A Flexible Architecture for Real-time Fisheye Correction using Soft-core Processors and FPGA's

Authors:
Rama Shankar, 4th Year Electrical and Electronics Engineering, MIT, Manipal
UG Intern, Manipal Dot Net Pvt. Ltd.
rama.shk@gmail.com, rama.shankar@manipal.net

Omveer Sihag, 4th Year Electrical and Electronics Engineering, MIT, Manipal
UG Intern, Manipal Dot Net Pvt. Ltd.
sihag.omveer@gmail.com, omveer.sihag@manipal.net

Dr. Narasimha B. Bhat, CEO, Manipal Dot Net Pvt. Ltd.
narasim@manipal.net

Dr. Vinay Kumar, Member of Technical Staff, Manipal Dot Net Pvt. Ltd.
vinay.kumar@manipal.net

Address: Manipal Dot Net Pvt. Ltd. (EHTP Unit)
#37 (2nd Floor), Ananth Nagar,
Manipal, Karnataka.
Phone: +919844543937/09741736027

Abstract: Fisheye lenses [1] provide very large wide-angle views but the images produced suffer from severe distortions as a result of the frontal hemispherical scene being projected onto a flat surface. This paper discusses an innovative architecture to perform real-time fisheye correction on an FPGA when basic camera parameters (field of view and focal length) are known. The algorithm involves the one time computation of a look-up table (LUT) to map that maps a fisheye image to a perspective image. Floating point operations needed for interpolation are avoided by using a 9-point interpolation scheme. The method is implemented on a Nios® II soft-core embedded processor and the results of processing video feeds from an OmniVision fisheye camera are presented.

Introduction

Fisheye lenses achieve extremely wide fields of view (FOVs) by foregoing the perspective (rectilinear) mapping common to non-fisheye lenses and opting instead for a special mapping (e.g., equisolid angle) that gives images the characteristic convex appearance shown in Figure 1 (left). The radial distortion caused by fisheye mappings is one in which image magnification decreases with distance from the optical axis leading to an apparent effect that of an image that has been mapped around a sphere. As result, fisheye images do not preserve the most important feature of rectilinear images, which is that they map straight lines in the scene onto straight lines in the image. There are two kinds of fisheye images:

1. *Circular (hemispherical) fisheye images* are formed when the entire hemispherical view is projected onto a circle within the film frame. In other words, the image circle is inscribed in the film or sensor area. These have a 180° angle of view along the vertical and the horizontal directions as shown in Figure 1(left).
2. *Full-frame fisheye images* are formed when the hemispherical image is circumscribed around the film or sensor area as depicted in Figure 1 (right). These have a 180° angle of view along the diagonal, while the horizontal and vertical angles of view are smaller.

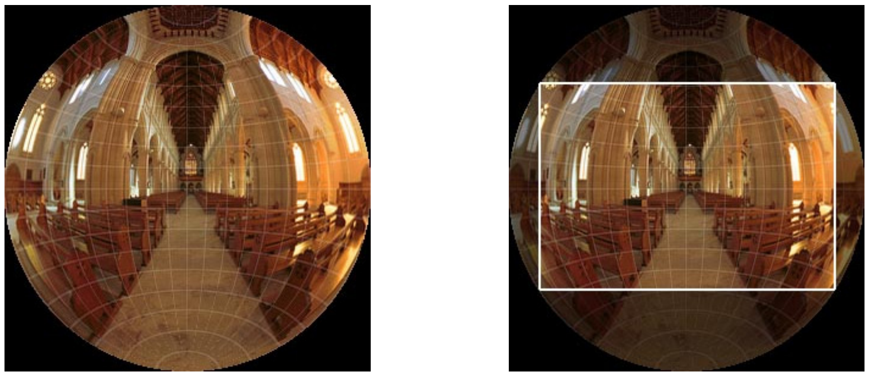

Figure 1: ***Fisheye Images—Circular (left) and Full-Frame (right)***

A fisheye lens is characterized by two basic parameters: the focal length and the FOV. Different fisheye lenses distort images differently and the nature of the distortion is defined by their mapping function. If θ is the angle between a point in the real world and the optical axis, which goes from the center of the image through the center of the lens, the radial position R of a point on the image is related to θ and to the focal length f of the lens for different mapping functions [1]:

1. Perspective projection (normal, non-fisheye lens): $R = f \tan(\theta)$. This simply works like a pinhole camera and is the basis for the rectilinear distortion-free mapping of normal cameras as shown in Figure 2.
2. Linear scaled (equidistant): $R = f\theta$ where θ is in radians as shown in Figure 4. This is the simplest mapping function for a fisheye lens and it clearly indicates that for a fisheye lens, the radial position of a point on the film is different from that of a perspective mapping and thus is shifted to a different position.

