transform and its use in noise removal.
- Noise removal in the spatial (Gaussian and median blurring).
- Histogram formation, equalization, and specification.
- Global and localized adaptive thresholding.
- Edge detection (Sobel, Prewitt, Roberts, LaPlace, zero crossing, and Canny).
- Morphological operators (Dilation, Erosion, Boundary Extraction, Skeletonization).

### Convolution

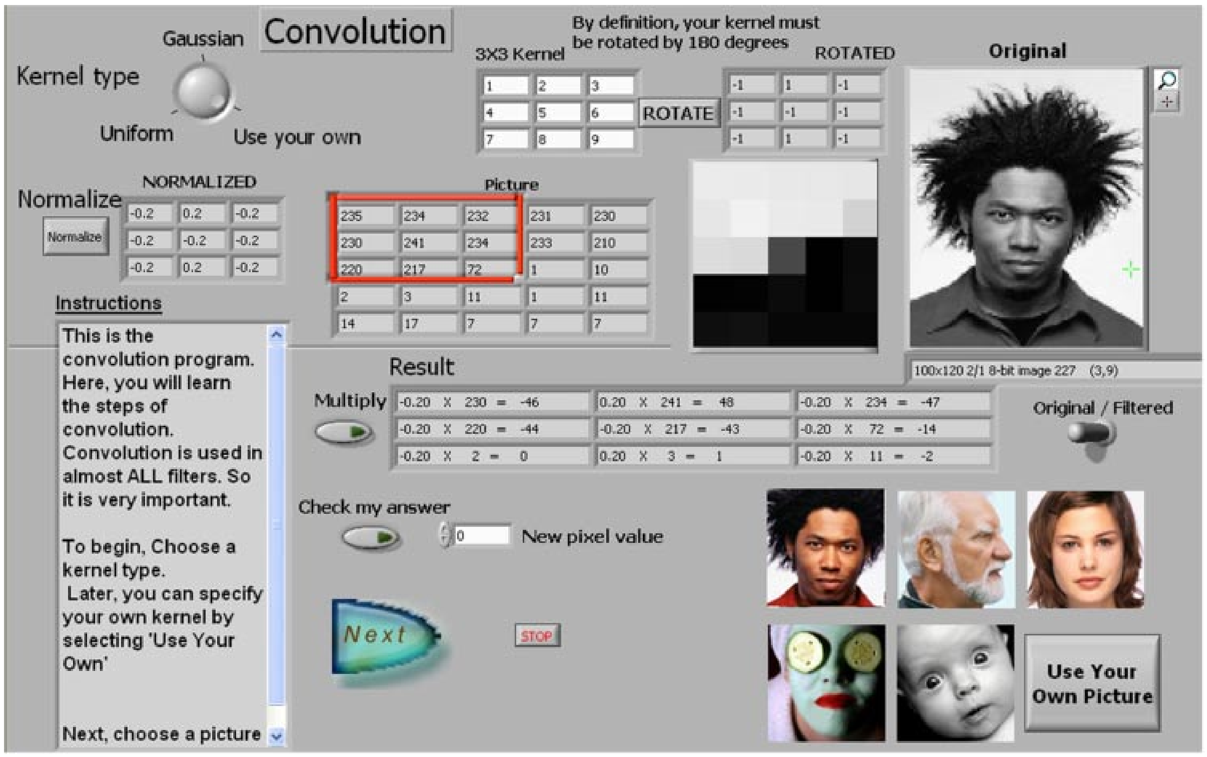
One of the most important concepts that students should be familiar with is convolution. This technique is used extensively in almost all filters, including morphology, edge detection, smoothing and sharpening, and noise removal. In this program, students are shown the actual numerical calculations so that nothing is hidden from them. First, the user selects one of the faces, which appears in the large window in Figure 1. The kernel type can be changed to a predefined or to the user's liking.

The versatility of this program stems from the way it presents the effects of the filter. Students are not only able to view the whole filtered image, but also a zoomed in subset of any part of the image at the click of the mouse. This helps students to understand the mathematical processes occurring on each pixel, which will be explained next. To illustrate, the intensity values of the subset image are first displayed in the $5 \times 5$ picture array with the red box. Basically, a $3 \times 3$ kernel window is moved around the array. Each member of the kernel is multiplied by each pixel in the subset of the image, shown in the moveable red box. This produces nine products shown in the "Result" array. The sum of these nine products is then used to replace the central pixel in the array. This is repeated for all pixels in the image by clicking on the "Next" button. The small image immediately updates itself each time a convolution has occurred. This moves the kernel to the next location. Later, the user can view the overall effect on the entire image by moving the switch to the "Filtered" mode. This is shown in Figure 2. The original picture has clearly been blurred from the convolution operation. Likewise, the picture can be sharpened if the user knows the appropriate kernel, Figure 3.

### Basic Operators

The basic operators application consists of two programs combined in one. Figure 4 shows the interface of the first program that performs additions, subtractions, and multiplications on each picture array. The central monitor shows three arrays: the original array, the operand, and the result. At the bottom of the interface, there are control knobs that will allow students to adjust the type of operation and the operand. For example, if an operator of addition and an operand of 50 are selected, the calculation will be performed and the numerical values in the arrays will reflect the result. In any 8-bit image, the highest integer for a pixel is 255. Likewise, the lowest possible integer is 0. If the result of the operation exceeds any of these limits, the corresponding rectangular LEDs below the result array will light up to show that the result is coerced.

After the student learns how basic operations are carried out on pixels, its time to see the actual effect on pictures. The student presses the "Experiment" button and selects one of the faces on the left and the



3X3 Kernel

|  |  |  |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

ROTATED

|  |  |  |
|---|---|---|
| -1 | 1 | -1 |
| -1 | -1 | -1 |
| -1 | 1 | -1 |

NORMALIZED

|  |  |  |
|---|---|---|
| -0.2 | 0.2 | -0.2 |
| -0.2 | -0.2 | -0.2 |
| -0.2 | 0.2 | -0.2 |

Picture

|  |  |  |  |  |
|---|---|---|---|---|
| 235 | 234 | 232 | 231 | 230 |
| 230 | 241 | 234 | 233 | 210 |
| 220 | 217 | 72 | 1 | 10 |
| 2 | 3 | 11 | 1 | 11 |
| 14 | 17 | 7 | 7 | 7 |

100x120 2/1 8-bit image 227 (3,9)

Instructions

This is the convolution program. Here, you will learn the steps of convolution. Convolution is used in almost ALL filters. So it is very important.

