RELATÓRIO

Nome: Henry Emanuel Leal Cagnini
Tema: Exercícios de Aplicação de Filtros

c) Crie um algoritmo para aplicar uma convolução em **um** ponto de uma imagem. A função deve receber as coordenadas do ponto, a matriz de convolução como um vetor de floats e a largura da matriz. Assume-se que a matriz é quadrada. O vetor deve armazenar a matriz desta forma:

MATRIZ

A	B	C
D	E	F
G	H	I

VETOR

A	B	C	D	E	F	G	H	I

Após a construção da função, aplique cada uma das matrizes de convolução abaixo em uma imagem. Note que a matriz deve passar em todos os pontos da imagem, excluindo-se as bordas.

-1	-1	-1
-1	9	-1
-1	-1	-1

-2	-1	0
-1	1	1
0	1	2

1/273 *

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Enunciado:
Resposta: O algoritmo faz exatamente o que o enunciado solicita: aplica um filtro convolucional (na forma de um *dot product*) sobre toda a figura. A figura utilizada é uma com ruído, portanto, o ruído é removido antes da aplicação do filtro convolucional, através de um filtro mediano. É importante salientar que o filtro convolucional é aplicado pixel-a-pixel; isto é, o filtro não pula N pixels para o lado/acima após a aplicação, mas sim apenas 1 pixel. O algoritmo faz uso de uma imagem *buffer* para não influenciar nos valores de outras janelas.
Código desenvolvido:

```c
#define FILTER_SIDE 3
/**
 * Troca os valores entre duas variáveis.
 * @param a primeiro valor
 * @param b segundo valor
 */
void swap(unsigned char* a, unsigned char* b) {
  unsigned char t = *a;
  *a = *b;
  *b = t;
}
/**
 * Parte que faz propriamente a ordenação de valores.
 *
 * @param arr Array
 * @param low Menor índice desse array
 * @param high Maior índice desse array
 * @return
 */
unsigned char partition(unsigned char arr[], int low, int high) {
  unsigned char pivot = arr[high];
  int i = (low - 1);
  for (int j = low; j <= high - 1; j++) {
    if (arr[j] < pivot) {
      i++;
      swap(&arr[i], &arr[j]);
    }
  }
  swap(&arr[i + 1], &arr[high]);
  return (i + 1);
}
```

```
/**
 * Algoritmo principal do quicksort. Ordena um array in-place.
 *
 * @param arr Array de valores a serem ordenados.
 * @param low Menor índice desse array
 * @param high Maior índice desse array
 */
void quickSort(unsigned char arr[], int low, int high) {
  if (low < high) {
    int pi = partition(arr, low, high);
    quickSort(arr, low, pi - 1);
    quickSort(arr, pi + 1, high);
  }
}
/**
 * Pega (se possível) o valor grayscale de um pixel da imagem.
 *
 * @param i Posição no eixo x do pixel. Se exceder a imagem, o valor da borda
 *          na altura j será retornado no lugar.
 * @param j Posição no eixo y do pixel. Se exceder a imagem, o valor da borda
 *          na largura i será retornado no lugar.
 * @param img Imagem de onde o pixel será retirado
 * @return O valor (grayscale) do pixel na posição (i, j).
 */
unsigned char getGrayscalePixel(int i, int j, ImageClass img) {
  unsigned char r, g, b;
  if(i < 0) {
    i = 0;
  } else if(i >= img.SizeX()) {
    i = img.SizeX() - 1;
  }
  if(j < 0) {
    j = 0;
  } else if(j >= img.SizeY()) {
    j = img.SizeY() - 1;
  }
  img.ReadPixel((GLint)i, (GLint)j, r, g, b);
  unsigned char gray_pixel = (unsigned char)((0.3 * r) + (0.59 * g) + (0.11 * b));
  return gray_pixel;
}
/**
 * Captura uma determinada região da imagem.
 *
 * @param side Lado da máscara.
 * @param centerX Coordanada x do centro da máscara
 * @param centerY Coordanada y do centro da máscara
 * @param img Imagem
 * @return a máscara, com valores atualizados.
 */
unsigned char *getMask(int side, int centerX, int centerY, ImageClass img) {
  int min_x = centerX - (int)(side / 2);
  int max_x = centerX + (int)(side / 2);
  int min_y = centerY - (int)(side / 2);
  int max_y = centerY + (int)(side / 2);
  unsigned char *mask = new unsigned char [side * side];
  int counter = 0;
  for(int i = min_x; i <= max_x; i++) {
    for(int j = min_y; j <= max_y; j++) {
      mask[counter] = getGrayscalePixel(i, j, img);
      counter += 1;
    }
  }
  return mask;
}
```

```
/**
 * Pega o valor mediano de uma posição na imagem.
 *
 * @param side Lado da máscara
 * @param i Posição central no eixo x da máscara
