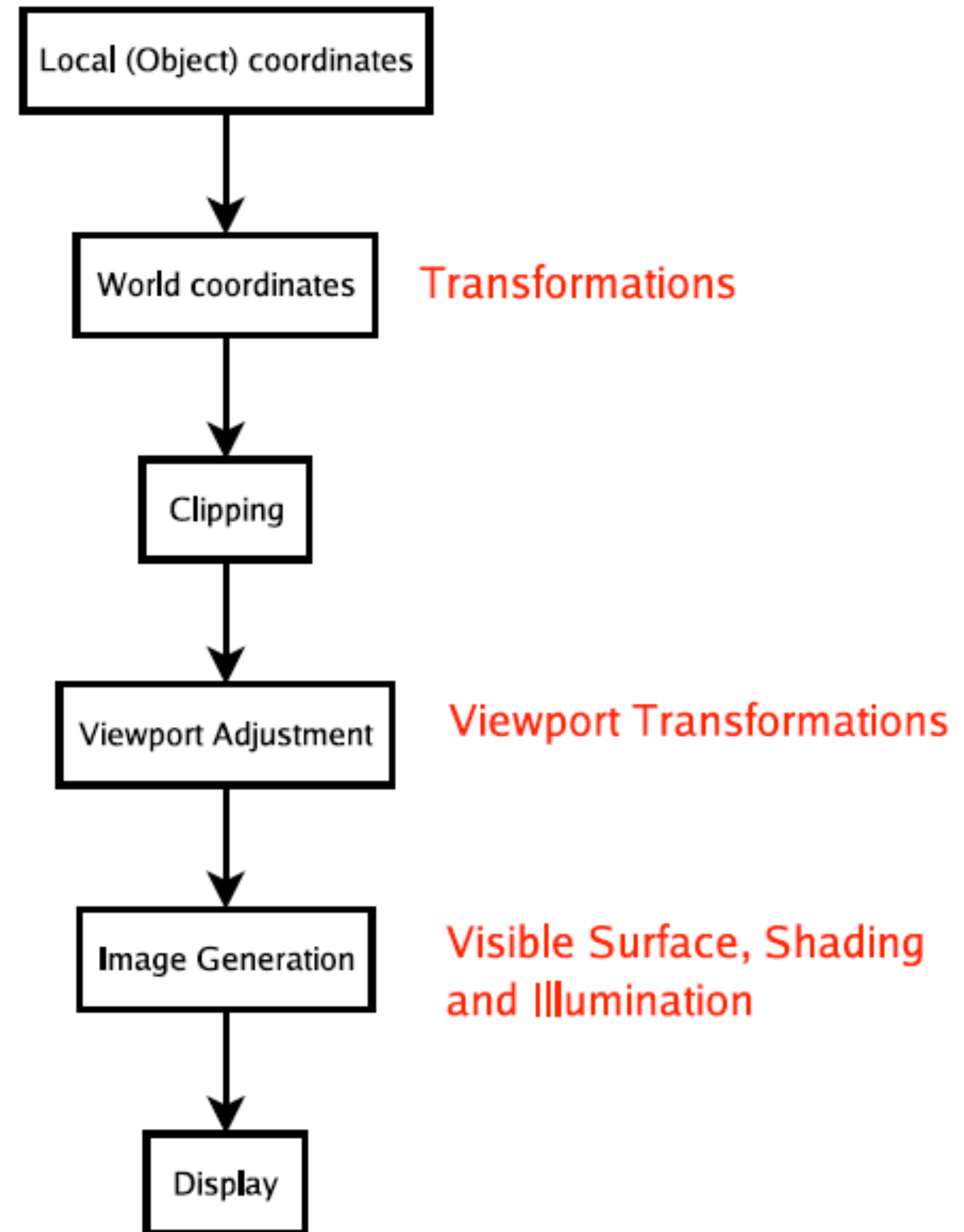


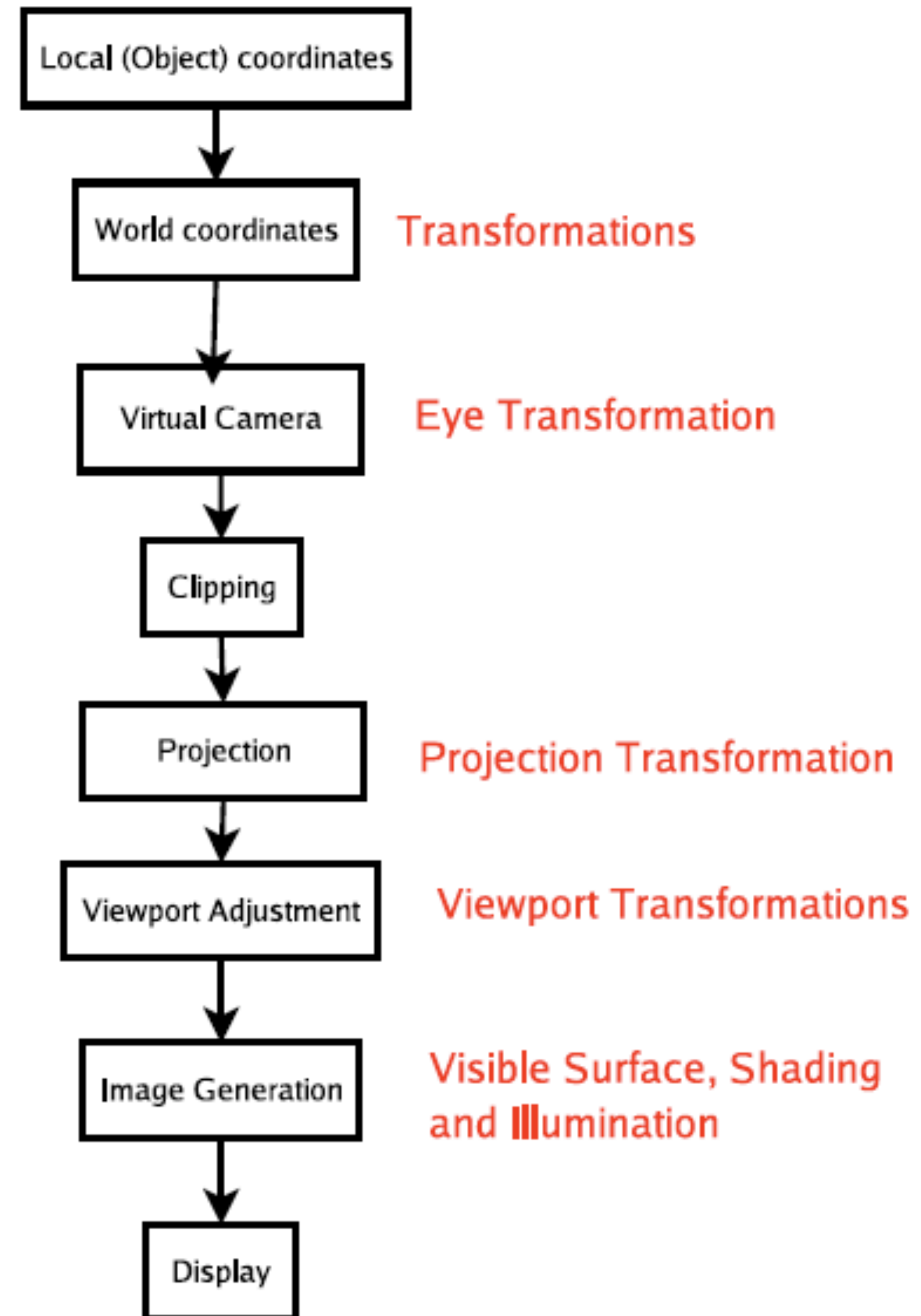
# Visualisation Pipeline :