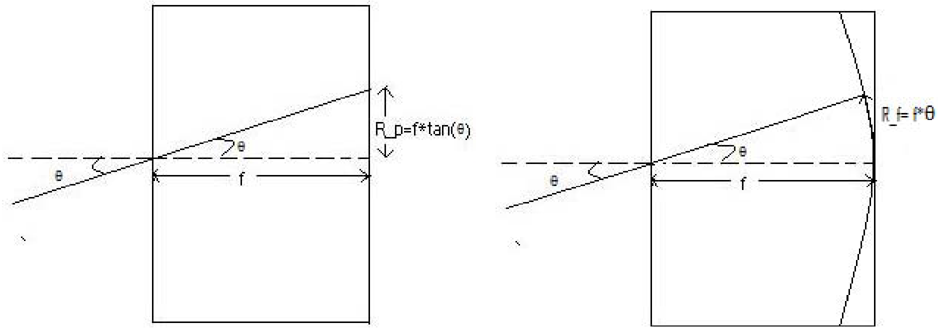

Figure 2: Perspective Projection (left) and Linear-Scaled Projection (right)

Similarly, other mapping functions for fisheye lens are possible like Equisolid angle ($R = 2f \sin(\theta/2)$: this popular mapping function is assumed for our analysis), Orthographic ($R = f \sin(\theta)$) *and* Stereographic (conform) ($R = 2f \tan(\theta/2)$).

Algorithm Description

As described above, the radial position of a point in a fisheye image (R_f) is different from that in a perspective image (Rp). Therefore, the task of correcting a distorted fisheye image is one of finding a relationship between Rp and R_f. This is found by solving for the unknowns in the two equations defining the perspective mapping and the fisheye mapping. Since solving those equations involves the computation of trigonometric functions, this is a difficult task to implement on an FPGA.

This paper presents a novel method that simplifies the procedure for correcting a distorted fisheye image. The basic idea is based on the observation that the relationship between Rp and (R_f) is completely determined by the camera geometry, i.e., the focal length (f) and the FOV. This implies that it can be pre-computed and stored in the form of a look-up table (LUT). The FPGA's task then is to use the LUT to map the pixels of the distorted fisheye input image to that of the corrected output image. Since this involves sub-pixel rendering, the method requires the FPGA perform some form of pixel interpolation. The 9-point interpolation method, a simple and very efficient form of pixel interpolation, produces a tolerable and distortion-free output image.

Computation of the LUT

With the input frame captured by the fisheye camera denoted as the source image and the corrected output as the target image, the task of correcting the source fisheye distorted image can be defined as follows: For every pixel location in the target image, compute its corresponding pixel location in the source image [2]. Let x_p and y_p be the x and y coordinates, respectively, of a target pixel in the output perspective (rectilinear) image, and similarly, let x_f and y_f be those of a source pixel in the input fisheye image. Then assuming an equisolid angle mapping for the fisheye image, the following equations hold true:

$$R_f = 2f\sin(\theta/2)$$
$$R_p = f\tan(\theta)$$
$$x_p/y_p = x_f/y_f$$

Where R_p is the radial position of a pixel on the perspective image from the center and R_f is the radial position of a pixel on the fisheye image from the center. Assuming the center of the image is the same as the center of the fisheye lens and eliminating θ between the above three equations gives:

$$x_f = \frac{2x_p\sin(\tan^{-1}(\lambda/2))}{\lambda}$$
$$y_f = \frac{2y_p\sin(\tan^{-1}(\lambda/2))}{\lambda}$$

Where $\lambda = \left(\sqrt[2]{(x_p^2 + y_p^2)}\right)/f$ and f is the focal length in pixels which can be calculated using

$$f = \frac{image\_width}{4\sin(FOV_{horz}/2)}$$

Thus for every target pixel, the corresponding pixel location in the fisheye image can be computed and this data can be stored in the form of an LUT. All parameters needed for LUT generation are known before-hand such as the imager characteristics (including imager FOV and size of the input frame), and display system characteristics (including the display size). Therefore for a given imager and display system, the LUT is computed only once and off-line.

