

```

1  /* C code for ECE 6310 Lab 1: Convolution, Separable filters, sliding windows */
2  #include<stdio.h>
3  #include<string.h>
4  #include<stdlib.h>
5  #include<time.h>
6
7  int main(int argc, char* argv[])
8  {
9      FILE          *fpt,*fpt_1,*fpt_2,*fpt_3;
10     unsigned char  *image;
11     unsigned char  *smoothed,*smoothed_1,*smoothed_final, *sep_smoothed_1,
12     *sep_smoothed_final;
13     char           header[320];
14     int            ROWS,COLS,BYTES;
15     int            r,c,r2,c2,sum,prev_val;
16     struct timespec tp1,tp2;
17
18     // Read and check original image:
19     if ((fpt=fopen("bridge.ppm","rb"))==NULL)
20     {
21         printf("Unable to open bridge.ppm for reading.\n");
22         exit(0);
23     }
24     fscanf(fpt,"%s %d %d %d",header,&COLS,&ROWS,&BYTES);
25
26     if(strcmp(header,"P5")!=0 || BYTES!=255)
27     {
28         printf("Not a greyscale 8-bit PPM image.\n");
29         exit(0);
30     }
31     image=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
32     header[0]=fgetc(fpt); //read whitespace char that separates header
33     fread(image,1,COLS*ROWS,fpt);
34     fclose(fpt);
35
36     //allocate memory for smoothed versions:
37     smoothed=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
38     smoothed_1=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
39     smoothed_final=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
40     sep_smoothed_1=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
41     sep_smoothed_final=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
42
43     // Assign a 0(black) to all border points in image:
44     for(r=0;r<3;r++)
45     {
46         for(c=0;c<COLS;c++)
47         {
48             image[r*COLS+c]=0;
49         }
50     }
51     for(r=ROWS-3;r<ROWS-1;r++)
52     {
53         for(c=0;c<COLS;c++)
54         {
55             image[r*COLS+c]=0;
56         }
57     }
58     for(r=3;r<ROWS-3;r++)
59     {
60         for(c=0;c<3;c++)
61         {
62             image[r*COLS+c]=0;
63         }
64     }
65     for(r=3;r<ROWS-3;r++)
66     {
67         for(c=COLS-3;c<COLS;c++)
68         {
69             image[r*COLS+c]=0;
70         }
71     }
72 }

```

```

69     }
70 }
71 smoothed=image;           // smoothed image for 2D convolution
72 smoothed_1=image;         // intermediary smoothed image after sliding window
73 smoothed_final=image;     // final smoothed image after sliding window
74 sep_smoothed_1=image;     // intermediary smoothed image after separable filters
75 sep_smoothed_final=image; // final smoothed image after separable filters
76
77 clock_gettime(CLOCK_REALTIME,&tp1); // Query timer for 2D convolution
78 printf("Total time for 2D convolution:\t");
79 for (r=3;r<ROWS-3;r++) // Excluding border row points
80 {
81     for (c=3;c<COLS-3;c++) // Excluding border column points
82     {
83         sum=0;
84         for (r2=-3;r2<=3;r2++)
85         {
86             for(c2=-3;c2<=3;c2++)
87             {
88                 sum+= image[(r+r2)*COLS+(c+c2)]; // 2D filter of size 7*7
89             }
90         }
91         smoothed[r*COLS+c]=sum/49;
92     }
93 }
94 clock_gettime(CLOCK_REALTIME,&tp2); // Query timer for 2D convolution
95 printf("%ld\n",tp2.tv_nsec-tp1.tv_nsec); // Report time to smooth using 2D convolution
96
97 clock_gettime(CLOCK_REALTIME,&tp1); // Query timer for separable filters
98 printf("Total time for sep filters:\t");
99 for(r=3;r<ROWS-3;r++)
100 {
101     for(c=3;c<COLS-3;c++)
102     {
103         sum=0;
104         for(c2=-3;c2<=3;c2++) // 1D filter of size 1*7
105         {
106             sum+= image[r*COLS+(c+c2)]; // Taking sum across column values
107         }
108         sep_smoothed_1[r*COLS+c]=sum/7; // Intermediary result image
109     }
110 }
111 for(r=3;r<ROWS-3;r++)
112 {
113     for(c=3;c<COLS-3;c++)
114     {
115         sum=0;
116         for(r2=-3;r2<=3;r2++) // 1D filter of size 7*1
117         {
118             sum+= sep_smoothed_1[(r+r2)*COLS+c]; // Taking sum across row values
119         }
120         sep_smoothed_final[r*COLS+c]=sum/7; // Final result image
121     }
122 }
123 clock_gettime(CLOCK_REALTIME,&tp2); // Query timer for separable filters
124 printf("%ld\n",tp2.tv_nsec-tp1.tv_nsec); // report time to smooth using separable
125 filters
126
127 clock_gettime(CLOCK_REALTIME,&tp1); // Query timer for sliding window
128 printf("Total time for sep filters and sliding window:\t");
129 sum=0;
130 prev_val=0;
131 for(r=3;r<ROWS-3;r++)
132 {
133     for(c=3;c<COLS-3;c++)
134     {
135         if(r==c==3) // Calculating sum for first 1D sliding window
136         {
137             for(c2=-3;c2<=3;c2++) // Separable 1D filter of size 1*7

