

# Introduction to Higher-order aberrations

Linghan Zheng  
Shanghai Jiao Tong University  
linghan0919@163.com

## ABSTRACT

The project Vision Correction Display is aimed to let the viewers who have eye problems see the display clearly without glasses or contact lenses. Shichao Yue has already implemented the algorithm that can almost correctly compensate for the defocus aberrations. However, it failed to deal with higher-order aberrations successfully. After reading a lot related materials, I have made some improvement and now the code can work for higher-order aberrations properly. This report would give a basic introduction of vision correcting algorithm for higher-order aberrations, including a simple description of Zernike polynomials and the ray tracing mechanism to map pixels between screen and sensor.

**Keywords:** Higher-order Aberrations, Zernike polynomials

## ZERNIKE POLYNOMIALS

The wavefront surface describes the directions of rays leaving the pupil at the same phase, and we only need to propagate these rays back to the display and determine the intersections. The Zernike polynomials are a set of functions that are orthogonal over the unit circle. They are useful for describing the shape of an aberrated wavefront in the pupil of an optical system or true height corneal topography[1].

The wavefront map is simply a height field defined on a circular plane where  $z = w(x^a, y^a) = w(r \cos(\theta), r \sin(\theta))$ . The wavefront map can be decomposed into a

series of mutually orthogonal circular basis functions as the following:

$$\begin{aligned}
w(x^a, y^a) &= c_0^0 Z_0^0(x^a, y^a) \\
&\quad + c_1^0 Z_1^0(x^a, y^a) + c_1^{-1} Z_1^{-1}(x^a, y^a) + c_1^{+1} Z_1^{+1}(x^a, y^a) \\
&\quad + c_2^0 Z_2^0(x^a, y^a) + c_2^{-1} Z_2^{-1}(x^a, y^a) + c_2^{+1} Z_2^{+1}(x^a, y^a) + c_2^{-2} Z_2^{-2}(x^a, y^a) + c_2^{+2} Z_2^{+2}(x^a, y^a) \\
&\quad + \dots \\
&= \sum_{n=0}^{\infty} \sum_{m=-n}^{+n} c_n^m Z_n^m(x^a, y^a)
\end{aligned}$$

where  $n$  is the radial index,  $m$  is the azimuth index,  $c_n^m$  are the scaling weights, and  $Z_n^m(x^a, y^a)$  are the Zernike polynomials.

Sometimes, it is useful to refer to a specific Zernike polynomial using just a single index, or describe the expansion coefficients as a vector. The single indexing scheme is defined by using the double indices  $n$  and  $m$ :

$$j = \frac{n(n+2) + m}{2}.$$

The Zernike polynomials can be represented in both polar coordinate and Cartesian coordinate. In our project, we use Cartesian coordinate. The following table can show the first fifteen terms of  $Z_n^m(x^a, y^a)$  and  $dZ_n^m/dx$ ,  $dZ_n^m/dy$ , which can describe the normal directions of rays leaving the pupil using tangents at the sampled locations.