## The Virtual Camera

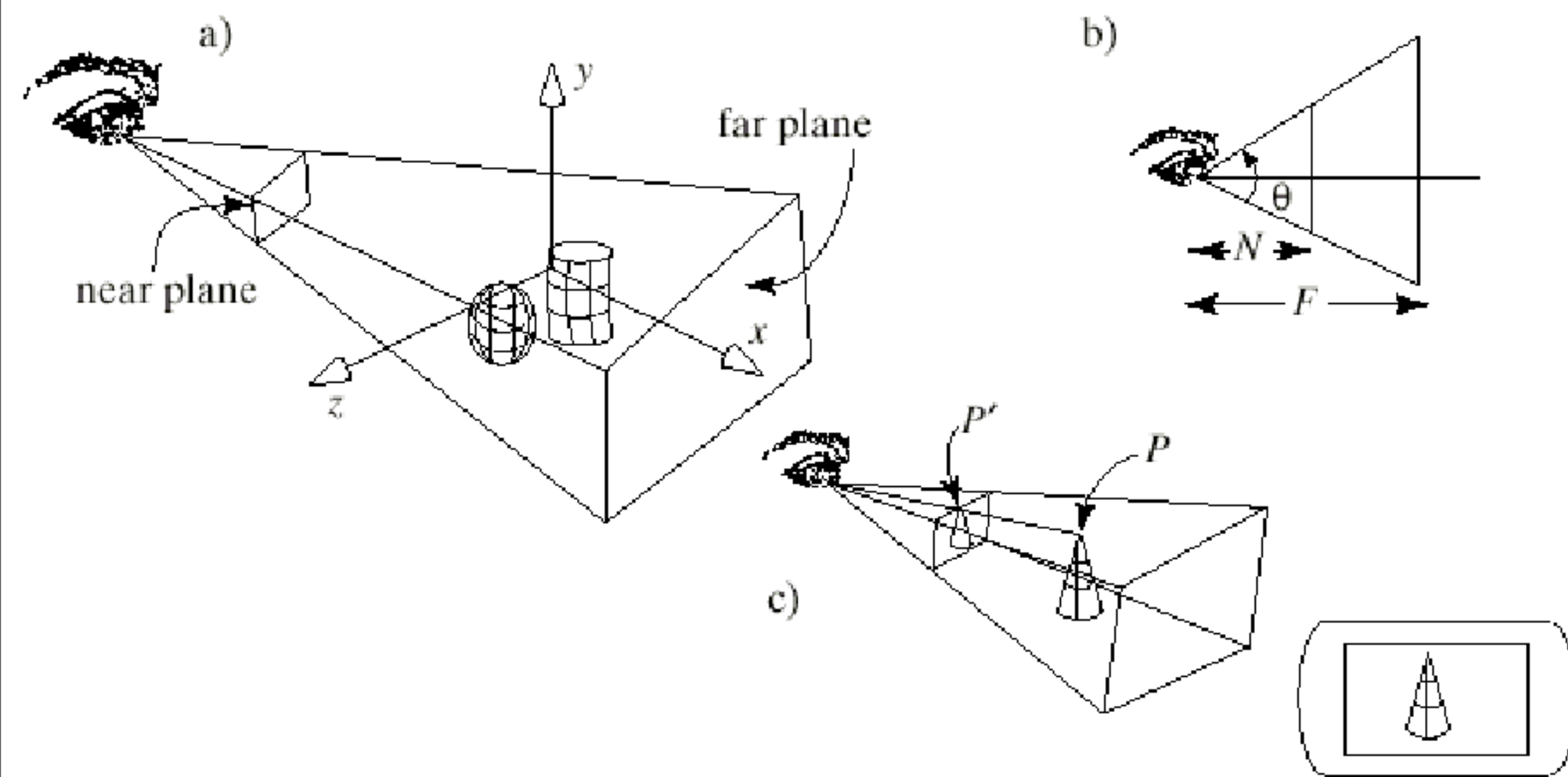
# The Graphics Pipeline



# 3D Pipeline



# The Virtual Camera

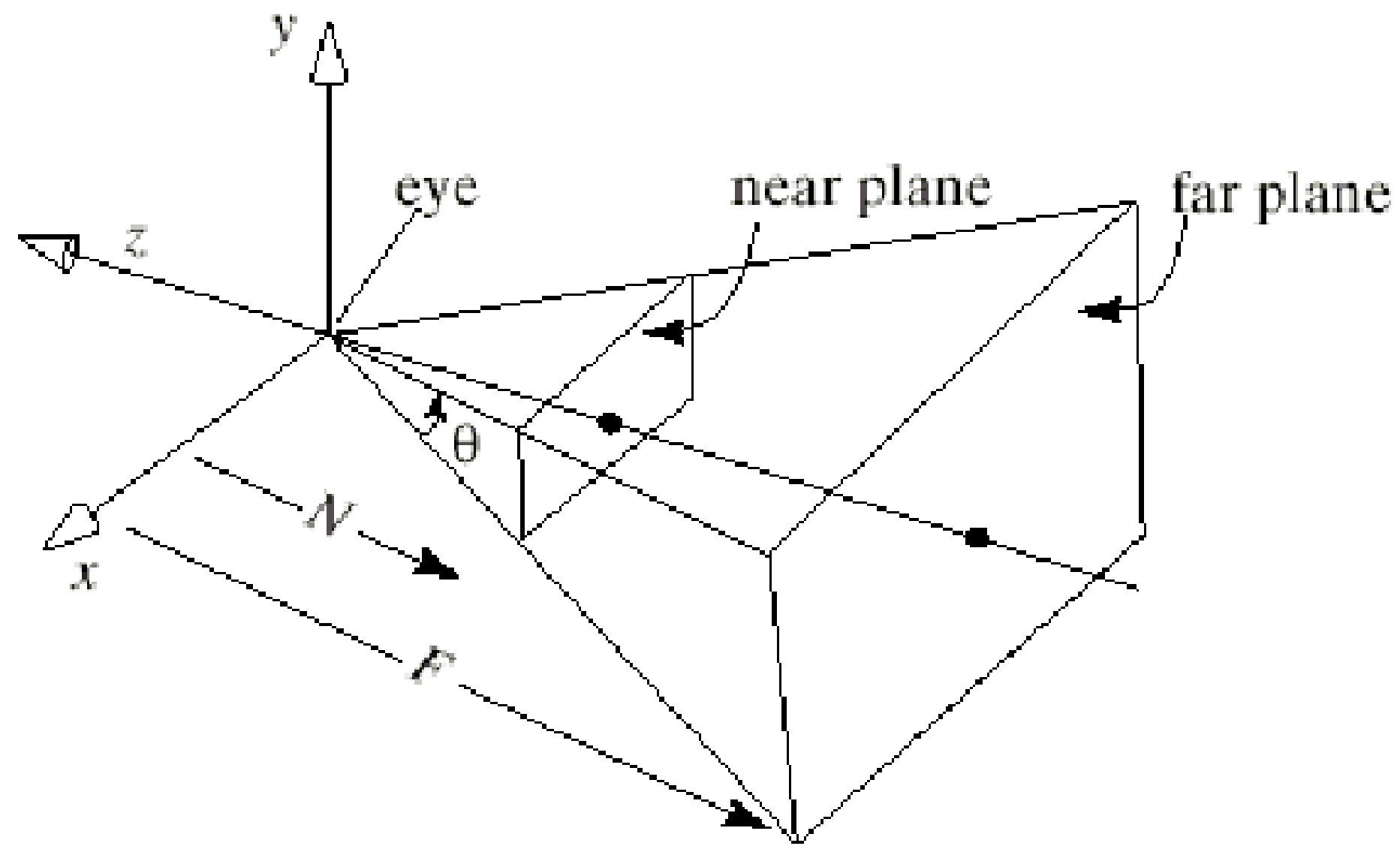


- The Camera is defined by using a parallelepiped as a view volume with two of the walls used as the near and far view planes
- Most applications also allow for a perspective view to be created, this is done by changing the shape of the view volume.