To begin, Choose a kernel type.
Later, you can specify your own kernel by selecting 'Use Your Own'

Next, choose a picture

Result

Multiply

|  |  |  |
|---|---|---|
| -0.20 X 230 = -46 | 0.20 X 241 = 48 | -0.20 X 234 = -47 |
| -0.20 X 220 = -44 | -0.20 X 217 = -43 | -0.20 X 72 = -14 |
| -0.20 X 2 = 0 | 0.20 X 3 = 1 | -0.20 X 11 = -2 |

Check my answer

0 New pixel value

**Figure 1** Convolution demonstration program. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

interface will change to Figure 5. The source pictures have been deliberately degraded as shown on the left large face. The student's job is to fix the picture by selecting the appropriate settings. For example, one way to correct the dark rectangular patch is to select ''Addition'' as the operator, an operand of about 50, and click and drag a region of interest around the dark area. The result is shown immediately on the right

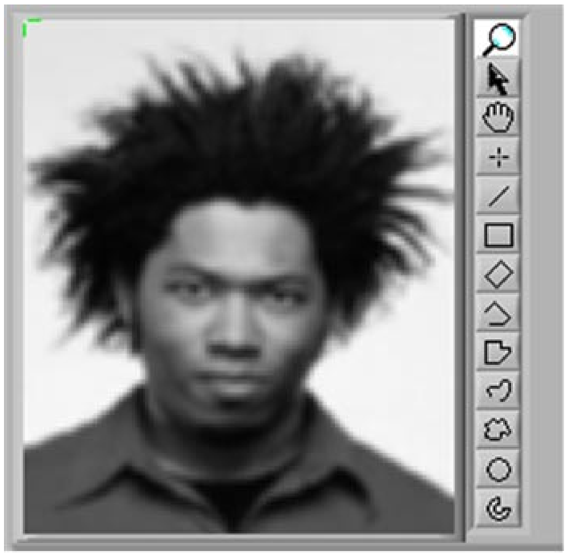


**Figure 2** The convolution program demonstrating blurring techniques. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

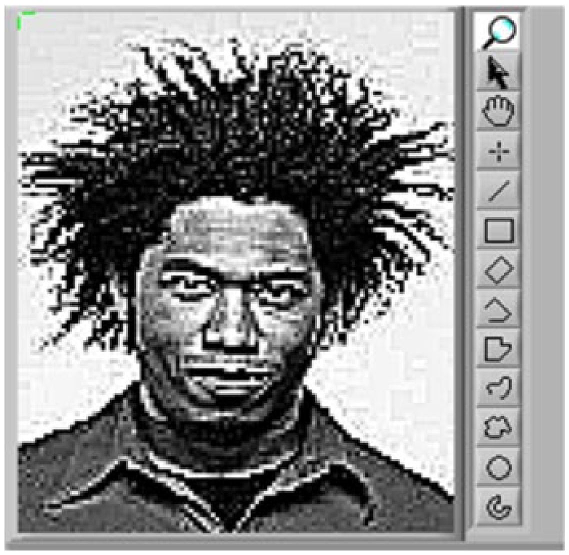
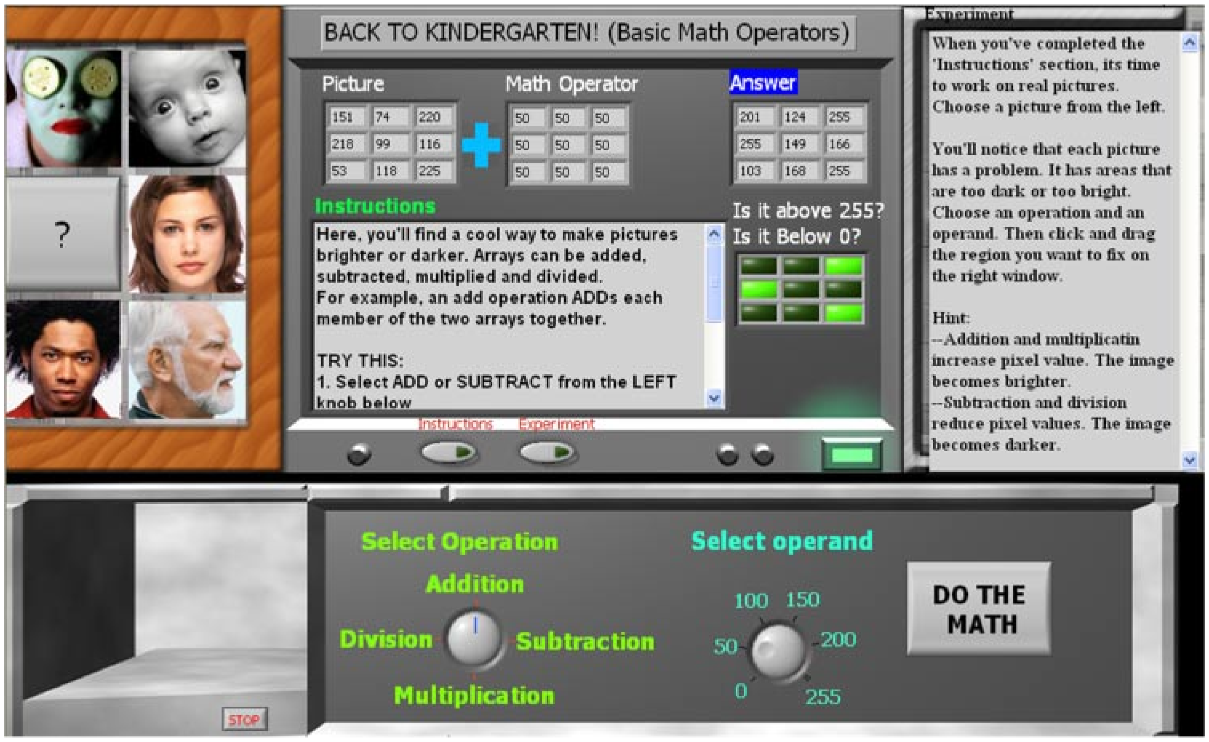


**Figure 3** The convolution program demonstrating sharpening techniques. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]



| Picture | | | Math Operator | | | Answer | | |
|---|---|---|---|---|---|---|---|---|
| 151 | 74 | 220 | 50 | 50 | 50 | 201 | 124 | 255 |
| 218 | 99 | 116 | 50 | 50 | 50 | 255 | 149 | 166 |
| 53 | 118 | 225 | 50 | 50 | 50 | 103 | 168 | 255 |

Here, you'll find a cool way to make pictures brighter or darker. Arrays can be added, subtracted, multiplied and divided.
For example, an add operation ADDs each member of the two arrays together.

