Software-based Correction of Radial Lens Distortion

wcrr51
Department of Computer Science
Durham University
Durham, United Kingdom
wcrr51@durham.ac.uk

***Abstract*—This paper looks at the implementation and evaluating of a virtual reality lens distortion correction system. Using a simplified version of Brown's radial distortion model, this paper provides two implementations for pincushion distortion pre-correction using the Unity Game Engine. Lateral Chromatic Aberration is also considered and a simple correction model is implemented and demonstrated.**

***Index Terms*—Virtual reality, lens, pincushion distortion, barrel distortion, lateral chromatic aberration, distortion correction.**

For implementation and runtime usage details, see `README.txt`.

I. Introduction and Background

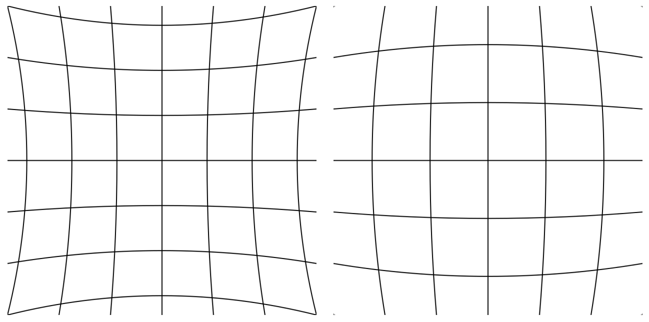
Head-mounted displays (HMDs) represent the current state of the art in virtual reality (VR). As the user requires a wide field of view (FOV) for immersion, lenses are used to project the emitted image (from the near-by screen) into the eye, making the image appear larger and further away.

However, the real-world physical properties of these lenses incur the introduction of visible distortions and optical aberrations. The class of these that will be considered in this paper are referred to as radial distortions. These distortions are rotation-wise invariant (about the centre of the image) and vary only in their radius.

One such example of these is pincushion distortion. Pincushion distortion appears as the stretching of objects the further they are from the centre of the lens, causing the corners of the image to become more *pointed* (hence a cushion shape as shown in Fig. 1 (a)). Playing in VR with this can quickly become disorientating or nauseating due to greater acceleration of object towards the periphery compared to the centre.

In order to correct for pincushion distortion, the opposite distortion can be used. This conversely causes objects to become more compressed (or magnified) the further away they are from the origin, making the corners more *rounded*. Because of this appearance, it is called barrel distortion (Fig. 1 (b)).

Another example that will be briefly considered is lateral chromatic aberration (LCA). Unlike longitudinal chromatic aberration, that causes a colour-variant blurriness across the whole image, LCA causes a fringing between colours dependent on how far they are from the centre of the lens. This is most visible towards the corners of an image appearing as colour-fringing on edges or high contrast areas. LCA is caused by the fact that since the different colours have different wavelengths, they are refracted differently to each other, meaning, for a given lens, they have a different focal point. This is shown in Fig. 2.



(a) Pincushion Distortion (b) Barrel Distortion

Fig. 1. Barrel and pincushion distortion (by WolfWings for Wikipedia, USPD 2008).

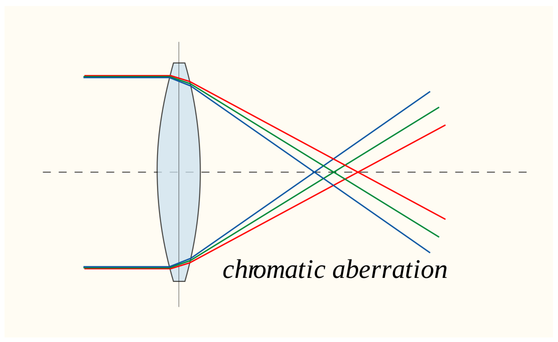


Fig. 2. Diagram of chromatic aberration (by Bob Mellish for Wikipedia, CC 2006).

II. Pincushion Distortion Correction

As previously mentioned, radial distortion is only dependent on the distance between the origin and the point being considered. Because of this, it can be thought of as a transformation which, according to some function f, moves the point to a different location on the line intersecting it and the origin. More specifically, for a point (r_u, θ_u) in polar coordinates, $r_d = f(r_u), \theta_d = \theta_u$ is a radial distortion where (r_d, θ_d) represents the distorted polar coordinates of the point.

