Efeitos de Distorção de Lentes Parametrizáveis a 1000+ FPS

Lucas Oliveira Maggi[2], João Marcelo Xavier Natário Teixeira[1], Lucas Silva Figueiredo[2], Veronica Teichrieb[2]

[1]DEINFO - Universidade Federal Rural de Pernambuco (UFRPE)

[2]Centro de Informática – Universidade Federal de Pernambuco (UFPE)

Cidade Universitária - 50740-560 – Recife – PE – Brazil

{lom,jmxnt,lsf,vt}@cin.ufpe.br

Abstract. *This paper describes the implementation of a parameterizable lens distortion effect technique running at 1000+ FPS on CPU. Different image resolutions were used to validate the implemented technique and the visual result obtained showed to be very satisfactory.*

Resumo. *Este artigo descreve a implementação de uma técnica de efeito de distorção de lentes parametrizáveis com desempenho superior a 1000 FPS em CPU. Diferentes resoluções de imagem foram utilizadas na validação da técnica implementada e o resultado visual obtido mostrou-se bastante satisfatório.*

1. Introdução

Há quase quatro décadas existem estudos voltados à distorção de lentes [Brown 1966]. Nas últimas duas décadas, merecem destaque trabalhos na área de distorção de lentes [Tsai 1987], em que frequentemente se utilizam distorções inversas de calibração de câmera [Brown 1971, Weng et al. 1992, Shar et Aggarwal 1996, Zhang 2000, Fitzgibbon 2001, Thirthala et Pollefeys 2005, Claus et Fitzgibbon 2005, De Villiers et al. 2008]. Tais trabalhos são de extrema importância para realidade aumentada, com a correta inserção dos elementos em cena, e visão computacional [Tsai 1987, Weng et al 1992] , para a identificação, rastreamento ou métricas mais precisas. O trabalho descrito neste artigo é voltado para o uso em efeitos de distorção de imagem, com distorções parametrizáveis de alto desempenho em CPU.

Este artigo está organizado da seguinte maneira: a seção 2 lista alguns trabalhos relacionados a implementações de algoritmos de distorção de lentes assim como aplicações relacionadas; a seção 3 descreve conceitos básicos necessários ao entendimento do trabalho apresentado; a seção 4 detalha o algoritmo escolhido e como o mesmo foi implementado de forma otimizada em CPU; a seção 5 ilustra os resultados obtidos com a técnica implementada, analisando o desempenho conseguido para diferentes configurações de resolução e número de threads; por fim, a seção 6 lista

conclusões acerca do projeto e indica possíveis caminhos a serem traçados como trabalhos futuros.

2. Trabalhos relacionados

Todos os trabalhos relacionados são voltados para distorção de lente e/ou calibração de câmera, e podem ser categorizados quanto à finalidade:

- Descrição de distorção paramétrica [Brown. 1966],
- Calibração [Brown 1971, Weng et al. 1992, Shar et Aggarwal 1996, Zhang 2000, Fitzgibbon 2001, Thirthala et Pollefeys 2005, Claus et Fitzgibbon 2005, De Villiers et al. 2008],
- Visão [Tsai 1987, Weng et al. 1992, Gribbon, K. T. et al. 2003].

Em [Tsai 1987, Weng et al. 1992, Gribbon et al. 2003, Thirthala et Pollefeys 2005] é considerada a distorção radial de lente para correção da imagem. Já em [Claus et Fitzgibbon 2005], a distorção radial é utilizada para fins de comparação com outras distorções.

3. Conceitos básicos

Distorções de lente são causadas pelo formato da lente utilizada, e pelo efeito de refração da luz na mesma. Tais distorções podem ser descritas e parametrizadas matematicamente. Distorções radiais são distorções que apresentam uma variação da distorção de acordo com a distância radial em relação a algum centro. São distorções radiais: Barrel Distortion e Pincushion Distortion, ambas ilustradas na Figura 1 [Gribbon el al 2003].

