Correspondence Analysis Between The Image Formation Pipelines of Graphics and Vision

Li Ming†
† Max-Planck-Institute for Computer Science,
Stuhlsatzenhausweg 85, 66123, Saarbrücken, Germany

Abstract

Augmented Reality can merge the computer-generated objects into the real image or video sequences, one key issue is to keep the geometric consistency between the real and virtual objects. This is achieved by using the camera parameters recovered from the real image to render the virtual objects.

Camera parameters' recovery is a basic problem in the area of computer vision. On the other hand, rendering is the major concern of computer graphics. Ususally in these two areas different image formation pipelines are used. This paper presents a side-by-side comparison between the pipelines, in order to make it easier to setup the recovered camera parameters to do the rendering for AR applications. Some experiment results are shown to verify the correctness of the analysis.

Keywords:Augmented Reality, Computer Vision, Camera Calibration, OpenGL.

1 Introduction

Augmented Reality is a promising technology that can mix the real and virtual world. In order to obtain a naturally-appearing combination, one must keep the consistency between the real and virtual objects. Usually this consistency can be classified into two kinds: geometric and photometric.

What we concern here is keeping the geometric consistency. This is achieved by using recovered camera parameters from real image to render the virtual objects. However, due to different image formation pipelines used in computer graphics and computer vision, some inconsistency of the camera parameters often leads to confusion. Our work mainly focuses on clarifying such confusion by comparing the image formation pipelines and presenting their relationship explicitly. We will discuss several variations of transform used in computer vision. For computer graphics, we choose OpenGL to do our analysis because it is a popular software interface for rendering.

The remainder of the paper is organized as follows. Some related works are reviewed in Section 2. Section 3 and 4 will discuss the image formation pipelines in computer vision and OpenGL respectively. Then the correspondence of the transform parameters between them will be analyzed in Section 5. We also show how to use recovered camera parameters to setup the transform in OpenGL properly. Finally some experiment results are demonstrated and a conclusion will be drawn.

2 Related Work

Camera calibration recovers camera parameters from 2D images. These parameters determine the process of image formation and they describe the internal camera characteristics(intrinsic parameters) and the 3D position and orientation of the camera frame(extrinsic parameters). Calibration has been extensively investigated in computer vision and numerous algorithms have been proposed[1, 2, 3]. However, various coordinate system setup and camera parameters are often used in them.

OpenGL is a cross-platform standard for 3D rendering. Unlike many different camera models in the computer vision society, it has a well-defined specification[4], in which the geometric transformation pipeline is precisely described.

To our knowledge, there are not much works on the comparison of the image formation pipelines used in graphics and vision. Lengyel[5] discussed the convergence of them in a methodological manner. Different approaches are classified and fitted in the spectrum of graphics and vision techniques. Debevec *et al.* [6] pointed out that different transforms are used in the two fields. They also presented the formulae for them. But no more in-depth analysis was given.

3 Image Formation Pipeline in Computer Vision

3.1 Overview of the transform

Usually four coordinate systems are invovled in the image formation process of a pinhole camera. They are world coordinate system $O_w x_w y_w z_w$, eye(camera) coordinate system $O_e x_e y_e z_e$, image coordinate system $O_i xy$, pixel coordinate system $O_p uv$. Fig. 1 illustrates how a point in the world frame is transformed through several stages.

Figure 1: Transform Stages in Computer Vision

In Fig. 1, M_e is the matrix built from extrinsic camera parameters. M_{i1}, M_{i2} are matrices related to intrinsic parameters. The transform pipeline can be formulated into the following equation.

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = M_{i1}M_{i2}M_e \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = \begin{pmatrix} 1/dx & 0 & u_c \\ 0 & 1/dy & v_c \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} & & & t_0 \\ & \mathbf{R} & & t_1 \\ & & & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \quad (1)$$

s is the scale factor of the homogeneous pixel coordinate. Rotation matrix R is determined by Euler angles[7] α, β, γ. The extrinsic camera parameters include $t_0, t_1, t_2, \alpha, \beta, \gamma$, and the intrinsic parameters include f:effective focal length, (u_c, v_c):principal point of image, (dx, dy):pixel's physical size.

3.2 Variations of transform equation

Different coordinate system configuration and camera parameters usually result in different transform equation. Equation 1 is obtained by assuming the camera viewing direction is negative Z axis and the coordinate system O_ixy, O_puv have same direction. Camera can also point to the positive Z axis. The reversal of camera's direction only changes -1 in the matrix M_{i2} into 1 because the image plane changes from $z_e = -f$ to $z_e = f$. Sometimes the axes of the image coordinate and pixel coordinate system have opposing directions. In this case, we needn't change the transform matrix because dx and dy will have the proper sign to accommodate the reversal of axes.

