

CS3500
Computer Graphics
Module: 3D Graphics

P. J. Narayanan
Spring 2009

3D Graphics: Objectives

- Representation and manipulation of 3D objects built using 3D primitives.
- Placing a camera in the 3D world and computing what it sees.
- Final output is a 2D image. A 3D-to-2D projection is involved.
- Additional complication: **Which object is in the front and which is at the back?**

2D Graphics

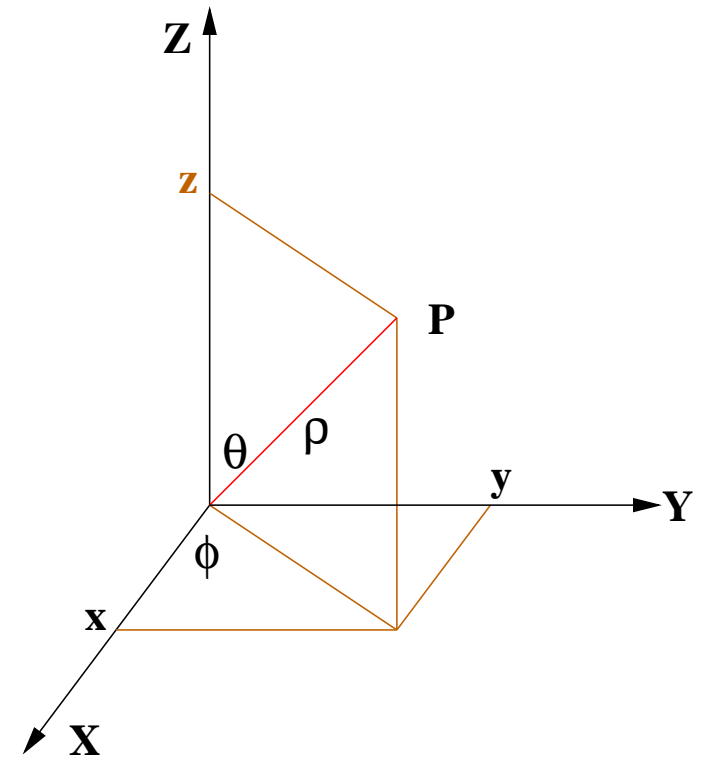
- Draw on a 2D canvas using 2D primitives.
- View it frontally from a certain distance away.
- Only **scaling** and **translation** between canvas coordinates and frame buffer coordinates.
- A window on the canvas is mapped to a window on screen or frame buffer using the **(window-to-)viewport** transformation.
- Translate to origin, scale to new size, translate to new position.

Perspective Effects

- Perspective distortion is relevant for 3D points.
- Farther points are scaled by larger numbers.
- Distances from the camera are important as everything happens with respect to the camera.
- First, compute the world with respect to the camera.
- Second, project the 3D world to a 2D plane.

3D Coordinates

- Cartesian: (x, y, z) .
- Polar: (ρ, θ, ϕ)
- $z = \rho \cos \theta,$
 $y = \rho \sin \theta \sin \phi$
 $x = \rho \sin \theta \cos \phi$
- $\rho^2 = x^2 + y^2 + z^2,$
 $\phi = \tan^{-1}(y/x),$
 $\theta = \tan^{-1}(\sqrt{x^2 + y^2}/z)$



Homogeneous Representation

- Convert to a 4-vector with a scale factor as fourth.
 $(x, y, z) \equiv [kx \ ky \ kz \ k]^T$ for any $k \neq 0$.
- Translation, rotation, scaling, shearing, etc. become linear operations represented by 4×4 matrices.
- What does $[x \ y \ z \ 0]^T$ mean?
- $[a \ b \ c \ d]^T$ could be a point or a plane. Lines are specified using two such vectors, either as join of two points or intersection of two planes!

Transformation Matrices: Rotations

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$R_y = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_z = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- CCW +ve; orthonormal; length preserving; rows give direction vectors that rotate onto each axis; columns

Translation, Scaling, Composite

$$T(a, b, c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad S(a, b, c) = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- A sequence of transforms can be represented using a composite matrix: $\mathbf{M} = \mathbf{R}_x \mathbf{T} \mathbf{R}_y \mathbf{S} \mathbf{T} \dots$
- Operations are not commutative, but are associative.
- \mathbf{RT} and \mathbf{TR} ??