# Perspective Camera

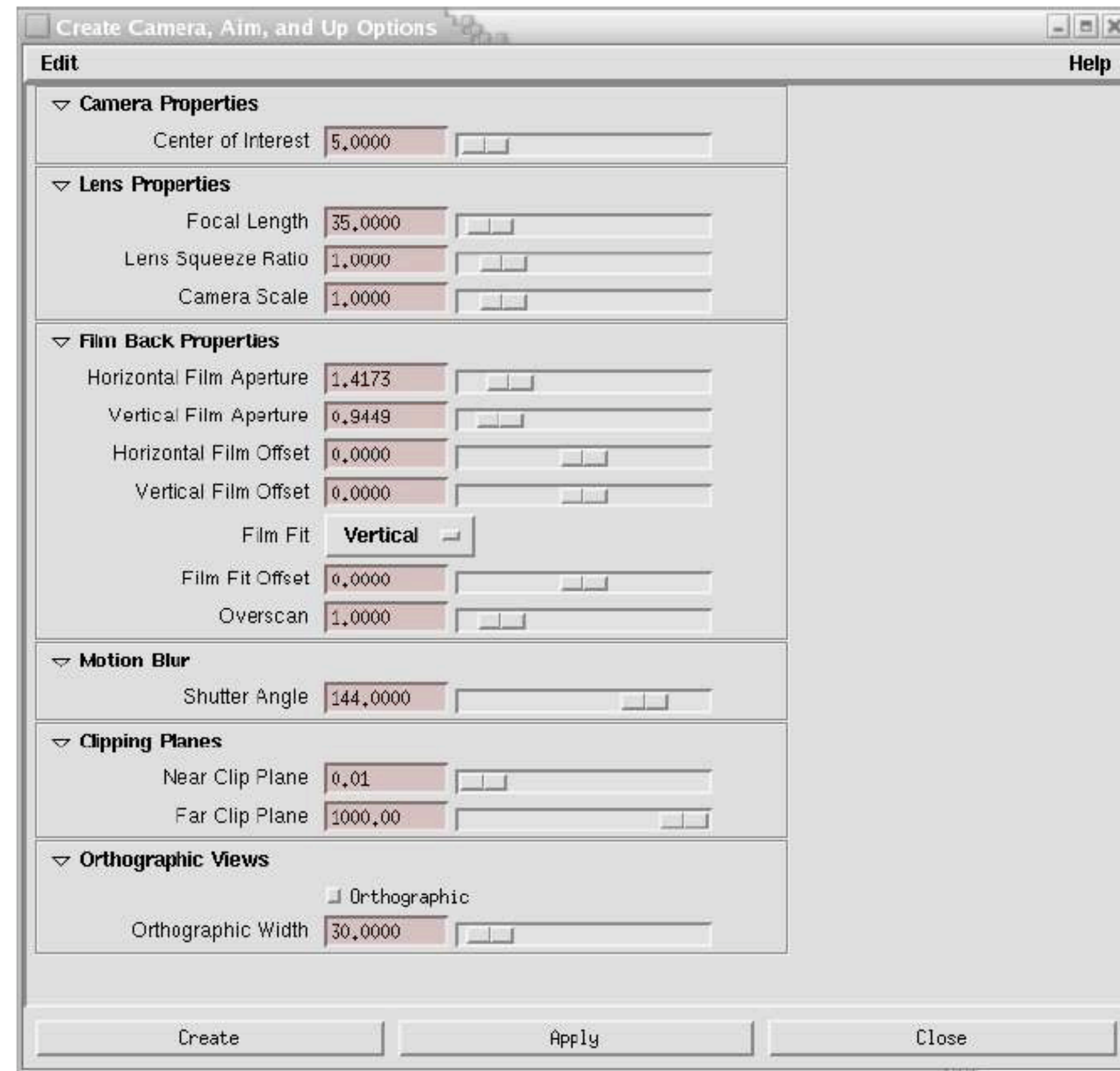
- The Camera has an eye positioned at some point in space.
- Its view volume is a portion of a rectangular pyramid, whose apex is at the eye.
- The opening of the pyramid is set by the viewangle  $\theta$  (part b of figure)
- Two planes are defined perpendicular to the axis of the pyramid : the near and the far plane.
- Where these planes intersect the pyramid they form rectangular windows which have an adjustable aspect ratio.
- The application clips points which lie outside the view volume. Points lying inside the the view volume are projected onto the viewplane to a corresponding point P'
- With a perspective projection the point P' is determined by finding where a line from the eye to P intersects the viewplane.

# Setting the View Volume



- Most applications provides a simple way to set the view volume in a program by setting the projection Matrix

# Maya Camera setup



# Modelling and Viewing

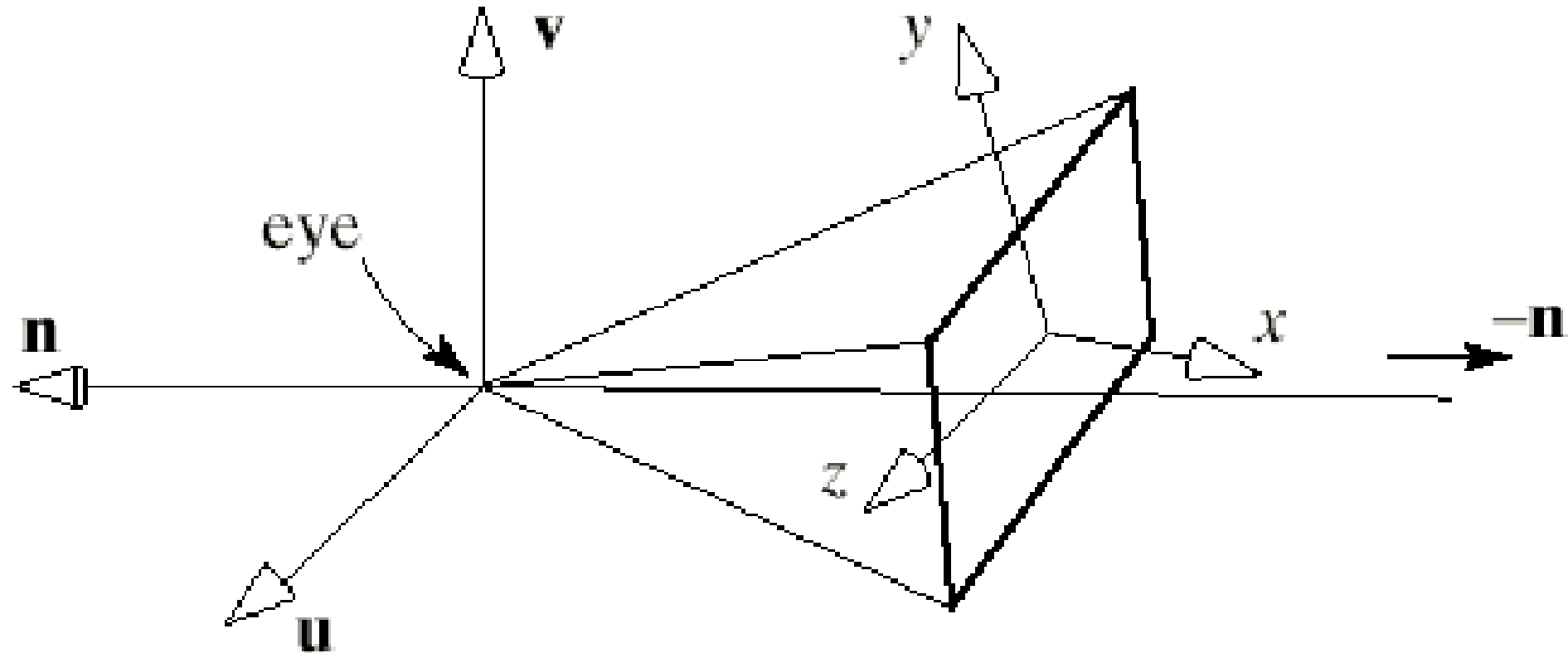
- Typically Modelling and Viewing are a combination of two matrices
- The Modelling Matrix contains the local (world) transformations used to create the model
- The Viewing Matrix contains the Camera transformations to position the model relative to the Camera.
- Usually these two matrices are combined by multiplication to set one matrix known as the MODELVIEW matrix
- All vertices are then passed through this to give the final matrix