It should be noted that the effective focal length f isn't the same as the one printed on the lens of a real camera. f is just a parameter in a pin-hole camera model, while the focal length on the real camera is a parameter for thick-lens camera model. Sometimes the effective focal length can be factored in matrix M_{i1}, and we will have another form of focal lengths $f_x = f/dx$, $f_y = f/dy$. Strictly speaking, the focal length now changes to 1. f_x, f_y just serve as two scale factors. This also suggests that the number of intrinsic parameters can be reduced from 5 to 4. They are f_x,f_y,u_c,v_c or $\eta(=\frac{dy}{dx})$,f_y,u_c,v_c.

So far, we have assumed an ideal pinhole camera without any distortion. But for more accurate camera model the radial distortion is often considered. Such distortion causes image points to shift radially to or from the image center.

Say the ideal image coordinate is $(X_i, Y_i, W_i)^T$, and the distorted one is $(X_d, Y_d, W_d)^T$. Then we can model the radial distortion as:

$$\begin{cases} X_d/W_d = X_i/W_i \cdot (1 + k \cdot r_d^2), & r_d^2 = (\frac{X_d}{W_d})^2 + (\frac{Y_d}{W_d})^2 \\ Y_d/W_d = Y_i/W_i \cdot (1 + k \cdot r_d^2) \end{cases} \quad (2)$$

Given X_i, Y_i, W_i and k(distortion term), The distorted radius r_d can be solved by Equation 2. If we denote $\lambda = 1 + k \cdot r_d^2$, Then we can include the term λ into the matrix M_{i1} and change $1/dx$,$1/dy$ to λ/dx,λ/dy respectively.

4 Image Formation Pipeline in OpenGL

OpenGL has a specific definition of perspective camera model. The camera points to the negative Z axis. The consecutive stages of the geometric transform are illustrated in Fig. 2:

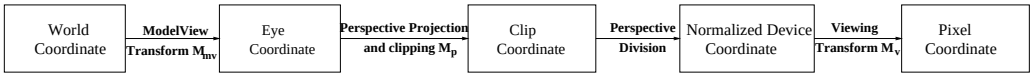

Figure 2: Transform Stages in OpenGL

The projection matrix can be constructed by *gluPerspective*. Equation 3 presents the perspective projection transform. $(X_c, Y_c, Z_c, W_c)^T$ is the clip coordinate. fovy, aspect, zFar, zNear are the four parameters of *gluPerspective*.

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ W_c \end{pmatrix} = \begin{pmatrix} \frac{\cot(fovy/2)}{aspect} & 0 & 0 & 0 \\ 0 & \cot(fovy/2) & 0 & 0 \\ 0 & 0 & \frac{zFar+zNear}{zNear-zFar} & \frac{2*zFar*zNear}{zNear-zFar} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} X_e \\ Y_e \\ Z_e \\ 1 \end{pmatrix} \tag{3}$$

The viewport transform is expressed by Equation 4. $(x_0, y_0, width, height)$ are the parameters provided in the glViewport function call.

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = M_v \begin{pmatrix} X_c \\ Y_c \\ W_c \end{pmatrix} = \begin{pmatrix} \frac{width}{2} & 0 & \frac{width}{2} + x_0 \\ 0 & \frac{height}{2} & \frac{height}{2} + y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ W_c \end{pmatrix} \tag{4}$$

Since the third row of projection matrix in Equation 3 is used to compute the depth and has nothing to do with the pixel's 2D position, we can simply discard it. Thus we obtain the concatenated transform from Equation 3 and Equation 4:

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{width}{2} & 0 & \frac{width}{2} + x_0 \\ 0 & \frac{height}{2} & \frac{height}{2} + y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\cot(fovy/2)}{aspect} & 0 & 0 & 0 \\ 0 & \cot(fovy/2) & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} M_{mv} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{5}$$

5 Correspondence Analysis and OpenGL setup

This section will give correspondence analysis between the parameters used in computer vision and OpenGL pipelines. And the OpenGL transform function calls using the recovered camera parameters will be listed.

Let's first compare the piplelines shown in Fig. 1 and Fig. 2. Obviously they have the same first step. But for step 2, perspective projection, computer vision uses one focal length f, while OpenGL uses different focal lengths in x and y direction($\frac{\cot(fovy/2)}{aspect}, \cot(fovy/2)$). Step 3 in Fig. 1 is a scale and bias transform, which corresponds to step 4 in Fig. 2, the viewing transform. And the last step in Fig. 1 is equivalent to step 3 in Fig. 2. They are both homogenization of coordinate.