Rotation Followed by Translation

$$\begin{bmatrix} \mathbf{I} & a \\ & b \\ & c \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & a \\ & b \\ & c \\ \mathbf{0} & 1 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} a \\ \mathbf{I} & b \\ & c \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{t} \cdot \mathbf{r}_1 \\ \mathbf{R} & \mathbf{t} \cdot \mathbf{r}_2 \\ & \mathbf{t} \cdot \mathbf{r}_3 \\ \mathbf{0} & 1 \end{bmatrix}$$

Commutativity

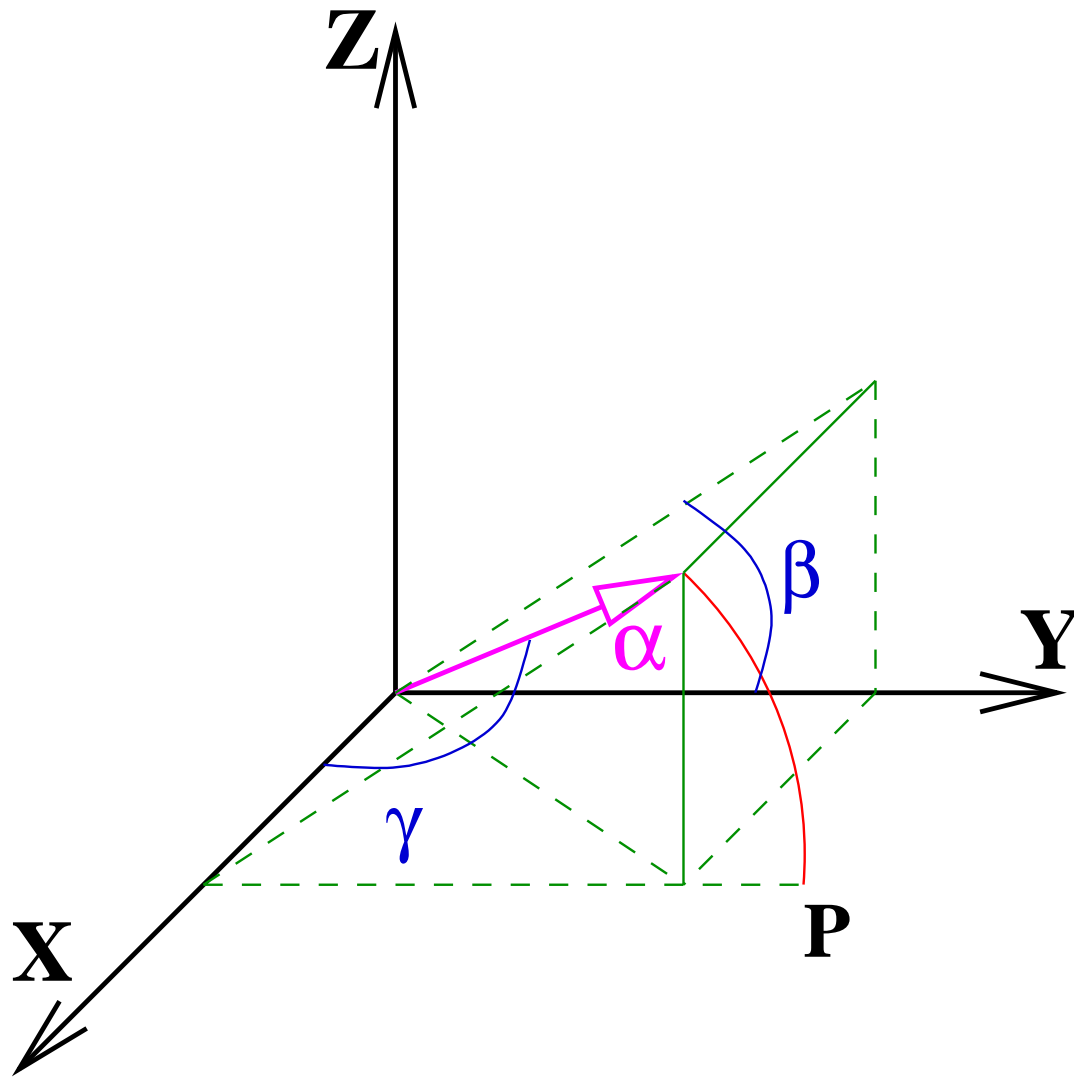
- Translations are commutative: $\mathbf{T}_1\mathbf{T}_2 = \mathbf{T}_2\mathbf{T}_1$.
- Scaling is commutative: $\mathbf{S}_1\mathbf{S}_2 = \mathbf{S}_2\mathbf{S}_1$.
- Rotations are NOT commutative: $\mathbf{R}_1\mathbf{R}_2 \neq \mathbf{R}_2\mathbf{R}_1$.
- Consider the effect on Z-axis of $\mathbf{R}_x(90)\mathbf{R}_y(90)$ and $\mathbf{R}_y(90)\mathbf{R}_x(90)$!
- Also, $\mathbf{R}_1\mathbf{T}_2 \neq \mathbf{T}_2\mathbf{R}_1$ etc....

