Project 1 - O TAD image8bit

Universidade de Aveiro

Algoritmos e Estruturas de Dados

Luis Sousa nMec:108583,
Gonçalo Oliveira nMec: 108405

No desenvolvimento deste projeto necessitamos de criar/modificar as seguintes funções:
ImageCreate; ImageDestroy; ImageStats; ImageValidRect; Static inline int G; ImageNegative; ImageThreshold; ImageBrighten; ImageRotate; ImageMirror; ImageCrop; ImagePaste; ImageBlend; ImageMatchSubImage; ImageLocateSubImage; ImageBlur;

```
Image ImageCreate(int width, int height, uint8 maxval) { ///
  assert (width >= 0);
  assert (height >= 0);
  assert (0 < maxval && maxval <= PixMax);
  Image image = malloc(sizeof(*image));
  if(!check(image != NULL, "Erro Malloc")) {
    return NULL;
  }
  image -> width = width;
  image -> height = height;
  image -> maxval = maxval;
  image -> pixel = malloc(width * height * sizeof(uint8));
  if(!check(image -> pixel != NULL, "Sem Pixeis")) {
      free(image);
      return NULL;
    }return image;
}
```

Após criar os macros prosseguimos para a criação da função ImageCreate. Tivemos que alocar espaço para a struct e em seguida verificamos se o Alloc realmente aconteceu. Após tal acontecer, alocamos também espaço para o array do valor dos pixeis verificando de seguida se o allox se verificou.

```
void ImageDestroy(Image* imgp) {
  assert (imgp != NULL);
  Image img = *imgp;
  free(img -> pixel);
  free(img);
  *imgp = NULL;
}
```

Na imageDestroy associamos o ponteiro a struc img e em seguida libertamos o espaço dos pixeis e o espaço da struct respetivamente.

```
void ImageStats(Image img, uint8* min, uint8* max) {
  int i,j;
  assert (img != NULL);
  for(i = 0; i < img -> width; i++){
    for(j = 0; j < img -> width; j++){
      uint8 px = ImageGetPixel(img,i,j);
      if(px < *min) *min = px;
      if(px > *max) *max = px;
    }
  }
}
```

Ao longo deste projeto definimos essa sequência de for's de modo a percorrer todos os pixeis presentes nas imagens.
Após isso verificamos quais são os maiores e os menores valores associados.

```
int ImageValidRect(Image img, int x, int y, int w, int h) {
  assert (img != NULL);
  if(x + w-1 > img -> width-1 || y+h-1 > img->height-1){
    return -1;
  }
  if (x + w > img -> width || y + h > img ->height){
    return -1;
  }
  return 1;
}
```

No ImageValidRect verificamos se a área da imagem se encontra dentro dos limites e de seguida fizemos o mesmo mas tendo em conta a altura e a largura, se tal não se verificasse dava-se um return de -1 que significa que a área retangular não está completamente dentro da imagem.

```
static inline int G(Image img, int x, int y) {
  int index;
  index = x + (img -> width*y);
  assert (0 <= index && index < img -> width*img -> height);
  return index;
}
```

A segunda linha da função foi adicionada de modo a calcular o índice de um ponto num array bidimensional (x,y).

```
void ImageNegative(Image img) { ///
  int x,y;
  assert (img != NULL);
  for(x=0; x<img->width; x++){
    for(y=0; y<img->height; y++){
      ImageSetPixel(img,x,y,img->maxval-ImageGetPixel(img,x,y));
    }
  }
}
```

Tal como foi explicado em cima, passamos por todos os pixeis da imagem de modo à linha em questão ser substituída por cada valor do pixel na imagem pelo seu complemento negativo.

```
void ImageThreshold(Image img, uint8 thr) { ///
  int x,y;
  assert (img != NULL);
  for(x=0; x<img->width; x++){
    for(y=0; y<img -> height; y++){
      uint8 pixelValue = ImageGetPixel(img, x, y);
      uint8 newPixelValue = (pixelValue < thr) ? 0 : img->maxval;
      ImageSetPixel(img, x, y, newPixelValue);
    }
  }
}
```