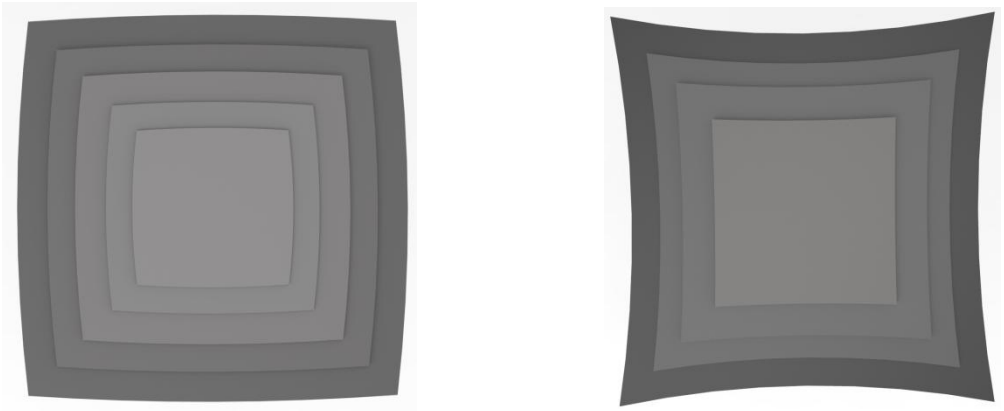


Figura 1. Efeitos de distorção de lentes: Distorção de Barril (esquerda) e Distorção de Almofada (direita).

4. O algoritmo

O algoritmo proposto leva em consideração apenas a distorção radial, desconsiderando a distorção tangencial, utilizando as seguintes equações:

$$x_i := x \times (1 + k_1 \times r^2 + k_2 \times r^4 + k_3 \times r^6)$$

Equação 1. Transformação não linear de Brown-Conrady em *x*.

$$y_i := y \times (1 + k_1 \times r^2 + k_2 \times r^4 + k_3 \times r^6)$$

Equação 2. Transformação não linear de Brown-Conrady em *y*.

x e y representam as coordenadas na imagem original, e r é o valor do raio calculado. r^2 é dado em função da seguinte equação:

$$r^2 := x^2 + y^2$$

Equação 3. Equação da circunferência no plano.

Uma vez que o valor do raio não é utilizado nas equações listadas anteriormente, apenas é necessário calcular os valores de r^2, r^4 e r^6, evitando assim a realização de operações de raiz quadrada, o que melhora o desempenho e a precisão do algoritmo.

Com base nas equações iniciais, é necessária a definição de três coeficientes (k_1, k_2, k_3) para o cálculo da distorção radial. Cada um deles atua sobre uma parte específica do polinômio. Sendo assim, a entrada da solução é dada pela imagem juntamente com os parâmetros k definidos.

As coordenadas x e y da imagem original devem ser convertidas para coordenadas canônicas, no intervalo $[-1,1]$, considerando que a origem está localizada no centro da imagem, com o valor de y invertido (diminui de cima para baixo, contrariamente ao que acontece na biblioteca OpenCV). Para isso, são usadas as seguintes equações:

$$x := (\frac{j}{larguraDaImagem}) \times 2 - 1$$

Equação 4. Conversão de coordenadas em *x*.

$$y := -((\frac{i}{larguraDaImagem}) \times 2 - 1)$$

Equação 5. Conversão de coordenadas em *y*.

Nas equações anteriores, i e j indicam as coordenadas dos pixels nas imagens representadas pela biblioteca OpenCV.

Como o mapeamento é realizado do espaço de imagem no OpenCV para o espaço de imagem no OpenCV, deve-se realizar agora o processo inverso, para que se tenha como resultado as coordenadas finais x e y já transformadas. Para isso, são usadas as seguintes equações:

$$xi_2 := \frac{x_i + 1}{2}$$

Equação 6. Reversão de coordenadas em *x*.

$$yi_2 := 1 - (\frac{y_i + 1}{2})$$

Equação 7. Reversão de coordenadas em *y*.

$$x_f := xi_2 \times cols$$

Equação 8. Rescalonando para a imagem final em *x*.

$$y_f := yi_2 \times rows$$

Equação 9. Rescalonando para a imagem final em *y*.

