$$p_w = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} = w_x \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + w_y \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + w_z \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= w_x e_x + w_y e_y + w_z e_z$$

Monash University • Clayton's School of Information Technology

CSE3313 Computer Graphics

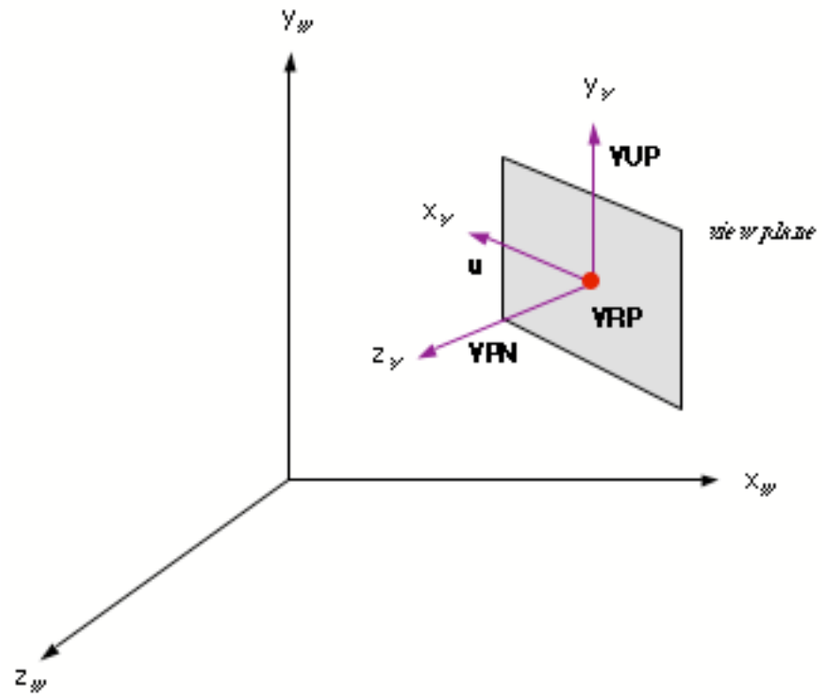
Lecture 19: Viewing in 3D

Specifying a View Plane

- A view plane is established by defining a **viewing coordinate system.**
- The user defines the origin for the viewing coordinate system by picking a world coordinate position as the **view reference point.**
- The orientation of the view plane is defined by specifying the **view plane normal** vector. This vector establishes the direction for the positive z axis of the viewing coordinate system.
- The view plane normal $\mathbf{N}$, can be specified via a world coordinate system position.
- The direction of $\mathbf{N}$ is the direction of a line from the origin to that world coordinate position.
- A vertical vector $\mathbf{V}$, called the view up vector defines the direction of the positive y axis. $\mathbf{V}$ can be specified in a similar manner to $\mathbf{N}$.

Specifying the View Plane (cont.)

- Some systems allow **V** to be not perpendicular to **N** and use the component of the specified vector which is perpendicular to **N** as the View up vector. (the vector is resolved in the direction orthogonal to **N**).
- We can think of the view plane as a logical device upon which the image is to be displayed.



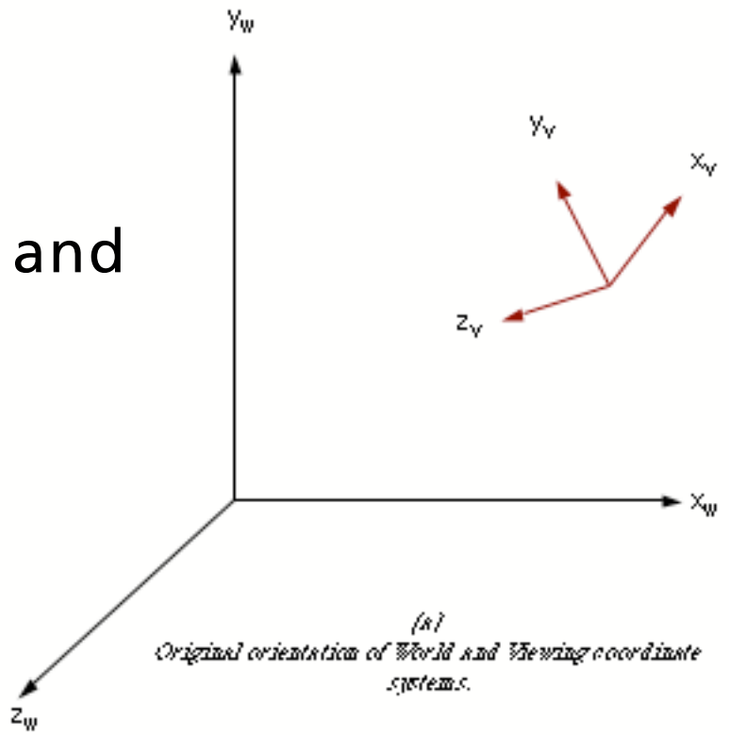
Specifying the View Plane (cont.)

