Fine Image Resampling Algorithm

Bronislav Přibyl*
Supervised by: Pavel Zemčík†

Department of Computer Graphics and Multimedia
Faculty of Information Technology
Brno University of Technology
Brno / Czech Republic

Abstract

This paper introduces a fine image resampling algorithm intended for corrections of image distortions caused by lenses or similar devices. The algorithm is designed for correction of small distortions in terms of pixel displacement but with high subpixel precision. The geometrical description of the correction is through bilinear interpolation within each node of a sparse square or rectangular mesh. The paper describes the algorithms itself, its features, implementation issues and data formats. Specifically discussed are the issues connected with programmable hardware (FPGA) implementation.

Keywords: Image resampling, Subpixel resampling, Lens distortion correction, FIR filter, Bilinear interpolation

1 Introduction

Image processing is one of the fields of computer science and applications that is developing very fast. The object of image processing is, of course, an image. Vast majority of image processing methods assumes that the image is a 2D signal represented through samples organized in a regular square or rectangular raster [2]. While the contemporary image acquisition devices and methods acquire images that relatively well fulfill the above assumption, in most cases, the images suffer from small geometrical imperfections caused e.g. by lenses (pincushion distortion, barrel distortion) used with the cameras that acquire the images.

The geometrical imperfections are in some cases not critical; however, many applications of image processing exist that suffer from the imperfections and where it is desirable to correct them. While the geometrical correction – calculation of new sample positions in the image – is relatively straightforward and can be e.g. performed through bilinear interpolation within square or rectangular mesh, the problem remains how to get the new samples values so that the signal properties of the image remain as much preserved as possible. Unfortunately, the nearest neighbor method, which completely destroys the image signal properties, and bilinear or bicubic interpolation [3] which can be better but by far is not ideal, are traditionally used for this purpose (see figure 1 for illustration). The main reason is that while the algorithms to preserve good signal properties, namely frequency spectrum, are known, they are often considered prohibitively computationally expensive. This paper proposes a method that is far better from the point of view of signal properties than the bilinear or bicubic interpolation while still preserves relatively low computational requirements. The limitation of the proposed method, however, is that it is limited to the cases where the distortions do not involve significant angular or scale changes – the method merely assumes only local subpixel shift limited to several pixels displacement [2, 3].

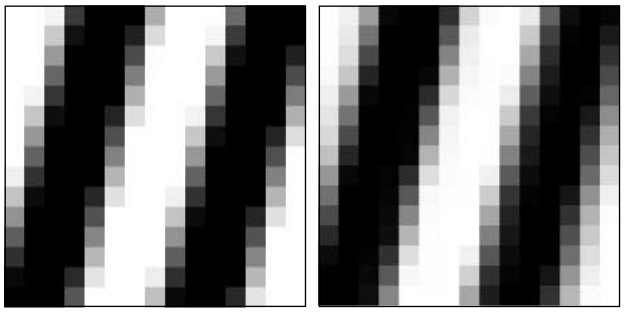

Figure 1: (**a**) Original pattern with 4 px thick synthetic lines 15 × magnified. (**b**) The same pattern resampled at scale 1.05 using bilinear interpolation. Note that edges of lines in the resampled image are significantly blurred.

2 Image Resampling

General image resampling problem is relatively straightforward mathematically – it is merely a problem of proper reconstruction of signal values in 2D space and proper application of sampling theorem. However, the efficient implementation of such resampling is still quite open problem. In our approach, we limit the general problem to

*xpriby12@stud.fit.vutbr.cz
†zemcik@fit.vutbr.cz

resampling in order to correct geometrical imperfections only. This limitation has the following implications:

- The displacement of pixel location of the original and resampled images is only units of pixels,
- no angular distortion is expected, and
- no scaling is expected.

The general approach for resampling in our case is to scan the output image raster pixel by pixel (sample by sample) and reconstruct the values from the original raster based on knowledge of the pixel displacement. Taking into account the above limitations, it is known that the sampling theorem cannot be violated and also it can be assumed that the function is separable:

$$r_{x,y} = f\left(o, d(x,y)\right) = f''\left(f'\left(o, d(x,y)\right), d(x,y)\right), \quad (1)$$

where r is the resampled image,
o is the original image,
d is the displacement function,
f is a resampling function,
and f' and f'' are the partial reconstruction functions after separation.