TRY THIS:
1. Select ADD or SUBTRACT from the LEFT knob below

When you've completed the 'Instructions' section, its time to work on real pictures. Choose a picture from the left.

You'll notice that each picture has a problem. It has areas that are too dark or too bright. Choose an operation and an operand. Then click and drag the region you want to fix on the right window.

Hint:
--Addition and multiplicatin increase pixel value. The image becomes brighter.
--Subtraction and division reduce pixel values. The image becomes darker.

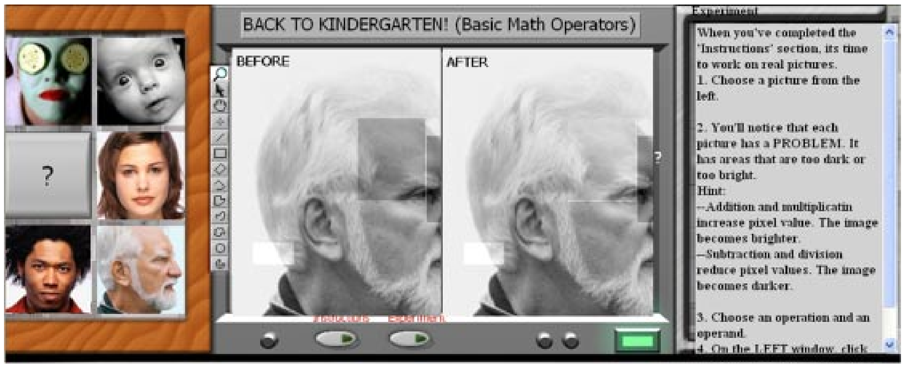
**Figure 4** The basic operators instructions program interface. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

window. This part of the program is very interactive as the student can try a different setting if the previous settings do not work. Later, any of the other four faces can be tried, each with different degradations for the student to try.

## Local and Global Adaptive Thresholding

In Figure 6, the thresholding program allows students to threshold images interactively. Any of the five pictures with an additional user defined picture can be selected. A defect can also be chosen, such as ramp gradient that causes half of the picture to be dark or bright. If the user attempts to threshold the image globally, all pixels in the image will be processed using this single threshold value. Chances are that the initial guess may not be suitable. Thus, an adaptive threshold transform is used. This algorithm suggests a better threshold value with each successive try. Even with the adaptive global threshold, the image may still not be thresholded correctly as shown in Figure 6. The left half of the thresholded image is too dark.

If, on the other hand, local thresholding is selected, the image is divided into 4, 9, or 16 subimages. Each

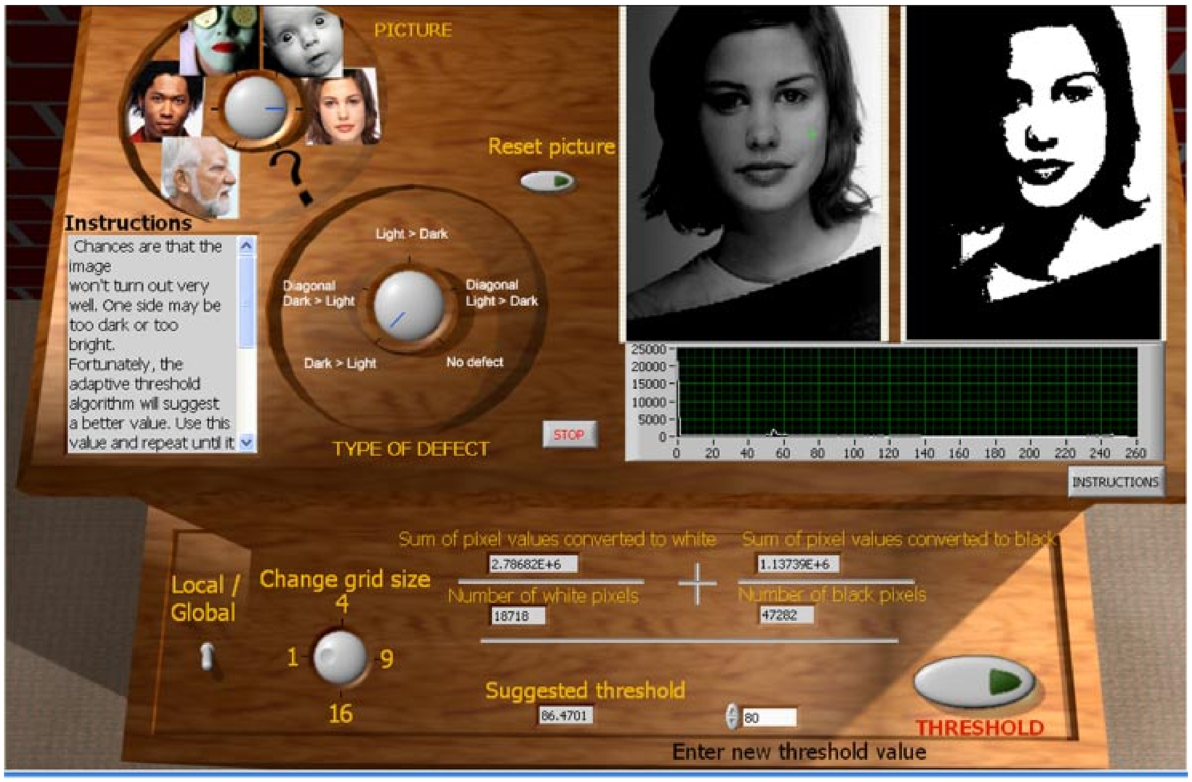


When you've completed the 'Instructions' section, its time to work on real pictures.
1. Choose a picture from the left.

2. You'll notice that each picture has a PROBLEM. It has areas that are too dark or too bright.
Hint:
--Addition and multiplicatin increase pixel value. The image becomes brighter.
--Subtraction and division reduce pixel values. The image becomes darker.

3. Choose an operation and an operand.
4. On the LEFT window, click

**Figure 5** The basic operators experiment program interface. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]



Chances are that the image won't turn out very well. One side may be too dark or too bright. Fortunately, the adaptive threshold algorithm will suggest a better value. Use this value and repeat until it

Sum of pixel values converted to white: 2.78682E+6
Number of white pixels: 18718
Sum of pixel values converted to black: 1.13739E+6
Number of black pixels: 47282
Suggested threshold: 86.4701
Enter new threshold value: 80

Figure 6 The threshold program set to global adaptive thresholding. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

subimage is thresholded individually and interactively and this normally gives better results as shown in Figure 7a. Increasing the number of subimages can also improve the processing, as can be seen from Figure 6b. A larger number of subregions helps to reduce any discontinuities such as those present in Figure 6a.