- The viewing coordinate system can be either left-handed or right-handed.
  - Left-handed system, increasing z coordinate means objects are further away from the viewer;
  - Right-handed system is consistent with right-handed master/ world coordinates.
- In this discussion, a left-handed coordinate system is used.
- In establishing the view plane, some systems use an extra parameter called the **view distance**.
- The view plane is defined as the plane parallel to the viewing coordinate x–y plane that is a specified distance from the view reference point.

View Coordinate Transformation

- As part of the viewing process, points defined in world coordinates must be converted to viewing coordinates.
- This transformation can be accomplished conceptually by a sequence of translations and rotations that map the viewing system axes onto the world coordinate axes.
- The matrix representing this transformation can be obtained by concatenating the following transformations:

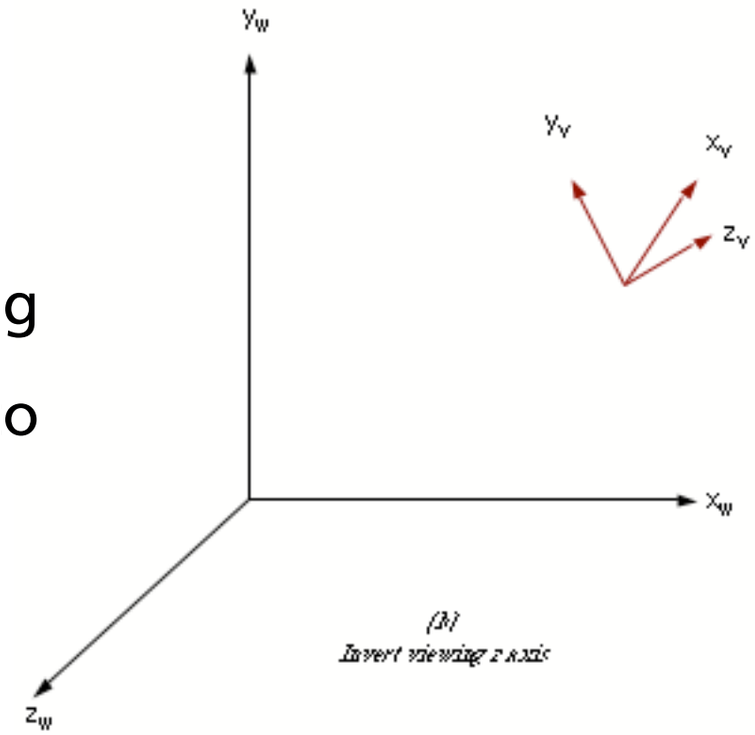
Right-handed world coordinate system and left-handed viewing coordinate system.