O primeiro par de equações é responsável por converter as coordenadas de volta, considerando a origem deslocada e a reflexão no eixo y. O segundo par de equações aplica um fator de escala nas coordenadas encontradas, proporcional aos valores de altura (rows) e largura (cols) da imagem resultante.

Caso as coordenadas finais encontradas estejam fora do intervalo de pixels da imagem, pode-se preencher aquele pixel com informação de uma cor estática. No nosso caso, foi escolhida a cor preta como cor de fundo.

4.1. Implementação

A solução desenvolvida foi implementada usando a linguagem de programação C++, incluindo a biblioteca padrão (STL) do C++11. Utilizou-se a biblioteca OpenCV para fins de carregamento das imagens/videos de entrada (arquivos ou webcam) e exibição do resultado em tempo real em janelas (resultado da distorção). Além disso, apenas a estrutura Mat foi utilizada, o que torna todo o código desenvolvido compatível com as versões 2.4 e 3 da biblioteca. A IDE de desenvolvimento utilizada foi o Microsoft Visual Studio 2015 Community. O desempenho da solução foi otimizado com paralelismo para um processador multicore através da biblioteca OpenMP em dois momentos: criação dos endereços de mapeamento da transformação e cópia dos valores da imagem original para a imagem transformada.

O trecho de código responsável por calcular o mapeamento entre imagem original e imagem distorcida, ou seja, os novos endereços para cada posição de pixel na imagem original é descrito na Figura 2:

```
void mapDistortion(Mat &img, uint *addresses) {
    #pragma omp parallel for num_threads(NUM_THREADS)
    for (int i = 0; i < img.rows; i++) {
        for (int j = 0; j < img.cols; j++) {
            float x, y;
            x = (j / (float)img.cols)*2.f - 1.f;
            y = -((i / (float)img.rows)*2.f - 1.f);
            float r2 = x*x + y*y;
            float k1 = valk1 / 100.f, k2 = valk2 / 100.f, k3 = valk3 / 100.f;

            float r4 = r2*r2, r6 = r4*r2;

            float xff = (x / (float)(1 + k1*r2 + k2*r4 + k3*r6) + 1.f) / 2.f;
            float yff = 1.f - ((y / (float)(1 + k1*r2 + k2*r4 + k3*r6)) + 1.f) / 2.f;

            int xf = xff*img.cols;
            int yf = yff*img.rows;

            if (xf < img.cols && xf > -1 && yf > -1 && yf < img.rows) {
                addresses[i*img.cols + j] = yf*img.cols + xf;
            } else {
                addresses[i*img.cols + j] = -1;
            }
        }
    }
}
```

Figura 2. Código do cálculo de endereço.

Na Figura 3 está presente o trecho de código responsável por realizar a cópia dos pixels da imagem original para a imagem distorcida, seguindo os endereços calculados previamente, é dado a seguir:

```
void applyDistortion(Mat &imgref, Mat &img) {
    uint size = img.cols * img.rows;
    bgrColor *data = (bgrColor*)img.data;
    bgrColor *org = (bgrColor*)imgref.data;
    bgrColor black;

    #pragma omp parallel for num_threads(NUM_THREADS)
    for (int i = 0; i < size; i++) {
        if (addresses[i] == -1) {
            data[i] = black;
        }
        else {
            data[i] = org[addresses[i]];
        }
    }
}
```

Figura 3. Código de cópia/aplicação da distorção.

5. Resultados

Apresentamos nesta seção os resultados obtidos, em imagens originais e distorcidas, e gráficos comparativos de desempenho. Utilizamos duas fontes diferentes de imagem, uma delas foi uma webcam integrada ao computador utilizado, e outra foi de vídeo, e três resoluções distintas. Dados de desempenho, além de sua análise, são detalhados.

5.1 Cenários de validação

Foram selecionadas três resoluções para realização dos testes de efeito visual e de desempenho, sendo estas: 480p, 720p e 1080p. O uso de tais resoluções se justifica pelo fato das mesmas serem comumente utilizados tanto na indústria cinematográfica/televisiva, quanto em computadores pessoais.