In our case, the functions f' and f'' are implemented through a bank of FIR[1] filters [4] indexed by subpixel location of the pixel. Moreover, the sampling is the same in both directions, so f' and f'' are implemented using the same FIR bank.

The above solution with FIR filters was chosen as it has well defined features and as it is quite flexible in terms of exchangeability of the filtering function.

3 Proposed Resampling

The proposed approach to resampling relies on the separability; however, in addition to the generally used approach, the separability is applied to both the resampling function itself and the geometrical distortion calculation so that the distortion calculation is separated in vertical and horizontal directions:

$$r_{x,y} = f''\left(f'\left(o, d_x(x,y)\right), d_y(x,y)\right), \quad (2)$$

where r is the resampled image,
o is the original image,
d_x and d_y are the displacement functions,
and f' and f'' are the partial reconstruction functions after separation.

[1] Abbreviation from Finite Impulse Response.

The resampling function itself is assumed to be some suitable filter function and in the presented approach it is implemented through a bank of FIR filters. The bank of FIR filters is indexed through a subpixel position of the desired sample in the raster (see equation 3). The reason is that the FIR coefficients are dependent on the subpixel position of the desired sample location. Of course, the size of FIR filters is limited. The filters in the bank can be e.g. Lanczos filters [7] for optimal exploitation of the bandwidth of the image signal given the size of the filter, or other filter design to achieve the desired image signal properties. The described approach is, in fact, not dependent on it.

$$r_{x,y} = FIR_{fp(x)}\left(FIR_{fp(y)}\left(o, ip\left(d_y(x,y)\right)\right), ip\left(d_x(x,y)\right)\right), \quad (3)$$

where r is the resampled image,
o is the original image,
d_x and d_y are the displacement functions,
FIR_t is the function of the bank for position t,
fp is the fractional part of a numerical value,
and ip is the integer part of a numerical value.

The distortion to be corrected can be described in different ways, e.g. analytically or by a offset texture. Distortion description acquisition is not subject of this work but it has been studied e.g. in [5, 6, 8]. In our approach the distortion is described with a sparse rectangular mesh with displacement specified for each node of the mesh. In fact, the mesh can be seen as a offset texture. While the displacement in each node (corner of the rectangles) of that mesh is known, the displacement inside the rectangles is computed via bilinear interpolation. Size of the rectangles depends on application and local change of the displacement – the smaller size of the rectangle, the more precise approximation of the distortion but more memory for the distortion description needed.

Distortion inside each rectangle is described by means of the following four precalculated coefficients:

- D0 – displacement of top left pixel in the rectangle.
- DC0 – difference of displacements between adjacent pixels in 1st row of the rectangle.
- DR – difference of displacements between 1st pixels of adjacent rows in the rectangle.
- DDC – change in difference of displacements between pixels of adjacent rows in the rectangle, that means $\mathrm{DC_{n+1}} - \mathrm{DC_n}$.

For more detailed description see figure 2. Note, please, that the displacement calculation can be subdivided into independent calculation of vertical and horizontal displacements.

Displacement calculation executed by the resampling algorithm in each rectangle of the mesh can bee seen in