In Figure 3 left and right, the results of correcting the fisheye images of Figure 1 left and right, respectively, are shown. Note that to get a corrected fisheye image with a 180° FOV along some direction requires that the size of the corrected image be infinite. Since this is impossible, the FOV of the target is restricted to less than 180°. In the case of a full-frame fisheye image, where the FOV of the source image along horizontal and vertical directions is less than 180°, if the FOV of the target image is greater than that of the source fisheye, then points outside the FOV of the source image are rendered black. This produces the characteristic “bow tie” effect in which the corners are stretched out as depicted in Figure 3 (right). This can be avoided by ensuring that the FOV of the target is always sufficiently smaller than that of the source fisheye.

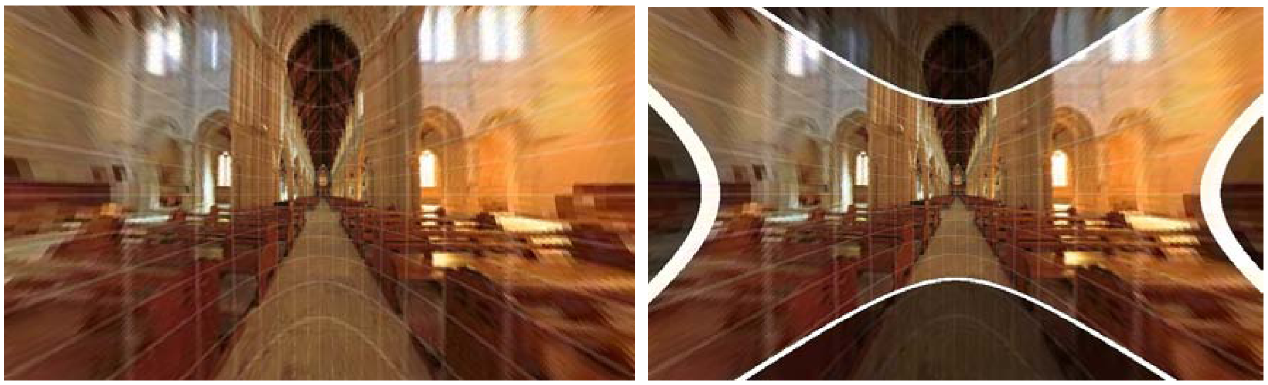
Figure 3: Fisheye-Corrected Images—Circular (left) and Full-Frame (right)

9-Point Interpolation

A LUT with real values (needing floating-point representations) will be unwieldy for an FPGA. However, the 9-point interpolation scheme addresses this issue by providing a simple and efficient method for pixel interpolation that avoids the storage of real numbers in the LUT. Alternate methods for pixel interpolation are less successful. Nearest neighbor interpolation is a simple and fast method for computation, but is also somewhat coarse and can lead to visible image artifacts in the interpolated image. More complex interpolation techniques such as bilinear interpolation would involve floating-point operations that the FPGA is not suited to handle efficiently. The 9-point interpolation scheme is a middle path between the nearest neighbor and bilinear interpolation schemes. It involves mapping the LUT's real-valued pixel to the nearest of its nine neighbors, as shown in Figure 4.

