TEAM 4 (FAST AND SOMEWHAT ACCURATE ALGORITHMS): MATHEMATICAL MODELING IN INDUSTRY XIX (AUGUST 5-14, 2015)

CHAI WAH WU, IBM (MENTOR), MARIO BARELA, SU CHEN, EMILY GUNAWAN, MORGAN SCHREFFLER, MINHO SONG, AND DOYEOB YEO

1. Introduction

We seek to produce algorithms using look-up-tables (LUT) for speeding up edge-detection and blurring filters. In order for any LUT-based algorithm to be useful, the size of the LUTs need to be reasonably small. To meet this aim, we use:

(1) a byte-truncation technique,
(2) convolution decompositon of a kernel 3×3 matrices (rank one),
(3) approximate convolution decompositon of a kernel 3×3 matrices (rank two)
(4) and taking advantage of the observation that edge-detection algorithm shows higher tolerance to byte-truncation.

2. instruction for using comments (while we are drafting the report)

morgan: to write a comment to Morgan

mario: to write a comment to Mario

minho: to write a comment

chensu: to write a comment

emily: to write a comment

doyeob: to write a comment

chaiwah: to write a comment

3. Wednesday August 5: Daily Notes

Decide that look-up tables are not suitable for a median filter algorithm.

4. Thursday August 6: Daily Notes

4.1. Compute an algorithm using a look-up table for a high-pass kernel matrix.

$\frac{1}{8}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ using a truncation technique using truncation matrix $\begin{bmatrix} 8 & 4 & 8 \\ 4 & 2 & 4 \\ 8 & 4 & 8 \end{bmatrix}$. This particular look-up table was created in about 15-19 seconds.

Date: August 10, 2015.

4.2. **Compare results of linear filters on Matlab with and without truncating digits.** Truncating with the following truncating matrices look acceptable for gray-scale images: $T_0 = \begin{bmatrix} 0 & 4 & 0 \\ 4 & 0 & 4 \\ 0 & 4 & 0 \end{bmatrix}, T_1 = \begin{bmatrix} 0 & 4 & 0 \\ 4 & 1 & 4 \\ 0 & 4 & 0 \end{bmatrix}$ $T_2 = \begin{bmatrix} 0 & 4 & 0 \\ 4 & 2 & 4 \\ 0 & 4 & 0 \end{bmatrix}$ $T_3 = \begin{bmatrix} 0 & 5 & 0 \\ 5 & 3 & 5 \\ 0 & 5 & 0 \end{bmatrix}$ $T_4 = \begin{bmatrix} 0 & 3 & 0 \\ 3 & 1 & 3 \\ 0 & 3 & 0 \end{bmatrix}$.

A normalized gradient magnitude from Sobel-Feldman operator [Wikipedia/Sobel] $\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$

where

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}.$$

Here * denotes the 2-dimensional convolution operation.

Some high-pass kernel matrices:

- $\frac{1}{8}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
- $\frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

A low-pass kernel matrix:

- $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

A Scharr Gradient kernel matrix:

- $\frac{1}{32}\begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$

An unsharp mask kernel filter:

- $\frac{1}{8}\begin{bmatrix} 0 & -\lambda & 0 \\ -\lambda & 1 - 0.3 * \lambda & -\lambda \\ 0 & -\lambda & 0 \end{bmatrix}$ where $\lambda = 0.1$.

Remark 4.1. For blurring filters (including the low-pass kernel we list above), we think that the truncation technique will cause too many errors.

4.3. **Matrix-decomposition?** Started looking into doing matrix decomposition to make the LUT smaller.

4.4. **Using the truncating technique for a blurring filter: Taking advantage of the fact that truncating works better for edge-detecting than for blurring.**

Remark 4.2. (1) The truncation method T can be used to reduce running time.
(2) Truncation introduces too much error for blurring, but is decent for edge-detecting.

Let I, L, and H denote the identity, a low-pass filter operation, and a high-pass filter operation on an image. Let $L(T)$, and $H(T)$ denote L and H with a truncation using the

truncation matrix T. To informally say that applying a truncation technique to a high-pass filter gives close enough result, we can write $H \approx H(T)$.

Since $I = L + H$, we have $L = I - H \approx I - H(T)$.

Example 4.3. For example, consider a high-pass filter $H = \frac{1}{8}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$. Let

$$T_1 = \begin{bmatrix} 8 & 4 & 8 \\ 4 & 2 & 4 \\ 8 & 4 & 8 \end{bmatrix}, T_2 = \begin{bmatrix} 8 & 5 & 8 \\ 5 & 3 & 5 \\ 8 & 5 & 8 \end{bmatrix} \text{ and } T_3 = \begin{bmatrix} 8 & 6 & 8 \\ 6 & 4 & 6 \\ 8 & 6 & 8 \end{bmatrix}$$

be truncation matrices. The table1 shows us how $L(T_i)$ and $I - H(T_i)$ are different from the low-pass filter $L = I - H$ for each i. $L(T_i)$ is Old and $I - H(T_i)$ is New in the table1.

