

Relatório: Ampliando e reduzindo imagens por interpolação bilinear

Davi de Lima Cruz
PROJETO 01-2024.2: Grupo 1

November 24, 2024

Resumo

Este relatório apresenta a implementação de um programa que amplia e reduz imagens por interpolação bilinear. O programa foi implementado em C e utiliza a biblioteca libtiff para ler e escrever imagens no formato TIFF. O programa foi testado com imagens de diversos tamanhos e resoluções e os resultados obtidos foram satisfatórios.

1 Discussão Técnica

1.1 Formato TIFF

O formato TIFF é um formato de arquivo de imagem flexível e poderoso que suporta vários tipos de dados, incluindo imagens monocromáticas, imagens coloridas e imagens em tons de cinza. O formato TIFF é amplamente utilizado em aplicações de imagem e é suportado por muitos programas de edição de imagem.

1.2 DPI

DPI (dots per inch) é uma unidade de medida que indica a resolução de uma imagem. A resolução de uma imagem é o número de pixels por polegada e é usada para determinar a qualidade da imagem. Uma imagem com uma resolução mais alta tem mais detalhes e é mais nítida do que uma imagem com uma resolução mais baixa. A resolução de uma imagem é importante ao imprimir a imagem, pois determina a qualidade da impressão. Podemos descobrir a resolução da nova imagem a partir da resolução da imagem original e do fator de escala, que é obtido pela razão dos DPIs da imagem original e da imagem modificada.

$$fator_escala = \frac{DPI_{original}}{DPI_{nova}} \quad (1)$$

1.3 Interpolação Bilinear

A interpolação bilinear é um método de interpolação que calcula o valor de um pixel interpolado com base nos valores dos pixels vizinhos. O método é usado para aumentar ou reduzir o tamanho de uma imagem e é amplamente utilizado em aplicações de processamento de imagem. A interpolação bilinear é um método simples e eficaz para aumentar ou reduzir o tamanho de uma imagem e produz resultados de alta qualidade. Dado as posições x_1 , x_2 , y_1 e y_2 dos pixels vizinhos, o valor do pixel interpolado é calculado pela fórmula:

$$\begin{aligned} I(x, y) = & \frac{1}{(x_2 - x_1)(y_2 - y_1)} (f(x_1, y_1)(x_2 - x)(y_2 - y) + \\ & + f(x_2, y_1)(x - x_1)(y_2 - y) + \\ & + f(x_1, y_2)(x_2 - x)(y - y_1) + \\ & + f(x_2, y_2)(x - x_1)(y - y_1)) \end{aligned} \quad (2)$$

2 Discussão dos Resultados

Para o relógio, as diferenças são pouco perceptíveis, mas a imagem com 1250 dpi (Figura 1c) é um pouco menos nítida que a imagem original (Figura 1a). Isso ocorre porque a primeira interpolação perdeu informações da imagem original.



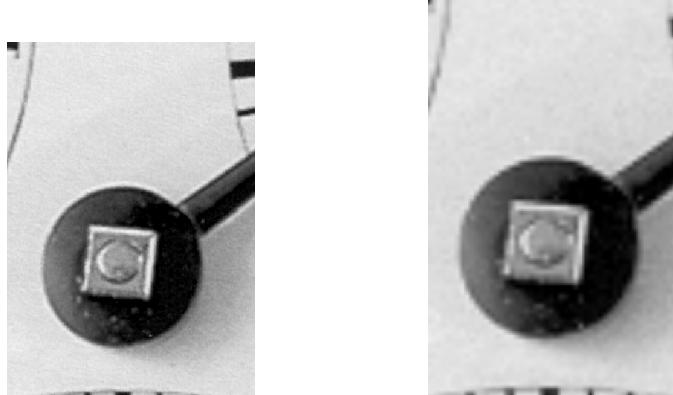
(a) Imagem original.

(b) Imagem com 300 dpi.

(c) Imagem com 1250 dpi.

