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

```
69
              }
 70
          }
 71
                                   // smoothed image for 2D convolution
          smoothed=image;
 72
          smoothed 1=image;
                                   // intermediary smoothed image after sliding window
                                   // final smoothed image after sliding window
 73
          smoothed final=image;
 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, &tpl); // 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
              for (c=3;c<COLS-3;c++)
 81
                                        // 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
                                            // 1D filter of size 1*7
                   for (c2=-3; c2 <= 3; c2++)
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
                                            // 1D filter of size 7*1
                   for(r2=-3;r2<=3;r2++)
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
          filters
125
126
          clock gettime(CLOCK REALTIME,&tp1); // Query timer for sliding window
127
          printf("Total time for sep filters and sliding window:\t");
128
          sum=0;
129
          prev val=0;
130
          for (r=3; r<ROWS-3; r++)
131
132
              for (c=3; c<COLS-3; c++)
133
               {
134
                   if(r==c==3)
                                         // Calculating sum for first 1D sliding window
135
                                                     // Separable 1D filter of size 1*7
136
                       for(c2=-3;c2<=3;c2++)
```

```
137
                       {
138
                           sum+= image[r*COLS+(c+c2)];
139
                           if (!prev val)
140
                           {
141
                               prev val=sum;
                                                     // Updating value for oldest column
142
                           }
143
                       smoothed 1[r*COLS+c]=sum/7; // Intermediary result image
144
145
                   }
146
                   else
147
                   {
148
                       sum-=prev val;
                                                    // Subtracting oldest column value from sum
                       prev val=image[r*COLS+c-3]; // Updating oldest column for new window
149
                       sum+= image[r*COLS+(c+3)]; // adding last column value to the existing
150
151
                       smoothed 1[r*COLS+c]=sum/7; // Intermediary result image
152
                   }
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
                       for (r2=-3; r2 <= 3; r2++) // Separable 1D filter of size 7*1
163
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
          version
196
          fwrite(smoothed final, COLS*ROWS, 1, fpt 3);// final image after sliding window version
197
198
          fclose(fpt 1);
199
          fclose(fpt 2);
200
          fclose(fpt 3);
201
      }
202
```