Os vídeos utilizados como entrada para o algoritmo foram:

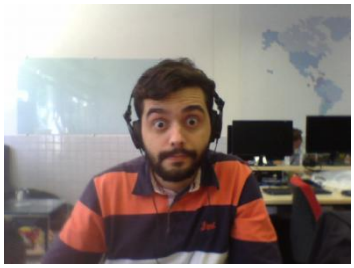
A: Webcam, resolução 640x480 (480p, VGA)

B: Steve Jobs - The Man In The Machine (2015), resolução 1280x720(720p, HD)

C: Inside Out (2015), resolução 1920x1080(1080p, Full HD)

A seguir estão as configurações utilizadas dos parâmetros k1, k2, k3, juntamente com os resultados obtidos.

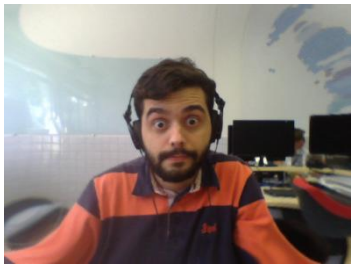
Originais: k1 = 0, k2 = 0, k3 = 0. (Sem distorção)

		
A	B	C

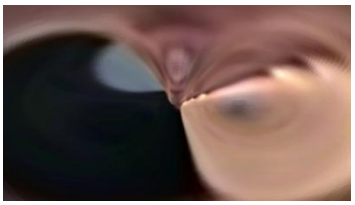
Distorção 1: k1 = 0, k2 = 0, k3 = -12.

		
A	B	C

Distorção 2: k1 = 0, k2 = 0, k3 = 12.

		
A	B	C

Distorção 3: k1 = 752, k2 = 0, k3 = 568.

		
A	B	C

Distorção 4: k1 = -752, k2 = 0, k3 = -1357.

		
A	B	C

5.2 Análise de desempenho

Para validar o desempenho observado, foram realizados vários conjuntos de testes de performance em um notebook. Temos como método para medição o uso de média aritmética simples de 1000 (mil) amostras de desempenho, ou seja, afere-se o tempo gasto no cálculo de cada etapa, Cálculo de endereços e Cópia, e para cada etapa é calculada uma média dessas 1000 (mil) amostras. Executamos o procedimento descrito anteriormente 10 (dez) vezes para realizar uma média da replicação do procedimento e utilizar como valor final para análise de desempenho, sendo até aqui o conjunto de teste. Foi realizado um conjunto de teste para diferentes quantidades de threads, sendo as mais significativas nas quantidades 1, 2, 4 e 8 threads (acima de 8 threads houve queda de desempenho, possivelmente devido ao processador utilizado na realização dos testes).

As especificações do notebook utilizado nos testes são: ASUS S46C, Intel(R) Core(™) i7-3537U @ 2.00 GHz (2,5GHz máximo), 2 núcleos físicos, 4 processadores lógicos, 4MB cache L3, 512KB cache L2, 128KB cache L1, 8GB RAM(2x4GB) DDR3. Sistema operacional: Microsoft Windows 8.1 Single Language. Prioridade do processo: Normal.

Constatou-se que o desempenho, medido em quadros por segundo, variava de acordo com a quantidade de threads e de pixels (resolução) da imagem. Observou-se também

que o comportamento do desempenho acompanhava o crescimento da quantidade de threads, no entanto com uma diferença de desempenho cada vez menor entre os números consecutivos, como pode ser visto a seguir nas Figuras 4 e 5.

