

Viewing

Introduction to Computer Graphics

Arizona State University

Dianne Hansford

February 13, 2005

1 Introduction

The Viewing Pipeline involves several coordinate systems, namely

object \rightarrow world \rightarrow eye \rightarrow clip \rightarrow NDC \rightarrow window.

These coordinate transformations have been designed to optimize the algorithms applied to a primitive traveling down the pipeline. The discussion here focuses on the transformation from world to eye coordinates. This transformation is commonly described in terms of setting-up a camera.

2 Setting-up the Camera

Let's assume that we have placed the geometry that we would like to display in a world coordinate system. Soon we will want to define the type of projection (orthographic or perspective) and the specifics of the projection. Standard practice is to make the xy -plane the projection plane. Thus to facilitate defining the projection step, we move our geometry to sit "centered" on the $-z$ -axis, and this is called eye coordinates. Just how we move and orient our geometry into eye coordinates is conveniently defined by the parameters analogous to setting up a camera in the world coordinate system.

Specifically, setting-up a camera involves choosing

- its location, \mathbf{e} (called the *eye point*),

- its line of sight, defined by the line through the eye and a point \mathbf{a} (called the *at point*), and
- its orientation, \mathbf{u} (called the *up vector*).

These parameters are illustrated in Figure 1. There are a few restrictions on these camera parameters. The eye and at point must not be identical. The up vector determines how our camera is rotated. For instance, will the final image appear horizontally or vertically? The up vector does not need to be orthogonal to the line of sight, $\mathbf{a} - \mathbf{e}$. Obviously the up vector must not be the zero vector. See Section 3 for a closer examination of the up vector.

Now we have the means for describing the orientation of our geometry on the $-z$ -axis. The viewing operation will take

- the eye point to the origin,
- the at point onto the $-z$ -axis, and
- the up vector will lie in the $+yz$ -plane.

This is illustrated in Figure 2

In OpenGL, defining the camera and moving the geometry to eye coordinates is as simple as calling

$$\text{gluLookAt}(\mathbf{e}, \mathbf{a}, \mathbf{u}).$$

3 How Does gluLookAt Work?

Suppose a vertex of our geometry in world coordinates is labeled \mathbf{p}_w , and the corresponding point in eye coordinates is labeled \mathbf{p}_e . Using our knowledge of the connection between linear maps and coordinate transformations, the gluLookAt transformation is very easy to construct!

To review: recall that a matrix $A = [\mathbf{a}_1 \ \mathbf{a}_2 \ \mathbf{a}_3]$ defines a linear map that maps a vector \mathbf{v} in the $[\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3]$ -coordinate system to a vector \mathbf{v}' in the $[\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3]$ -coordinate system,

$$\mathbf{v}' = A\mathbf{v}.$$

Our problem here is simply a coordinate transformation, so the goal is to stage it in this context.

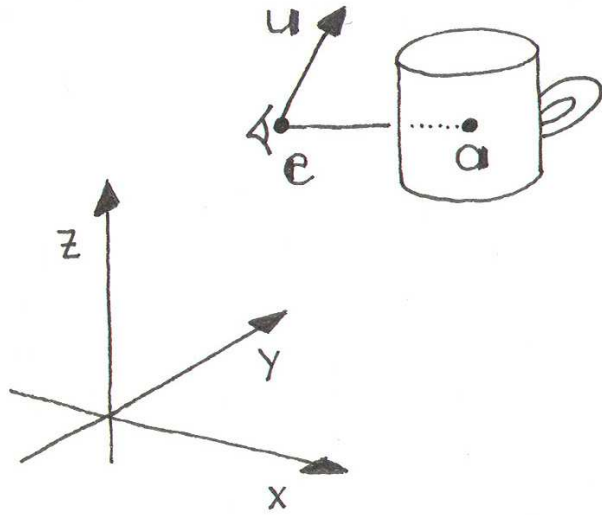


Figure 1: Setting-up the camera in world coordinates.