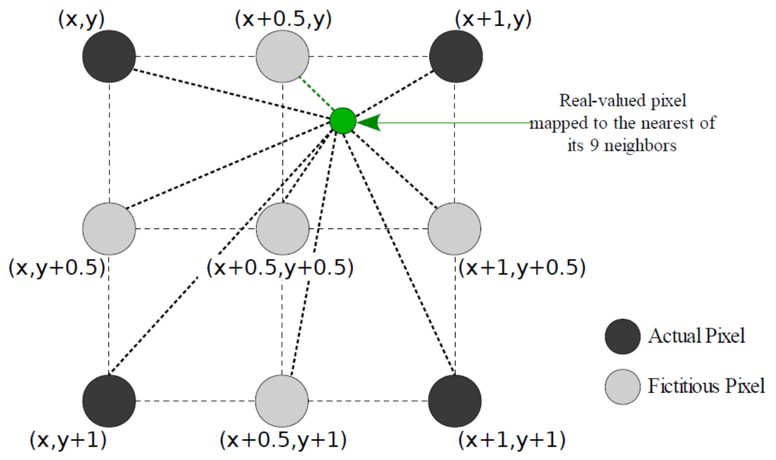

Figure 4: The 9-Point Interpolation Scheme

In this method, the intensity values of all fictitious pixels are calculated by taking the average of the intensities of its adjacent actual pixels. For example, the color intensity of the pixel (x, y+0.5) is the average of the color intensities of the actual pixels (x, y) and (x, y+1). The intensity value of (x+0.5, y+0.5) is the average of the intensities of the actual pixels (x, y), (x+1, y), (x, y+1) and (x+1, y+1). The main advantage this technique possesses over the two formerly discussed methods is simplified computation without significant sacrifice in the quality of the corrected image. This is because the only computation involved is taking the averages of either two or four quantities. Division by 2 can be realized by "right-shifting" the sum of the numbers by one, while division by 4 can be done by "right-shifting" the sum by two bits. This sort of computation is very simple for an FPGA. Note that with the 9-point interpolation scheme there is no need for the LUT to store any real-valued pixel locations. It can directly store the location of either the actual or the fictitious pixel to which the real-valued pixel is mapped. This can be easily achieved with fixed-point representations.

Design Implementation

This section discusses the implementation of the fisheye correction using devices from the Altera® Cyclone® FPGA series and the Nios® II soft-core embedded processors [3], [4]. The Nios II architecture is a RISC soft-core architecture, which is implemented entirely in the programmable logic and memory blocks of Altera FPGAs, and is capable of handling a wide range of embedded computing applications, from DSP to system-control. Altera's Nios® II Embedded Evaluation Kit (NEEK) is used as the development platform.
As shown in Figure 5, the hardware architecture is based on the following modules: Nios II soft-core embedded processor, a BT-656 video input, an I^2C configuration, a DDR-SDRAM controller and an LCD controller.