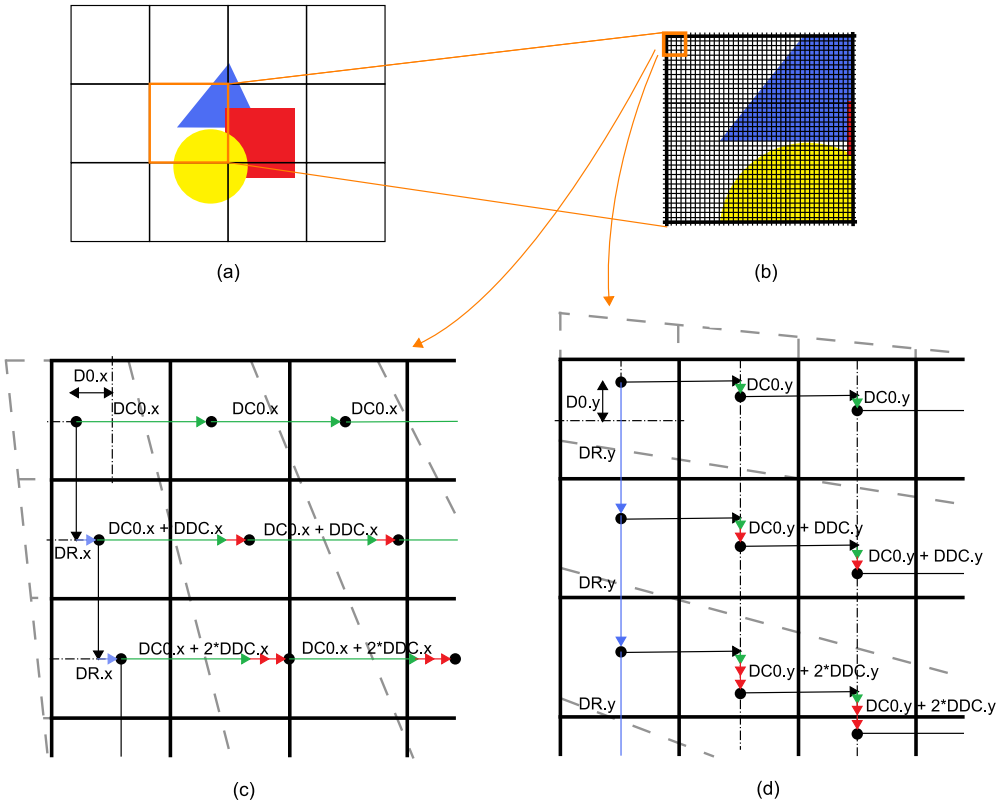

Figure 2: Displacement interpolation inside rectangles. (**a**) Output image is covered with rectangular mesh, source image distortion is defined within each mesh node. (**b**) Rectangular area of the output image with individual pixels visible. (**c**, **d**) Graphical representation of precalculated coefficients describing the source image distortion for horizontal and vertical resampling respectively. Pixels of distorted source image are plotted with gray dashed lines, pixels of output image are plotted with black solid lines. Big black dots are centers of source image pixels. Meaning of precalculated coefficients is marked with coloured vectors.

algorithm 1. The input of the algorithm is the original image, distortion description (through the above mentioned coefficients), and FIR filter; the output is the resampled pixels within the given rectangle. Note, please, that two instances of the algorithm are being used, one for vertical and one for horizontal displacement and filtering.

```
1:  var DoR, DC, D;
2:  DoR = D0;
3:  DC = DC0;
4:  for all rows in rectangle do
5:      D = DoR;
6:      for all pixels in row do
7:          Output FIR[fp(D)](O,ip(D));
8:          D += DC;
9:      end for
10:     DoR += DR;
11:     DC += DDC;
12: end for
```

Algorithm 1: Displacement calculation.

Time complexity class of the displacement calculation algorithm is $O(n^2)$ because displacement for each pixel of the output image needs to be computed directly. Cost of displacement calculation for a pixel is just cost of one addition operation (algorithm 1, line 8) plus cost of FIR filter response computation (algorithm 1, line 7). When the proposed FIR filtering is used, the time complexity class does not change and remains $O(n^2)$. This fact is, however, not important as the size of the image is fixed anyway and the proposed approach significantly reduces the computational cost.

As it can be seen from the pseudocode, three variables are needed in the displacement calculation algorithm. Their meaning is as following:

- DoR – displacement of 1st pixel in a row.
- DC – difference of pixel displacements.
- D – displacement of current pixel.

As shown in the algorithm, the displacement is subdivided into integer and fractional parts. The integer part is used to determine the pixel placement of the filter while the fractional part is used to determine the set of coefficients within the filter bank. When the number of filters in the bank is N (e.g. 16), the fractional part is multiplied by

N and then rounded to nearest integer. Then it is used for filter bank index.

4 FPGA Implementation

Two implementations of proposed approach have been realized so far; On CPU[2] using C[3] language and in FPGA[4] [1] using VHDL[5] language. The former one was originally intended as support and template for implementation of the latter one, so it does not fully exploit potential of the CPU. However it closely simulates processes running in programmable hardware with bit-precision, so outputs of both implementations are identical.

An FPGA platform has been chosen as it offers high degree of parallelism, which is exploitable e.g. in convolution-like operations. In our case it has been employed together with pipelining[6] to accelerate computation of the FIR filter response.