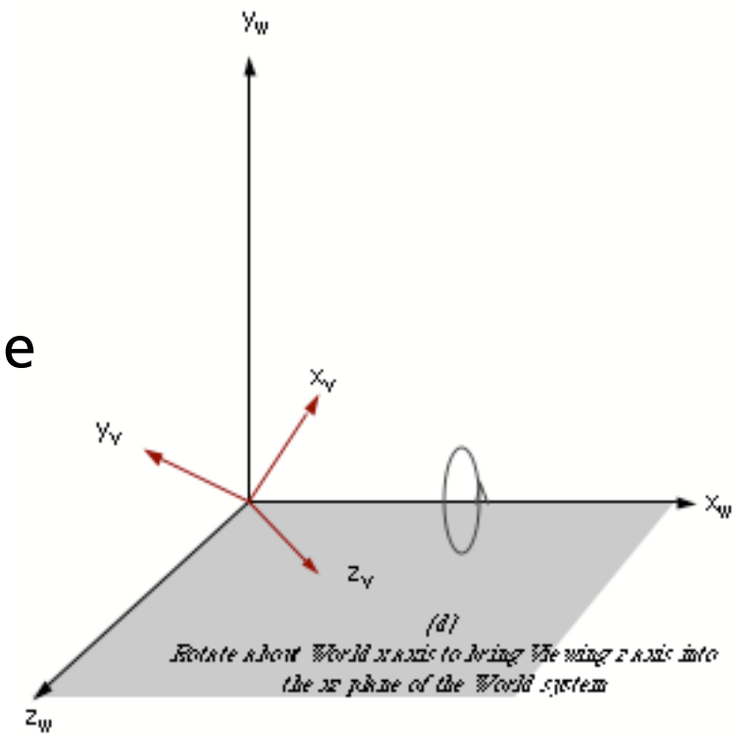
(a) Original orientation of World and Viewing coordinate systems.

View Coordinate Transformation (cont.)

(1) Reflect relative to the x–y plane, reversing the sign of each z coordinate. This changes to a right-handed coordinate system.

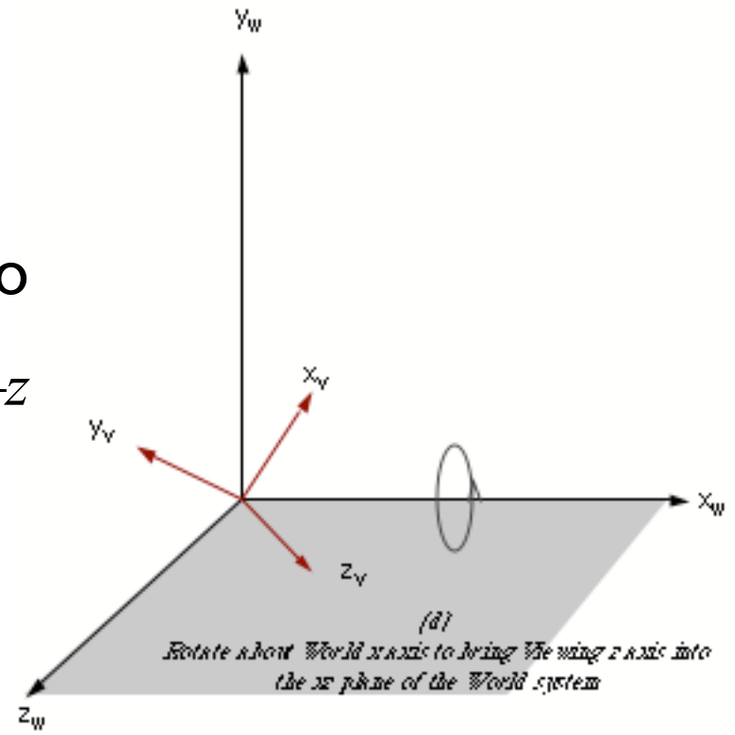


(2) Translate the view reference point to the origin of the world coordinate system.