# Positioning and Pointing the Camera

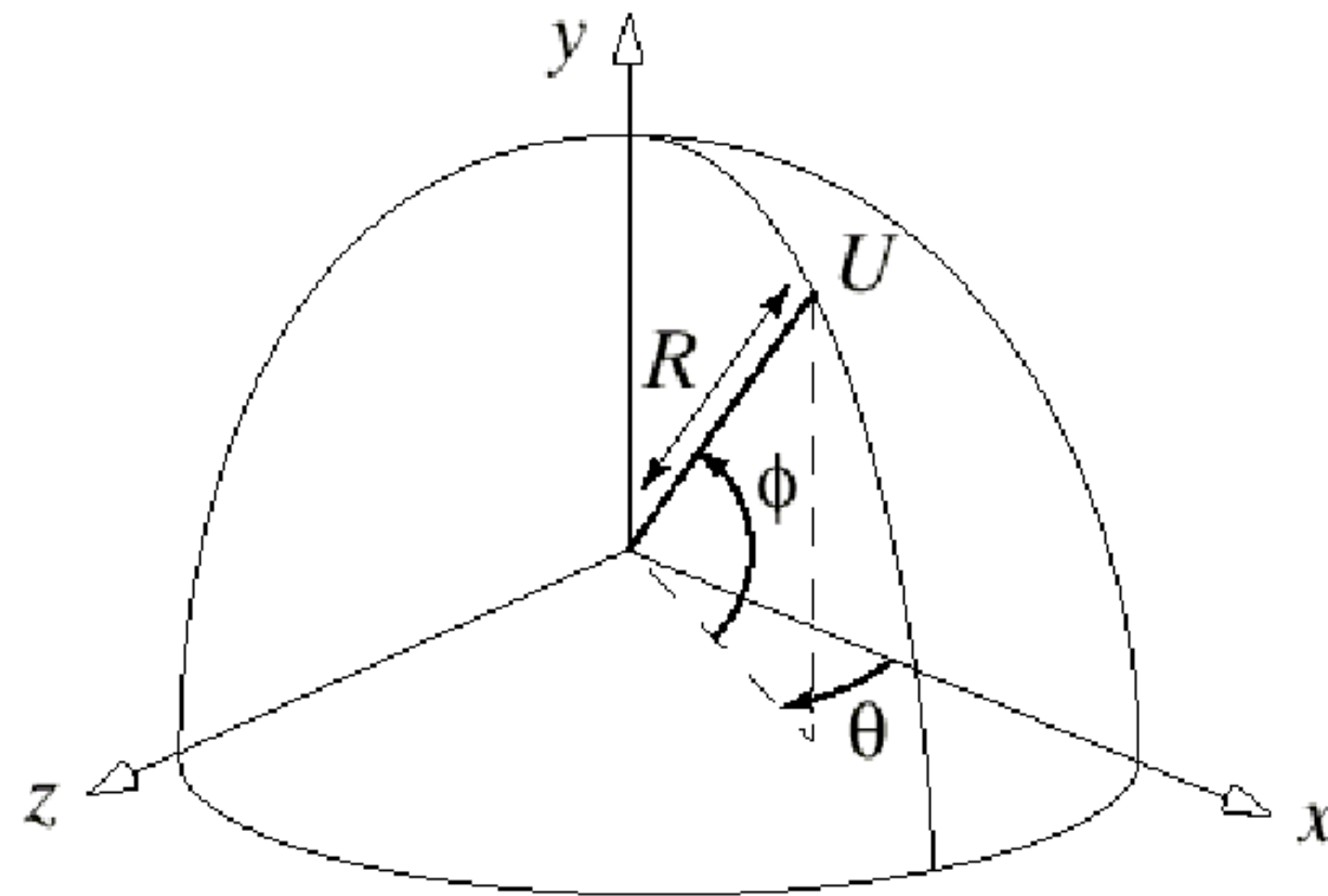
- In order to obtain the desired view of a scene, we move the camera from its default position as shown in the previous slide
- To point it in a particular direction we need to perform rotations and translations which result in changes to the modelview matrix

# The General Camera with Arbitrary Orientation and Position



- It is useful to attach an explicit co-ordinate system to the camera as shown in the figure above.
- This co-ordinate system has its origin at the eye and has three axes, usually called the  $\mathbf{u}$ -,  $\mathbf{v}$ -, and  $\mathbf{n}$ -axis which define the orientation of the camera
- The axes are pointed in directions given by the vectors  $\mathbf{u}$ ,  $\mathbf{v}$ , and  $\mathbf{n}$
- Because, by default, the camera looks down the negative  $z$ -axis, we say in general that the camera looks down the negative  $\mathbf{n}$ -axis in the direction  $-\mathbf{n}$
- The direction  $\mathbf{u}$  points off “to the right of” the camera and the direction  $\mathbf{v}$  points “upward”

# Spherical Geometry



- The above figure shows how a point  $U$  is defined in spherical coordinates.
- $R$  is the radial distance of  $U$  from the origin, and  $\phi$  is the angle that  $U$  makes with the  $xz$ -plane, known as the **latitude** of the point  $U$ .
- $\theta$  is the **azimuth** of  $U$ , the angle between the  $xy$ -plane and the plane through  $U$  and the  $y$ -axis.
- $\phi$  lies in the interval  $-\frac{\pi}{2} \leq \phi < \frac{\pi}{2}$  and  $\theta$  lies in the range  $0 \leq \theta < 2\pi$

# Spherical Co-ordinates

- Using trigonometry we can work out a relationship between spherical co-ordinates and Cartesian coordinates  $(u_x, u_y, u_z)$  for  $U$  the equations are

$$u_x = R \cos(\phi) \cos(\theta), \quad u_y = R \sin(\phi), \quad \text{and} \quad u_z = R \cos(\phi) \sin(\theta)$$

- We can also invert the relations to express  $(R, \phi, \theta)$  in terms of  $(u_x, u_y, u_z)$

$$R = \sqrt{u_x^2 + u_y^2 + u_z^2}, \quad \phi = \sin^{-1} \left( \frac{u_y}{R} \right), \quad \theta = \arctan(u_z, u_x)$$

- The function  $\arctan(,)$  is the two argument form of the arctangent, defined as (atan2 in C)

$$\arctan(y, x) = \begin{cases} \tan^{-1}(y/x) & \text{if } x > 0 \\ \pi + \tan^{-1}(y/x) & \text{if } x < 0 \\ \pi/2 & \text{if } x = 0 \text{ and } y > 0 \\ -\pi/2 & \text{if } x = 0 \text{ and } y < 0 \end{cases}$$

# Example

Suppose that Point  $U$  is at a distance 2 from the origin, is  $60^\circ$  up from the  $xz$ -plane, and is along the negative  $x$ -axis.

Hence  $U$  is in the  $xy$ -plane. Then  $U$  is expressed in spherical coordinates as  $(2, 60^\circ, 180^\circ)$

To convert this to Cartesian coordinates we use the equations :

$$u_x = R \cos(\phi) \cos(\theta),$$

$$u_y = R \sin(\phi),$$

$$u_z = R \cos(\phi) \sin(\theta)$$

Where

$$u_x = 2 \cos(60^\circ) \cos(180^\circ) = -1$$

$$u_y = 2 \sin(60^\circ) = 1.732$$

$$u_z = 2 \cos(60^\circ) \sin(180^\circ) \approx 0$$

# C Example

```
1  #include <stdio.h>
2  #include <math.h>
3
4
5  int main(void)
6  {
7      //cos and sin need angle in radians
8      //to convert we mult by PI/180
9      float scale=M_PI/180;
10     float phi=60*scale;
11     float theta=180*scale;
12     float radius=2.0;
13
14     printf("x=%f\n", radius*cos(phi)*cos(theta));
15     printf("y=%f\n", radius*sin(phi));
16     printf("z=%f\n", radius*cos(phi)*sin(theta));
17
18     return 1;
19 }
```

# Direction Cosines

- The direction of point  $U$  in the previous example is given in terms of two angles : the **azimuth** and the **latitude**.
- Directions are often specified in an alternative useful way through direction cosines.
- The direction cosines of a line through the origin are the cosines of the three angles it makes with the  $x$ –,  $y$ – and  $z$ –axes respectively.
- Recall that the cosine of the angle between two unit vectors is given by their dot product.
- Using the given point  $U$  we form the position vector  $(u_x, u_y, u_z)$
- We also know that the length of the vector is  $R$  so we normalise it to unit length thus  $\mathbf{m} = (u_x/R, u_y/R, u_z/R)$

