

Ampliando e Reduzindo Imagens por
Interpolação Bilinear
Projeto 01-2024.2: Gp. 1

André Filipe de Medeiros - 542104
Ryan Erik Lima da Silva - 542103

November 27, 2024

1 Resumo

O presente relatório descreve a implementação de um programa em C++ para ampliar e reduzir imagens com interpolação bilinear. Esse método calcula os valores dos pixels em uma nova resolução com base nos quatro vizinhos mais próximos. O programa permite redimensionar uma imagem de acordo com a resolução desejada (em dpi). Durante a atividade, reduziu-se imagem de 1250 dpi para 100 dpi e, em seguida, ampliou-se de volta para 1250 dpi, observando a perda de qualidade. O programa também foi aplicado em outras imagens, demonstrando a eficácia da técnica na ampliação e redução de imagens.

2 Discussão Técnica

2.1 DPI

Pontos por polegadas (dpi) é uma unidade de resolução comumente utilizada em imagens digitais. Essa grandeza indica a quantidade de pixels que existem em uma área de uma polegada quadrada. Em geral, quanto maior for o dpi de uma imagem, maior será sua nitidez. Assim, o dpi é um fator importante para a qualidade da imagem. A razão de dpis é utilizada na mudança de resolução de uma imagem, seu valor é calculado por:

$$r = \frac{DPI_1}{DPI_2} \quad (1)$$

Em que, r é a razão de dpis, DPI_1 é a dpi original e DPI_2 é a dpi desejada.

2.2 Interpolação Bilinear

A interpolação bilinear é um método para redimensionamento de imagens. Esse método estima o valor de um pixel em uma nova posição com base nos quatro pixels vizinhos mais próximos, utilizando uma média ponderada das distâncias horizontais e verticais. Embora seja mais precisa que métodos simples, como a interpolação por vizinho mais próximo, a bilinear pode introduzir artefatos, como suavização excessiva ou perda de detalhes, especialmente ao ampliar excessivamente uma imagem. A fórmula da interpolação bilinear para calcular o valor $f(x, y)$ é dada por:

$$f(x, y) = \frac{(x_2 - x) \cdot (y_2 - y) \cdot f(x_1, y_1) + (x - x_1) \cdot (y_2 - y) \cdot f(x_2, y_1)}{(x_2 - x_1) \cdot (y_2 - y_1)} + \frac{(x_2 - x) \cdot (y - y_1) \cdot f(x_1, y_2) + (x - x_1) \cdot (y - y_1) \cdot f(x_2, y_2)}{(x_2 - x_1) \cdot (y - y_1)} \quad (2)$$

Onde: $f(x_1, y_1), f(x_2, y_1), f(x_1, y_2), f(x_2, y_2)$ são os valores dos pixels nos quatro pontos mais próximos. $(x_1, y_1), (x_2, y_2)$ são as coordenadas dos quatro pixels vizinhos mais próximos ao ponto (x, y) . (x, y) são as coordenadas do pixel que estamos calculando na nova imagem (redimensionada).

3 Discussão dos Resultados

Após a escrita do *script* em C++ que amplia e reduz (em dpi) uma imagem através de interpolação bilinear, submetemos a Figura ?? a uma redução de 1250 dpi a 100 dpi e chegamos à Figura ???. Depois, através de interpolação bilinear, ampliamos de volta a Figura ?? de 100 dpi a 1250 dpi e a Figura ?? é o resultado.

Figure 1: Imagens do relógio.



(a) Imagem original 1250 dpi.

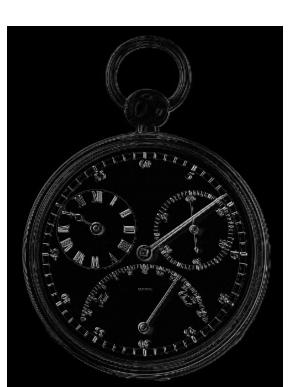
(b) Imagem modificada com 100 dpi.

(c) Imagem modificada com 1250 dpi.