Brown's model for lens distortion [1] builds upon that presented by Conrady [2], presenting a mathematical representation of barrel/pincushion and tangential distortion. The rest of this report will consider a simplified version of this model in which there is no tangential distortion and is evaluated up to r^5 (and r^8 for its inverse). This simplified version of Brown's radial distortion model is shown in Eq. 1. The approximation of its inverse is shown in Eq. 2.

$$r_d = f(r_u) = r_u + c_1 r_u^3 + c_2 r_u^5 \tag{1}$$

$$r_u = f^{-1}(r_d) \approx \frac{c_1 r_d^2 + c_2 r_d^4 + c_1^2 r_d^4 + c_2^2 r_d^8 + 2c_1 c_2 r_d^6}{1 + 4c_1 r_d^2 + 6c_2 r_d^4} \tag{2}$$

Where c_1 and c_2 are per-lens constants that determine the extremity of the distortion. These are usually calculated experimentally, however, for the static renders in this paper, they will be set to arbitrary values of $c_1 = 0.15$ and $c_2 = 0.10$. While these are relatively extreme, they clearly exemplify the concepts being demonstrated. Its worth noting that, due to the approximate nature of the inverse distortion function, the greater values of c_1 and c_2, the more $f^{-1}(f(r))$, $f(f^{-1}(r))$ and r diverge.

For practical implementations, computers represent points using cartesian coordinates. Because of this, any transformations first need to convert the coordinates to polar form. Given a two dimensional cartesian point $p = (x, y)$, its polar form is $\left(r = \sqrt{x^2 + y^2} = \sqrt{p \cdot p}, \theta = \text{arctan2}\,(y, x)\right)$. Given a two dimensional polar point (r, θ), its cartesian form is $(x = r\cos\theta, y = r\sin\theta)$.

By using the fact that only r is varied during a radial distortion, an optimisation can be used whereby only r, or the required powers of r, are calculated, and the cartesian coordinates are simply scaled by that applied during the distortion (removing the need to calculate θ, $\sin\theta$, and $\cos\theta$). Applying this to Brown's model, for a point (x_u, y_u), [3] calculates its distorted point (x_d, y_d) as shown in Eq. 3.

$$\begin{pmatrix} x_u \\ y_u \end{pmatrix} = \begin{pmatrix} x_d \\ y_d \end{pmatrix} \left(1 - f^{-1}\left(\sqrt{x_d^2 + y_d^2}\right)\right), \tag{3}$$

All distortions are applied on top of the scene shown in Fig. 3. Any intermediary render textures are 1024×1024 pixels and the output resolution is 500×500 pixels unless otherwise specified.

A. *Pixel-based Distortion Correction*

The pixel-based distortion correction method works by transforming UV positions in the texture map for each pixel (or fragment). In a normal fragment shader, the graphics card provides texture UV positions (between $(0, 0)$ and $(1, 1)$) interpolated between those specified in the vertex shader. A `sampler2D` is then used to retrieve the colour of the pixel in the texture corresponding to the UV position. In order to perform distortion, the UV position being sampled can be changed.

Firstly, both the forward and inverse radial distortions functions require points to be normalised to between $(-1, -1)$ and $(1, 1)$. For input UV position p_{UV}, this is achieved simply