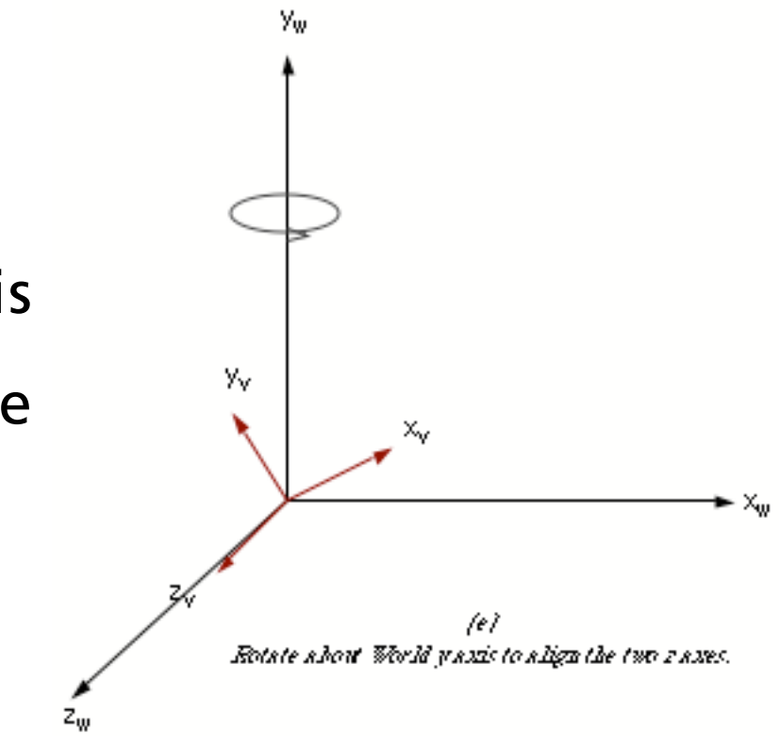


View Coordinate Transform (cont.)

(3) Rotate about the world coordinate x axis to bring the viewing coordinate z axis into the x–z plane of the world coordinate system.

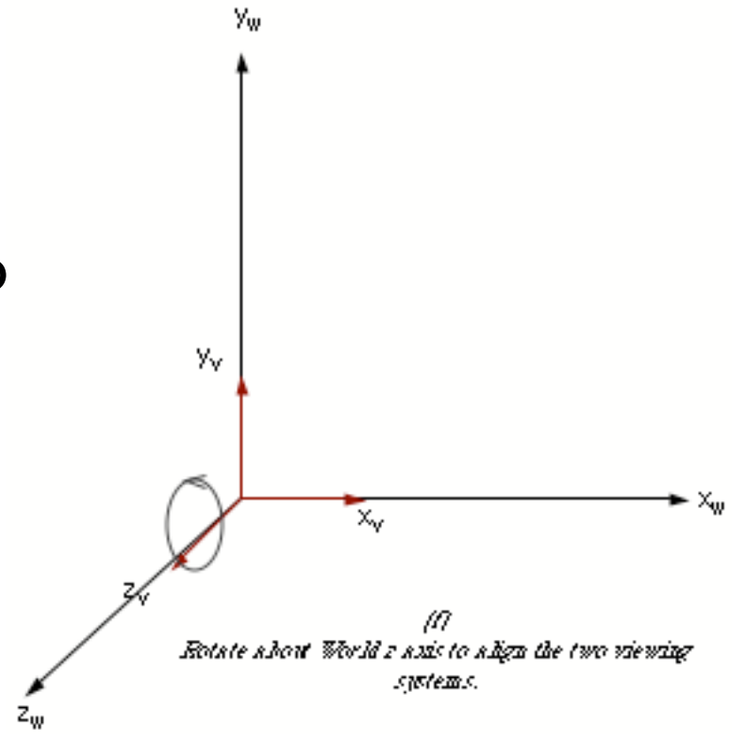


(4) Rotate about the world coordinate y axis until the z axes of both coordinate systems are aligned.



View Coordinate Transform (cont.)

(5) Rotate about the world coordinate z axis to align the viewing and world axes.



- When applied to world coordinate definitions of objects in the scene, this sequence of transformations converts them to their positions within the viewing coordinate system.
- This sequence has much in common with the transformation sequence that rotates an object about an arbitrary axis.

Specifying the View Plane

- It turns out that there is a more efficient method for performing the rotation component of the transformation rather than multiplying 3 rotation transformation matrices together. We can use the orthogonal properties of the axes and fact that the inverse of a rotation matrix is its transpose.
- Given a view plane normal , **n** and a view up vector, **v**, we require that **v** is orthogonal to **n**. If the **v** specified by the user is not orthogonal to **n** we can resolve the component of **v** orthogonal to **n**. i.e.:

$$v \leftarrow v - \frac{v \cdot n}{n \cdot n} n$$

Where **v.n** is the vector dot product: $v \cdot n = v_x n_x + v_y n_y + v_z n_z$

Specifying the View Plane (cont.)

- We need to derive a third vector which is orthogonal to both **n** and **v**. This can be calculated from:

$$u = v \times n = \begin{bmatrix} v_y n_z - v_z n_y \\ v_z n_x - v_x n_z \\ v_x n_y - v_y n_x \end{bmatrix}$$

E.g.:

$$v = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad n = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad v \times n = \begin{bmatrix} 0 \cdot 0 - 0 \cdot 1 \\ 0 \cdot 0 - 1 \cdot 0 \\ 1 \cdot 1 - 0 \cdot 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

In this case, **v**, **u** and **n** will generate a right-handed coordinate system.