When implemented on CPU, time complexity class of filtering operation is generally $O(n^2)$ (when using only row or columnar filter it is $O(n)$) as a function of the filter size. This is because all pixels in the filtered area have to be multiplied by their respective filter coefficients sequentially and also summed up sequentially. On the other hand FPGA offers the possibility to multiply all pixels with filter coefficients simultaneously. However impact of this can be eliminated by pipelining which causes that results of the filtering operations are available in each cycle, only with some delay.

The overall structure of the resampling system is based on the fact that in vast majority of applications, the image is acquired "line by line" "pixel by pixel" from top left corner to bottom right corner. Also, in most of the memory storage structures, the images are stored this way. Therefore, it can be assumed that the image is best processed in the same order and the dataflow is adjusted so.

The overall structure is illustrated in figure 3. First, the image is fed into a buffer that can hold several complete image lines. The buffer must be large enough to accomodate as many lines of the image as it corresponds to the maximal vertical displacement in both directions plus the size of vertical FIR filter. In this case, the height of the buffer should be at least 31 pixels (2×12 px maximal vertical displacement + 7 px FIR filter height).

Next, the data fed into the buffer is processed so that for each line of the output image and for each position of a pixel on that line, the corresponding vertical location in the buffer is found. Then columnar neighborhood of that location, which is of same size as the FIR filter, is fed into the vertical resampling unit (in our case 7 samples). Thanks to parallelism of FPGA, the buffer allows to access the neighborhood in one clock cycle.

Then, the resampling unit calculates on output value of the vertical part of the resampling process through FIR''_q, where q is the index of a set of coefficients in the FIR'' filter bank determined from the fractional part of the vertical position. This concludes the vertical resampling process.

After the vertical resampling, the horizontal resampling takes place. The horizontal resampling part can take advantage from the fact that the data used in it is image line data only – the result of the vertical resampling unit. As the sampling frequency of the resampled image is very similar to the original image, it can be additionally assumed that for one pixel produced by the vertical resampling unit, approximately one pixel of the output image will be produced by the horizontal resampling unit as well. This allows for usage of only a small buffer (units of samples) between the vertical and horizontal resampling. In fact, the data buffer can even be in a form of a small shift register where the shift factor is 0, 1, or 2 samples at a time.

When the location of the data in the horizontal direction is determined, the actual horizontal resampling takes place using similar mechanism to the vertical one – the FIR'_q filter is applied based on the subpixel position in horizontal direction, where q is the index of a set of coefficients in the FIR' filter bank.

The horizontal resampling unit directly produces the resampled image pixels that should be stored into the resampled image data structures and this concludes the processing of the complete resampling unit.

In both vertical and horizontal resampling units, the control is done based on the calculation of displacements of resampled pixels according to the method described in section 3.

An FPGA implementation of the resampling algorithm has been prepared as part of the experiments with the design. The dataflow in the resampling unit can be seen in figure 4. Functional blocks are associated in two groups – one group handles vertical resampling while the other handles horizontal resampling. Each group consists of a FIR module, a Displacement interpolation module and respective Resampling module.

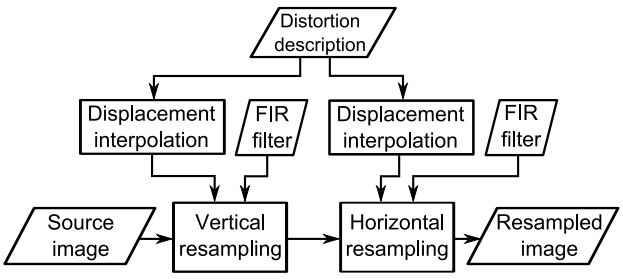

Figure 4: Dataflow of the resampling algorithm in FPGA.

Data formats used in the algorithm are the fixed deci-

[2] Abbreviation from Central Processing Unit – a processor.

[3] A general purpose programming language.

[4] Abbreviation from Field-programmable Gate Array – a type of integrated circuit which can be programmed after manufacturing.

[5] Abbreviation from Very-high-speed integrated circuit Hardware Description Language.

[6] Pipeline is a series of computational elements which work in parallel. Output of one element is input of the next one.