Rotation about an axis parallel to Z

- An axis parallel to Z axis, passing through point $(x, y, 0)$.
- Translate so that the axis passes through the origin:
 $\mathbf{T}(-x, -y, k)$ for any k !!
- Overall: $\mathbf{M} = \mathbf{T}(x, y, -k) \mathbf{R}_Z(\theta) \mathbf{T}(-x, -y, k)$
- Why shouldn't k matter? \mathbf{R}_Z doesn't affect the z coordinate. So, whatever k is added first will be subtracted later

Rotation about an axis α

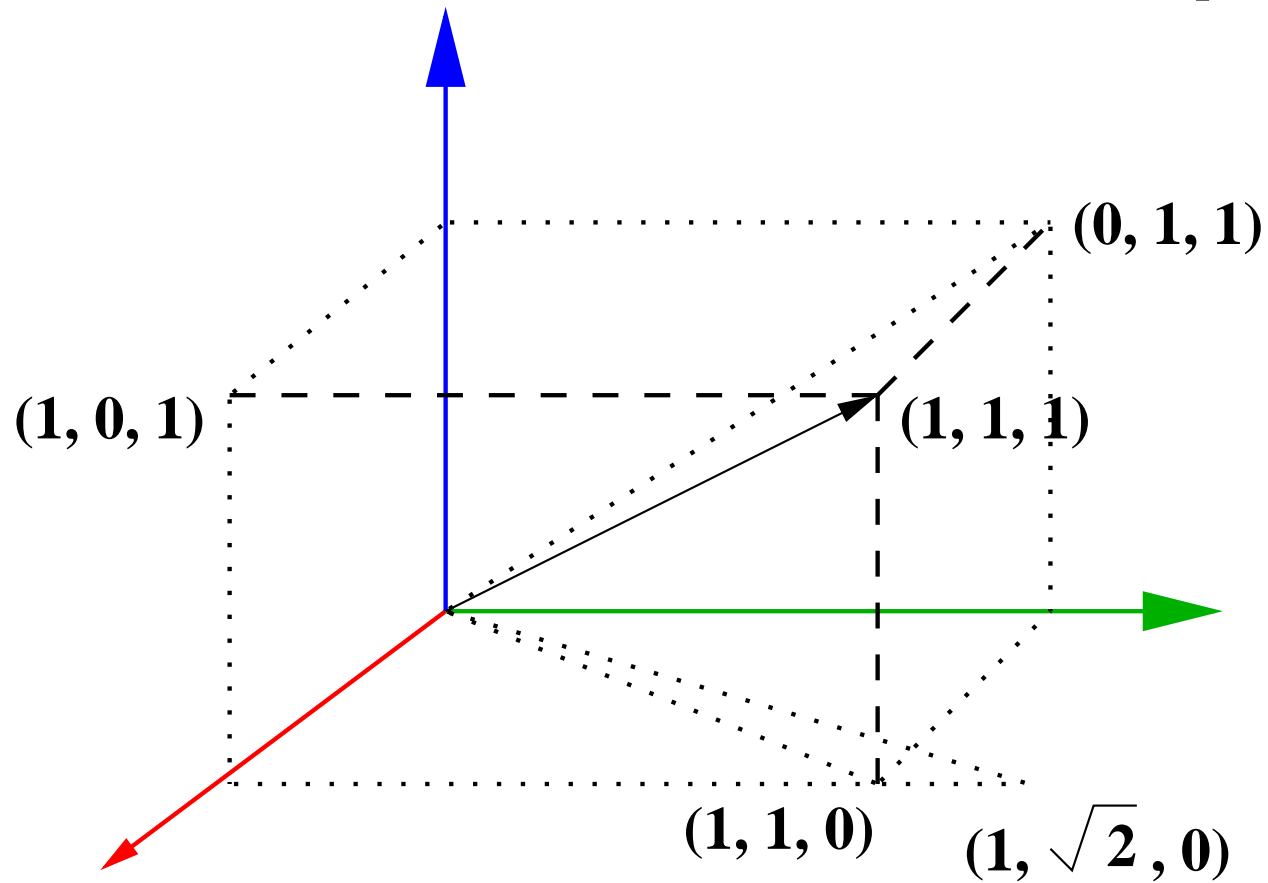
- What is $\mathbf{R}_\alpha(\theta)$?
- 3-step process:
 1. Apply $\mathbf{R}_{\alpha\mathbf{x}}$ to align α with the X axis.
 2. Rotate about X by angle θ .
 3. Undo the first rotation using $\mathbf{R}_{\alpha\mathbf{x}}^{-1}$.
- Net result: $\mathbf{R}_\alpha(\theta) = \mathbf{R}_{\alpha\mathbf{x}}^{-1}\mathbf{R}_{\mathbf{x}}(\theta)\mathbf{R}_{\alpha\mathbf{x}}$



Computing \mathbf{R}_α

- First rotate by $-\beta$ about X axis. Vector α would lie in the XY plane, with tip at point \mathbf{P} .
- Next rotate by $-\gamma$ about Z axis. Vector α would coincide with the X axis.
- Also, $\tan \beta = \frac{\alpha_z}{\alpha_y}$ and $\tan \gamma = \frac{\sqrt{\alpha_y^2 + \alpha_z^2}}{\alpha_x} = \frac{\sqrt{1 - \alpha_x^2}}{\alpha_x}$ if $|\alpha| = 1$.
- Overall: $\mathbf{R}_{\alpha\mathbf{x}} = \mathbf{R}_z(-\gamma)\mathbf{R}_x(-\beta)$ and $\mathbf{R}_{\alpha\mathbf{x}}^{-1} = \mathbf{R}_x(\beta)\mathbf{R}_z(\gamma)$.
- Could have aligned α with Y or Z axis with same results.