	T_1		T_2		T_3	
	Old	New	Old	New	Old	New
PSNR	38.12	39.58	30.80	32.84	22.59	25.59
l_2 - error	0.011	0.012	0.029	0.023	0.074	0.053
l_∞ - error	0.033	0.029	0.077	0.062	0.17	0.14

TABLE 1. Comparison of errors

doyeob: Please add paper references for PSNR

l_2, l_∞-differences between two images A,B are given as:

$$l_2\text{-difference} = \sqrt{\frac{\sum_{i,j}\{A(i,j) - B(i,j)\}^2}{(\text{size of image})}}$$

$$l_\infty\text{-difference} = \max_{i,j}|A(i,j) - B(i,j)|.$$

The PSNR (in dB) is defined as

$$\begin{aligned} \text{PSNR} &= 10 \cdot \log_{10}\left(\frac{\text{MAX}_B^2}{\text{MSE}}\right) \\ &= 20 \cdot \log_{10}\left(\frac{\text{MAX}_B}{\sqrt{\text{MSE}}}\right) \end{aligned}$$

where

$$\text{MSE} = \frac{1}{(\text{size of image})}\sum_{i,j}\{A(i,j) - B(i,j)\}^2.$$

PSNR can be easily calculated by using MATLAB command $psnr(A, B)$. PSNR is the abbreviation for Peak Signal-to-Noise Ratio. In brief, it shows us how image is different. As PSNR is higher, it becomes harder to recognize the difference between original image and filtered image. Typical values for the PSNR in lossy image are between 30 and 50 dB, provided the bit depth is 8 bits. As you see, our new approach gives us the better results no matter which truncation matrix you use. See also Figures 1 made via the truncation matrix T_3.

As you see, you can easily check that our new idea reduces staircase effects against the usual truncation method.

(A) LPF (B) LPF with Truncation (C) Result of the New Idea

FIGURE 1. Comparison of filtered images

Likewise, we can apply our method to any other low-pass filter as follows:

1. Let L be the our target low-pass filter.
2. Set $\tilde{H} := I - L(T)$ for a certain truncation T.
3. Find $\tilde{L} := I - \tilde{H}$ as an approximation of L.

Note that the second and third step stands for avoiding staircase effects.

5. FRIDAY AUGUST 7

Group meeting

(1) Testing filters with truncation
(2) Generating LUT to match with above.
(3) Decomposition into multiple LUTs.
(4) $L \approx Id - H(T)$
(5) Color (RGB) filters (see what Matlab does with edge detection for color pictures to make sure we are doing filters correctly.)
(6) Sobel / Scharr magnitude operator (see what Matlab does with color).

5.1. **A rank 1 decomposition.** Truncation can be a very powerful tool we can use to reduce the size of our LUTs. We of course have to be aware of the errors we sacrifice for computation time. In effort to reduce the size of our LUTs before truncation we explore a decomposition of our kernal K.
Idea: For any $(n \times n)$ matrix K that has rank 1 we can decompose the matrix K as:

$$K = c * b^T$$

where $c, b \in \mathbb{R}^n$ and "*" represents the matrix convolution. The convolution of two $n-$ *dimensional* vectors is actually the same a multiplication of the two vectors.
Example: Consider the Low-pass filter K below and the decomposition.

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Exploiting symmetry (or not necessarily).

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Optimization toolbox in Matlab: $fminunc(f, x_o)$.

Try $||H_{\text{high-pass kernel}} - M * N||_{\text{FRO or something else}}$.

mario: is writing the code.

5.2. **More matrix decomposition.** Morgan had an idea to take advantage of this:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

5.3. **Sobel magnitude filter.** Ran an RGB picture with Mario's color code and the results look similar as the one in Wikipedia (but not as nice), and the results look acceptable even with truncating to 5,6 digits.

6. Saturday

6.1. **A rank 2 decomposition approximation.** Recall that a rank 1 decomposition of our kernal K had the form

$$K = c * b^T$$

where $c, b \in \mathbb{R}^n$. This resulted in two LUTs of size 2^{24} bytes instead of the full LUT of size 2^{72} bytes. This drastically reduces the size of the LUT before any truncation but it requires us to have a rank 1 kernal K. What can we do for a filter that is not rank 1?

Question: Can we decompose the filter K in a better way?

Consider the decomposition of our (3×3)filter K into the convolution:

$$K = A * B$$

where

$$A = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 0 \end{bmatrix}, B = \begin{bmatrix} g & h & i \\ k & m & n \\ 0 & 0 & 0 \end{bmatrix}$$

Ran an algorithm using fmin

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 0 \end{bmatrix} * \begin{bmatrix} g & h & i \\ j & k & \ell \\ 0 & 0 & 0 \end{bmatrix}$$

$$A = \begin{bmatrix} -0.372739294911747 & -0.443690961374357 & -0.457110102971229 \\ -0.509997537573061 & 2.683304035822869 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.681646089142412 & -0.509741093569434 \\ -0.456856367067759 & -0.443428387610633 & -0.372501463359094 \end{bmatrix}$$

```
conv2(A,B,'same')
```

$$\begin{bmatrix} -0.999554872469786 & -0.999821595552767 & -0.999640004082445 \\ -0.999646676848929 & 8.000063237563769 & -0.999819205287410 \\ -0.999737147772875 & -0.999878353018540 & -0.999534679981381 \end{bmatrix}$$

```
norm(F-conv2(A,B,'same'),'fro')
ans =
```

$9.063656510413503e-04$

7. ideas about what to do (Emily)

- Filter color images with these http://r0k.us/graphics/kodak/

8. Monday

emily: todo: Check errors $||L(A-B)||_F$ where L is a low-pass filter. Our eyes are essentially a low-pass filter. Chai thinks that PSNR-HVS is essentially this.

References

[Wikipedia/Sobel] Sobel operator