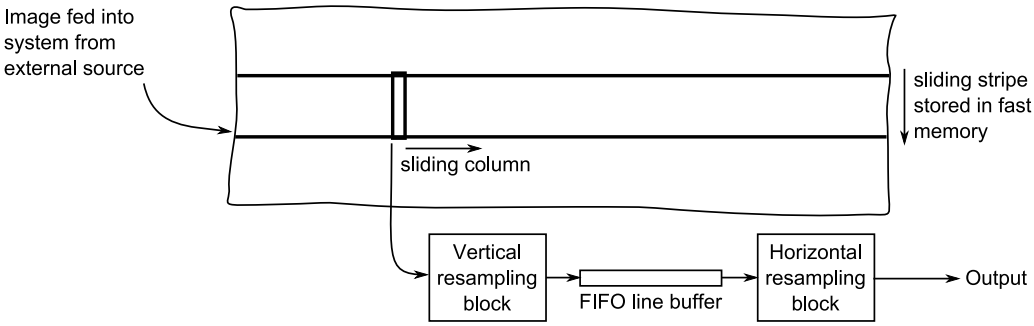

Figure 3: Overall processing of the image resampling.

mal point numbers in order to represent the data accurately enough while maintaining the design simple to enable its simple implementation.

The actual data formats used in the experiments are described below.

- Pixel data – 16 bit signed or unsigned. The pixel processing is assumed in 16 bit format in order to support the standard dynamic range of contemporary video cameras, which is 10 to 14 bits, plus an overhead for absorption of rounding errors of FIR.
- Coordinate – 12+4 bits unsigned. The subpixel resolution is assumed to be 16 subpixel positions which is in practical terms enough to avoid measurable adverse effects of granularity in subpixel position.
- Difference of coordinates – 2+8 bits signed. The difference of positions must be precise enough to represent the change of displacement.

Data formats of precalculated coefficients used as parameters of the displacement interpolation algorithm (see section 3) as well as data format of its output can be seen in table 1. Data formats specified in the table may seemingly not correspond to data formats described above. This is because all the coefficients and variables used in the interpolation algorithm do not represent coordinates of pixels. Instead of this, they represent displacements of output image pixels relative to input image pixels which have smaller range. Therefore upper bits of integer parts of the variables do not have to be used. On the other hand, more bits of fractional parts of the variables are utilized in order to absorb a cumulative error which emerges during execution of the interpolation algorithm.

The experimental design and synthesis of the resampling unit was performed for Xilinx[7] Virtex-II xc2v1000 FPGA device. XST[8] version H.38 was chosen for this task. As the unit is relatively generic, the following parameters were used: Image size 256×1024 px and square size 64 px which results in square mesh of 4×16 squares (and also displacement coefficient sets). Device utilization with configuration mentioned above is shown in table 2.

Items on chip	Used	Capacity	% capacity
Slices	3 947	5 120	77 %
Slice Flip Flops	2 112	10 240	21 %
4 input LUTs	3 103	10 240	30 %
BRAMs	20	40	50 %

Table 2: Exploitation of FPGA unit Virtex II-1000.

The device clock frequency is up to 105 MHz. While the resampling unit produces one output pixel per 2 clock cycles, the output resampling data rate for a single unit is up to 52.5 Mpixels per second. Thus the device is able to process 720p high definition video format (1280×720 px) at framerate 50 fps. This demonstrates the real-time potential of the design.

5 Results

The algorithm has been evaluated with images of artificial lines, photographic patterns acquired by camera and other images with results identical for standard CPU implementation and FPGA implementation. The resampling itself was performed with 7-sample Lanczos filter and the subpixel resolution was 16. These values are the limit values for the current implementation. These values can be seen as limits for efficient exploitation in real-life applications; However, they do not represent any limit for FPGA hardware.

Because no ground truth of an resampled image is available, we decided to use energy of power spectrum as a measure of error. The exact measure is ratio of energy of the resampled image and energy of the source image. An ideal algorithm would have the ratio 100 %.

The image of artificial lines (figure 5) was resampled at constant scale 1.05 in whole area using bilinear interpolation as well as our proposed algorithm. Power spectra of respective images are also shown. Bilinear interpolation algorithm proved to preserve 88.96 % of the spectrum energy, our proposed algorithm preserved 91.77 %.

[7] A company producing programmable logic devices, such as FPGAs.

[8] Abbreviation from Xilinx Synthesis Technology; A application for synthesizing device designs from hardware description language code.