Percebe-se que houve uma redução de nitidez quanto se reduziu a imagem de 1250 a 100 dpi. Após aplicação da interpolação de 100 a 1250 dpi, percebe-se uma suavização na intensidade dos pixels da imagem.

Com a diferença entre as imagens, percebe-se a perca de informação após a redução de dpi em relação à imagem original.

Figure 2: Imagens diferença do relógio.



(a) Imagem de diferença da imagem original (1250 dpi) com a de 300 dpi.



(b) Imagem de diferença da imagem original (1250 dpi) com a de 1250 dpi interpolada.

Aplicamos o mesmo raciocínio a outras duas imagens escolhidas: imagem do coqueiro e imagem da chuva.

Figure 3: Imagens do coqueiro.



(a) Imagem original 300 dpi.



(b) Imagem modificada com 60 dpi.



(c) Imagem modificada com 300 dpi.

Figure 4: Imagens diferença do coqueiro.



(a) Imagem de diferença da imagem original (300 dpi) com a de 60 dpi.



(b) Imagem de diferença da imagem original (300 dpi) com a de 300 dpi interpolada.

H

Figure 6: Imagens da chuva.



(a) Imagem original 300 dpi.

(b) Imagem modificada com 60 dpi.

(c) Imagem modificada com 300 dpi.

Figure 7: Imagens diferença da chuva.



(a) Imagem de diferença da imagem original (300 dpi) com a de 60 dpi.

(b) Imagem de diferença da imagem original (300 dpi) com a de 300 dpi interpolada.

Conclui-se que, em determinadas condições, a interpolação bilinear é um ótimo método para interpolar pixels a uma imagem. Em especial, quando a diferença de intensidade entre pixels vizinhos é considerável, a interpolação é uma boa estratégia para aumentar a densidade de pixels (dpi). Além disso, por ser uma média ponderada, a interpolação bilinear acaba suavizando a diferença de intensidade entre pixels de uma imagem.

4 Código Fonte em C++

Listing 1: Código fonte do programa de interpolação bilinear em C++.

```
1 #include <iostream>
2 #include <opencv2/opencv.hpp>
3
4 cv::Vec3b bilinearInterpolate(const cv::Mat& image, float x, float
5     y) {
6     int x1 = std::floor(x);
7     int y1 = std::floor(y);
8     int x2 = std::min(x1 + 1, image.cols - 1);
9     int y2 = std::min(y1 + 1, image.rows - 1);
10
11     float dx = x - x1;
12     float dy = y - y1;
```

```

13     cv::Vec3b p1 = image.at<cv::Vec3b>(y1, x1);
14     cv::Vec3b p2 = image.at<cv::Vec3b>(y1, x2);
15     cv::Vec3b p3 = image.at<cv::Vec3b>(y2, x1);
16     cv::Vec3b p4 = image.at<cv::Vec3b>(y2, x2);
17
18     cv::Vec3b result;
19     for (int c = 0; c < 3; ++c) {
20         float top = p1[c] * (1 - dx) + p2[c] * dx;
21         float bottom = p3[c] * (1 - dx) + p4[c] * dx;
22         result[c] = cv::saturate_cast<uchar>(top * (1 - dy) +
23             bottom * dy);
24     }
25     return result;
26 }
27 int main() {
28     std::string imagePath = "imgs/Fig0220(a)(chronometer 3692x2812
29     2pt25 inch 1250 dpi).tif";
30     cv::Mat image = cv::imread(imagePath, cv::IMREAD_COLOR);
31
32     if (image.empty()) {
33         std::cerr << "Error" << std::endl;
34         return -1;
35     }
36
37     int currentDpi = 1250;
38
39     int desiredDpi;
40     std::cout << "DPI desejado:";
41     std::cin >> desiredDpi;
42
43     float scalingFactor = (float)desiredDpi / currentDpi;
44
45     int newWidth = (int)(image.cols * scalingFactor);
46     int newHeight = (int)(image.rows * scalingFactor);
47
48     cv::Mat resizedImage(newHeight, newWidth, image.type());
49
50     for (int y = 0; y < newHeight; ++y) {
51         for (int x = 0; x < newWidth; ++x) {
52             float srcX = x / scalingFactor;
53             float srcY = y / scalingFactor;
54
55             resizedImage.at<cv::Vec3b>(y, x) = bilinearInterpolate(
56                 image, srcX, srcY);
57         }
58     }
59     cv::namedWindow("DPI original", cv::WINDOW_NORMAL);
60     cv::imshow("DPI original", image);
61
62     cv::namedWindow("DPI reduzido", cv::WINDOW_NORMAL);
63     cv::imshow("DPI reduzido", resizedImage);
64
65     std::cout << "Pressione qualquer tecla" << std::endl;
66     cv::waitKey(0);