# Direction Cosines II

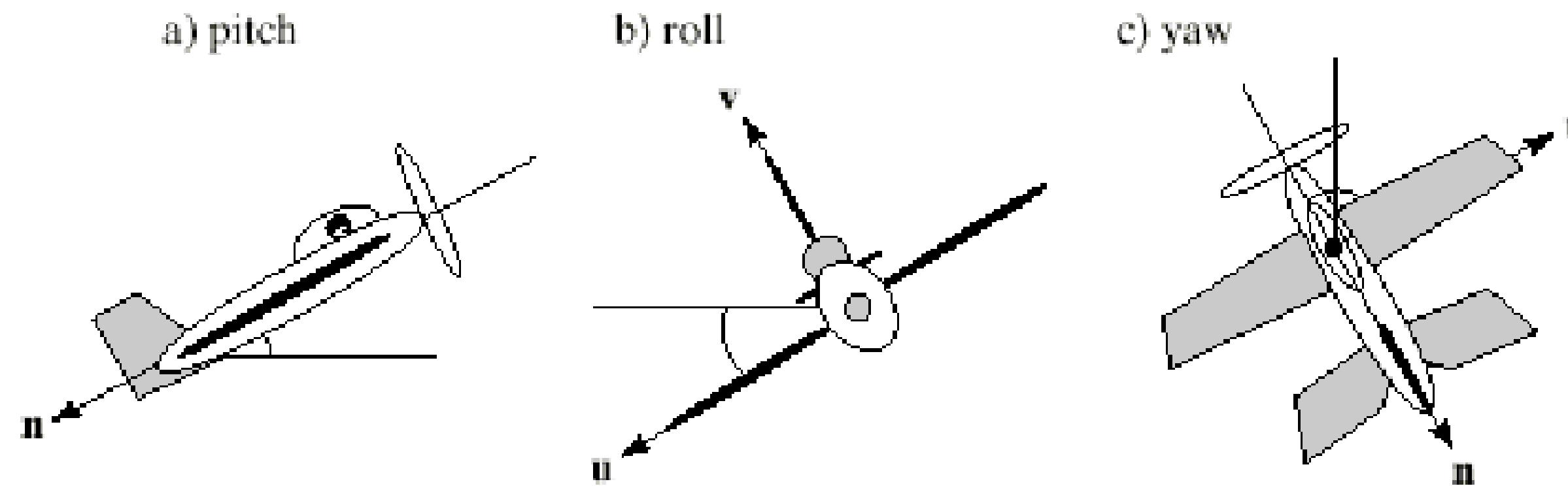
- Then the cosine of the angle it makes with the  $x$ -axis is given by the dot product  $\mathbf{m} \bullet \mathbf{i} = u_x/R$  which is the first component of  $\mathbf{m}$
- Similarly the second and third components of  $\mathbf{m}$  are the second and third direction cosines respectively
- Calling the angles made with the  $x$ -,  $y$ - and  $z$ -axis  $\alpha$ ,  $\beta$  and  $\gamma$  respectively, we have, for the three direction cosines for the line from 0 to  $U$

$$\begin{aligned} \cos(\alpha) &= \frac{u_x}{R}, \\ \cos(\beta) &= \frac{u_y}{R}, \\ \text{and} \\ \cos(\gamma) &= \frac{u_z}{R} \end{aligned}$$

- Note that the three direction cosines are related, since the sum of their squares is always unity

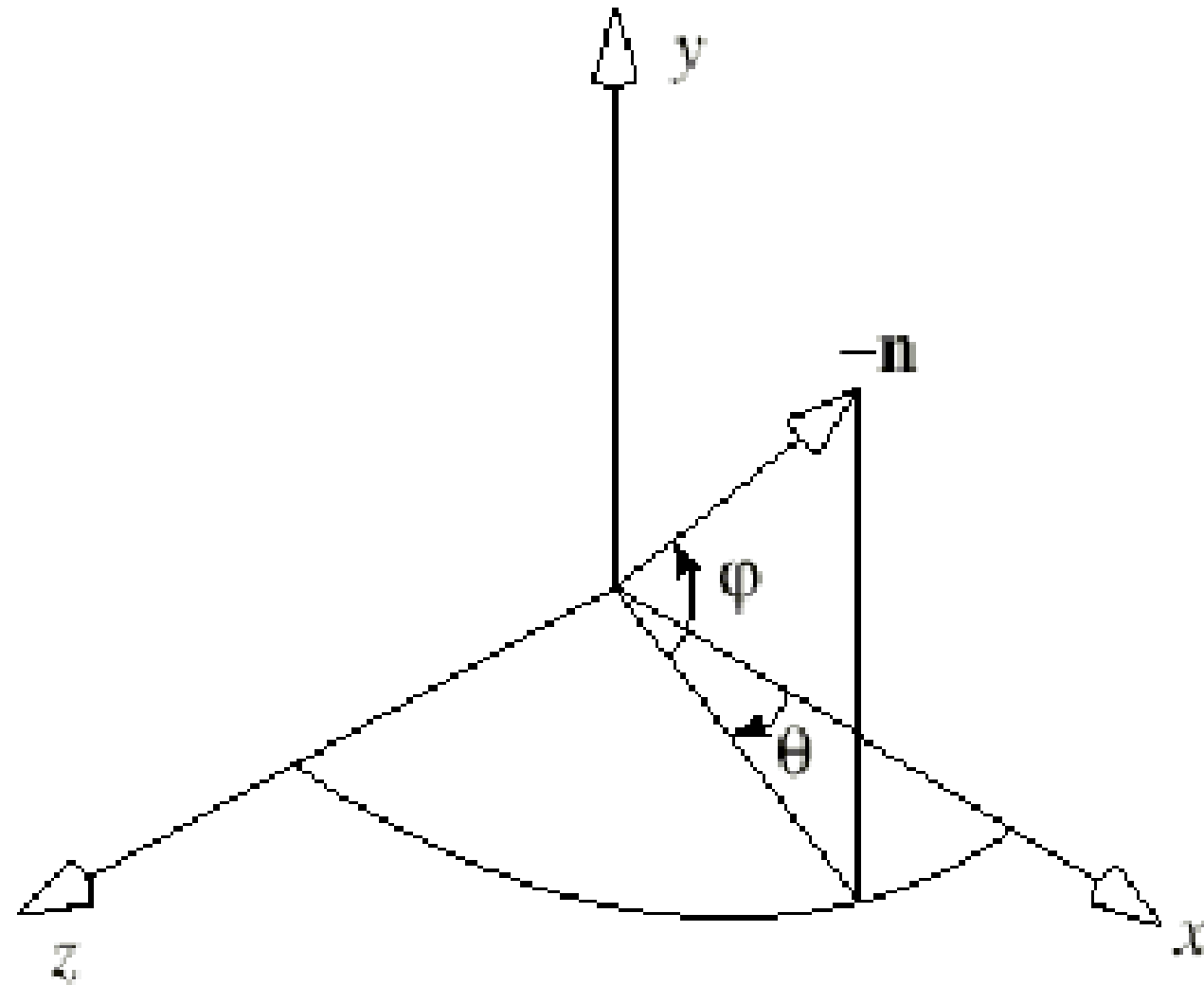


# Describing Orientation



- Position is easy to describe, but orientation is difficult. It helps to specify orientation using the aviation terms **pitch, heading, yaw** and **roll** as shown in the figure above
- The pitch of an air plane is the angle that its longitudinal axis (running from tail to nose having direction **-n**) makes with the plane.
- A roll is a rotation about the longitudinal axis; the roll is the amount of rotation relative to the horizontal.
- An airplane's heading is the direction it is headed (This is also called azimuth and bearing)

# Describing Orientation



- To find the heading and the pitch, given  $\mathbf{n}$  simply express  $-\mathbf{n}$  in spherical coordinates, as shown in the figure
- The vector  $-\mathbf{n}$  has longitude and latitude given by the angles  $\theta$  and  $\phi$  respectively.
- The heading of the plane is given by the longitude of  $-\mathbf{n}$  and the pitch is given by the latitude of  $-\mathbf{n}$