Suppose:

$$p_w = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} = w_x \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + w_y \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + w_z \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$
$$= w_x e_x + w_y e_y + w_z e_z$$

then $\mathbf{e}_x$, $\mathbf{e}_y$ and $\mathbf{e}_z$ are all unit vectors and are all at right angles to each other.

Transforming the View Reference Point

- In the viewing coordinate system we normalise the new coordinate axes to be unit vectors.

$$v \leftarrow \frac{v}{|v|}, \quad |v| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

The viewing transformation can be conveniently specified using this information.

$$p_w = vrp_w + p_v \qquad (1)$$

where $\mathbf{p}_w$ is a point's position in the world coordinate system;

$\mathbf{vrp}_w$ is the view reference point, given in world coordinates;

$\mathbf{p}_v$ is the point's position relative to the viewing coordinate system axis, i.e. its view system coordinates.

Transforming the View Reference Point (cont.)

- The first stage of converting from world coordinates to viewing coordinates is a translation from the view reference point to the origin.
- Suppose **u′**, **v′** and **n′** are the unit vectors for the coordinate axes in the viewing coordinate system.

$$\mathbf{u}'_v = \begin{bmatrix}1\\0\end{bmatrix}_v = \begin{bmatrix}u'_x\\u'_y\\u'_z\end{bmatrix}_w, \quad \mathbf{v}'_v = \begin{bmatrix}0\\1\end{bmatrix}_v = \begin{bmatrix}v'_x\\v'_y\\v'_z\end{bmatrix}_w, \quad \mathbf{n}'_v = \begin{bmatrix}0\\0\\1\end{bmatrix}_v = \begin{bmatrix}n'_x\\n'_y\\n'_z\end{bmatrix}_w$$

where the subscript v denotes coordinates with respect to the viewing coordinate system, and the subscript w denotes coordinates with respect to the world coordinate system.

The world coordinate representation of $\mathbf{u}'$ is $\begin{bmatrix}u'_x\\u'_y\\u'_z\end{bmatrix}$

View Coordinate Rotations using Basis Vectors

$$C = \begin{bmatrix} c_{x,u} & c_{x,v} & c_{x,n} \\ c_{y,u} & c_{y,v} & c_{y,n} \\ c_{z,u} & c_{z,v} & c_{z,n} \end{bmatrix}$$

- is a matrix that converts coordinates using **basis vectors**, which converts coordinates expressed in terms of $\mathbf{u}'$, $\mathbf{v}'$ and $\mathbf{n}'$ to coordinates expressed in terms of $\mathbf{e}_x$, $\mathbf{e}_y$ and $\mathbf{e}_z$.

$$Cu'_v = \begin{bmatrix} c_{x,u} & c_{x,v} & c_{x,n} \\ c_{y,u} & c_{y,v} & c_{y,n} \\ c_{z,u} & c_{z,v} & c_{z,n} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_{x,u} \\ c_{y,u} \\ c_{z,u} \end{bmatrix}, \quad Cv'_v = \begin{bmatrix} c_{x,v} \\ c_{y,v} \\ c_{z,v} \end{bmatrix}, \quad Cn'_v = \begin{bmatrix} c_{x,n} \\ c_{y,n} \\ c_{z,n} \end{bmatrix}$$

The world coordinate representation of u' is $\begin{bmatrix} u'_x \\ u'_y \\ u'_z \end{bmatrix}$

so the columns of $\mathbf{C}$ are the world coordinates of the new axis system.