 * @param j Posição central no eixo y da máscara
 * @param img Imagem
 * @return O valor mediano, quando aplicada uma máscara (de centro (i, j)) sobre uma imagem.
 */
unsigned char medianFilterToRegion(int side, int i, int j, ImageClass img) {
  unsigned char *mask = getMask(side, i, j, img);
  quickSort(mask, 0, side * side);
  if(((side * side) % 2) == 0) {
    return (mask[(side * side) / 2] + mask[((side * side) / 2) + 1]) / 2;
  }
  unsigned char new_pixel = mask[(side * side) / 2];
  delete[] mask;
  return new_pixel;
}
/**
 * Aplica um filtro mediano sobre uma figura inteira. Gera uma nova figura.
 * @param img Imagem que será processada.
 * @return A mesma imagem, agora pós passada do filtro mediano.
 */
ImageClass medianFilter(ImageClass img) {
  int length = FILTER_SIDE * FILTER_SIDE;
  unsigned char *window = new unsigned char [length];
  ImageClass newImg = ImageClass(img.SizeX(), img.SizeY(), img.Channels());
  img.CopyTo(&newImg);
  for (int i = 0; i < img.SizeX(); i++) {
    for (int j = 0; j < img.SizeY(); j++) {
      unsigned char new_pixel = medianFilterToRegion(FILTER_SIDE, i, j, img);
      newImg.DrawPixel(i, j, new_pixel);
    }
  }
  delete[] window;
  return newImg;
}
/**
 * Aplica um filtro convolucional à uma imagem, na forma de um dot product.
 * @param side Lado do filtro
 * @param filter Filtro propriamente dito, de tamanho side * side
 * @param i Centro do filtro no eixo x
 * @param j Centro do filtro no eixo y
 * @param img Imagem onde o filtro será aplicado
 * @return O pixel resultante da aplicação do filtro
 */
unsigned char applyFilter(int side, float *filter, int i, int j, ImageClass img) {
  unsigned char *mask = getMask(side, i, j, img);
  float sum = 0;
  for(int k = 0; k < side * side; k++) {
    sum += (filter[k] * (float)mask[k]);
  }
  return (unsigned char)min(max((float)0, sum), (float)255);
}
/**
 * Aplica o filtro convolucional sobre uma imagem. Retorna uma nova imagem no lugar.
 *
 * @param img Imagem original
 * @param side Lado do filtro
 * @param filter O filtro convolucional
 * @return Uma nova imagem, processada pelo filtro
 */
ImageClass convolutionalFilter(ImageClass img, int side, float *filter) {
  ImageClass newImg = ImageClass(img.SizeX(), img.SizeY(), img.Channels());
  img.CopyTo(&newImg);
  for (int i = 0; i < img.SizeX(); i++) {
    for (int j = 0; j < img.SizeY(); j++) {
      unsigned char new_pixel = applyFilter(side, filter, i, j, img);
      newImg.DrawPixel(i, j, new_pixel);
    }
  }
  return newImg;
}
```

```
// ************************************************************************
//  void init(void)
// ************************************************************************
void init(int argc, char **argv) {
  int r;
  // Carrega a uma imagem
  if (argc < 2) {
    cout << "Usage:" << endl <<
      "\t./example_image_manipulation <image_path>" << endl;
    throw 6;
  }
  r = original.Load(argv[1]); // Carrega uma imagem
  if (!r) {
    exit(1); // Erro na carga da imagem
  } else {
    cout << ("Imagem carregada!\n");
  }
  float firstFilter[] = {-1, -1, -1, -1, 9, -1, -1, -1, -1};
  float secondFilter[] = {-2, -1, 0, -1, 1, 1, 0, 1, 2};
  float thirdFilter[] = {0.003663  , 0.01465201, 0.02564103, 0.01465201, 0.003663  ,
              0.01465201, 0.05860806, 0.0952381 , 0.05860806, 0.01465201,
              0.02564103, 0.0952381 , 0.15018315, 0.0952381 , 0.02564103,
              0.01465201, 0.05860806, 0.0952381 , 0.05860806, 0.01465201,
              0.003663  , 0.01465201, 0.02564103, 0.01465201, 0.003663};
  original = medianFilter(original);
  img1 = convolutionalFilter(original, 3, &firstFilter[0]);
  img2 = convolutionalFilter(original, 3, &secondFilter[0]);
  img3 = convolutionalFilter(original, 5, &thirdFilter[0]);
  blankImage.SetSize(LARGURA_JAN, ALTURA_JAN, original.Channels());
}
```

Exemplos de Processamento:
Aplicação do filtro convolucional sobre uma imagem pré-filtrada:

- Canto inferior esquerdo: imagem original;
- Canto inferior direito: aplicação do firstFilter (função init);
- Canto superior esquerdo: aplicação do secondFilter (função init);
- Canto superior direito: aplicação do thirdFilter (função init).