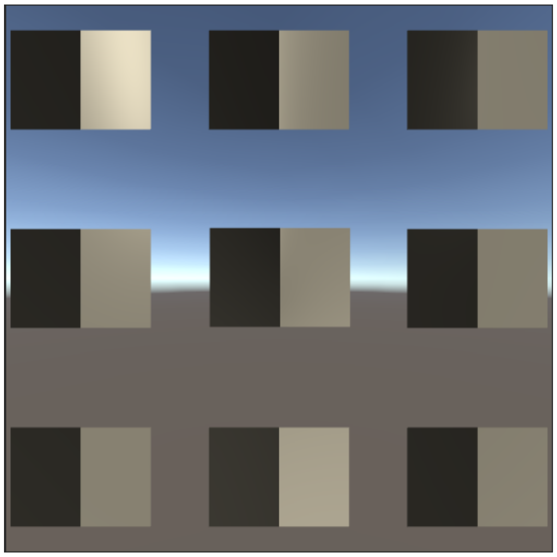
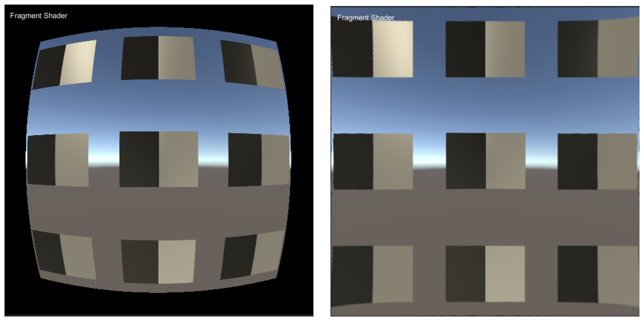


Fig. 3. Untouched and undistorted 9-cube scene.

using $p_{\texttt{norm}} = 2p_{\text{UV}} - 1$. After the distortion transformation is completed, the normalised points are converted back to UV positions using $p'_{\text{UV}} = 0.5(p'_{\texttt{norm}} + 1)$.

Somewhat non-trivially, to perform a radial barrel or pincushion distortion, the opposite equation must be used (Eq. 1 for barrel and Eq. 2 for pincushion). This is because sampling a different UV value to the current one is already the inverse operation. Instead of imagining the current UV point A being transformed to its distorted UV point B, its better to think of it as finding the UV point that should replace A, and calculating this to be B. Hence, for barrel distortion, considering the point A, it needs to be replaced by a point B further away from the origin (effectively distorting by compression). In this case, B is found by inverting the distortion that lands on A, therefore using Eq. 1 (and vice versa for pincushion distortion).

The fragment shader implementation simply performs this operation for each pixel/fragment in the image, and discards any fragments who's distorted UV positions lie outside the bounds of the texture. The results of this and simulation of it running through a lens are shown in 4.



(a) Pre-distortion (b) Correction and distortion

Fig. 4. Pixel-based pre-distortion and lens simulation.

B. Mesh-based Distortion Correction

The mesh-based distortion correction method works by transforming the vertex positions of a mesh (instead of transforming texture UV positions). In doing so, it looks to exploit the graphics rendering pipeline for a performance gain with minimal residual artifacts.

The meshes were produced in Blender by continually subdividing a plane (the 8×8 mesh is shown in Fig. 5). An $n \times n$ mesh has n^2 quads, $2n^2$ triangles, or $(n+1)^2 = n^2 + 2n + 1$ vertices. As each face is subdivided into four subfaces, n is a power of 2.

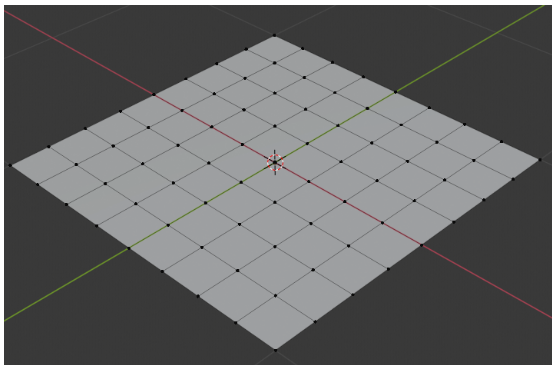


Fig. 5. 8×8 mesh in Blender.