Na Threshold após passar pelos pixeis todos vamos armazenar o valor dos pincéis em pixelValue para determinar o seu novo valor, recorremos ao operando '?' que serve de if else que fará com que se o valor for menor que o limite o valor do pixel será 0 caso contrário será o valor máximo. Após tal acontecer definimos o valor no ponto (x,y) para o novo valor

```
void ImageBrighten(Image img, double factor) { ///
  int x,y;
  assert (img != NULL);
  assert (factor >= 0.0); //estava comentado originalmente
  for(x=0;x<img -> width; x++){
    for(y=0; y<img -> height; y++){
      uint8 bright = ImageGetPixel(img,x,y);
      if (factor != 1.0){
        bright = (uint8)((bright * factor) + 0.5);
        ImageSetPixel(img, x, y, (bright > img->maxval) ? img->maxval : bright);
      }
    }
  }
}
```

Na ImageBrighten vamos armazenar o valor do pixel e verificar o seu fator de brilho, se for diferente de 1 iremos calcular o novo valor adicionando 0.5 por questões de arredondamento pois não queremos que o programa dê floor automaticamente, a adição de 0.5 previne tal coisa. A última linha define o valor do pixel para o novo valor ajustado, mas se o novo valor for maior que o valor máximo permitido este será ajustado para o valor máximo.

```
Image ImageRotate(Image img) { ///
  int x,y;
  assert (img != NULL);
  Image image = ImageCreate(img ->height, img->width, img->maxval);
  for(x=0; x<img -> width; x++){
    for(y=0; y<img->height;y++){
      ImageSetPixel(image,y,img -> width-1-x,ImageGetPixel(img,x,y));
    }
  }return image;
}
```

A função Rotate é usada para pegar na imagem e passar a linha da img para a coluna da image definida previamente.

```
Image ImageMirror(Image img) { ///
  int x,y;
  assert (img != NULL);
  Image image = ImageCreate(img->width, img->height, img->maxval);
  for(x=0; x<img -> width; x++){
    for(y=0; y<img -> height; y++){
      ImageSetPixel(image, img -> width-1-x, y, ImageGetPixel(img,x,y));
    }
  }return image;
}
```

A função Mirror vai pegar nos pixeis e inverte-los na posição horizontal.

```c
Image ImageCrop(Image img, int x, int y, int w, int h) { ///
  int i;
  assert (img != NULL);
  assert (ImageValidRect(img, x, y, w, h));
  Image image = ImageCreate(w,h,img->maxval);
  for(i = x; i < x+w; i++){
    for(int j = y; j < y+h; j++){
      ImageSetPixel(image,i-x,j-y,ImageGetPixel(img,i,j));
    }
  }return image;
}
```

Esta função vai verificar que a imagem não é nula. Após verificação vai ver se a área de recorte está dentro dos limites da imagem. Se tal se confirmar uma nova imagem será criada e passaremos por todos os pixeis que vão ser recortados de modo a copiá-los e colocá-los na nova posição.

```c
void ImagePaste(Image img1, int x, int y, Image img2) { ///
  int i,j;
  assert (img1 != NULL);
  assert (img2 != NULL);
  assert (ImageValidRect(img1, x, y, img2->width, img2->height));
  for(i = x; i < x+img2->width; i++){
    for(j = y; j < y+img2->height; j++){
      ImageSetPixel(img1,i,j,ImageGetPixel(img2,i-x,j-y));
    }
  }
}
```

A função do paste vai verificar se ambas as fotos, img1 e img2 já definidas não são nulas. Se tal se verificar o ciclo for passará por todos os pixeis da img2 de modo a substituir os pixeis da img1.