Now it's the time to give the precise relations between the pipelines in Formulae. For computer vision, considering the distortion discussed in Section 3.2, the following equation can be derived from Equation 1:

$$\begin{cases} u = \lambda \cdot f/dx \cdot (-X_e/Z_e) + u_c \\ v = \lambda \cdot f/dy \cdot (-Y_e/Z_e) + v_c \end{cases} \tag{6}$$

For OpenGL, we can also get pixel coordinate from Equation 5:

$$\begin{cases} u = \frac{\cot(fovy/2)}{aspect} \cdot \frac{-X_e}{Z_e} \cdot (width/2) + (width/2) + x_0 \\ v = \cot(fovy/2) \cdot \frac{-Y_e}{Z_e} \cdot (height/2) + (height/2) + y_0 \end{cases} \tag{7}$$

Comparing Equation 6 and 7, we can give the correspondence between the transform parameters used in computer vision and OpenGL:

$$\begin{cases} u_c = width/2 + x_0 \\ v_c = height/2 + y_0 \\ f = (dy/\lambda) \cdot \cot(fovy/2) \cdot (height/2) \\ dx/dy = aspect \cdot (height/width) \end{cases} \tag{8}$$

Given an image with its dimension and the recovered camera parameters t_0,t_1,t_2,α, β,γ, $f, u_c, v_c, \frac{dx}{dy}, k$, the OpenGL functions should be called as the following:

- $glViewport(u_c - width/2, v_c - height/2, width, height)$;
- $gluPerspective(2 * atan(\frac{height}{2}/\frac{\lambda * f}{dy}))/M\_PI * 180, (\frac{width}{height}) * (\frac{dx}{dy}), 0.1, 200.0)$;

  The last two arguments only serve for clipping 3D objects in the depth direction. M_PI is the radian value of π.
- $glTranslatef(t_0, t_1, t_2)$;
  $glRotatef(\gamma/M\_PI * 180, 0, 0, 1)$;
  $glRotatef(\beta/M\_PI * 180, 0, 1, 0)$;
  $glRotatef(\alpha/M\_PI * 180, 1, 0, 0)$;

6 Experiment Results

Tsai's calibration algorithm[1] is one of the most widely used calibration algorithms. [8] provided the implementation of this algorithm. We modified the source code and adapted it to our analysis in Section 5. The camera parameters to be recovered are t_0, t_1, t_2,α, β, γ, f, u_c, v_c, k. $\frac{dx}{dy}$ is assumed to 1. Fig. 3 shows the image(608*401) for camera calibration. Input data are 3D coordinates of 81 coplanar points on the board and their correspondent 2D positions.

The calibration result is (t_0, t_1, t_2,α, β, γ, u_c, v_c, f, k) = (-167.079642, -60.533580, -393.260938, 26.508423, 19.010273, -24.158055, 308.428414, 201.420045, 7.762790, 1.226731e-03). These parameters are used to setup the OpenGL function as explained in the previous section. We overlay the rendered points on the original image and get the merged result image, shown in Fig. 4. The rendered points and real corners in the image coincide very well.

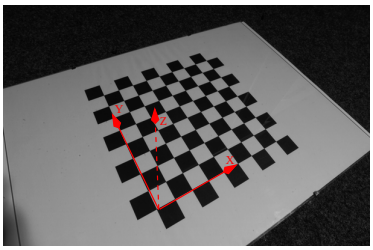

Figure 3: Checkerboard

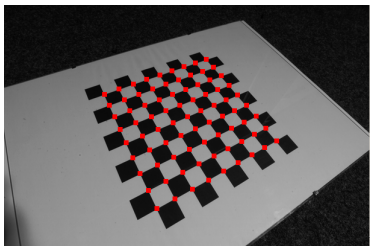

Figure 4: Reprojection of points

7 Conclusion

We have explained image formation pipelines used in computer vision and OpenGL and analyzed the correspondent relationship of the two pipelines. The main contribution of this paper is presenting such correspondence in an organized manner to allow better comparison of the geometric transform used in them. We also show how to setup such transform in OpenGL when lens distortion is taken into account.

For other camera models in vision or graphics, one can refer to this paper and perform his specific analysis. For example, the skewness [3] of two axes of the pixel coordinate system is left out in our discussion. But it can be added into the above analysis without difficulty.

Acknowlegement

Thanks to Michael Goesele for taking the calibration board picture in Fig. 3.

References

[1] R. Y. Tsai. An efficient and accurate camera calibration technique for 3-D machine vision. In *Proc. IEEE Computer Vision and Pattern Recognition*, pages 364–374, June 1986.

[2] Olivier Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint.* MIT Press, Cambridge, Massachusetts, 1993.

[3] Zhengyou Zhang. A flexible new technique for camera calibration. Technical Report 98-71, Microsoft Research, 1998.

[4] SGI. Opengl 1.2.1 specification. ftp://sgigate.sgi.com/pub/opengl/doc/opengl1.2/opengl1.2.1.pdf.

[5] Jed Lengyel. The convergence of graphics and vision. *IEEE Computer*, 31(7):46–53, July 1998.

[6] P. Debevec, S. Gortler, L. McMillan, R. Szeliski, and C. Bregler. Projective image warping. *Image-Based Modeling and Rendering, SIGGRAPH98 course 15*, 1998.

[7] Ken Shoemake. *Euler Angle Conversion.* Graphics Gems IV. Academic Press, 1994.

[8] Reg Willson. http://www.cs.cmu.edu/afs/cs.cmu.edu/user/rgw/www/TsaiCode.html.