Example: Rotation about $[1\ 1\ 1]^T$



Computing $\mathbf{R}_{\alpha\mathbf{x}}$: Method 1

- Rotate by $-\pi/4$ about X. $\mathbf{R}_{\mathbf{X}}(-\frac{\pi}{4}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$
- $\mathbf{R}_{\mathbf{Z}}(-\arctan(\sqrt{2})) = \begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{2}{\sqrt{6}} & 0 \\ \frac{-2}{\sqrt{6}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- $\mathbf{R}_{\alpha\mathbf{x}}^{\mathbf{I}} = \mathbf{R}_{\mathbf{Z}}(-\tan^{-1}(\sqrt{2})) \mathbf{R}_{\mathbf{X}}(-\frac{\pi}{4}) = \begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ \frac{-2}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & 0 \\ 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Alternate Way to get $\mathbf{R}_{\alpha\mathbf{x}}$

- After rotation, α will align with X-axis. Hence that is the first row \mathbf{r}_1 .
- Find an axis orthogonal to α to be the second row. Take an arbitrary vector v (not parallel to α). Now, $\mathbf{r}_2 = \alpha \times v$.

- Lastly, $\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$ and $\mathbf{R}_{\alpha\mathbf{x}} = \begin{bmatrix} \alpha & 0 \\ v \times \alpha & 0 \\ \alpha \times v \times \alpha & 0 \\ \mathbf{0} & 1 \end{bmatrix}$

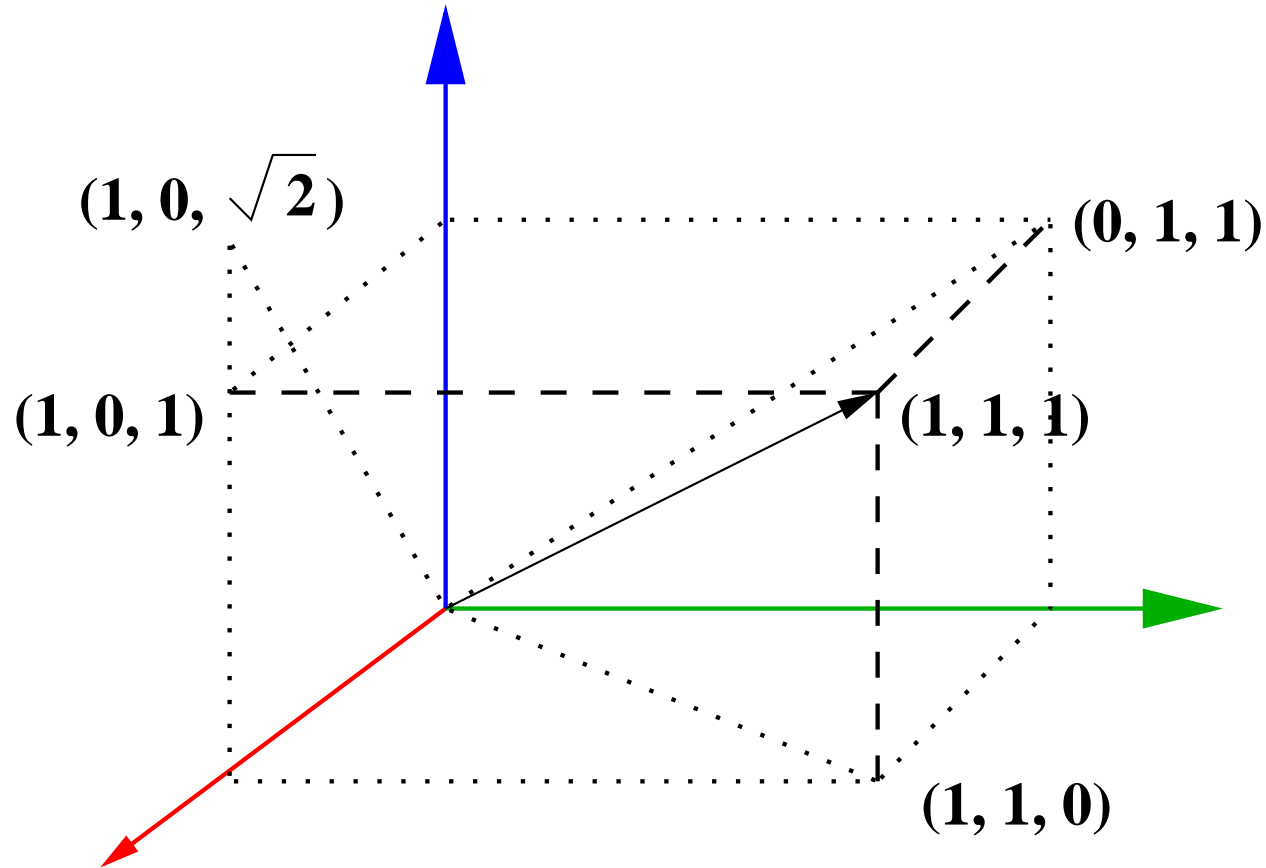
- Many possibilities, all with the same result!