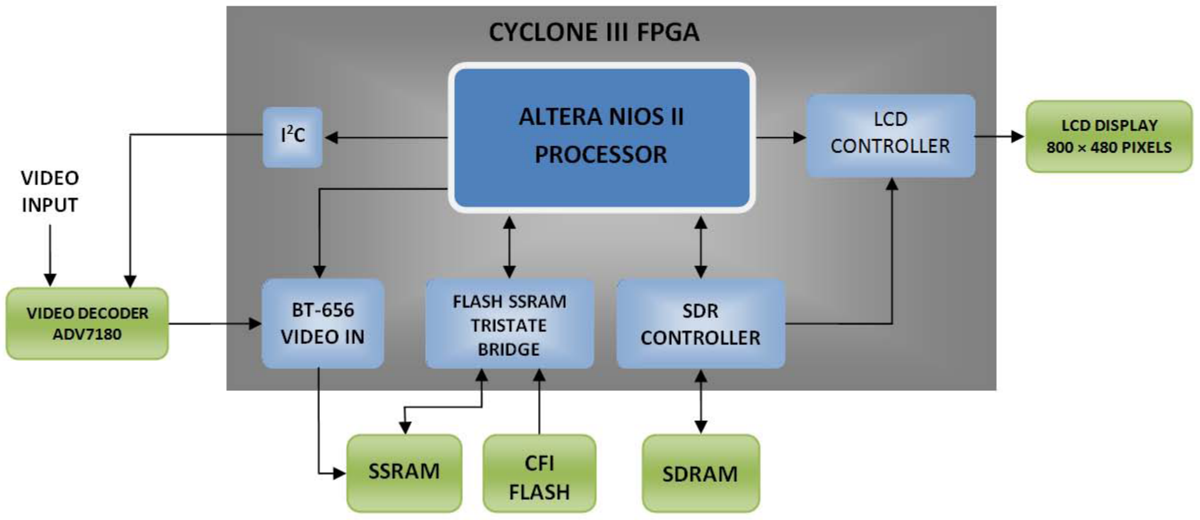

Figure 5: Internal Architecture

Memory Requirement and Management

There is continuous data input from the camera and this requires quick real-time processing for optimum throughput. Effective memory management becomes critical for storing and retrieving image and additional processing data required for fisheye correction. The architecture uses the available memory resources on the NEEK for the input frame buffer (implemented on the SS RAM), output frame buffer (implemented on DDR-SDRAM), and for storing and reading the static LUT (implemented on DDR-SDRAM). All these memory functions are performed using control signals sent from the CPU to the SDR controller. The SDR controller connects to the SDRAM chip and handles all SDRAM protocol requirements. The resolution of the input camera used is 720×576 pixels (interlaced input 60 fields/sec).The Pseudo Code for the following setup is shown below:

```
DETERMINE memory size of LUT by width of display × height of display × 4 bytes
        ASSIGN memory for static LUT on SDRAM
```

```
    OPEN flash memory device
    COPY the static LUT from flash memory to SDRAM
    CLOSE the flash memory device

COMPUTING THE OUPTUT FRAME
FOR each row of the output image
    FOR each column of the output image
        OBTAIN the value of pixel position from the static LUT
            IF pixel location is INVALID THEN
                ASSIGN output pixel as BLACK
            ELSE
                IF pixel location represents REAL pixel THEN
                    ASSIGN the output pixel with the corresponding input
                    pixel intensity value
                ELSE IF pixel location represents FICTITIOUS pixel THEN
                    COMPUTE the average intensity of the nearest actual input
                    pixels depending on whether field is even or odd
                    ASSIGN the value to the output pixel
                ENDIF
            ENDIF
        ENDFOR
    ENDFOR
DISPLAYING THE FRAME
    ASSIGN the screen buffer with the computed output frame
```

Results

The architecture discussed above was used to process the continuous video input (NTSC with two interlaced fields per frame) from an OmniVision OV7950 camera with an FOV of 120°, and the output was displayed on the NEEK's LCD display with a resolution of 800x480. The performance of the system while using the 9-point interpolation scheme was compared to that of the nearest neighbour scheme. Further it was found that correcting a single field and updating the display buffer while the other field is being written to the input image buffer almost doubles the speed compared to correcting an entire frame and updating the display buffer. These results are summarized in the following table:

< Please put a table here with the following entries >

Interpolation technique	Frames per second (fps)
9-point (correcting a frame at a time)	4
9-point (correcting a field at a time)	8
NN (correcting a frame at a time)	9
NN (correcting a field at a time)	17

Conclusions

Thus this paper discusses an efficient and novel architecture for real-time fisheye correction in wide-angle cameras using FPGAs and soft-core embedded processor technology. This architecture is flexible, scalable, and makes efficient use of the FPGA's resources. This algorithm can be suitably accelerated using custom hardware on FPGA's. Because the architecture's Nios II processor is versatile and powerful enough to take on additional embedded processor functions, this technology is ideally suited for use in applications where wide-angle cameras are used such as automotive rear-view cameras and others.

Acknowledgement

The authors thank Manipal Dot Net Pvt. Ltd. and Altera ® Corporation for providing the facilities and infrastructure for research and implementation of the algorithm.

References

[1] "Fisheye lens," Wikipedia: http://en.wikipedia.org/wiki/Fisheye_lens
[2] David Salomon*, "Transformations and Projections in Computer Graphics"*, Springer, 2006.
[3] *Nios II Processor Reference Handbook*: www.altera.com/literature/lit-nio2.jsp
[4] Nios II Embedded Evaluation Kit, Cyclone® III Edition, User Guide: www.altera.com/literature/ug/niosii_eval-user-guide.pdf
[5] Nios II Flash Programmer User Guide: www.altera.com/literature/ug/ug_nios2_flash_programmer.pdf