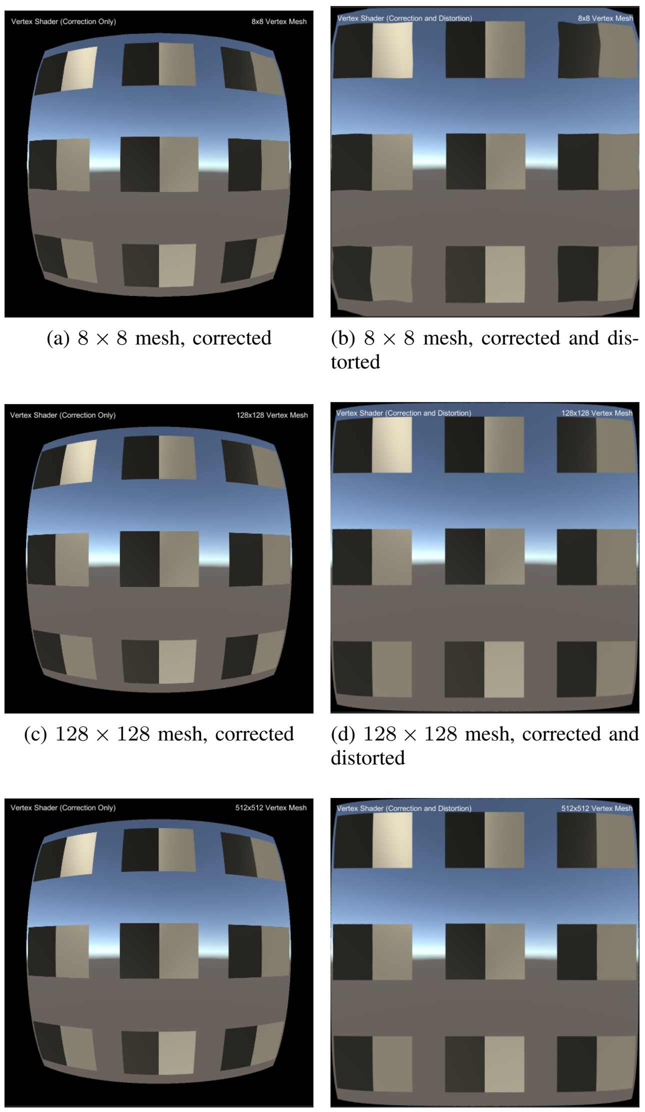
In the vertex shader, vertices are first transformed from world-space to clip-space using its model-view-projection matrix. Since clip-space is between $(-1,-1)$ and $(1,1)$, it matches the domain of both Eq. 1 and Eq. 2. This means that the respective radial transforms can be applied to the vertices in clip-space to distort their positions.

Opposite to the pixel-based method, applying pre-distortion correction to the output requires the inverse radial transform (Eq. 2), and simulating the distortion requires the forward radial transform (Eq. 1). This works by translating the vertices (as if they were distorted) to their pre-distorted positions. As part of the rendering pipeline, during the fragment shader, each of these pre-distorted vertex positions are linearly interpolated by the graphics card to produce estimates for pre-distorted fragments/pixels.

In order to simulate a lens being applied to the pre-distorted image, as mentioned, the same technique is applied using the forward radial transform (Eq. 1). This then cancels out the pre-distortion to produce an image similar to that which was input ($f^{-1}(f(r)) = r$).

As an aside, similar results could be achieved by performing the same texture UV transformation in the vertex shader as that used by the pixel-based method in the fragment shader. While this would cause the same distortion, as vertices remain on the edge, a mesh image of original size would be output, causing redundant additional distortion and calculations near the edge (instead of the black background).

Fig. 6 shows the mesh-based method applied to three meshes (8×8, 128×128, and 512×512) with correction only (a, c, and e) and correction with distortion (b, d, and f). The main visible difference is between 8×8 and 128×128, where clear straight lines blend into curved lines. There is also a clear difference in correction and distortion applied together, with the 8×8 mesh exhibiting clear artifacts towards the corners. Between 128×128 and 512×512 there is no perceivable difference in output, implying that at this level the difference is sub-pixel. This makes logical sense since the number of quads (and by extension vertices) exceed the number of pixels in the image.


Vertex Shader (Correction Only)
8x8 Vertex Mesh
Vertex Shader (Correction and Distortion)
8x8 Vertex Mesh
Vertex Shader (Correction Only)
128x128 Vertex Mesh
Vertex Shader (Correction and Distortion)
128x128 Vertex Mesh
Vertex Shader (Correction Only)
512x512 Vertex Mesh
Vertex Shader (Correction and Distortion)
512x512 Vertex Mesh

(a) 8×8 mesh, corrected

(b) 8×8 mesh, corrected and distorted

(c) 128×128 mesh, corrected

(d) 128×128 mesh, corrected and distorted

(e) 512×512 mesh, corrected

(f) 512×512 mesh, corrected and distorted

Fig. 6. LCA pre-distortion and corrected LCA with distortion.