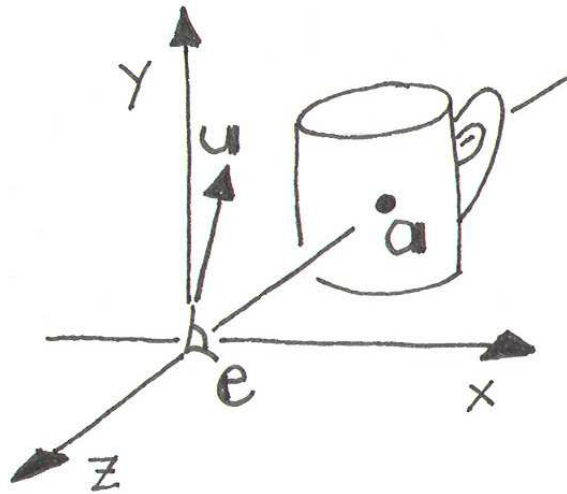


Figure 2: The camera's position in eye coordinates.

The first step is to form an orthonormal frame (a set of orthogonal unit vectors) from the camera parameters,

$$\mathbf{l} = \frac{\mathbf{a} - \mathbf{e}}{\|\mathbf{a} - \mathbf{e}\|}, \quad \mathbf{r} = \frac{\mathbf{l} \wedge \mathbf{u}}{\|\mathbf{l} \wedge \mathbf{u}\|}, \quad \mathbf{s} = \mathbf{r} \wedge \mathbf{l},$$

where \mathbf{l} is called the *line of sight vector*. Notice that \mathbf{r} is on our right as we look down the line of sight. And then \mathbf{s} will correspond to the up-direction. Now we see that all \mathbf{u} vectors that form the same plane with \mathbf{l} fall into two groups, and each group forms the “upside-down” image of the other. Thus the angle between \mathbf{u} and \mathbf{l} is inconsequential, however the side of \mathbf{l} in this plane is a determining factor in the resulting orientation.

In Section 2, the camera’s transformation from world to eye coordinates was outlined. In terms of this orthonormal frame, we want

$$\mathbf{l} \rightarrow -\mathbf{e}_3 \quad \mathbf{r} \rightarrow \mathbf{e}_1 \quad \mathbf{s} \rightarrow \mathbf{e}_2.$$

However, before we align our orthonormal frame with the coordinate axes, we need to translate the geometry by $-\mathbf{e}$, that is $\mathbf{p}_w - \mathbf{e}$.

Bringing in the geometric interpretation of a linear map, we can construct a matrix such that

$$\begin{aligned} [\mathbf{r} \ \mathbf{s} \ -\mathbf{l}] \mathbf{p}_e &= (\mathbf{p}_w - \mathbf{e}) \\ A\mathbf{p}_e &= (\mathbf{p}_w - \mathbf{e}). \end{aligned}$$

However, \mathbf{p}_e is unknown, so we need the matrix A^{-1} which forms

$$\mathbf{p}_e = A^{-1}(\mathbf{p}_w - \mathbf{e}).$$

Now we can take advantage of having formed an orthonormal frame: $A^{-1} = A^T$. Hence

$$\mathbf{p}_e = A^T(\mathbf{p}_w - \mathbf{e}) = \begin{bmatrix} \mathbf{r}^T \\ \mathbf{s}^T \\ -\mathbf{l}^T \end{bmatrix} (\mathbf{p}_w - \mathbf{e}), \quad (1)$$

and we have our transformation to eye coordinates.