```

```

137         {
138             sum+= image[r*COLS+(c+c2)];
139             if (!prev_val)
140             {
141                 prev_val=sum;           // Updating value for oldest column
142             }
143         }
144         smoothed_1[r*COLS+c]=sum/7; // Intermediary result image
145     }
146     else
147     {
148         sum-=prev_val;           // Subtracting oldest column value from sum
149         prev_val=image[r*COLS+c-3]; // Updating oldest column for new window
150         sum+= image[r*COLS+(c+3)]; // adding last column value to the existing
151         sum
152         smoothed_1[r*COLS+c]=sum/7; // Intermediary result image
153     }
154 }
155 sum=0;
156 prev_val=0;
157 for (c=3; c<COLS-3; c++)
158 {
159     for (r=3; r<ROWS-3; r++)
160     {
161         if (r==c==3) // Calculating sum for first 1D sliding window
162         {
163             for (r2=-3; r2<=3; r2++) // Separable 1D filter of size 7*1
164             {
165                 sum+= smoothed_1[(r+r2)*COLS+c];
166                 if (!prev_val)
167                 {
168                     prev_val=sum; // Updating value for oldest row
169                 }
170             }
171             smoothed_final[r*COLS+c]=sum/7; // Final result image
172         }
173         else
174         {
175             sum-=prev_val; // Subtracting oldest row value from sum
176             prev_val=smoothed_1[(r-3)*COLS+c]; // Updating oldest row for new window
177             sum+= smoothed_1[(r+3)*COLS+c]; // Adding last row value to sum
178             smoothed_final[r*COLS+c]=sum/7; // Final result image
179         }
180     }
181 }
182 clock_gettime(CLOCK_REALTIME, &tp2); // Query timer for sliding window
183 printf("%ld\n", tp2.tv_nsec-tp1.tv_nsec); // report time to smooth using sliding window
184
185 // write out smoothed images to see result:
186 fpt_1=fopen("smoothed_by_convolution_1.ppm", "wb");
187 fpt_2=fopen("smoothed_by_sep_filters_1.ppm", "wb");
188 fpt_3=fopen("smoothed_by_sliding_window_1.ppm", "wb");
189
190 fprintf(fpt_1, "P5 %d %d 255\n", COLS, ROWS);
191 fprintf(fpt_2, "P5 %d %d 255\n", COLS, ROWS);
192 fprintf(fpt_3, "P5 %d %d 255\n", COLS, ROWS);
193
194 fwrite(smoothed, COLS*ROWS, 1, fpt_1); // final image after 2D convolution version
195 fwrite(sep_smoothed_final, COLS*ROWS, 1, fpt_2); // final image after separable filters
196 // version
197 fwrite(smoothed_final, COLS*ROWS, 1, fpt_3); // final image after sliding window version
198
199 fclose(fpt_1);
200 fclose(fpt_2);
201 fclose(fpt_3);
202 }

```