Computing $\mathbf{R}_{\alpha\mathbf{x}}$: Method 2

- $[1\ 1\ 1]^T$ will lie on X-axis. First row $\mathbf{r}_1 = \left[\frac{1}{\sqrt{3}}\ \frac{1}{\sqrt{3}}\ \frac{1}{\sqrt{3}}\right]^T$.
- Second row $\mathbf{r}_2 = \alpha \times [1\ 0\ 0]^T = \left[0\ \frac{1}{\sqrt{2}}\ \frac{-1}{\sqrt{2}}\right]^T$.

- Thus, $\mathbf{R}_{\alpha\mathbf{x}}^{\text{II}} = \begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{-2}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{R}_Y(\arctan(\sqrt{2}))\mathbf{R}_X\left(\frac{\pi}{4}\right)$

$R_{\alpha x}$ Method 2: Contd



Relationship between \mathbf{R}^I and \mathbf{R}^{II}

- $\mathbf{R}_{\alpha X}^I$ and $\mathbf{R}_{\alpha X}^{II}$ are related by a rotation about \mathbf{X} . That is, $\mathbf{R}_{\alpha X}^I [\mathbf{R}_{\alpha X}^{II}]^{-1} = \mathbf{R}_X(\Delta)$ for some Δ .
- $\mathbf{R}_\alpha(\theta) = [\mathbf{R}_{\alpha X}^I]^{-1} \mathbf{R}_X(\theta) \mathbf{R}_{\alpha X}^I =$
 $[\mathbf{R}_X(\Delta) \mathbf{R}_{\alpha X}^{II}]^{-1} \mathbf{R}_X(\theta) [\mathbf{R}_X(\Delta) \mathbf{R}_{\alpha X}^{II}] =$
 $[\mathbf{R}_{\alpha X}^{II}]^{-1} [\mathbf{R}_X(-\Delta) \mathbf{R}_X(\theta) \mathbf{R}_X(\Delta)] \mathbf{R}_{\alpha X}^{II} =$
 $[\mathbf{R}_{\alpha X}^{II}]^{-1} \mathbf{R}_X(-\Delta + \theta + \Delta) \mathbf{R}_{\alpha X}^{II} =$
 $[\mathbf{R}_{\alpha X}^{II}]^{-1} \mathbf{R}_X(\theta) \mathbf{R}_{\alpha X}^{II}$
- In this example: $\mathbf{R}_{\alpha X}^I [\mathbf{R}_{\alpha X}^{II}]^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} = \mathbf{R}_X(-\frac{\pi}{2})$

