ECE 6310 Introduction to Computer Vision

Lab #1 – Convolution, separable filters, sliding windows

Submitted by:
Christin Wilson
CUID: C21515896

In this project each student must implement three versions of a 7x7 mean filter. The first version should use basic 2D convolution. The second version should use separable filters (1x7 and 7x1). The third version should use separable filters and a sliding window.

All three versions of the filter should produce the exact same output. This must be verified by comparing the images using “diff” or a similar program and showing the method used and result. Each version of the filter should be timed, and the typical amount of time reported (for example, the average amount of time over ten runs).

Original image:

Smoothed image:

I got identical smoothed images for all the three versions. This is shown later by applying diff across these images.

Time Comparisons: (in picoseconds)

No.	2D Convolution	Separable filters	Separable filters + Sliding Window
1	32.246	11.276	6.754
2	31.295	13.441	7.361
3	32.160	11.949	8.316
4	32.204	14.020	6.874
5	32.938	12.795	7.401
6	36.581	12.826	6.873
7	32.123	13.341	6.727
8	34.216	11.160	7.105
9	34.261	13.581	7.329
10	34.747	13.123	7.508
Avg:	**33.2771**	**12.7512**	**7.2248**

From the average time it took for each version, we can see that the separable filters takes much less time (12.7512 picoseconds) when compared to the 2d convolution filter (33.2771 picoseconds). This can further be reduced by using the sliding window approach which took an average of 7.2248 picoseconds.

Diff Checker:

```
Christins-Mac:computer vision christinwilson$ diff smoothed.ppm smoothed77.ppm
files smoothed.ppm and smoothed77.ppm differ
Christins-Mac:computer vision christinwilson$ diff smoothed77.ppm smoothed1d.ppm
Christins-Mac:computer vision christinwilson$ diff smoothed77.ppm smoothedslide.ppm
Christins-Mac:computer vision christinwilson$ diff smoothed1d.ppm smoothedslide.ppm
Christins-Mac:computer vision christinwilson$ diff smoothed77.ppm bridge.ppm
files smoothed77.ppm and bridge.ppm differ
Christins-Mac:computer vision christinwilson$
```

As already stated above, comparing the 3 versions of the images from each other using ‘diff’ proves that the generated images are identical with each other but different from the original and the output image generated by 3x3 filter.

Here the file names are:

bridge.ppm	input image
smoothed.ppm	2D convolution generated (3x3)
smoothed77.ppm	2D convolution generated (7x7)
smoothed1d.ppm	Separable filters generated
smoothedslide.ppm	Seperable filters and sliding window generated

Conclusion:

I implemented three versions of a 7x7 mean filter. For the basic 2D convolution, I used the same C code as the one that was provided for 3x3 convolution and changed the dimensions of the filter accordingly. For the separable filter, I created a temporary image generated after the first 1d filter and then ran the other filter on this temporary image. To not lose information I performed the division together at the second filter stage. since I didn't find the mean in the first filter loop and just calculated the sum. The temporary image I created was declared as int data type so as to store the value because sum can go above 255 here. For the third filter, I calculated the sum for the first filter values and from then I removed the fourth pixel behind the center of the filter and added the new pixel that was 3 spaces ahead.

All three algorithms generated the same output image that was verified using the diff command

By getting the time for each piece of code, we can prove that the **sliding window approach is the most efficient method in terms of time** and takes about 1/5th the time of a basic 2d convolution filter in my case.

Code:

2D Convolution

time-smooth77.c

```c
/*
** This program reads bridge.ppm, a 512 x 512 PPM image.
** It smooths it using a standard 7x7 mean filter.
** The program also times the piece of code.
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
int main()

{
FILE    *fpt;
unsigned char *image;
unsigned char *smoothed;
char    header[320];
int   ROWS,COLS,BYTES;
int   r,c,r2,c2,sum;
struct timespec tp1,tp2;

  /* read image */
if ((fpt=fopen("bridge.ppm","rb")) == NULL)
  {
  printf("Unable to open bridge.ppm for reading\n");
  exit(0);
  }
fscanf(fpt,"%s %d %d %d",header,&COLS,&ROWS,&BYTES);
if (strcmp(header,"P5") != 0  ||  BYTES != 255)
  {
  printf("Not a greyscale 8-bit PPM image\n");
  exit(0);
  }
image=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
header[0]=fgetc(fpt); /* read white-space character that separates header */
fread(image,1,COLS*ROWS,fpt);
fclose(fpt);

  /* allocate memory for smoothed version of image */
smoothed=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));

  /* query timer */
clock_gettime(CLOCK_REALTIME,&tp1);
printf("%ld %ld\n",(long int)tp1.tv_sec,tp1.tv_nsec);

  /* smooth image, skipping the border points */
for (r=3; r<ROWS-3; r++)
  for (c=3; c<COLS-3; c++)
    {
    sum=0;
    for (r2=-3; r2<=3; r2++)
      for (c2=-3; c2<=3; c2++)
        sum+=image[(r+r2)*COLS+(c+c2)];
    smoothed[r*COLS+c]=sum/49;
    }
  /* query timer */
clock_gettime(CLOCK_REALTIME,&tp2);
printf("%ld %ld\n",(long int)tp2.tv_sec,tp2.tv_nsec);

  /* report how long it took to smooth */
printf("%ld\n",tp2.tv_nsec-tp1.tv_nsec);

  /* write out smoothed image to see result */
fpt=fopen("smoothed77.ppm","w");
fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
fwrite(smoothed,COLS*ROWS,1,fpt);
fclose(fpt);

}
```