Order of bit (2^n)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16
D0 (signed)											s	•	•	•	•	•	,	•	•	•	•												
DC0 (signed)																	,		s	•	•	•	•	•	•	•	•						
DR (signed)																	,		s	•	•	•	•	•	•	•	•						
DDC (signed)																	,				s	•	•	•	•	•	•	•	•	•	•	•	•
Displacement (signed)											s	•	•	•	•	•	,	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

Table 1: Precision of the interpolation algorithm coefficients and computed displacement. • denotes a used bit, "s" stands for sign, void bits are not used. Fixed decimal point is marked between bits nr. 0 and -1 with a comma.

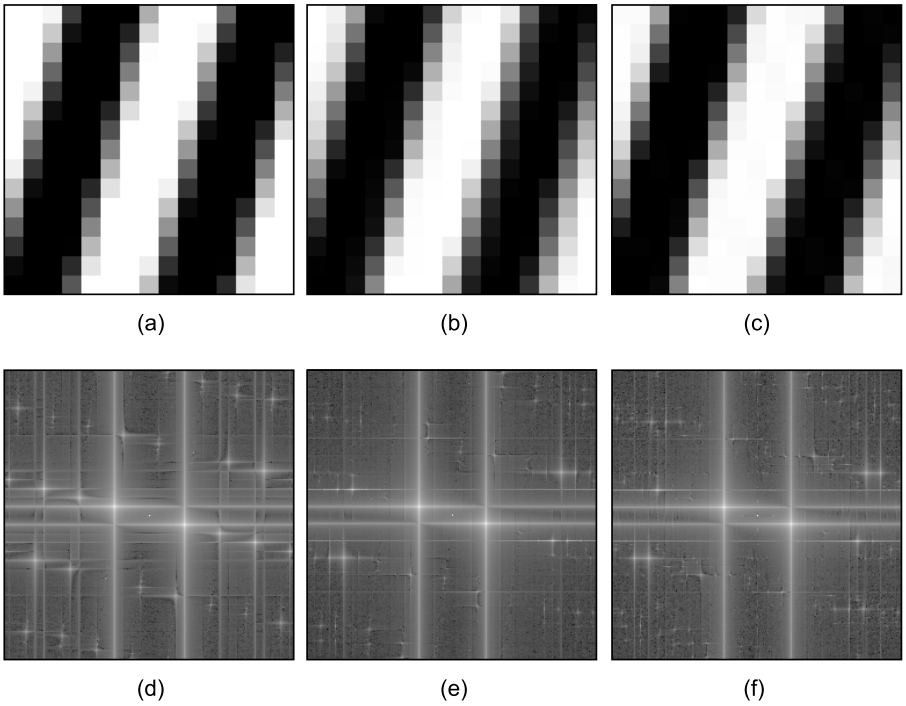

Figure 5: (**a**) Original pattern with 4 px thick synthetic lines 15 × magnified and (**d**) its power spectrum. (**b**) The same pattern resampled at scale 1.05 using bilinear interpolation and (**e**) its power spectrum. 88.96 % of energy of the power spectrum has been preserved. (**c**) The same pattern resampled at scale 1.05 using our proposed algorithm and (**f**) its power spectrum. 91.77 % of energy of the power spectrum has been preserved.

The image of photographic pattern (figure 6) was resampled simulating geometrical correction of barrel distortion using bilinear interpolation and our proposed algorithm too. Power spectra of respective images are shown as well because there are no observable differences in the output images. Bilinear interpolation preserved 90.32 % of the spectrum energy, our proposed algorithm preserved 98.45 %.

6 Conclusions

In this paper we described the accelerated fine image resampling approach based on a combination of a set of algorithms. Its intention is to correct geometrical image distortions caused by lenses or similar devices. Additionally, the implementation in software and FPGA is mentioned.

The presented approach and set of selected algorithms is based on approximation of the image distortion using a sparse rectangular mesh with bilinear interpolation of the positions within the nodes. The resampling itself is based on separable FIR filtering with a bank of filters indexed by a subpixel position.

As shown in the paper, the selected approach proved to be functional and lead into efficient implementation of resampling in both software and programmable hardware. The FPGA implementation is able to process up to 52.5 Mpixels per second which demonstrates the real-time potential.