j	n	m	$Z_n^m$ in Cartesian coordinate	$dZ_n^m/dx$	$dZ_n^m/dy$
0	0	0	1	0	0
1	1	-1	2y	0	2
2		+1	2x	2	0
3	2	-2	$2\sqrt{6}xy$	$2\sqrt{6}y$	$2\sqrt{6}x$
4		0	$2\sqrt{3}(x^2 + y^2) - \sqrt{3}$	$4\sqrt{3}x$	$4\sqrt{3}y$
5		+2	$\sqrt{6}(x^2 - y^2)$	$2\sqrt{6}x$	$-2\sqrt{6}y$
6	3	-3	$\sqrt{8}(3x^2y - y^3)$	$6\sqrt{8}xy$	$\sqrt{8}(3x^2 - 3y^2)$
7		-1	$\sqrt{8}(3x^2y + 3y^3 - 2y)$	$6\sqrt{8}xy$	$\sqrt{8}(3x^2 + 9y^2 - 2)$
8		+1	$\sqrt{8}(3xy^2 + 3x^3 - 2x)$	$\sqrt{8}(9x^2 + 3y^2 - 2)$	$6\sqrt{8}xy$
9		+3	$\sqrt{8}(x^3 - 3xy^2)$	$\sqrt{8}(3x^2 - 3y^2)$	$-6\sqrt{8}xy$
10	4	-4	$\sqrt{10}(4x^3y - 4xy^3)$	$\sqrt{10}(12x^2y - 4y^3)$	$\sqrt{10}(4x^3 - 12xy^2)$
11		-2	$\sqrt{10}(8x^3y + 8xy^3 - 6xy)$	$\sqrt{10}(24x^2y + 8y^3 - 6y)$	$\sqrt{10}(8x^3 + 24xy^2 - 6x)$
12		0	$\sqrt{5}(x^2 + y^2)^2 - 6\sqrt{5}(x^2 + y^2) + \sqrt{5}$	$\sqrt{5}(24x^3 + 24xy^2 - 12x)$	$\sqrt{5}(24y^3 + 24x^2y - 12y)$
13		+2	$\sqrt{10}(x^2 - y^2)(x^2 + 7y^2)$	$\sqrt{10}(16x^3 - 6x)$	$\sqrt{10}(-16y^3 + 6y)$
14		+4	$\sqrt{10}(x^4 - 6x^2y^2 + y^4)$	$\sqrt{10}(4x^3 - 12xy^2)$	$-\sqrt{10}(12x^2y - 4y^3)$

In our code, we only consider the first ten cases, they are (0) piston, (1) y-tilt, (2)

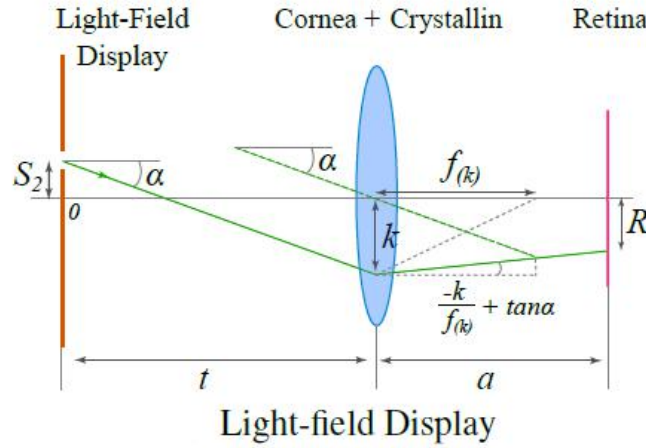
x-tilt, (3) oblique astigmatism, (4) defocus, (5) vertical astigmatism, (6) vertical trefoil, (7) vertical coma, (8) horizontal coma and (9) oblique trefoil.

The principle of Zernike polynomial is very complex for a non-optics-major students like me. But the information above is enough to help me design the algorithm. Anybody who is interested in this part can study further by oneself.

## RAY-TRACING

Now we can start to use Zernike polynomials to help describe the wavefront, and then calculate the mapping relationship between screen pixel and sensor point.

In paper[3], there displays an figure showing abstract light-field display:



Here, the light-field setup is placed a distance  $t$  from the eye, which has axial length (size of the eyeball) equals to  $a$ . A light-field ray from  $S_2$  at an angle  $\alpha$  reaches a retinal location  $R$  through a corneal point  $k$ . The focal length  $f(k)$  comes from the eye's wavefront map and drives the refraction of the light ray. In our program, we also calculate the refraction in a similar way.

In order to make the description clear, we define some symbols first :

**Tipoint screen\_pos** : the (x,y) position on screen, just as  $S_2$  above.

**Tipoint sensor\_pos** : the (x,y) position on sensor, just as  $R$  above.

**Tipoint lens\_pos** : the (x,y) position on lens, just as  $k$  above.

**disObj** : the distance between object and lens, just as  $t$  above.

**eyeDepth**<sup>1</sup> : the distance between lens and retina, just as  $a$  above.

**TiDirection origin\_dir** = (lens\_pos - screen\_pos) / disObj : the original direction when a ray entry the eye, just as  $\tan \alpha$  above.