View Coordinate Rotations

$$C = \begin{bmatrix} u'_x & v'_x & n'_x \\ u'_y & v'_y & n'_y \\ u'_z & v'_z & n'_z \end{bmatrix}$$

- $\mathbf{C}$ is a matrix whose columns have unit length and are orthogonal to each other, thus $\mathbf{C}$ is an orthogonal matrix.
- $\mathbf{C}$ is orthogonal and square, thus $\mathbf{C}^{-1} = \mathbf{C}^{\mathbf{T}}$

  $\mathbf{C}^{-1}$ converts from coordinates expressed in terms of $\mathbf{e}_x$, $\mathbf{e}_y$ and $\mathbf{e}_z$ into coordinates expressed in terms of $\mathbf{u}'$, $\mathbf{v}'$ and $\mathbf{n}'$.
- The overall transformation of $\mathbf{p}_w$ into $\mathbf{p}_v$ is:
  1. A translation which brings the view reference point to the origin.
  2. $\mathbf{C}^{-1}$

Camera Control: Euler Angles

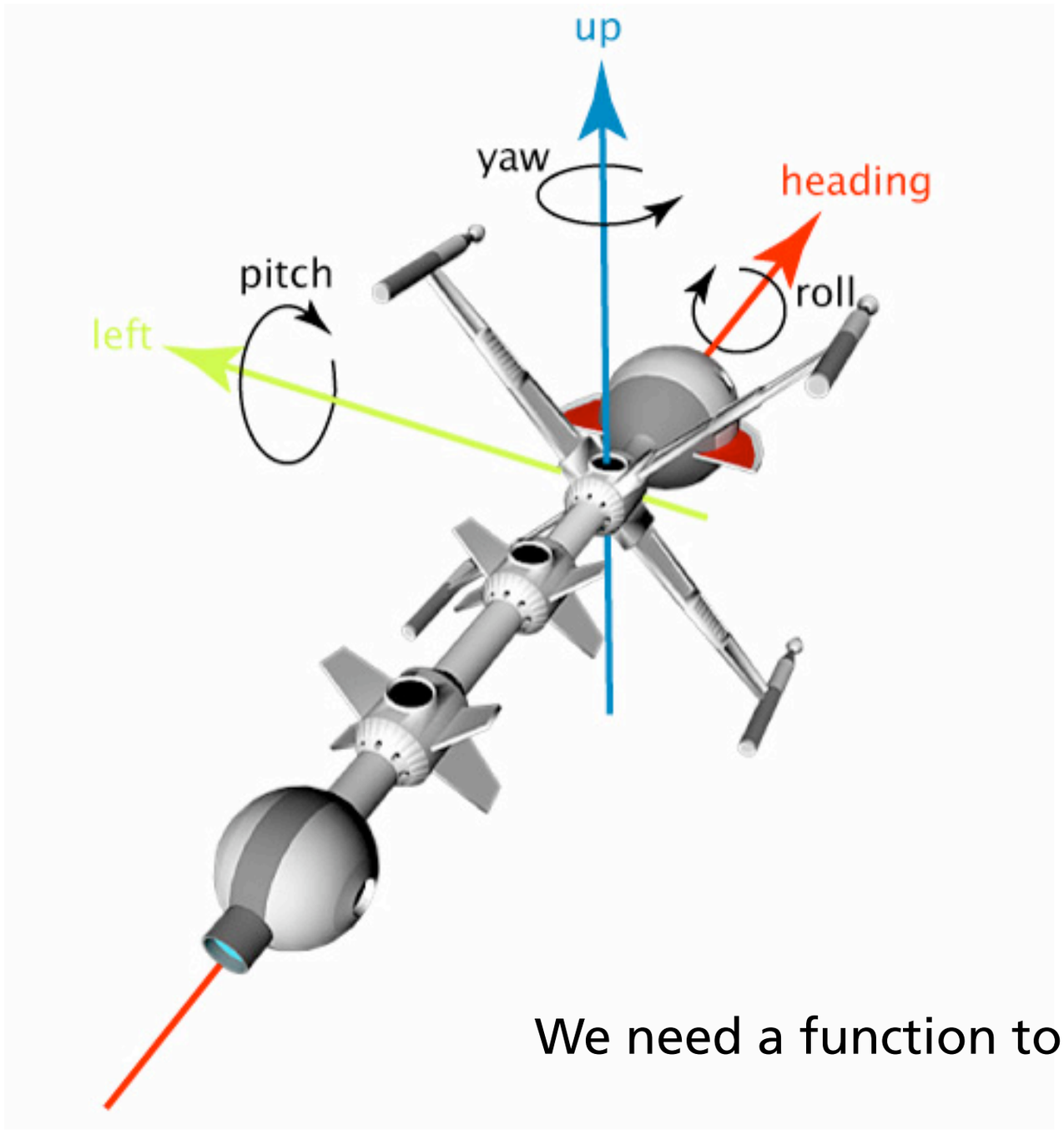
- Orientation in 3D space can be represented with three *Euler Angles: pitch, yaw* and *roll* (also: *elevation, azimuth* and *roll)* corresponding to rotations in the *y, z* and *x* axes respectively.