The adaptive algorithm uses a user specified initial threshold value. From there, the program thresholds the image and counts the number of pixels that have been set to either black or white. The number of black pixels and white pixels is denoted $N_B$ and $N_W$, respectively. For each category, the program also sums their actual pixel values. For example, the sum of all pixel values that were converted to white is denoted $S_W$. The same is done for all black pixels, $S_B$. After that, (3) is used to calculate a suggested threshold value. The result of the threshold can be seen to improve after each iteration

$$\text{New Threshold Value} = \frac{S_B/N_B + S_W/N_W}{2} \quad (1)$$

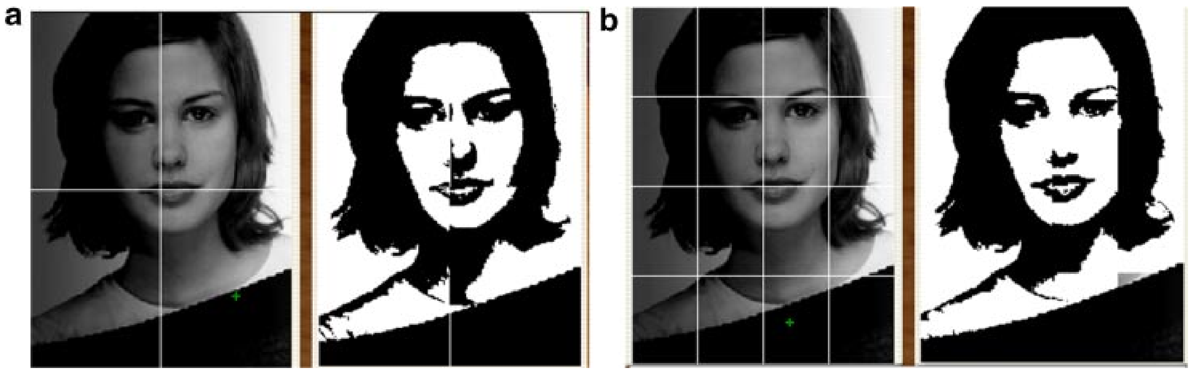


Figure 7 Adaptive local thresholding with (a) 4 subregions, (b) 16 subregions. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

## Histogram Formation

Figure 8 is a program that demonstrates how a histogram is made. The user can choose the picture and the resolution of the histogram. Generally, increasing the resolution, or number of ‘‘bins,’’ improves the quality of the histogram functions. For example, if two bins were chosen, any pixel would fall into only one of these bins. Any histogram processing would simply result in a thresholded 1-bit image. But with 16 bins, the resolution increases to 4-bits.

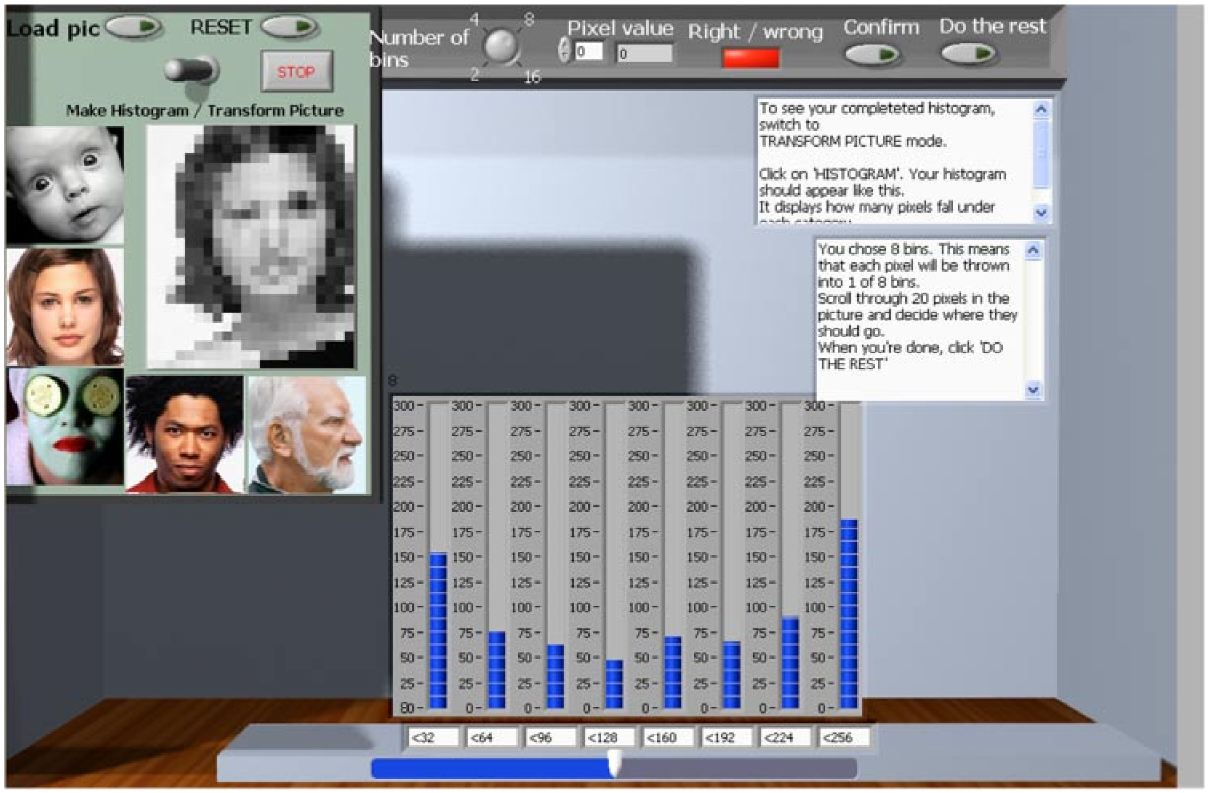
To form the histogram, the user will be presented with pixels directly from the image. The correct bin that each pixel belongs in will be decided by the user by selecting that particular bin and pressing confirm. If the correct bin is chosen, an LED turns from red to green and the histogram immediately updates itself. Although this process is repetitive, the user does have the choice of letting the computer finish the rest of the image but only after the user has completed 20 pixels.