It has also been shown that the selected approach gives better results than widely used bilinear interpolation algorithm. Additionally, we solved the problem of efficient implementation of a resampling algorithm, which properly reconstructs signal values in 2D space. However the general problem is limited by this approach to cases without angular distortion, scaling and significant pixel displace-

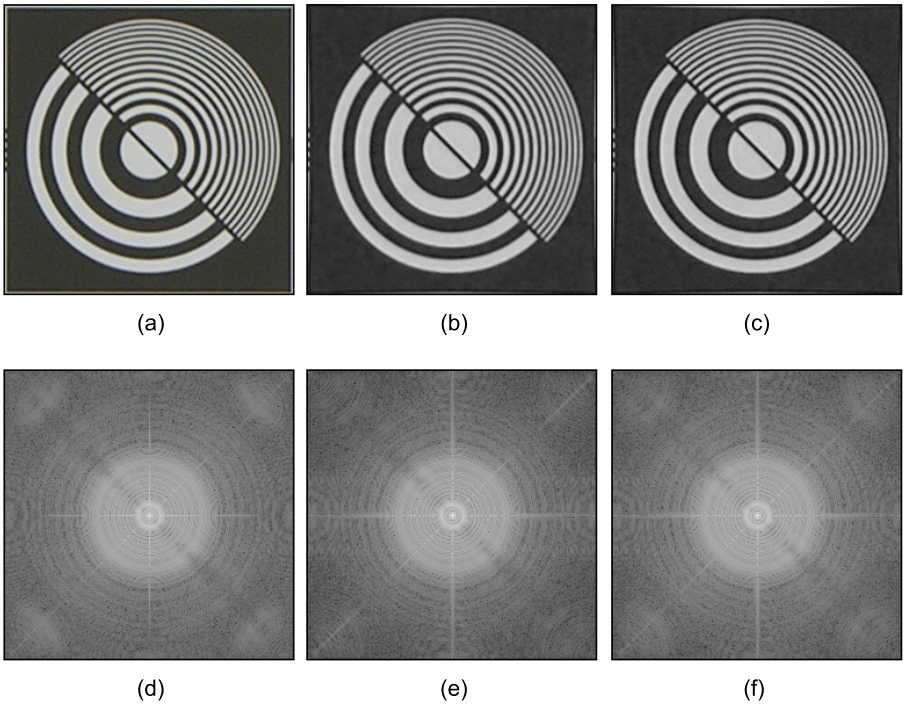

Figure 6: (**a**) Original image with a photographic pattern acquired by a camera and (**d**) its power spectrum. (**b**) The same image resampled using bilinear interpolation simulating barrel distortion correction and (**e**) its power spectrum. 90.32 % of energy of the power spectrum has been preserved. (**c**) The same image resampled using our proposed algorithm simulating barrel distortion correction and (**f**) its power spectrum. 98.45 % of energy of the power spectrum has been preserved.

ments.

Future work includes exploitation of the design in real-time applications of image processing. For example an efficient GPU[9] implementation will be considered. The work also includes further improvements of the structure and possible extension to 3D raster data.

7 Acknowledgements

This work was supported by the grant project of the Ministry of Education, Youth and Sports of CR, (MSMT 2B06052) project "BioMarker".

References

[1] S. Brown and J. Rose. FPGA and CPLD Architectures: A Tutorial. *IEEE Design and Test of Computers*, pages 42–57, 1996.

[2] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[3] A.C. Gallagher. Detection of Linear and Cubic Interpolation in JPEG Compressed Images. In *Proceedings of the 2nd Canadian conference on Computer and Robot Vision*, page 72. IEEE Computer Society, 2005.

[4] L.R. Rabiner and B. Gold. *Theory and application of digital signal processing*. Prentice Hall, 1975.

[5] C. Ricolfe-Viala and A.J. Sánchez-Salmerón. Robust metric calibration of non-linear camera lens distortion. *Pattern Recognition*, 43(4):1688–1699, 2010.

[6] GP Stein. Lens distortion calibration using point correspondences. In *1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1997. Proceedings.*, pages 602–608, 1997.

[7] T. Theußl, H. Hauser, and E. Gröller. Mastering windows: Improving reconstruction. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 101–108. ACM New York, NY, USA, 2000.

[8] T. Thormählen and H. Broszio. Automatic line-based estimation of radial lens distortion. *Integrated Computer-Aided Engineering*, 12(2):177–190, 2005.

[9] Abbreviation from Graphics Processing Unit. A specialized processor mainly used for accelerating computer graphics algorithms.