Figure 1: Comparação entre as imagens do relógio.

A seguir, apresentamos um zoom do centro do relógio nas duas imagens com 1250 dpi:



(a) Imagem original.

(b) Imagem com 1250 dpi.

Figure 2: Comparação entre as imagens do relógio com zoom.

Podemos ver que o relógio original (Figura 2a) tem muito mais detalhes do que a imagem transformada com 1250 dpi (Figura 2b).



(a) Imagem original.



(b) Imagem com 10 dpi.



(c) Imagem com 100 dpi.

Figure 3: Comparação entre as imagens da lua.

A imagem da lua com 10 dpi (Figura 3b) é muito pixelada, enquanto a imagem com 100 dpi (Figura 3c) é mais nítida, mas ainda apresenta pixelização. Isso ocorre porque a imagem original (Figura 3a) é muito pequena e a interpolação não consegue recuperar as informações perdidas.



(a) Imagem original.



(b) Imagem com 10 dpi.



(c) Imagem com 100 dpi.

Figure 4: Comparação entre as imagens da cidade.

A cidade apresentou o pior resultado. A imagem com 10 dpi (Figura 4b) é extremamente pixelada, e a imagem com 100 dpi (Figura 4c) também é pixelada e apresenta detalhes distorcidos. Resumindo os resultados, a interpolação bilinear é eficaz para aumentar e diminuir o tamanho de uma imagem se não tiver muitos detalhes. A qualidade da imagem ampliada depende da resolução da imagem original e do fator de escala. Se a imagem original for muito pequena, a interpolação não conseguirá recuperar as informações perdidas e a imagem ampliada será pixelada. Se a imagem original for grande, a interpolação produzirá uma imagem ampliada de alguma qualidade.

3 Código Fonte em C

Listing 1: Código fonte do programa em C.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "tiffio.h"
4
5 int discovery_dimensions_resolutions(char* filename, uint32_t* w,
6     uint32_t* h, float* xres, float* yres) {
7     TIFF* tiff = TIFFOpen(filename, "r");
```

```

7   TIFFGetField(tiff, TIFFTAG_IMAGEWIDTH, w);
8   TIFFGetField(tiff, TIFFTAG_IMAGELENGTH, h);
9   TIFFGetField(tiff, TIFFTAG_XRESOLUTION, xres);
10  TIFFGetField(tiff, TIFFTAG_YRESOLUTION, yres);
11  printf("Resolution: %fx%f\n", *xres, *yres);
12  TIFFClose(tiff);
13  return 1;
14 }
15
16 int readTiff(char* filename, uint8_t** pixels) {
17  uint32_t w, h;
18  float xres, yres;
19  TIFF* tiff = TIFFOpen(filename, "r");
20  TIFFGetField(tiff, TIFFTAG_IMAGEWIDTH, &w);
21  TIFFGetField(tiff, TIFFTAG_IMAGELENGTH, &h);
22  TIFFGetField(tiff, TIFFTAG_XRESOLUTION, &xres);
23  TIFFGetField(tiff, TIFFTAG_YRESOLUTION, &yres);
24  uint32_t npixels = w * h;
25  *pixels = (uint8_t*) malloc(npixels * sizeof(uint8_t));
26  uint32_t* raster = (uint32_t*) _TIFFmalloc(npixels * sizeof(
27    uint32_t));
28  TIFFReadRGBAIImage(tiff, w, h, raster, 0);
29  printf("Successfully read the TIFF image %dx%d.\n", w, h);
30  for (uint32_t row = 0; row < h; row++) {
31    for (uint32_t col = 0; col < w; col++) {
32      uint32_t pixel = raster[row * w + col];
33      uint8_t r = TIFFGetR(pixel);
34      uint8_t g = TIFFGetG(pixel);
35      uint8_t b = TIFFGetB(pixel);
36      (*pixels)[row * w + col] = (uint8_t)(0.299 * r + 0.587 * g +
37        0.114 * b);
38    }
39  }
40  _TIFFfree(raster);
41  TIFFClose(tiff);
42  return 1;
43 }
44
45 int writeTiff(char* filename, uint32_t w, uint32_t h, float xres,
46   float yres, uint8_t* pixels) {
47  TIFF* tiff = TIFFOpen(filename, "w");
48  TIFFSetField(tiff, TIFFTAG_IMAGEWIDTH, w);
49  TIFFSetField(tiff, TIFFTAG_IMAGELENGTH, h);
50  TIFFSetField(tiff, TIFFTAG_BITSPERSAMPLE, 8);
51  TIFFSetField(tiff, TIFFTAG_SAMPLESPERPIXEL, 1);
52  TIFFSetField(tiff, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_MINISBLACK);
53  TIFFSetField(tiff, TIFFTAG_ORIENTATION, ORIENTATION_BOTLEFT);
54  TIFFSetField(tiff, TIFFTAG_XRESOLUTION, xres);
55  TIFFSetField(tiff, TIFFTAG_YRESOLUTION, yres);
56  TIFFSetField(tiff, TIFFTAG_RESOLUTIONUNIT, RESUNIT_INCH);
57  for (uint32_t row = 0; row < h; row++) {
58    TIFFWriteScanline(tiff, &pixels[row * w], row, 0);
59  }
60  TIFFClose(tiff);
61  return 1;
62 }
```