Later, another interface is used to create a transformation function based on the histogram just formed. Once the transformation function is found, the user is presented with 20 pixels. Each of these input pixels is ‘‘looked up’’ using the transformation function to obtain the new output pixel. This new pixel value will replace the old pixel value in the picture.

Once the user has completed this program, he or she can do the same for another picture. Alternatively, the user can continue to the advanced program to experiment with full size images and 256 bin histograms, Figure 9. Because of the nature of equalization, it simply may not produce the best result. Thus, a key feature that has been added allows the user to specify their own transformation functions to tailor their requirements and see the effects on the image in real time. The marked difference between histogram equalization and specification can be observed in Figure 9.

## Noise Removal in the Frequency Domain

Images can be converted to the frequency domain using the Fourier transform, a complex but crucial tool in the field of image processing. Although images are inherently two-dimensional ($x$ and $y$ coordinate), the Fourier transform can also be applied to one-dimensional functions ($x$ coordinate only). Since the Fourier transform is already complicated when dealing with 2D signals, it may be easier to demonstrate it on 1D signals first.

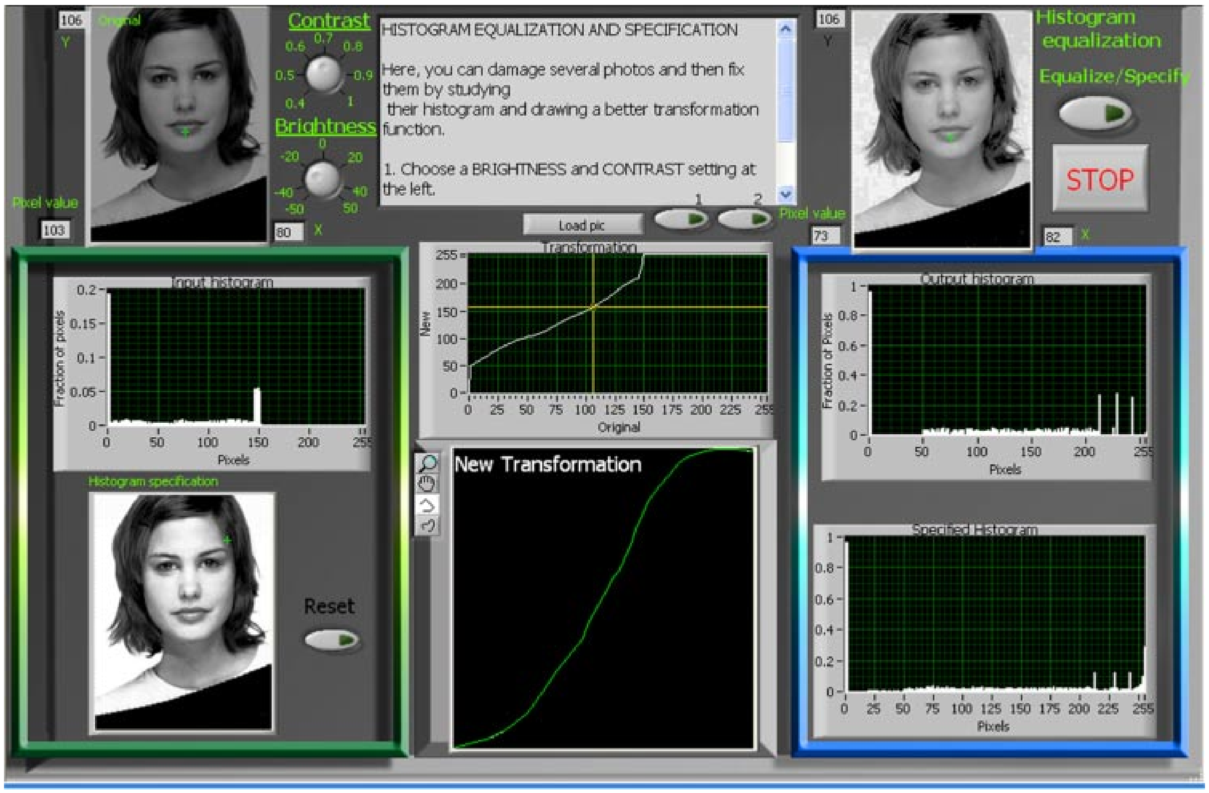


To see your completeted histogram, switch to TRANSFORM PICTURE mode.

Click on 'HISTOGRAM'. Your histogram should appear like this. It displays how many pixels fall under

You chose 8 bins. This means that each pixel will be thrown into 1 of 8 bins. Scroll through 20 pixels in the picture and decide where they should go. When you're done, click 'DO THE REST'

**Figure 8** The histogram formation program showing how a histogram is made. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]


HISTOGRAM EQUALIZATION AND SPECIFICATION

Here, you can damage several photos and then fix them by studying
their histogram and drawing a better transformation function.

1. Choose a BRIGHTNESS and CONTRAST setting at the left.

Figure 9 Histogram specification and equalization. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

The basic idea of the Fourier transform is that any function, no matter how complicated, can be separated into a set of pure sinusoidals that, when combined together, reproduce the original function. For example, Figure 10 shows a typical complicated signal.

Fortunately, according to Fourier, this is just a combination of two sinusoidal waveforms that can be easily separated into two waveforms (Fig. 11).

While this is a simple example, many students have difficulty understanding how such a waveform can be separated. In Figure 12, the interface for a program is shown, which could make this concept clearer by demonstration and illustration. This application combines no more than two waveforms, and because of such simplicity, it is hoped that the student is not bogged down. Later, however, students will have the ability to work with some very complicated Fourier plots from actual pictures.

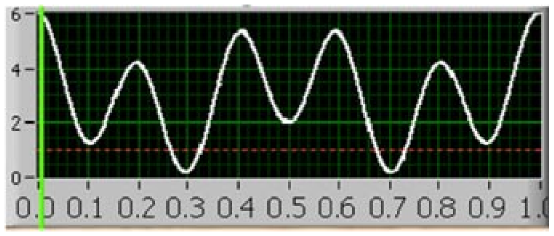


Figure 10 A complicated waveform. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

The selected waveform is first shown in the two graphs. If the waveform represented pixel intensities, the picture would appear as shown at the top. To begin analyzing the waveform's constituents, the user starts with a test waveform of 0 Hz and compares it to the subject waveform. To compare the waveforms, the scrollbar located between the graphs is dragged from side to side. This multiplies the subject and test waveforms together at regular intervals. The products of the waveforms are shown below each graph. Finally, this value is plotted at the bottom right using a symmetrical plot where 0 and 5 Hz are represented at the center and at the outer edges respectively. If the product is a high value, this means that the test waveform has a higher amplitude than other components, and the plot will appear brighter at that specific point. To completely analyze the waveform, the user must test using all six frequencies from 0 to 5 Hz.