For benchmarking and profiling, on an i7 6700K with an RTX 2080 at 500×500 pixels, all three meshes performed the same (about 1.7ms per frame). However, increasing the resolution to 8192×8192 yields more reliable variation on the frame rate and a clear performance drop as mesh complexity increases. The profiling results are shown in Tab. I. These results, combined with the minimal visual difference between the more detailed meshes, show that a low-poly mesh is a good trade-off between frame rate and visual quality.

TABLE I
PROFILING RESULTS FOR THREE MESHES AT 8192×8192 RESOLUTION.

Mesh	FPS	CPU (ms)	GPU (ms)	Total (ms)
8×8	~ 135	7.3	0.3	7.3
128×128	~ 128	7.8	0.3	7.9
512×512	~ 104	9.4	0.3	9.5

C. Pixel-based vs Mesh-based Distortion Correction

Both techniques have advantages and drawbacks.

Firstly, pixel-based pre-distortion in the fragment shader involves calculating Eq. 1 for each fragment/pixel in the scene, including those which are eventually discarded. On the other hand, mesh-based pre-distortion in the vertex shader involves calculating Eq. 2 for each vertex in the mesh. This means that, for a mesh with far less vertices than pixels being rendered to the screen, the mesh-based method performs far less direct distortion calculations than the pixel-based method, giving it better theoretical performance.

Secondly, while the pixel-based method produces the most accurate theoretical discrete output, in practice, the mesh-based method is able to produce indistinguishable results for a low number of vertices relative to pixels/fragments.

From an implementation efficiency standpoint, one drawback of the mesh-based method is that using Eq.2 means it needs to perform more operations per vertex calculation than the pixel-based method does per pixel. It does however have the added benefit that the input coordinates are already normalised to between -1 and 1. Additionally, since it only uses even powers of r, r^2 can be trivially calculated using $r^2 = x^2 + y^2 = p \cdot p$, removing the need for an expensive square root.

For the frame buffer resolution of 1024×1024 pixels used in the render targets, the fragment shader in the pixel-based method should perform $1024^2 = 1048576$ distortion calculations and up to 1048576 texture lookups per frame (dependent on the number of discarded fragments). On the other hand, the mesh-based method only performs $(n+1)^2$ distortion calculations for an $n \times n$-quad mesh (81 calculations for 8×8, 16384 for 128×128, and 262144 for 512×512), but will still require up to 1048576 texture lookups as the fragment shader samples the texture for each fragment in the buffer.

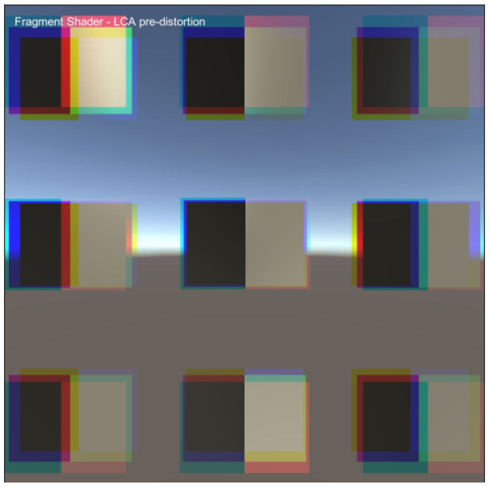
Finally, while not used in this report, the mesh-based method can be further refined by pre-baking its distorted vertex positions, removing the need to do any distortion calculations at runtime.