# Finding roll, pitch, and heading given vectors $\mathbf{u}$ , $\mathbf{v}$ , and $\mathbf{n}$ .

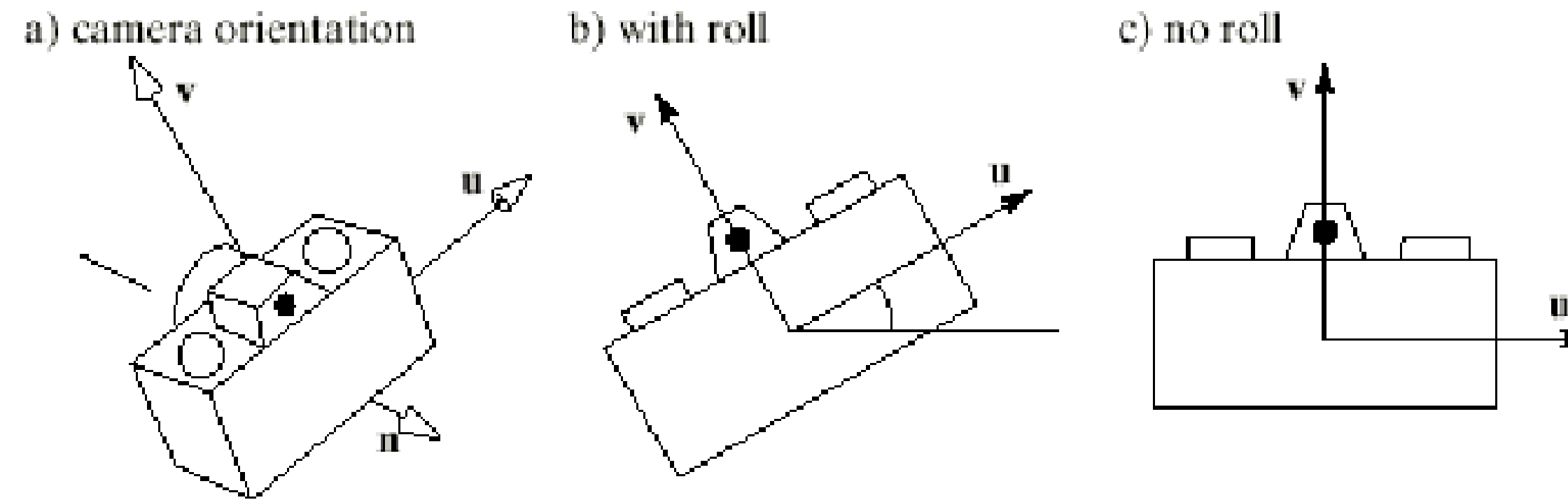
Assuming the camera is based on a coordinate system with axes in the directions  $\mathbf{u}$ ,  $\mathbf{v}$ , and  $\mathbf{n}$ , all unit vectors. The heading and pitch of the camera is denoted by  $\theta$  and  $\phi$ . The relationship of Cartesian and spherical coordinates system is given by:

$$\begin{aligned} n_x &= \cos(\phi)\cos(\theta) \\ n_y &= \sin(\phi) \\ n_x &= \sin(\phi)\cos(\theta) \Rightarrow \begin{cases} \theta = \tan^{-1}(-n_z / -n_x) \\ \phi = \sin^{-1}(-n_y) \end{cases} \end{aligned}$$

The roll of the camera is the angle its  $u$ -axis makes with the horizontal, To find it construct a vector

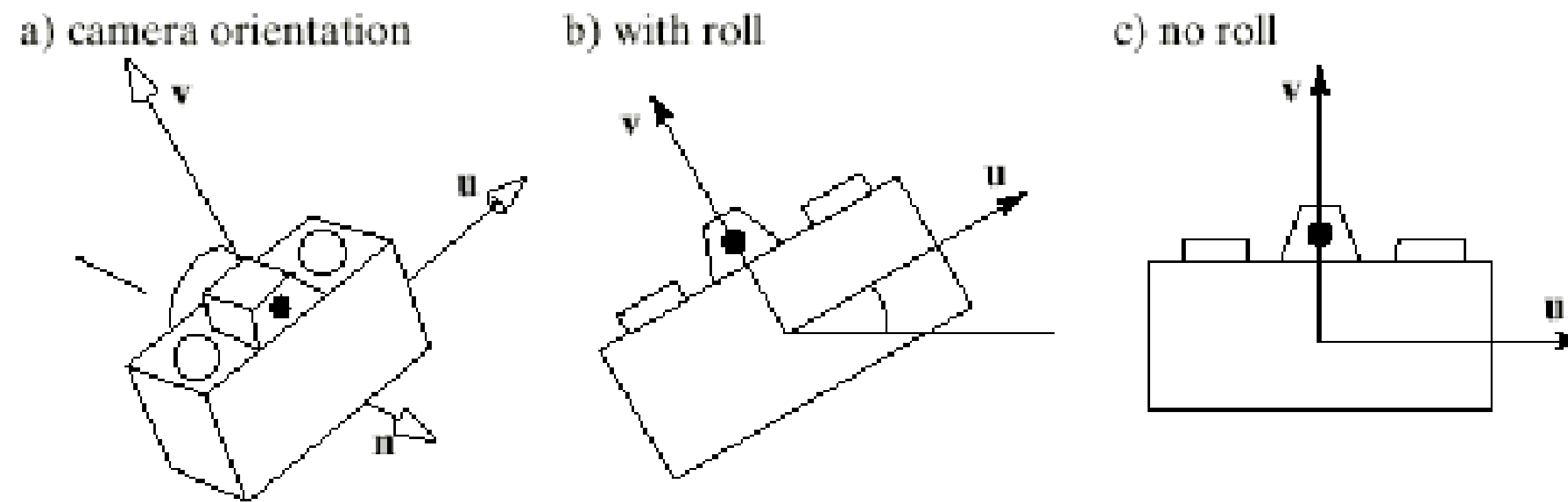
$$\begin{aligned} \mathbf{b} &= \mathbf{j} \times \mathbf{n} = (n_z, 0, -n_x) \\ \mathbf{b} \bullet \mathbf{j} &= (\mathbf{j} \times \mathbf{n}) \bullet \mathbf{j} = 0 \Rightarrow \mathbf{b} \text{ is horizontal} \\ \mathbf{b} \bullet \mathbf{n} &= (\mathbf{j} \times \mathbf{n}) \bullet \mathbf{n} = 0 \Rightarrow \mathbf{b} \text{ is perpendicular to } \mathbf{n} \text{ and therefore lies in the } uv\text{-plane} \\ \text{Since} \\ |\mathbf{b}| &= \sqrt{n_x^2 + n_z^2} \\ \mathbf{b} \bullet \mathbf{u} &= u_x n_z - u_z n_x \\ \text{The angle between } \mathbf{b} \text{ and } \mathbf{u} \text{ is given by} \\ \cos(roll) &= \frac{\mathbf{b} \bullet \mathbf{u}}{|\mathbf{b}|} = \left( \frac{u_x n_z - u_z n_x}{\sqrt{n_x^2 + n_z^2}} \right) \text{ so the roll of the camera can be expressed as} \\ roll &= \cos^{-1} \left( \frac{u_x n_z - u_z n_x}{\sqrt{n_x^2 + n_z^2}} \right) \\ heading &= \arctan(-n_z, -n_x) \text{ and } pitch = \sin^{-1}(-n_y) \end{aligned}$$