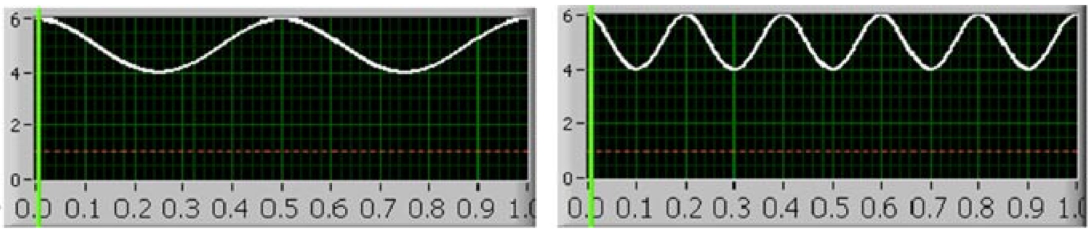


**Figure 11** The two separated components using Fourier Transform. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

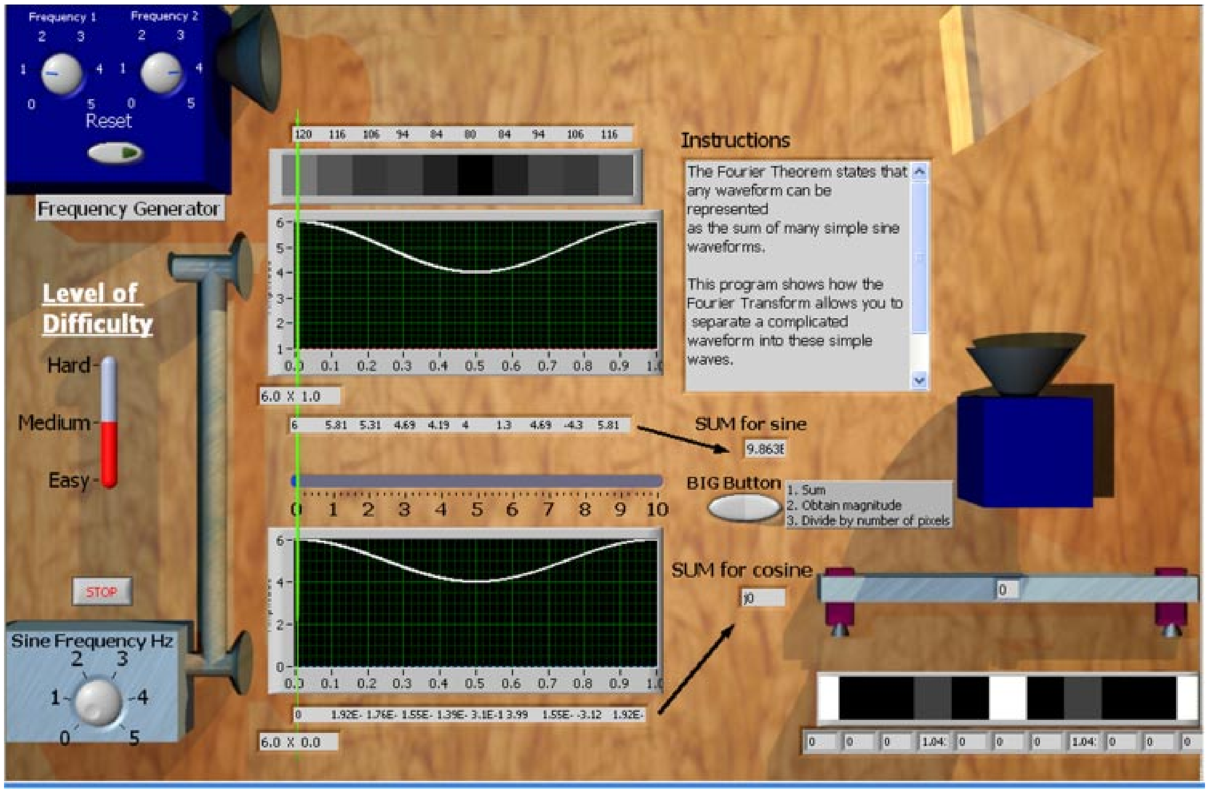
The interface in Figure 13 allows students to remove periodic noise from face images in the frequency spectrum

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\,\mathrm{e}^{-\mathrm{j}2\pi(ux/M+vy/N)} \qquad (2)$$

Firstly the image is converted to its Fourier transform using (1). Since images and their Fourier transforms are two dimensional, (1) has two summations, one for each dimension. It works by selecting an initial frequency for both *u* and *v*, say, 0. Then, it starts at the first pixel and works out the result of (2) for every other pixel in the image in relation to it. The final value is plotted on the Fourier transform plot at a coordinate that corresponds to the selected values of *u* and *v*, in this case 0 for both. The values of *u* and *v* are incrementally increased so that all possible values for both variables have been tried

$$f(x,y)\,\mathrm{e}^{-\mathrm{j}2\pi(ux/M+vy/N)} \qquad (3)$$

In Figure 13, the source image is clearly degraded with periodic noise. This noise is randomly generated each time so that students will always get a different problem. From the Fourier transform shown on the right, one can see that the periodic noise is caused by the two prominent white dots representing the