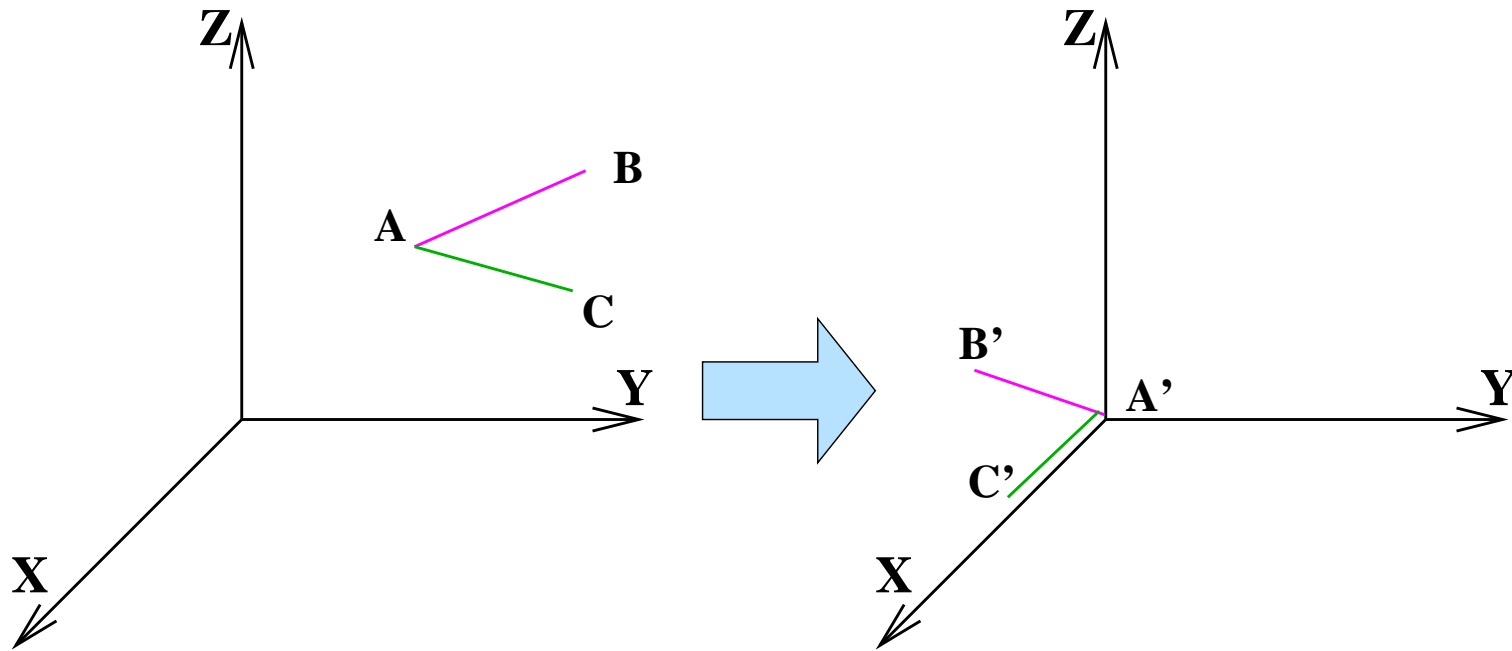
Rotation: Arbitrary Axis, Point

- An arbitrary axis may not pass through the origin.
- Translate by \mathbf{T} so that it passes through the origin.
- Apply \mathbf{R}_α .
- Translate back using \mathbf{T}^{-1} .
- Composite transformation: $\mathbf{T}^{-1} \mathbf{R}_\alpha \mathbf{T}$.

3D Transformations

- Many ways to *think about* a given transform.
- Ultimately, there is only one transform given the starting and ending configurations.
- Different methods let us analyze the problem from different perspectives.

Another Example



Working the Example

- Translate by $-\mathbf{A}$ to bring it to the origin.
- After the rotation, \mathbf{AC} sits on the X axis.
- The first row of the rotation matrix is $\mathbf{AC} / |\mathbf{AC}|$.
- The vector normal to the plane \mathbf{ABC} sits on the Y axis.
- The second row of the rotation matrix is the unit vector along $\mathbf{AB} \times \mathbf{AC} = (\mathbf{AB} \times \mathbf{AC}) / |\mathbf{AB} \times \mathbf{AC}|$.
- Third row is a cross product of the first two.

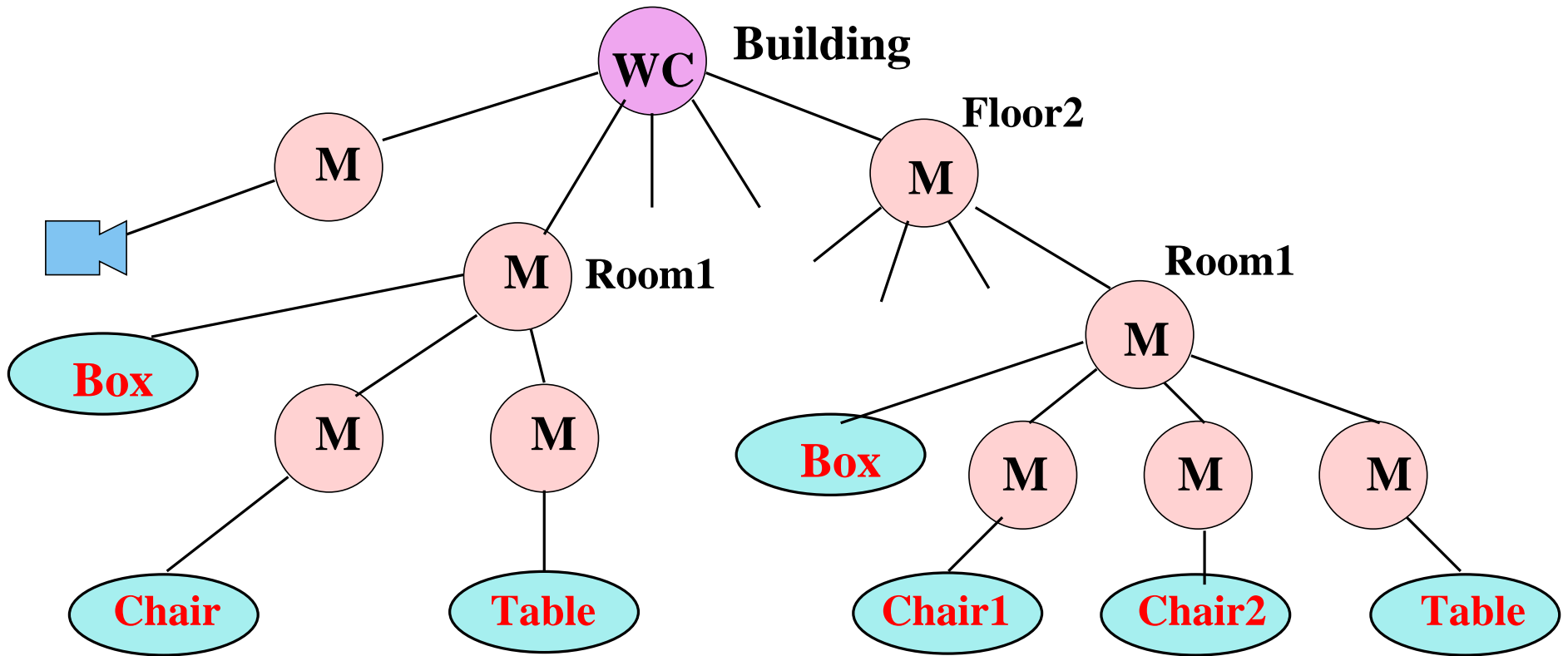
Transforming Lines

- A composite transformation can be seen as changing points in the coordinate system.
- These transformations preserve collinearity. Thus, points on a line remain on a (transformed) line.
- Take two points on the line, transform them, and compute the line between new points.
- Lines are defined as a join of 2 points or intersection of 2 planes in 3D. The same holds for transformed lines using transformed points or planes!