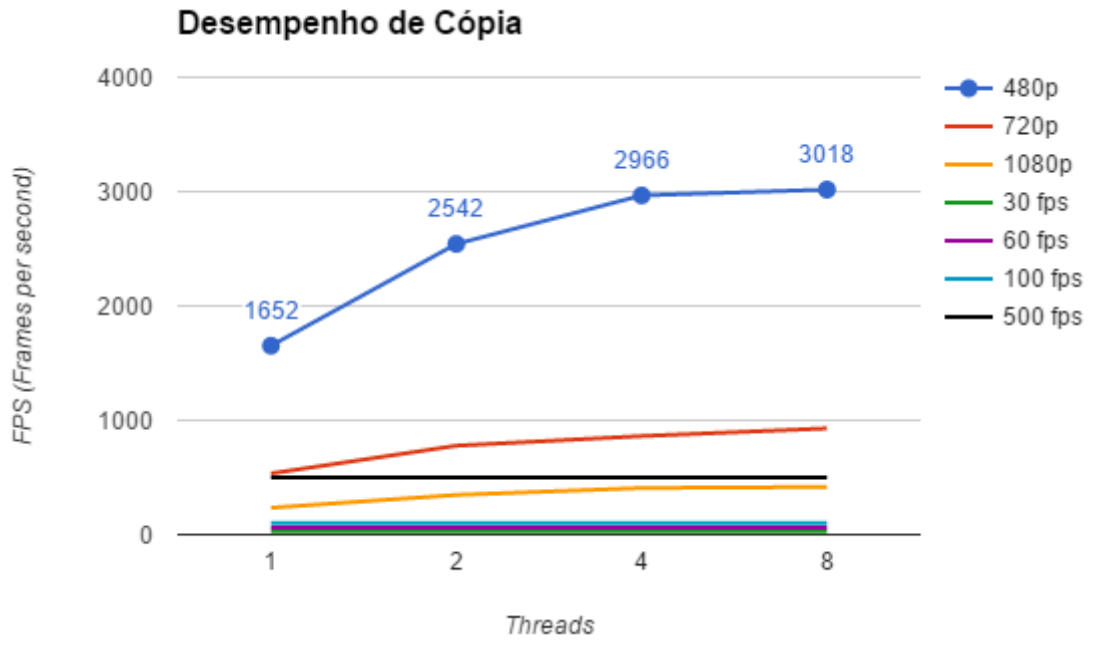


Figura 4. Comparação de desempenho de cópia.

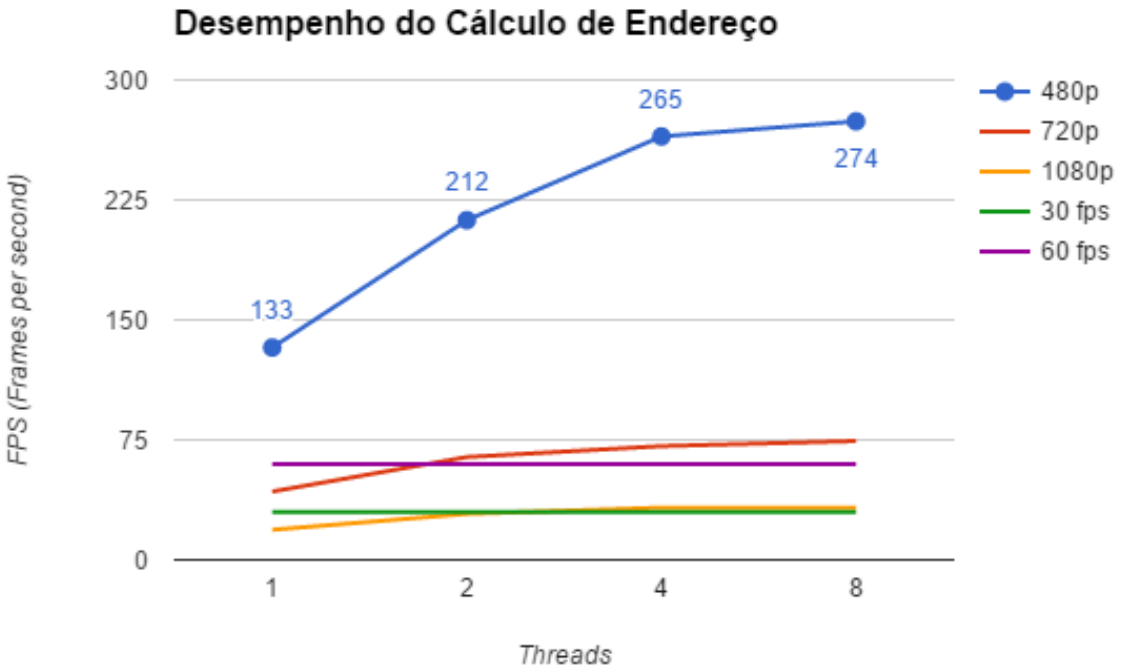


Figura 5. Comparação de desempenho do cálculo de endereço.