**TiDirection wave\_dir** : the deflection direction, just as  $\frac{-k}{f(k)}$  above.

**TiDirection final\_dir** = wave\_dir + origin\_dir : the final direction of the ray when leave lens. With final\_dir, we can use “sensor\_pos = lens\_pos + final\_dir \* eyeDepth;” to get sensor postion.

Obviously, the most important work of ray tracing to compute wave\_dir, which is done by function wavefront\_derive :

**TiDirection wavefront\_derive**

**(const TiPoint& p, const std::vector<float>& c, double aperture)**

Where p is lens\_pos, c is a vector recording Zernike coefficients and aperture is the radius of lens.

Wave\_dir includes two main parts. The first is the aberrations described by Zernike polynomials while the second is the intrinsic focus of eyes.

First, as what we discussed above, Zernike polynomials are used in the unit circle. So we should normalize lens position p by using “x = p[0] / aperture; y = p[1] / aperture;”. Next, we can calculate the directions resulted by Zernike functions.

$$\begin{aligned}zx = & c[1]*0 + c[2]*2 + \\ & c[3]*2\sqrt{6}y + c[4]*4\sqrt{3}x + c[5]*2\sqrt{6}x + \\ & c[6]*6\sqrt{8}xy + c[7]*6\sqrt{8}xy + \\ & c[8]*\sqrt{8}(9x^2 + 3y^2 - 2) + \\ & c[9]*\sqrt{8}(3x^2 - 3y^2)\end{aligned}$$

---

<sup>1</sup> eyeDepth was called as dislmg in previous code, but I have changed the name later.

$$\begin{aligned}
zy = & c[1]*2 + c[2]*0 + \\
& c[3]*2\sqrt{6}y + c[4]*4\sqrt{3}x + c[5]*(-2\sqrt{6}x) + \\
& c[6]*\sqrt{8}(3x^2 - 3y^2) + c[7]*\sqrt{8}(3x^2 + 9y^2 - 2) + \\
& c[8]*6\sqrt{8}xy + \\
& c[9]*(-6\sqrt{8}xy)
\end{aligned}$$

Second, because the eye has ability to deflect rays except aberrations, we should let “ $zx = zx - p[0]/focal$ ” and “ $zy = zy - p[1]/focal$ ”. Why? It is inferred by let Zernike part equal to zero and assume the lens is a perfect convex.

As a result, the `wave_dir` is calculated, and then it is easy to compute the sensor position. Below are part of codes from `HOAprojection.cpp` created by Linghan.

```

TiPoint compute_sensor_pos(const TiPoint& screen_pos, const TiPoint& lens_pos) {
    if (lens_pos[0] > 1e5 || lens_pos[1] > 1e5) return TiPoint(std::numeric_limits<float>::infinity, std::numeric_limits<float>::infinity);
    TiDirection wave_dir = wavefront_derive(lens_pos, zernike, aperture);
    TiDirection origin_dir = (lens_pos - screen_pos) / disObj;
    TiDirection final_dir = wave_dir + origin_dir;
    TiPoint sensor_pos = lens_pos + final_dir * disImg;
    return sensor_pos / projected_pixel + TiPoint(projected_size / 2.0, projected_size / 2.0);
}