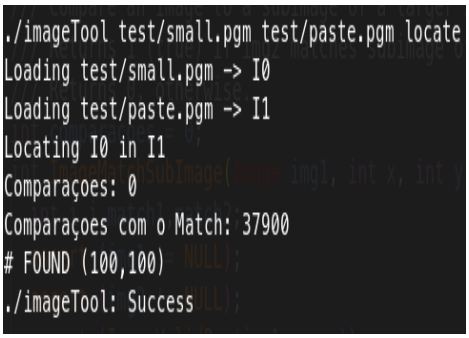
```c
void ImageBlend(Image img1, int x, int y, Image img2, double alpha) {
  int i,j;
  double blend1,blend2;
  assert (img1 != NULL);
  assert (img2 != NULL);
  assert (ImageValidRect(img1, x, y, img2->width, img2->height));
  if (alpha > 1.0){
    ImagePaste(img1,x,y,img2);
  }else if(alpha >= 0.0 && alpha <= 1.0){
    for(i = x; i-x < img2 -> width; i++){
      for(j = y; j-y < img2 -> height; j++){
        blend1 = (1-alpha)*ImageGetPixel(img1,i,j);
        blend2 = alpha * ImageGetPixel(img2,i-x,j-y);
        ImageSetPixel(img1, i,j, (uint8)(blend1+blend2+0.5));
      }///+0.5 para arredondar, 7.51 nao vai arredondar para 8
    } ///porque 0.01 nao da para arredondar, da sempre floor
  }
}
```

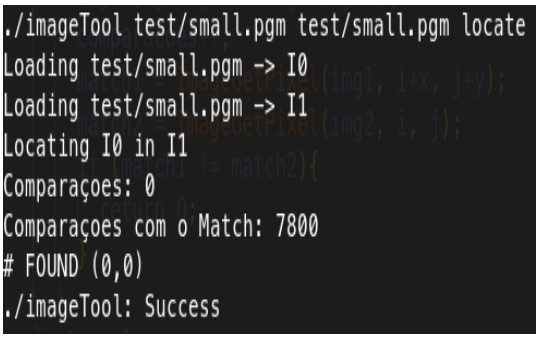
A função do paste vai verificar se ambas as fotos, img1 e img2 já definidas não são nulas. Se tal se verificar e o valor de alpha for maior que 1.0 a segunda imagem vai ser diretamente colada na primeira imagem mas se tal não se verificar procederemos para dar blend. O novo valor do pixel vai ser calculado com a média dos pixeis das duas imagens. A nova imagem será criada por esta média mais 0.5 cuja explicação está escrita como comentário no código.

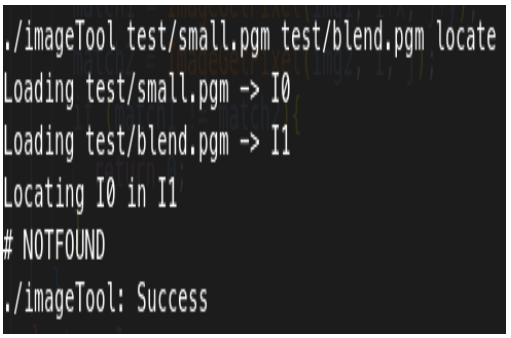
```c
int comparacoes = 0;
int ImageMatchSubImage(Image img1, int x, int y, Image img2) {
  int i,j,match1,match2;
  assert (img1 != NULL);
  assert (img2 != NULL);
  assert (ImageValidPos(img1, x, y));
  for(i=0;i<img2 -> width; i++){
    for(j=0; j<img2 ->height; j++){
      comparacoes++;
      match1 = ImageGetPixel(img1, i+x, j+y);
      match2 = ImageGetPixel(img2, i, j);
      if (match1 != match2){
        return 0;
      }
    }
  }return 1;
}
```

```c
int ImageLocateSubImage(Image img1, int* px, int* py, Image img2) { ///
  int i,j;
  int comparacoes2 = 0;
  assert (img1 != NULL);
  assert (img2 != NULL);
  for (i = 0; i < img1 -> width; i++){
    for (j = 0; j < img1 -> height; j++){
      if(ImageValidRect(img1,i,j,img2->width,img2->height)) {
        if(ImageMatchSubImage(img1,i,j,img2)){
          *px = i;
          *py = j;
          printf("Comparaçoes: %d\n", comparacoes2);
          printf("Comparaçoes com o Match: %d\n", comparacoes);
          return 1;
        }
      }
    }
  }return 0;
}
```