Transforming Planes

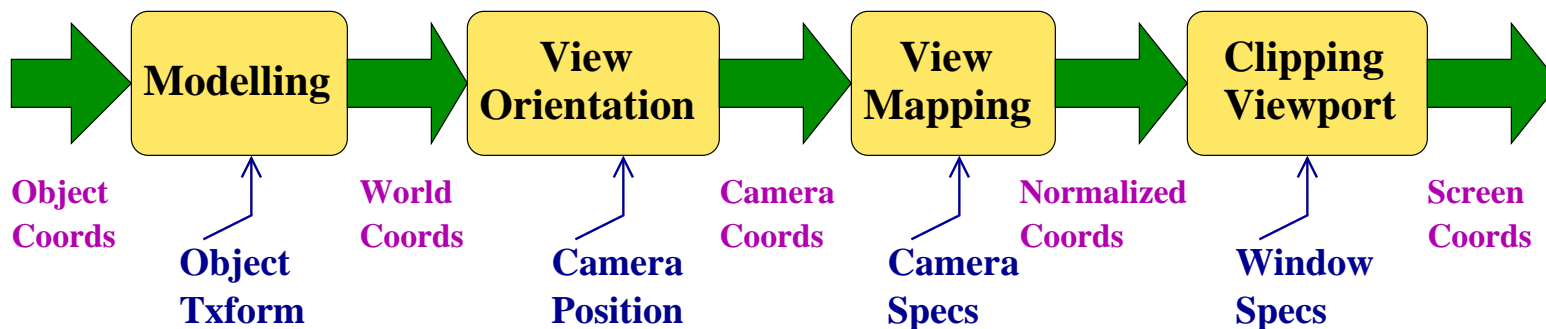
- A plane is defined by a 4-vector \mathbf{n} (called the **normal vector**) in homogeneous coordinates.
- The plane consists of points \mathbf{p} such that $\mathbf{n}^T \mathbf{p} = 0$.
- Let \mathbf{Q} transform \mathbf{n} when points are transformed by \mathbf{M} .
- Coplanarity is preserved: $(\mathbf{Q}\mathbf{n})^T \mathbf{M}\mathbf{p} = 0 = \mathbf{n}^T \mathbf{Q}^T \mathbf{M}\mathbf{p}$.
- True when $\mathbf{Q}^T \mathbf{M} = \mathbf{I}$, or $\mathbf{Q} = \mathbf{M}^{-T}$.
- \mathbf{Q} is the Matrix of cofactors of \mathbf{M} in the general case when \mathbf{M}^{-1} doesn't exist.