```

```

TiDirection
wavefront_derive(const TiPoint& p, const std::vector<float>& c, double aperture) {
    float x = p[0] / aperture; // limited in the unit circle
    float y = p[1] / aperture;
    //printf("x:%f, y:%f\n", x, y);
    float zerderx =
        c[3] * 2 * sqrt(6) * y +
        c[4] * 4 * sqrt(3) * x +
        c[5] * 2 * sqrt(6) * x +
        c[6] * 6 * sqrt(8) * x * y +
        c[7] * 6 * sqrt(8) * x * y +
        c[8] * sqrt(8) * (9 * x * x + 3 * y * y - 2) + // added by Yoyo
        c[9] * sqrt(8) * (3 * x * x - 3 * y * y);

    float zerdery =
        c[3] * 2 * sqrt(6) * x +
        c[4] * 4 * sqrt(3) * y +
        c[5] * (-2) * sqrt(6) * y +
        c[6] * 3 * sqrt(8) * (x * x - y * y) +
        c[7] * sqrt(8) * (3 * x * x + 9 * y * y - 2) +
        c[8] * 6 * sqrt(8) * x * y + // added by Yoyo
        c[9] * (-6) * sqrt(8) * x * y;




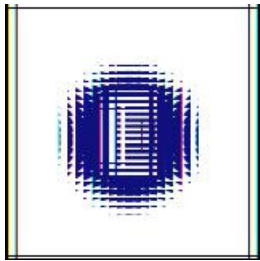


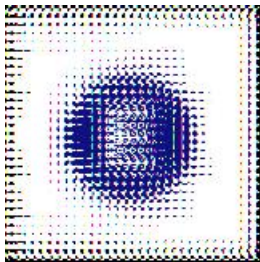

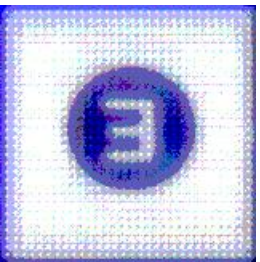
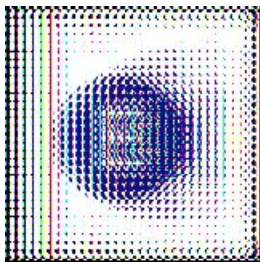

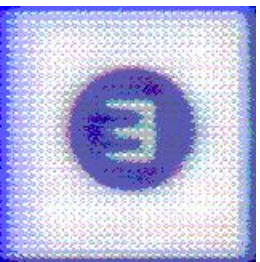
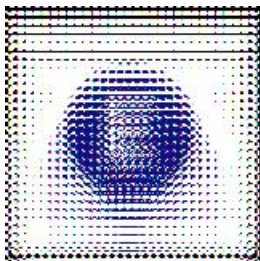

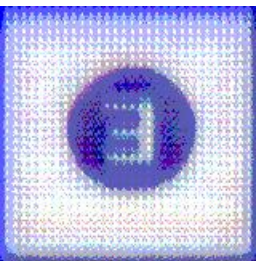
    zerderx = zerderx - p[0]/focal;
    zerdery = zerdery - p[1]/focal;

    return TiDirection(zerderx, zerdery);
}

```

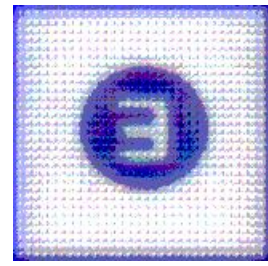
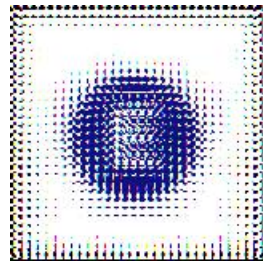
## RESULTS

The following are some results of my corrected program:

Aberrations	Pre-filtered image	LensArray	Pinhole Mask
● Oblique Astigmatism:			
● Vertical Astigmatism:			
● Vertical Trefoil:			
● Vertical Coma:			
● Horizontal Coma:			



- Oblique Trefoil:



## REFERENCE

- [1]. Jim Schwidgerling. *Optical Specification, Fabrication, and Testing*. October 2014.  
2.3.4 Zernike polynomials.
- [2]. Fu-Chung Huang. *A Computational Light Field Display for Correcting Visual Aberrations*. Chapter 8. Light Field Wavefront Aberrometry. December,2013.
- [3]. Pamplona V F, Oliveira M M, Aliaga D G, et al. Tailored displays to compensate for visual aberrations[J]. 2012.