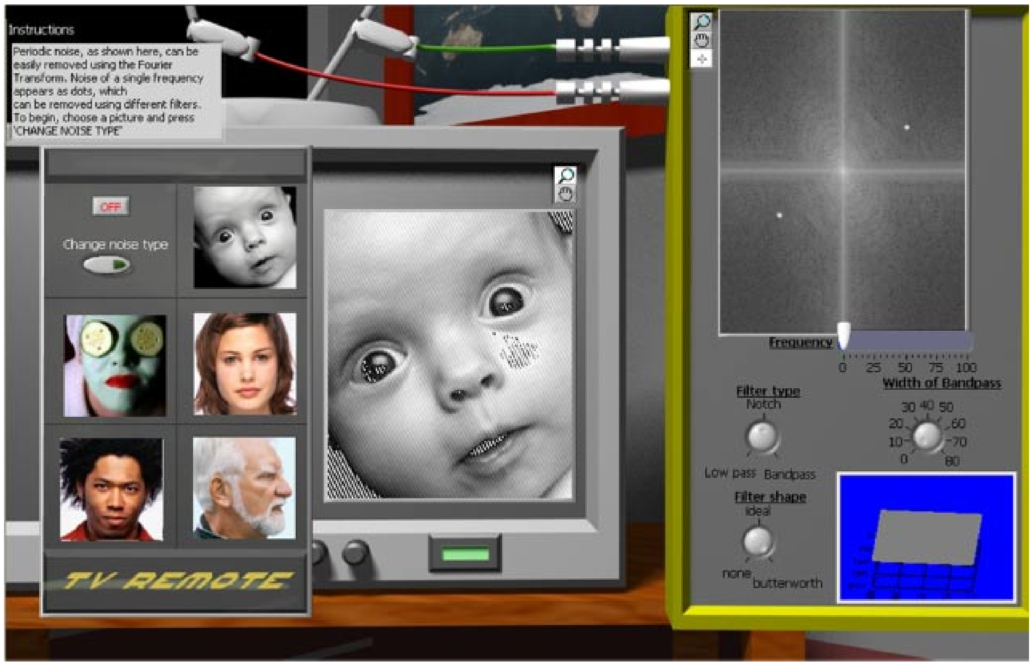
Instructions

The Fourier Theorem states that any waveform can be represented
as the sum of many simple sine waveforms.

This program shows how the Fourier Transform allows you to separate a complicated waveform into these simple waves.

```
120   116   106   94   84   80   84   94   106   116
6.0 X 1.0
6   5.81   5.31   4.69   4.19   4   1.3   4.69   -4.3   5.81
SUM for sine 9.863E
BIG Button
1. Sum
2. Obtain magnitude
3. Divide by number of pixels
SUM for cosine j0
0   1.92E-  1.76E-  1.55E-  1.39E-  3.1E-1  3.99   1.55E-  -3.12   1.92E-
6.0 X 0.0
0   0   0   1.04:   0   0   0   1.04:   0   0   0
```

**Figure 12** Program showing Fourier Transform used to separate a waveform into two components. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]


Instructions

Periodic noise, as shown here, can be easily removed using the Fourier Transform. Noise of a single frequency appears as dots, which can be removed using different filters. To begin, choose a picture and press 'CHANGE NOISE TYPE'

**Figure 13** A noisy image in the noise removal in the frequency spectrum program. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

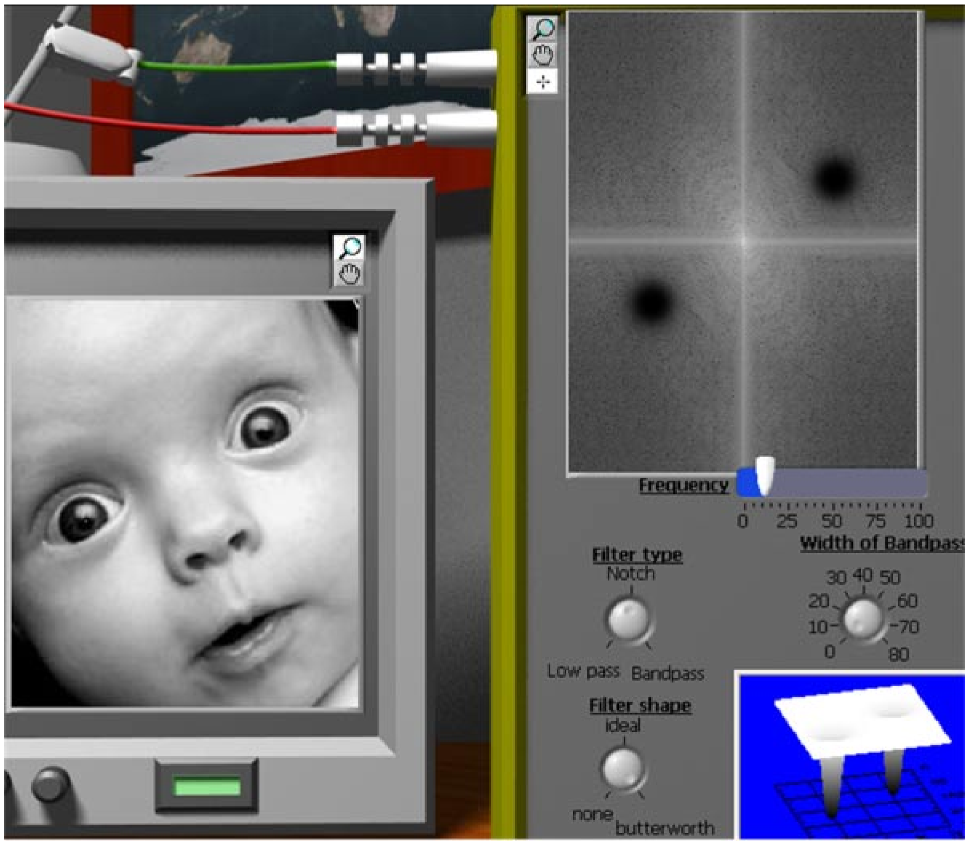


**Figure 14** The corrected picture after removing noise in the frequency spectrum. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

**Table 1** Students' Responses to an Evaluation Survey

| Question | Strongly disagree (%) | Disagree (%) | Neutral (%) | Agree (%) | Strongly agree (%) |
|---|---|---|---|---|---|
| The learning objectives of the topic were made clear to me | 0 | 0 | 0 | 70 | 30 |
| The program enabled me to achieve the learning objectives | 0 | 0 | 20 | 50 | 30 |
| I found the unit to be intellectually stimulating | 0 | 0 | 30 | 40 | 30 |
| The instructions were clear and informative | 0 | 0 | 50 | 20 | 30 |
| This program is useful for teaching image processing | 0 | 0 | 0 | 60 | 40 |

guilty frequency components. The user's job is to remove those dots with the most appropriate settings. These are:

- Filter type—Low pass filter, bandpass, notch.
- Frequency—to control the strength of the filter.
- Filter shape—Ideal or Butterworth.
- Width of bandpass (when using bandpass filter).

As the user changes the parameters, the 3D plot at the bottom left corner gives a visual representation of the filter. This aids the user in visualizing the filter and can help in predicting the possible output. The 3D plot can also be seen in Figure 14.

Figure 14 shows a possible remedy using a notch filter with a Butterworth shape. The notch filter is capable of removing specific frequencies as shown. A bandpass filter would have removed a ring of components, which would be excessive in this case. Selecting the Butterworth helps to minimize artifacts called "banding," which are found when using the Ideal shape. These settings seem to have cleared the picture of noise, however, there are many possible solutions.