```

61 int interpolate(uint8_t* pixels, uint32_t oldW, uint32_t oldH,
62                 uint8_t* newPixels, uint32_t newW, uint32_t newH,
63                 float xres, float yres, float newres) {
64     for (uint32_t row = 0; row < newH; row++) {
65         for (uint32_t col = 0; col < newW; col++) {
66             uint32_t x1, x2, y1, y2;
67             float x, y;
68             x = ((float) col) * (xres / newres);
69             y = ((float) row) * (yres / newres);
70             x1 = (uint32_t) x;
71             x2 = x1 + 1;
72             y1 = (uint32_t) y;
73             y2 = y1 + 1;
74             if (x2 >= oldW) x2 = oldW - 1;
75             if (y2 >= oldH) y2 = oldH - 1;
76             newPixels[row * newW + col] = (uint8_t)(pixels[y1 * oldW + x1
77                     ] * (x2 - x) * (y2 - y) +
78                     pixels[y1 * oldW + x2] * (x - x1) * (y2 -
79                         y) +
80                     pixels[y2 * oldW + x1] * (x2 - x) * (y -
81                         y1) +
82                     pixels[y2 * oldW + x2] * (x - x1) * (y -
83                         y1));
84         }
85     }
86     return 1;
87 }
88
89 int main(int argc, char* argv[]) {
90     char* filename = argv[1];
91     float newdpi = (float) atoi(argv[2]);
92
93     uint32_t oldW, oldH;
94     float xres, yres;
95     uint8_t* oldPixels;
96     discovery_dimensions_resolutions(filename, &oldW, &oldH, &xres, &
97                                         yres);
98     oldPixels = (uint8_t*) malloc(oldW * oldH * sizeof(uint8_t));
99     readTiff(filename, &oldPixels);
100
101    uint32_t newW, newH;
102    newW = (uint32_t) (oldW * (newdpi / xres));
103    newH = (uint32_t) (oldH * (newdpi / yres));
104    printf("new w and h: %d %d\n", newW, newH);
105    uint8_t* newPixels = (uint8_t*) malloc(newW * newH * sizeof(
106                                         uint8_t));
107    printf("Interpolating...\n");
108    interpolate(oldPixels, oldW, oldH, newPixels, newW, newH, xres,
109                 yres, newdpi);
110    printf("Writing the new image...\n");
111    writeTiff("output.tif", newW, newH, newdpi, newdpi, newPixels);
112    free(oldPixels);
113    free(newPixels);
114    return 0;
115 }
```