Seperable filters

time-smooth1d.c

```c
/*
** This program reads bridge.ppm, a 512 x 512 PPM image.
** It smooths it using seperable mean filters (7x1 and 1x7).
** The program also times the piece of code.
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
int main()
{
FILE    *fpt;
unsigned char *image;
unsigned char *smoothed;
int *TEMP;
char    header[320];
int   ROWS,COLS,BYTES;
int   r,c,r2,c2;
int sum;
struct timespec tp1,tp2;

  /* read image */
if ((fpt=fopen("bridge.ppm","rb")) == NULL)
  {
  printf("Unable to open bridge.ppm for reading\n");
  exit(0);
  }
fscanf(fpt,"%s %d %d %d",header,&COLS,&ROWS,&BYTES);
if (strcmp(header,"P5") != 0  ||  BYTES != 255)
  {
  printf("Not a greyscale 8-bit PPM image\n");
  exit(0);
  }
image=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
header[0]=fgetc(fpt); /* read white-space character that separates header */
fread(image,1,COLS*ROWS,fpt);
fclose(fpt);

  /* allocate memory for temp version of image. int used to store values above 255 */
TEMP=(int *)calloc(ROWS*COLS,sizeof(int));
  /* allocate memory for smoothed version of image */
smoothed=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));

  /* query timer */
clock_gettime(CLOCK_REALTIME,&tp1);
printf("%ld %ld\n",(long int)tp1.tv_sec,tp1.tv_nsec);

  /* smooth image, skipping the border points */
for (c=0; c<COLS; c++)
  for (r=3; r<ROWS-3; r++)
    {
    sum=0;
    for (r2=-3; r2<=3; r2++)
        sum+=image[(r+r2)*ROWS+(c)];
    TEMP[r*COLS+c]=sum;
    }
for (r=3; r<ROWS-3; r++)
  for (c=3; c<COLS-3; c++)
    {
    sum=0;
    for (c2=-3; c2<=3; c2++)
      sum+=TEMP[(r)*ROWS+(c+c2)];
    smoothed[r*COLS+c]=sum/49;
  }

  /* query timer */
clock_gettime(CLOCK_REALTIME,&tp2);
printf("%ld %ld\n",(long int)tp2.tv_sec,tp2.tv_nsec);

  /* report how long it took to smooth */
printf("%ld\n",tp2.tv_nsec-tp1.tv_nsec);

  /* write out smoothed image to see result */
fpt=fopen("smoothed1D.ppm","w");
fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
fwrite(smoothed,COLS*ROWS,1,fpt);
fclose(fpt);
}
```

Separable filters with Sliding window

```
47    /* smooth image, skipping the border points */
48   for (c=0; c<COLS; c++)
49     {sum=0;
50     for (r=3; r<ROWS-3; r++)
51       {if(r==3)
52         {
53           for (r2=-3; r2<=3; r2++)
54           sum+=image[(r+r2)*COLS+(c)];
55         }
56       else{
57           sum+=image[(r+3)*COLS+(c)];
58           sum-=image[(r-4)*COLS+(c)];
59
60       }
61       TEMP[r*COLS+c]=sum;
62       }}
63
64   for (r=3; r<ROWS-3; r++)
65     {
66     sum=0;
67     for (c=3; c<COLS-3; c++)
68       {
69       if(c==3)
70         {
71           for (c2=-3; c2<=3; c2++)
72           sum+=TEMP[(r)*COLS+(c+c2)];
73         }
74       else{
75           sum+=TEMP[(r)*COLS+(c+3)];
76           sum-=TEMP[(r)*COLS+(c-4)];
77
78       }
79         smoothed[r*COLS+c]=sum/49;
80     }}
81
```