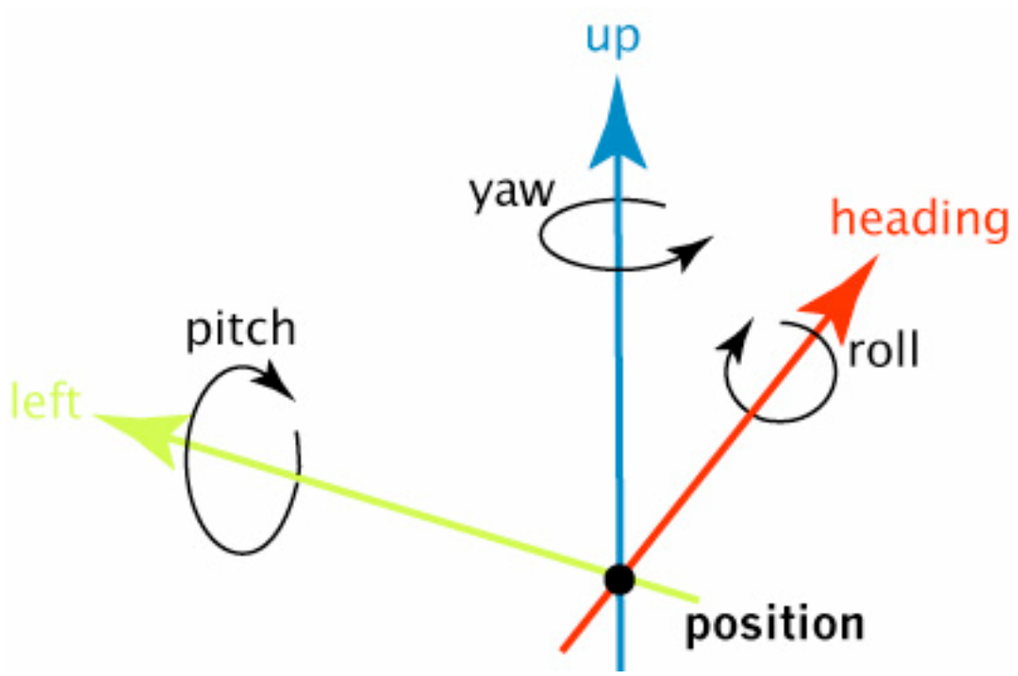
- To control a camera's movement (or indeed any object's movement) in 3D space we need both a **position** and **orientation**. This information forms a *coordinate reference frame.*

- Position can be specified as a 3D vector

- Orientation can be expressed in terms of three vectors representing the **H**eading, **L**eft and **U**p vectors of the camera's coordinate system

- Transform objects in world coordinates so that they are relative to the camera's coordinate reference frame requires the inverse transform to that of transforming from camera to world coordinates.





We need a function to convert Euler angles to **[H,L,U]**

Euler to HLU conversion function

```
void eulerToReference(EulerFrame * ef, ReferenceFrame * r)
{
    float cosPitch, cosYaw, cosRoll;
    float sinPitch, sinYaw, sinRoll;

    cosPitch = cos(ef->pitch);
    cosYaw = cos(ef->yaw);
    cosRoll = cos(ef->roll);
    sinPitch = sin(ef->pitch);
    sinYaw = sin(ef->yaw);
    sinRoll = sin(ef->roll);

    r->h[0] = sinYaw * cosPitch;
    r->h[1] = sinPitch;
    r->h[2] = cosPitch * -cosYaw;

    r->u[0] = -cosYaw * sinRoll - sinYaw * sinPitch * cosRoll;
    r->u[1] = cosPitch * cosRoll;
    r->u[2] = -sinYaw * sinRoll - sinPitch * cosRoll * -cosYaw;

    crossVect3(r->l, r->u, r->h); /* l = u x h */

    copyVect3(r->pos, ef->pos); /* copy the position vector */
}
```