III. LCA CORRECTION

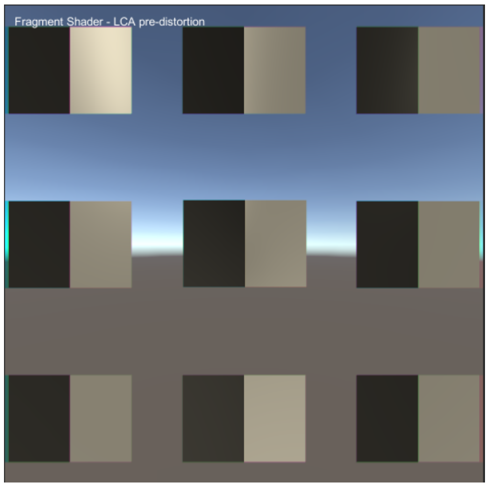
As monitors emit red, green, and blue light, this report will focus on these for LCA correction, however, it is worth noting that LCA applies to all wavelengths.

LCA causes blue light to be refracted further away from the normal than green and red due to its lower wavelength. Similarly, red light is refracted less further away from the normal than green due to its higher wavelength. As this happens radially, the further away from the centre of the lens, the greater the colour separation. This occurs linearly in r, and is controlled by respective red, green, and blue coefficients.

Because of this, a pre-distortion correction for LCA is implemented in the fragment shader such that red light is magnified further than green light and green light is magnified further than blue light. To show how this cancels out LCA, lens simulation is applied to the camera which includes both LCA and pincushion distortion. LCA correction before and after lens distortion simulation are shown in Fig. 7 (a) and (b) respectively.



(a) Correction



(b) Correction and Distortion

Fig. 7. LCA pre-distortion and corrected LCA with distortion.

IV. FURTHER CONSIDERATION FOR VR

This section will assess some further considerations regarding implementation in real-world VR systems and the problems that they might incur.

A. Assistance from Eye Tracking

Foveated rendering [4] is a technique whereby eye-tracking is used to dynamically adjust rendering quality dependent on what part of the screen is being looked at. LCA undistortion, aside from gaining the benefits of foveated rendering, may additionally benefit from eye tracking to allow areas of the screen being looked at to incorporate more advanced (but expensive) localised LCA models, while offering a worse but more computationally efficient alternative elsewhere on screen. Tying into the next topic, barrel distortion (as a correction for pincushion distortion) can make use of eye tracking by restoring objects being looked at to a higher resolution. Movement of the eye places the pupil off-centre relative to the origin of the lens. This deviation from the focal point causes additional distortion (mentioned in [5, ch. 5]). While the eye tracking-based barrel distortion is being calculated, it also provides a convenient time to pre-distort the image to account for the pupil being misaligned with the focal point.

B. Barrel Distortion and Resolution

When performing barrel distortion, some parts of the image are more compressed than others. The primary effect this has on the image is the production of a fair amount of unusable pixels around the edge, effectively wasting the extra resolution this provides. As a result, the image reaching the eye (having undergone pincushion distortion) is of a lower effective resolution (as the space required by a larger number of pixels are actually output from a lower number, effectively causing a radial reduction in perceived quality). [5] mentions a solution which involves radially increasing the hardware resolution of the screen in line with the barrel distortion.

For a given uniform display panel resolution, supersampling the rendering resolution (increasing it relative to that of the display panel) helps make use of the wasted space around the barrel distortion. It is worth noting that no matter the super-sampling resolution, for a uniform display, there will always be a radial reduction in perceivable resolution. For modern headsets, if using a display panel of uniform resolution, such a resolution should match the lowest perceivable resolution at the corners of the screen. The more sensible option is to adopt the previously mentioned radially increasing resolution.

References

[1] D. C. Brown, “Decentering distortion of lenses,” *Photogrammetric Engineering and Remote Sensing*, 1966.

[2] A. Conrady, “Lens-systems, decentered,” *Monthly notices of the royal astronomical society*, vol. 79, pp. 384–390, 1919.

[3] J. Mallon and P. F. Whelan, “Precise radial un-distortion of images,” in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 1. IEEE, 2004, pp. 18–21.

[4] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, and A. Lefohn, “Towards foveated rendering for gaze-tracked virtual reality,” *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, pp. 1–12, 2016.

[5] S. M. LaValle, *Virtual Reality*. Cambridge University Press, 2020. [Online]. Available: http://vr.cs.uiuc.edu/