# Camera



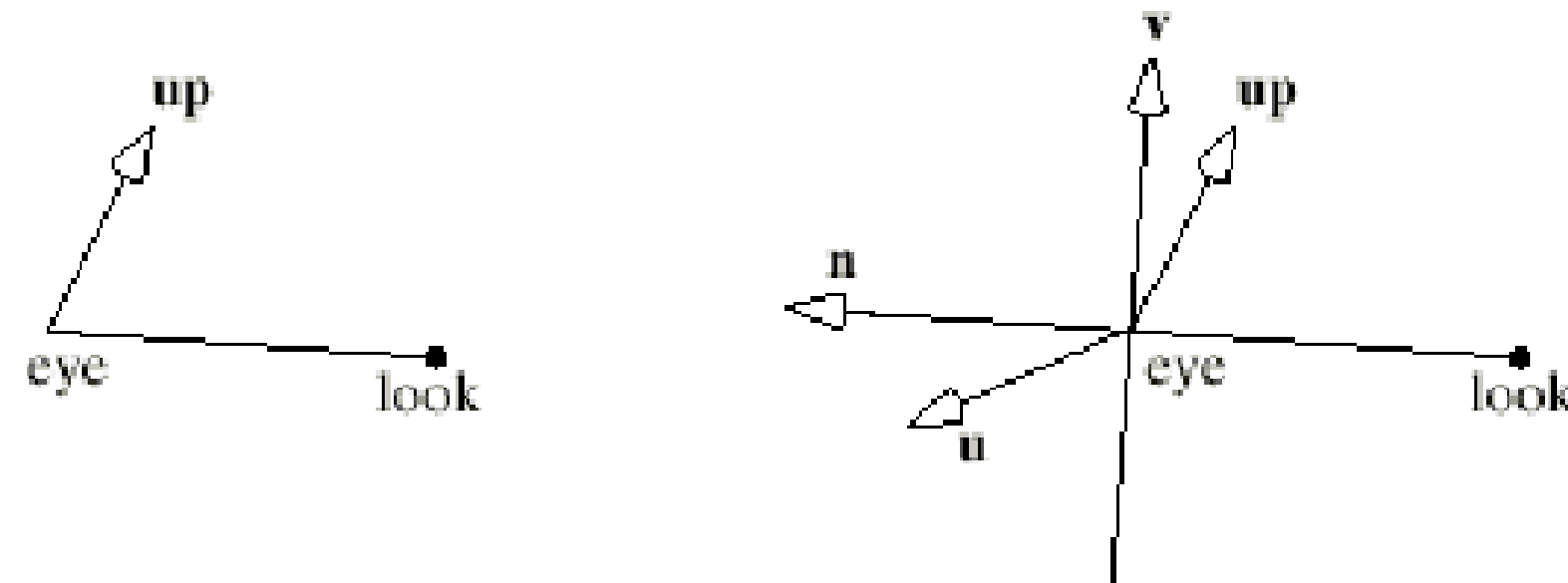
- The above figure shows a camera with the same coordinate system attached to it
- It has **u**-, **v**- and **n**-axes and an origin at position eye
- b) shows the camera with a roll applied to it
- c) shows the camera with zero roll or “no-roll” camera
- The  $u$ -axis of a no-roll camera is horizontal, that is perpendicular to the  $y$ -axis of the “world”
- Note that a no-roll camera can still have an arbitrary **n** direction, so it can have any pitch or heading.

# Camera



- The above figure shows a camera with the same coordinate system attached to it
- It has  $u$ -,  $v$ - and  $n$ -axes and an origin at position eye
- b) shows the camera with a roll applied to it
- c) shows the camera with zero roll or “no-roll” camera
- The  $u$ -axis of a no-roll camera is horizontal, that is perpendicular to the  $y$ -axis of the “world”
- Note that a no-roll camera can still have an arbitrary  $n$  direction, so it can have any pitch or heading.

# How a typical camera works inside a modelling application



- What are the directions of **u**, **v** and **n** when we place our camera with given values for eye, look and up
- If given the locations of eye, look and up, we immediately know that **n** must be parallel to the vector **eye-look**, as shown above. so we can set **n=eye-look**
- We now need to find a **u** and a **v** that are perpendicular to **n** and to each other.
- The **u** direction points “off to the side” of a camera, so it is natural to make it perpendicular to **up** which the user has said is the “upward” direction.

# How a camera works II

- An easy way to build a vector that is perpendicular to two given vector is to form their cross product, so we set  $\mathbf{u} = \mathbf{up} \times \mathbf{n}$
- The user should not choose an  $\mathbf{up}$  direction that is parallel to  $\mathbf{n}$ , because  $\mathbf{u}$  then would have zero length.
- We choose  $\mathbf{u} = \mathbf{up} \times \mathbf{n}$  rather than  $\mathbf{n} \times \mathbf{up}$  so that  $\mathbf{u}$  will point “to the right” as we look along  $-\mathbf{n}$
- With  $\mathbf{u}$  and  $\mathbf{n}$  formed it is easy to determine  $\mathbf{v}$  as it must be perpendicular to both and is thus the cross product of  $\mathbf{u}$  and  $\mathbf{n}$  thus  $\mathbf{v} = \mathbf{n} \times \mathbf{u}$
- Notice that  $\mathbf{v}$  will usually not be aligned with  $\mathbf{up}$  as  $\mathbf{v}$  must be aimed perpendicular to  $\mathbf{n}$  whereas the user provides  $\mathbf{up}$  as a suggestion of “upwardness” and the only property of  $\mathbf{up}$  that is used is its cross product with  $\mathbf{n}$

# Camera III

- To summarise, given eye look and up, we form

$$\mathbf{n} = \textit{eye} - \textit{look}$$

$$\mathbf{u} = \mathbf{up} \times \mathbf{n}$$

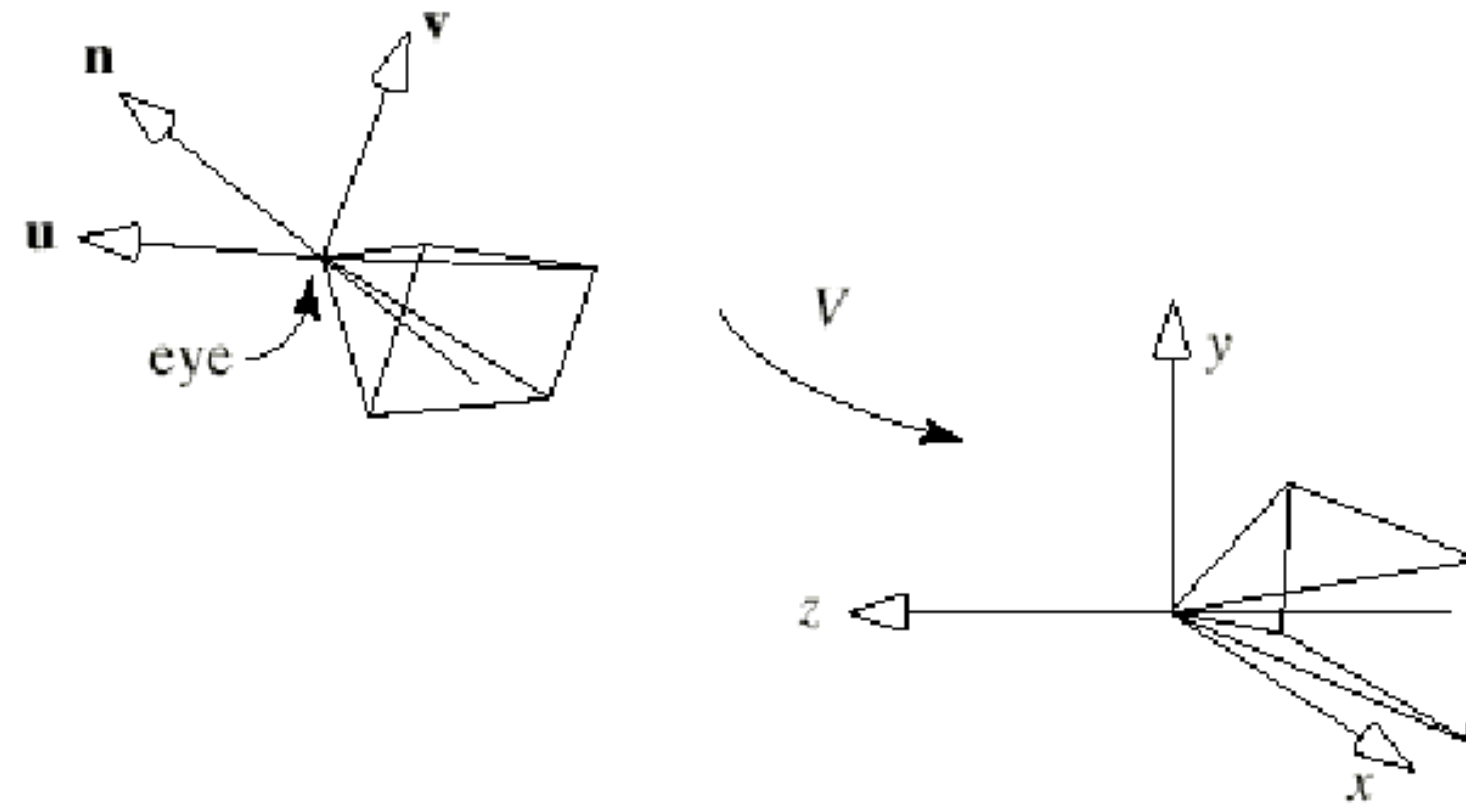
and

$$\mathbf{v} = \mathbf{n} \times \mathbf{u}$$

and then normalise the vectors to unit length.