4 A Numerical Example

This is a simple example to keep the calculations easy to follow. Suppose our modeling transformations place our object on the $-x$ -axis as illustrated

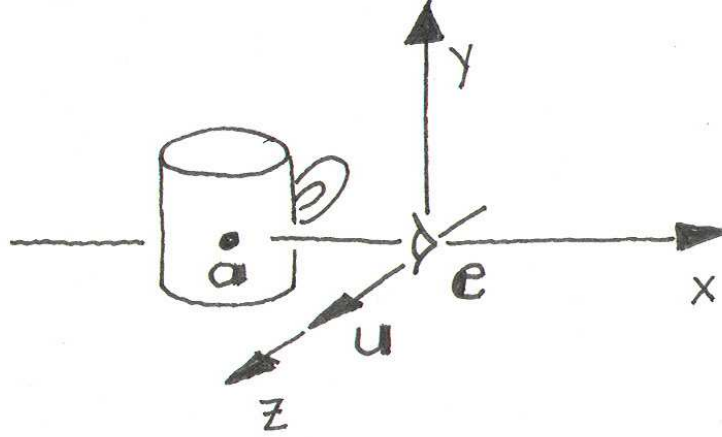


Figure 3: An example object and camera in world coordinates.

in Figure 3. Let's set up our camera with the following parameters

$$\mathbf{e} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} .$$

From this camera set-up, we form the following orthonormal frame

$$\mathbf{l} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{r} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{s} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} ,$$

which is also illustrated in Figure 3.

Since the camera is already located at the origin, a translation is not necessary, and

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} .$$

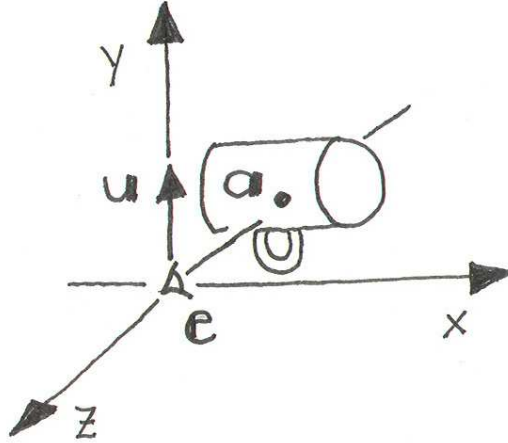


Figure 4: An example object and camera in eye coordinates.

Thus the eye coordinate points \mathbf{p}_e are calculated as

$$\mathbf{p}_e = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \mathbf{p}_w.$$

Let's check that our original goals, that the at point is mapped to the $-z$ -axis and the up vector is mapped to the $+yz$ -plane are satisfied:

$$A^T \mathbf{a} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \quad \text{and} \quad A^T \mathbf{u} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix},$$

which indeed does satisfy the requirements.

The final view of the object is illustrated in Figure 5.

5 OpenGL and the ModelView Matrix

In OpenGL, the transformation of vertices from world to eye coordinates is done via the ModelView matrix. The modeling operations, such as glRo-

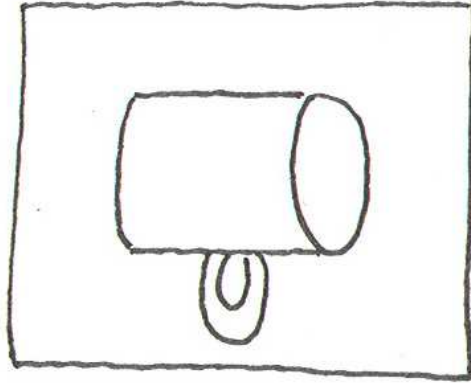


Figure 5: The final view resulting from the example camera parameters.

tate or `glTranslate`, are the tools for positioning our geometry in the world system. The viewing transformation tool, `gluLookAt`, simply utilizes the rotate and translate operations, so in that sense, modeling and viewing are interchangeable. In fact, consider achieving the displayed effect of an object rotating about its centroid — this could be achieved by either rotating the object or rotating the eye about the object. Thus modeling and viewing have been concatenated to form "ModelView".