```

```

67
68     float inverseScalingFactor = 1.0f / scalingFactor;
69
70     int enlargedWidth = (int)(resizedImage.cols *
71         inverseScalingFactor);
71     int enlargedHeight = (int)(resizedImage.rows *
72         inverseScalingFactor);
72
73     cv::Mat enlargedImage(enlargedHeight, enlargedWidth,
74         resizedImage.type());
74
75     for (int y = 0; y < enlargedHeight; ++y) {
76         for (int x = 0; x < enlargedWidth; ++x) {
77             float srcX = x / inverseScalingFactor;
78             float srcY = y / inverseScalingFactor;
79
80             enlargedImage.at<cv::Vec3b>(y, x) = bilinearInterpolate
81                 (resizedImage, srcX, srcY);
81         }
82     }
83
84     cv::namedWindow("DPI interpolado", cv::WINDOW_NORMAL);
85     cv::imshow("DPI interpolado", enlargedImage);
86
87     cv::waitKey(0);
88
89     cv::imwrite("imgs/resized_image_bilinear.tif", resizedImage);
90     cv::imwrite("imgs/enlarged_image_bilinear.tif", enlargedImage);
91
92     return 0;
93 }
```

Listing 2: Código fonte do programa de diferença de imagens em C++.

```

1 #include <iostream>
2 #include <opencv2/opencv.hpp>
3
4 int main() {
5     cv::Mat img1 = cv::imread("imgs/Fig0220(a)(chronometer 3692
6         x2812 2pt25 inch 1250 dpi).tif", cv::IMREAD_COLOR); // 
    Abrir com leitura em cor
6     cv::Mat img2 = cv::imread("imgs/resized_image_bilinear.tif", cv
        ::IMREAD_COLOR);
7
8     if (img1.empty() || img2.empty()) {
9         std::cerr << "Erro ao carregar as imagens" << std::endl;
10        return -1;
11    }
12
13    std::cout << "Imagen 1 - Tamanho: " << img1.size() << ", Tipo:
    " << img1.type() << std::endl;
14    std::cout << "Imagen 2 - Tamanho: " << img2.size() << ", Tipo:
    " << img2.type() << std::endl;
15
16    if (img1.size() != img2.size()) {
17        std::cout << "Redimensionando a imagem 2 para o tamanho da
            imagem 1" << std::endl;
18        cv::resize(img2, img2, img1.size());
```

```
19     }
20
21     if (img1.channels() != img2.channels()) {
22         std::cout << "Convertendo para o mesmo numero de canais"
23             << std::endl;
24         if (img1.channels() == 3) {
25             cv::cvtColor(img2, img2, cv::COLOR_BGR2GRAY);
26         } else {
27             cv::cvtColor(img1, img1, cv::COLOR_BGR2GRAY);
28         }
29
30         cv::Mat diff;
31         cv::absdiff(img1, img2, diff);
32
33         bool sucesso = cv::imwrite("imgs/diferenca.tif", diff);
34         if (!sucesso) {
35             std::cerr << "Erro ao salvar a imagem de diferenca" << std
36                 ::endl;
37         }
38
39         cv::imshow("Imagen de diferenca", diff);
40         cv::waitKey(0);
41
42         return 0;
43 }
```