# The Camera and the modelview Matrix



- The modelview is the product of two matrices the matrix  $\mathbf{V}$  that accounts for the transformation of the world point into camera coordinates.
- and  $\mathbf{M}$  that embodies all of the modelling transformations applied to the points.
- Typically we build the  $\mathbf{V}$  matrix and post-multiplies the current matrix by it.
- Because the job of the  $\mathbf{V}$  matrix is to convert world coordinates to camera coordinates it must transform the camera's coordinate system into the generic position for the camera as shown in the figure.

# MODEL /VIEW Matrix

- This means that  $V$  must transform eye into the origin,  $\mathbf{u}$  into the vector  $\mathbf{i}$ ,  $\mathbf{v}$  into  $\mathbf{j}$ , and  $\mathbf{n}$   $\mathbf{k}$
- The easiest way to define  $V$  is to use the following matrix

$$V = \begin{pmatrix} u_x & u_y & u_z & d_x \\ v_x & v_y & v_z & d_y \\ n_x & n_y & n_z & d_z \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Where  $(d_x, d_y, d_z) = (-eye \bullet \mathbf{u}, -eye \bullet \mathbf{v}, -eye \bullet \mathbf{n})$

The matrix  $V$  is created by *gluLookAt* and post-multiplies the current matrix.

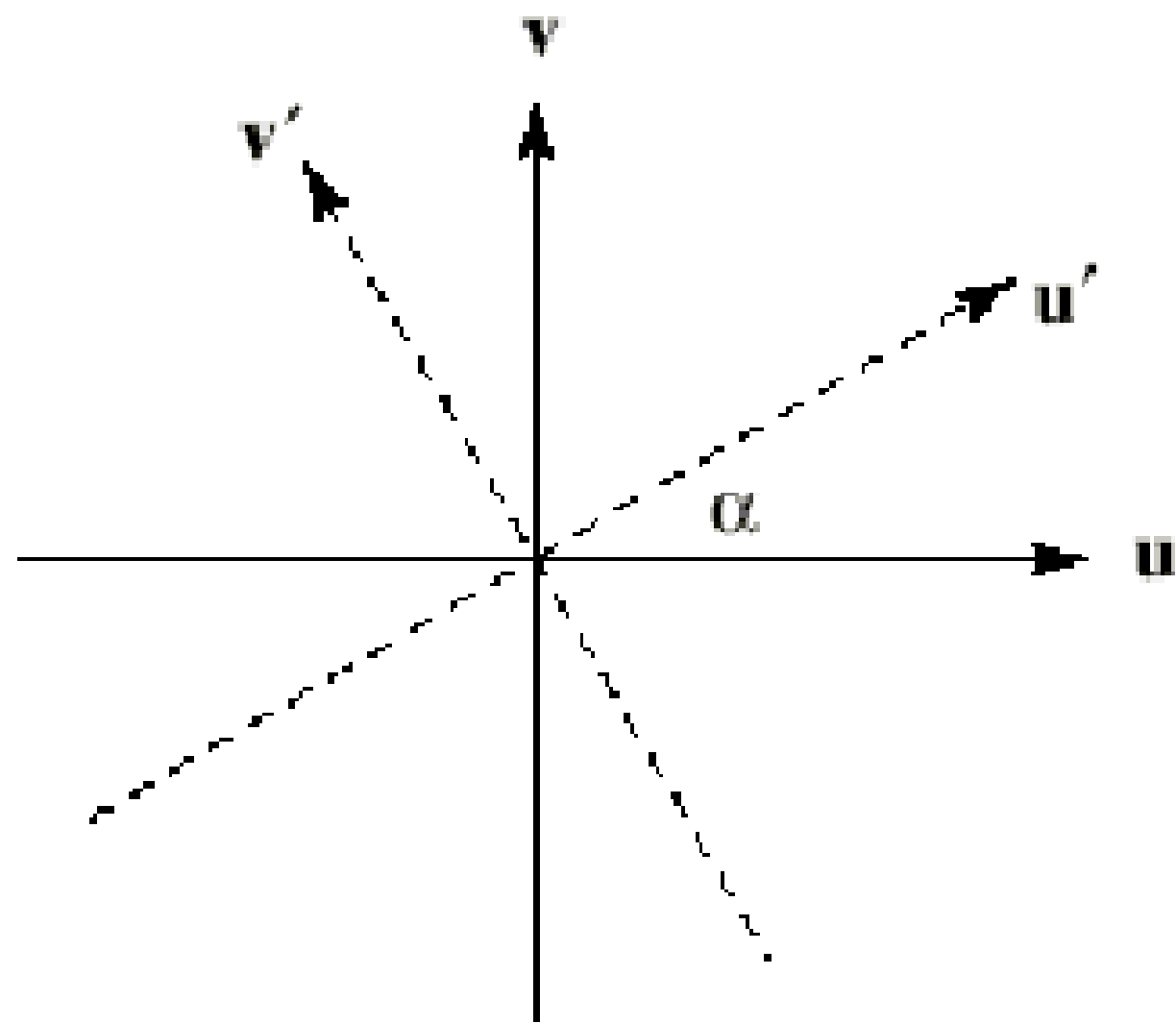
# The Camera Revisited

- In order to have full control over the viewing of a scene we need to create our own Camera
- After each change to the camera's state the camera will “tell” the applicaion where the current View of the scene is to be modifying the MODELVIEW matrix
- When we use the Virtual Camera we create a camera by setting the EYE, LOOK as points and UP as a vector.
- We then call the functions **roll**, **pitch**, **yaw** and **slide** to move the camera's position (or equivalent).

# Sliding the Camera

- Sliding the camera means moving it along one of its own axis, that is in the **u**, **v** or **n** axis without rotating it.
- Since the camera is looking along the negative **n**-axis, movement along **n** is “forward” or “back”
- Similarly, movement along **u** is “left” or “right” and along **v** is “up” or “down”
- It is simple to move the camera along one of its axes. To move it a distance  $D$  along its  $u$ -axis, set  $\text{eye}$  to  $\text{eye} + D\mathbf{u}$
- This can be done for each of the axes and combined into a single function

# Rotating the Camera



- To roll the camera is to rotate it about its own  $n$ -axis. This means that both the directions  $u$  and  $v$  must be rotated as shown below
- We form two new axes  $u'$  and  $v'$  that lie in the same plane as  $u$  and  $v$ , yet have been rotated through the angle  $\alpha$  radians.
- To do this we need to form  $u'$  as the appropriate linear combination of  $u$  and  $v$  and similarly for  $v'$

# rolling the Camera

$$\begin{aligned}\mathbf{u}' &= \cos(\alpha)\mathbf{u} + \sin(\alpha)\mathbf{v} \\ \mathbf{v}' &= -\sin(\alpha)\mathbf{u} + \cos(\alpha)\mathbf{v}\end{aligned}$$

- To roll the camera we need to replace the current  $\mathbf{u}$  and  $\mathbf{v}$  axes with the new  $\mathbf{u}'$  and  $\mathbf{v}'$  axes
- The Function to do this is shown next

# References

- Computer Graphics With OpenGL, F.S. Hill jr, Prentice Hall (most images from the instructors pack of this book)
- Basic Algebra and Geometry. Ann Hirst and David Singerman. Prentice Hall 2001
- "Essential Mathematics for Computer Graphics fast" John Vince Springer-Verlag London
- "Geometry for Computer Graphics: Formulae, Examples and Proofs" John Vince Springer-Verlag London 2004