Temos que a limitação de FPS será dada pelo processo que for mais lento, ou seja, o que possui menor FPS. Sendo assim, para uma execução em tempo real com a distorção sendo modificada a cada quadro, o desempenho de todo o processo de distorção é limitado pelo cálculo do endereço (Figura 5). No caso de uma distorção ser aplicada e fixada ao longo da execução, o desempenho segue o desempenho de cópia (Figura 4). De acordo com os testes realizados conseguimos executar o cálculo de endereço com FPS acima de 30 com as resoluções A e B para todas as quantidades de threads utilizadas. Com a resolução C (1080p), a única a não apresentar esse comportamento, conseguimos executar a o cálculo de endereço com FPS acima de 30 quando a quantidade de threads foi 4 e 8, havendo pouca diferença entre os dois.

6. Conclusão

Este artigo apresentou a possibilidade de realizar efeitos de distorções de lentes com três parâmetros em tempo real em formatos comuns de reprodução, mostrando o potencial para efeitos de pós-processamento da imagem, ou para correção de distorção da imagem em tempo real.

Como trabalhos futuros tem-se a implementação de distorção de múltiplos centros, neste trabalho foi utilizado para testes apenas o centro da imagem como centro de distorção de lente, a disponibilização de uma API pública com as técnicas implementadas e uma versão da distorção implementada em GPU ao invés de CPU, utilizando o poder de processamento paralelo massivo dessa plataforma.

Referências

YING Xiang-Hua e HU Zhan-Yi. (2003) “Fisheye Lense Distortion Correction Using Spherical Perspective Projection Constraint”, Chinese Journal Of Computers.

Juyang Weng, Paul Cohen, e Marc Herniou. (1992) "Camera Calibration with Distortion Models and Accuracy Evaluation", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, No. 10.

Andrew W Fitzgibbon. (2001) "Simultaneous linear estimation of multiple view geometry and lens distortion", IEEE Computer Vision and Pattern Recognition 2001 (CVPR 2001).

Shishir Shah, J. K. Aggarwal. (1996) "Intrinsic parameter calibration procedure for a (high-distortion) fish-eye lens camera with distortion model and accuracy estimation", Vol. 29, No. 11., Pages 1775-1788.

G. P. Stein, (1997) "Lens Distortion Calibration Using Point Correspondences", IEEE, Computer Vision and Pattern Recognition 1997 (CVPR 1997).

Roger Y. TSAI. (1987) "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-shelf TV Cameras and Lenses", IEEE, Journal of Robotics and Automation, Vol. RA-3, No. 4.

SriRam Thirthala e Marc PolleFeys. (2005) "The Radial Trifocal Tensor: A tool for calibrating the radial distortion of wide-angle cameras", IEEE, Computer Society Conference on Computer Vision and Pattern Recognition 2005 (CVPR 2005).

David Claus e Andrew W. Fitzgibbon (2005) "A Rational Function Lens Distortion Model for General Cameras", IEEE, Computer Society Conference on Computer Vision and Pattern Recognition 2005 (CVPR 2005).

Zhengyou Zhang. (2000) "A Flexible New Technique for Camera Calibration", IEEE, Transactions On Pattern Analysis and Machine Intelligence, Vol. 22, No. 11.

Brown, D. C. (1966). “Decentering distortion of lenses”. Photometric Engineering, 32(3), 444-462.

Brown, D. C. (1971) “Close-range camera calibration”. Photogrammetric Engineering, v. 37, n. 8, p. 855-866, 1971.

De Villiers, J. P., Leuschner, F. W., & Geldenhuys, R. (2008, November). "Centi-pixel accurate real-time inverse distortion correction". Na International Symposium on Optomechatronic Technologies (pp. 726611-726611). International Society for Optics and Photonics.

Gribbon, K. T., Johnston, C. T., & Bailey, D. G. (2003, November). A real-time FPGA implementation of a barrel distortion correction algorithm with bilinear interpolation. In Image and Vision Computing New Zealand (pp. 408-413).

Ojanen, H. (2001). “Automatic correction of lens distortion by using digital image processing”, 1999.