The `gluLookAt` affine map (1) is stored in the 4×4 ModelView matrix. To do this, first we express our points in homogenous form,

$$\bar{\mathbf{p}}_w = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}.$$

The translation followed by the rotation in (1) is written as

$$\bar{\mathbf{p}}_e = \begin{array}{|ccc|c|} \hline & A^T & & \mathbf{o} \\ \hline 0 & 0 & 0 & 1 \\ \hline \end{array} \begin{array}{|ccc|c|} \hline & I & & -\mathbf{e} \\ \hline 0 & 0 & 0 & 1 \\ \hline \end{array} \bar{\mathbf{p}}_w.$$

Multiplying the translation and rotation matrices, we have

$$\bar{\mathbf{p}}_e = \begin{array}{|ccc|c|} \hline & A^T & & A^T(-\mathbf{e}) \\ \hline 0 & 0 & 0 & 1 \\ \hline \end{array} \bar{\mathbf{p}}_w$$

Keep in mind that we specify the transformation commands in our program in the reverse order than we want them applied. Let R be the matrix formed from a modeling command such as `glRotate`, and L be the matrix formed from `gluLookAt`. Normally we apply our modeling transformations and then set up the camera, so for a vertex \mathbf{v} we want $LR\mathbf{v}$. In OpenGL this would require our code to take the form

```

:
gluLookAt
glRotate
draw v
:

```

6 Getting Started: Creating the Eye, At, and Up Parameters

Often times we input a geometric object from a file, and without knowledge of its exact shape or orientation, we need to determine suitable parameters for `gluLookAt`. One suggestion is as follows.

Upon reading the vertices, record the minimum and maximum values in x , y , and z . This is called the *minmax box*. Thus, the minmax box contains the object and it is aligned with the coordinate axes. Let the lower-left-most corner and upper-right-most corner be

$$\mathbf{m}_0 = \begin{bmatrix} m_{0,0} \\ m_{1,0} \\ m_{2,0} \end{bmatrix} \quad \text{and} \quad \mathbf{m}_1 = \begin{bmatrix} m_{0,1} \\ m_{1,1} \\ m_{2,1} \end{bmatrix},$$

respectively.

A good choice for the at point \mathbf{a} is the centroid of the minmax box,

$$\mathbf{a} = \frac{1}{2}(\mathbf{m}_0 + \mathbf{m}_1).$$

A good choice for the up vector is one of the coordinate axes, for instance the z -axis,

$$\mathbf{u} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

And finally, the eye point could be chosen some distance away from a face of the minmax box, but not so that the up vector is parallel to the line of sight. For instance, we could view the object from in front of an xy -face of the minmax box:

$$\mathbf{e} = \mathbf{a} - 2.0 \times \left(\begin{bmatrix} m_{0,0} \\ m_{1,1} \\ m_{2,0} \end{bmatrix} - \begin{bmatrix} m_{0,0} \\ m_{1,0} \\ m_{2,0} \end{bmatrix} \right),$$

which steps back from \mathbf{a} , twice the y -dimension of the minmax box.

7 Camera Handling

In Section 6, a very simple solution for specifying the camera's set-up was presented. For animated graphics, such as games, the camera set-up is much more involved, and this is commonly known as *camera handling*.

Considering the gaming application, some of the concerns include the following.

- Should the gamer's avatar always be in view?
- Are important elements of the game, such as enemies or cliffs, in full view?
- Is the camera in a physically impossible location, such as behind a wall?
- Should the gamer have control of the camera?

Key: There is no one, correct solution. Gaming companies are constantly experimenting with different camera handling schemes to achieve a novel look for their game. For instance, "FPS" (First Person Shooter) was novel when it came out because it was the first use of the camera at the eyes of the avatar, that is, first person I. You cannot actually see your avatar; perhaps only your arms or firearm. This approach gained widespread acceptance, as

it was used in games such as Doom, Quake, and Unreal Tournament. This immersive viewpoint allows gamers to identify closely with an avatar.

Another popular camera handling approach is “third person”, where the camera would be just behind your head. Now you can see a bit more of your avatar, and peripheral vision is simulated. Tomb Raider implements this approach.