## EVALUATION/FEEDBACK

A survey involving 10 students was conducted after allowing them to test some of the programs. Table 1 lists their responses.

All students who were surveyed agreed that the program was generally useful and successful in teaching image processing. They shared a common idea that the programs helped to stimulate an interest in image processing through its interactive ideas and problem solving activities. Furthermore, they also agreed that they would prefer to learn through such a program instead of learning by programming.

According to the results of question 3, about one-third of the students strongly agreed that the contents were stimulating in such a manner that it allowed them to want to develop further ideas and continue using the program. However, one-third of the responses also showed that the content could be further improved. This could be done by adding more forms of multimedia such as audio annotations and more animations. On the other hand, students generally pointed out that the current interface was a good start and that the use of various forms of controls were crucial to the learning process.

For question 4, the instructions obviously needed some enhancement as only half the students found the instructions to be excellent. One student commented that the instructions for the Salt and Pepper noise removal program were easy to understand, but the same could not be said for the Fourier Transform program. The reason is likely to be that the Fourier Transform is a complex technique that warrants a more detailed set of instructions. It is possible that the Fourier Transform also deserves audio and visual demonstrations rather than text based. Another student suggested that the instructions should be given one step at a time after each action. This is in contrast to displaying all instructions in a list. At the time of writing, this problem has now been rectified in several of the programs. Currently, the instructions update whenever a student performs an action.

## CONCLUSION

Image processing is an essential interdisciplinary course and should be taught in a stimulating and engaging manner. When demonstrating concepts, the needs and the skills of the student must be taken into consideration. Asking a student who is weak in programming to create their own filter in C or Java can be hindering their ability to understand its general characteristics. Simply incorporating programming sessions in labs does not fully explain why, when, and how the filter works down to the fundamental pixel level.

The work described here shows a series of applications created in LabVIEW where each filter is presented in two ways. The first method shows the calculations and processes that each pixel encounters on a small and manageable picture. The second method allows students to apply filters to full sized images while providing opportunities to adjust a wide array of parameters and displaying results in real time. As a bonus, LabVIEW was capable of producing engaging learning environments with rich multimedia. This includes images, video, and sound, which have been (or will be) incorporated into these applications.

## ACKNOWLEDGMENTS

The authors would like to thank Monash University Malaysia for providing a conducive environment to undertake this project. Special thanks should also go to the program testers and evaluators: Darwin Gouwanda, Arif Oyong, Niran Singh Khaira, Charles Lean, Yogeswaran, Ang Kok Seong, William Lavi, Chua Chong Wu, Ang Jin Leong, Jesse Cornelius, and Hari Benpoly.

## REFERENCES

[1] G. Bebis, D. Egbert, and M. Shah, Review of computer vision education, IEEE Trans Educ 46 (2003), 2–21.
[2] J. Lee, Y. Cho, H. Heo, and O. Chae, MTES: Visual programming environment for teaching and research in image processing, Lecture Notes Comput Sci 3514 (2005), 1035–1042.
[3] A. Tretiakov and T. Kinshuk, Designing multimedia support for situated learning, Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies 2003, July 9–11, 2003, pp 32–36.
[4] L. Wayne, Teaching machine vision to engineers, IEE Colloquium on Teaching of Digital Image Processing in Universities, October 21, 1993, pp 7/1–7/4.
[5] A. McAndrew and A. Venables, A ‘‘secondary’’ look at digital image processing, Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, 2005, pp 337–341.
[6] J. Robinson, A software system for laboratory experiments in image processing, IEEE Trans Educ 43 (2000), 455–459.
[7] R. C. Gonzalez and R. E. Woods, Digital image processing, 2nd ed., Prentice-Hall, NJ, 2002.
[8] J. Cubillo and M. Fisher, Digital image processing and machine vision—An education support package, IEE Colloquium on Teaching of Digital Image Processing in Universities, October 21, 1993, pp 9/1–9/4.
[9] D. Sage and M. Unser, Easy Java programming for teaching image-processing, Proceedings of the International Conference on Image Processing 2001, Vol. 3, October 7–10, 2001, pp 298–301.
[10] U. Rajashekar, G. Panayi, F. Baumgartner, and A. Bovik, The SIVA demonstration gallery for signal, image, and video processing education, IEEE Trans Educ 45 (2002), 323–335.
[11] E. Ageenko and G. Russa, A visualization toolkit for teaching, learning and experimentation in image processing, 35th ASEE/IEEE Frontiers in Education Conference, 2005, F2H-21–26.
[12] C. S. Rathore, Digital image processing simulator (DIPS)—A software for teaching digital image processing concepts, http://www.gisdevelopment.net/education/papers/edpa0017.htm (accessed on 18th August 2005).
[13] M. Morozov, A. Tanakov, A. Gerasimov, D. Bystrov, and E. Cvirco, Virtual chemistry laboratory for school education, Proceedings of the IEEE International Conference on Advanced Learning Technologies 2004, August 30–September 1, 2004, pp 605–608.
[14] A. Bovik, Handbook of image and video processing, Academic Press, Canada, 2005.

## BIOGRAPHIES

**Clarence Yapp** earned his first degree with first-class honors in engineering (Mechatronics) from Monash University Malaysia (2006) and received the Best Overall Graduate award. He is currently pursuing a master's degree in the Department of Engineering and Science at Oxford University in the field of biomedical engineering and is planning to continue onto a DPhil at Oxford's Centre for Tissue Engineering and Bioprocessing. One of his main interests is in integrating the disciplines of engineering and the life sciences to develop even more effective methods for regenerative medicine.

**Alex See** was an overseas research scholar at the University of Leicester, United Kingdom, and he earned his PhD in instrumentation and measurement at University of Leicester in 2001. He also earned his BEng and was awarded first-class honors in electrical and electronics at the University of Leicester. He has over eight years of experience using National Instruments products. As an engineer with Defense Science Organisation (DSO) National Laboratory, he worked in the area of underwater sonar acoustics and signal processing. He served as an engineering lecturer at Monash University at Malaysia for about four years. He is a senior member (SMIEEE) of the Institute of Electrical and Electronics Engineers (IEEE) and a member of the Institute of Engineers, Singapore (IES). He is a certified LabVIEW Associate Developer (CLAD). He served as an adjunct lecturer in the School of Computer Science and Electrical Engineering, University of NewCastle. He is currently a lecturer with the School of Engineering, Division of Electrical Engineering, at Ngee Ann Polytechnic, Singapore.