A ImageMatchSubImage vai verificar se a img2 corresponde a uma sub-imagem de img1 a partir da posição (x,y). A função vai percorrer cada pixel da img2 e vai comparar com os da img1 e se forem semelhantes a função vai retornar 1 o que significa que efetivamente a img2 corresponde a uma sub-imagem da img1 caso contrário retorna 0.
A ImageLocateSubImage vai procurar a img2 dentro da img1 percorrendo cada pixel da img1 recorrendo à função ImageMatchSubImage. Se tal se verificar vai retornar 1 e definir as variáveis dx e dy conforme o verificado. Caso contrário retorna 0.







No Makefile criamos 3 testes para testar o Locate comparando a small.pgm com paste.pgm, small.pgm e blend.pgm

```c
void ImageBlur(Image img, int dx, int dy) {
  int x, y; ///definir i,j da erro por alguma razao, diz que nao sao usadas mas sao
  double count, some, average;
  Image image = ImageCreate(img->width, img->height, img->maxval);
  for(int g = 0; g < img -> width; g++ ){
    for(int v = 0; v < img-> height; v++){
      ImageSetPixel(image,g,v,ImageGetPixel(img,g,v));
    }
  }
  for (x = 0; x < img->width; x++) {
    for (y = 0; y < img->height; y++) {
      some = 0.0;
      count = 0.0;
      for (int i = x - dx; i <= x + dx; i++) {
        for (int j = y - dy; j <= y + dy; j++) {
          if (i >= 0 && i < img->width && j >= 0 && j < img->height) {
            some += ImageGetPixel(img, i, j);
            count++;
          }
        }
      }
      average = some / count;
      ImageSetPixel(image, x, y, (uint8)(average+0.5));
    }///+0.5 para arredondar, 7.51 nao vai arredondar para 8
    ///porque 0.01 nao da para arredondar, da sempre floor
  }
  for (x = 0; x < img->width; x++) {
    for (y = 0; y < img->height; y++) {
      ImageSetPixel(img, x, y, ImageGetPixel(image, x, y));
    }
  }
  ImageDestroy(&image);
}
```

Esta função vai ser usada para desfocar a imagem original usando dx e dy como variáveis para saber a área de desfoque. A função vai criar uma nova imagem com as mesmas dimensões da original e de seguida copiar todos os pixeis originais para esta nova imagem. Após tal se verificar os for's permitem-nos percorrer cada pixel da imagem e calcular o valor médio somente dos pixeis que estão dentro da área previamente definida a partir de dx e dy. Então a média é calculada adicionando 0.5 de modo a ter um arredondamento correto tal como é explicado no código. Para finalizar a função copia todos os pixeis da nova imagem para a original e em seguida destrói a nova imagem.

Imagem	1,1	3,3	6,6	9,9
art3_222x217	1.18938	1.200644	1.224994	1.238627
art4_300x300	1.190865	1.236639	1.242115	1.290929
bird_256x256	1.1939	1.196565	1.22566	1.25563
airfield-05_640x480	1.188068	1.225897	1.330877	1.537683
tac-pulmao_512x512	1.193656	1.232055	1.318322	1.507071
tools_2_765x460	1.200338	1.239494	1.360014	1.606701
airfield-05_1600x1200	1.215585	1.515708	2.146761	3.471263
einstein_940x940	1.222505	1.343065	1.652113	2.296147
ireland_03_1600x1200	1.261467	1.522206	2.145831	3.487882

No Makefile criamos 36 testes para o blur recorrendo a diversas imagens. Os pixeis definidos em cima correspondem à área de desfoque e os resultados são referentes ao time.

Conclusão:

Concluindo foi desenvolvido um programa de processamento de imagens com diversas funções. Todas as opções foram devidamente testadas e o seu funcionamento aprovado.