A Practical Viewing System

We will now discuss the design of a viewing system. In particular, we are going to be implementing a *pinhole camera*. The name is a reference to the way cameras originally worked. The principle was to have a box whose insides were painted completely black with the exception of one face, which was coated in a photo-sensitive compound. Nowadays this is known as the *film*. On the opposing face there was a single pinhole through which light could come through. The light would then etch the film and the image would be formed. It is worth noting that in the actual pinhole camera, the image would be inverted, while in our implementation it won't be. We will therefore call this type of system a *virtual camera*.

The Interface

Our previous viewing system was axis aligned. We will now remove that restriction in order to allow the camera to be placed anywhere in the scene. To accomplish this, we will require the following data:

- an arbitrary eye point,
- an arbitrary view direction,
- an arbitrary orientation about the view direction, and
- a distance between the eye and the view plane.

We will represent this information with the following points and vectors:

- The eye point $\mathbf{e}$,
- The look-at point $\mathbf{l}$,
- the up vector $\mathbf{up}$, and
- the view-plane distance d.

The eye point was already defined, so we will leave it as is. The look-at point gives us a way of defining which way our camera is looking. The up vector is used to determine the orientation of the camera, while d is used to determine the field of view.

Before we move on, we will define the space in which these vectors are given. The vectors that conform the camera system, and in fact the positions of all objects in our scene are specified in *world space*. The reason why we are making this distinction is that the camera system will have its own set of coordinates, which will be called the *viewing coordinates* or *view space*.

Viewing Coordinates.

In order to define a new coordinate space, we first need to construct the axes that form it. Since we are in 3D space, we will require 3 coordinate axes. These will be defined as follows:

- $\mathbf{w} = (\mathbf{e} - \mathbf{l})/|\mathbf{e} - \mathbf{l}|$

- $\mathbf{u} = (\mathbf{up} \times \mathbf{w})/|\mathbf{up} \times \mathbf{w}|$
- $\mathbf{v} = \mathbf{w} \times \mathbf{u}$

The first vector w defines the view direction. In particular, this vector is defined to be in the opposite direction of the view. We will discuss why later. The next vector is constructed by taking the cross-product of $\mathbf{w}$ with the up vector. We will take the up vector to be (canonically) the vector $(0, 1, 0)$. This means that the resulting vector $\mathbf{u}$ is parallel to the x_w, z_w plane, where x_w, z_w are the world x, z coordinate axes. The final vector is obtained by the cross-product of $\mathbf{w}, \mathbf{u}$. The main reason why we require this vector is because there is no reason to expect the up vector to be perpendicular to $\mathbf{u}, \mathbf{w}$. Since we need 3 axes that are all perpendicular to each other, we must therefore construct the final axis in this way. The final step is to define the origin of our coordinate space, which we will set to be the eye point. What we have constructed is called an *orthonormal basis*.

Due to the way we have constructed the axes, it so happens that $\mathbf{v}$ and $\mathbf{u}$ will line up with the columns and rows of the viewing plane. This makes sense, as it matches what we would see if we were looking down along the view vector. This means that we can tilt the view direction in any way we want and the orientation of the viewing plane remains consistent with the way we view the world and take photographs. In addition, these new axes allow us to easily manipulate the camera by specifying the rotation in relation to the existing coordinate axes.

Now the reason why $\mathbf{w}$ is defined to be in the opposite direction is the following: if we examine the view plane, we would expect that positive $\mathbf{u}$ would be to the right and positive $\mathbf{v}$ is up (lining up with our previous axis-aligned representation). This is convenient for when we compute the direction of the rays, since we maintain a consistent orientation. In order to allow this, we require $\mathbf{w}$ to point away from the look at point. Another consequence of this is that the resulting coordinate system is right-handed (like the world space).

Ray Calculation

Given that we have our new coordinate space, what we now need is an expression for the direction of the rays in world coordinates. First, the origin will be set to the eye. Since the coordinates for the eye are specified in world space, then we are done. All that is left is to specify the direction vectors. The (x_v, y_v) coordinates of a sample point p on the pixel in row r and column c are

$$x_v = c - w/2 + p_x$$
$$y_v = r - h/2 + p_y$$

Given that we have our orthonormal basis, we can express any vector as a linear combination of its basis vectors. We can therefore compute the direction of our rays as follows:

$$\mathbf{d} = x_v\mathbf{u} + y_v\mathbf{v} - d\mathbf{w}$$

Where $x_v, y_v, -d$ are the components in the $\mathbf{u}, \mathbf{v}, \mathbf{w}$ directions, respectively. Since the vectors were constructed in world coordinates, then the resulting vector $\mathbf{d}$ is itself in world coordinates. The final step is then to normalize $\mathbf{d}$.

There are two last things to mention with this camera system:

- We can easily implement a zoom level by simply changing the value of d.
- The camera will have a singularity. In particular, notice that the construction of $\mathbf{u}$ depends on the up vector. The singularity occurs whenever the view direction is parallel with the up vector. In our case, since we are defining the up vector to be $(0, 1, 0)$, then this will happen when the camera is looking straight up or down. Whenever this happens, what we can do is to simply change the basis vectors to align with the canonical axes, where $\mathbf{u} = (0, 0, 1)$, $\mathbf{v} = (1, 0, 0)$, and $\mathbf{w} = (0, 1, 0)$. This point is important to keep in mind, as it will appear later on once we shift to real-time graphics.