A Model

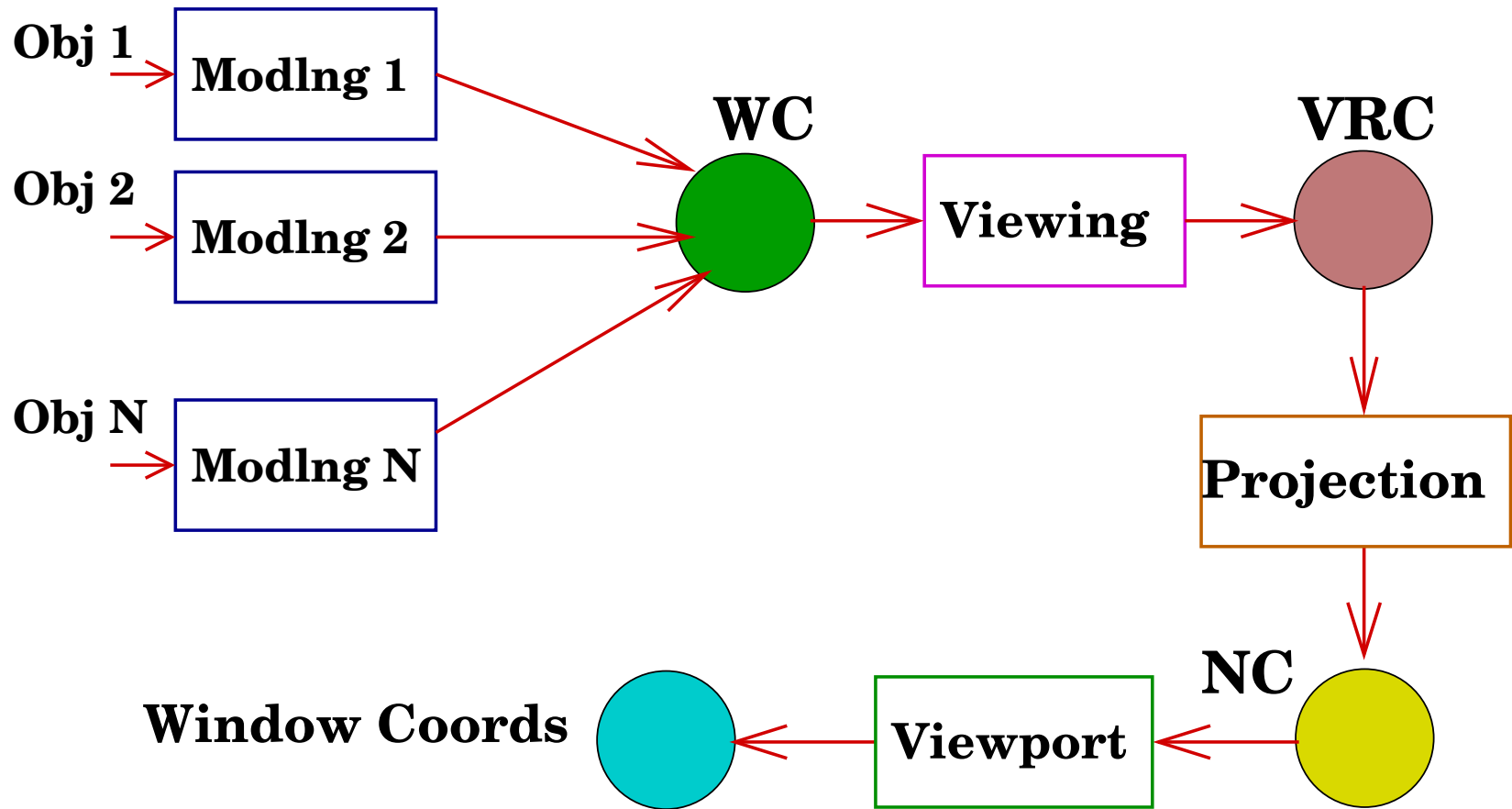


3D Graphics: Block Diagram

- Objects are specified in their own coordinate system and placed in the world coordinate frame.
- Camera is also placed in the world coordinate frame.
- Camera-to-world geometry is first projected to normalized coordinates and then to screen.



3D Graphics: Block Diagram



Different Coordinates

- **Object Reference:** Object is described in an internal coordinate frame called **ORC**.
- **World:** Common reference frame to describe different objects, called **WC**.
- **Camera/View Reference:** Describe with respect to the current camera position/orientation, called **VRC**.
- **Normalized Projection:** A standard space from which projection is easy, called **NPC**.
- **Screen:** Coordinates in the output device space.

Transformations

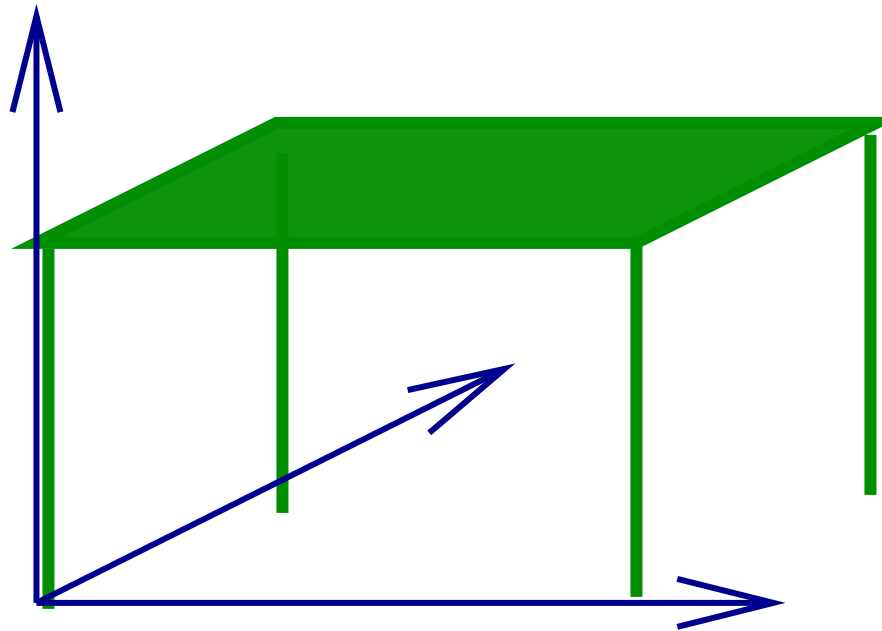
- **Modelling:** Convert from object coordinates to world coordinates (ORC to WC).
- **View Orientation or Viewing:** From world coordinates to camera coordinates (WC to VRC).
- Simple coordinate transformations.
- **View Mapping or Projection:** From VRC to Normalized Coordinates (NC).
- **Viewport:** From NC to window coordinates.

Modelling and Viewing

- Transform points from object coordinates (ORC) to world coordinates (WC) to camera coordinates (VRC).
- A series of transformations for each object or point.
- $P_{VRC} = V M P_{ORC}$.
- Points move to new coordinates in the same global coordinate system when going from right to left.
- Coordinate system undergoes transformation with points getting new interpretations when going from left to right.

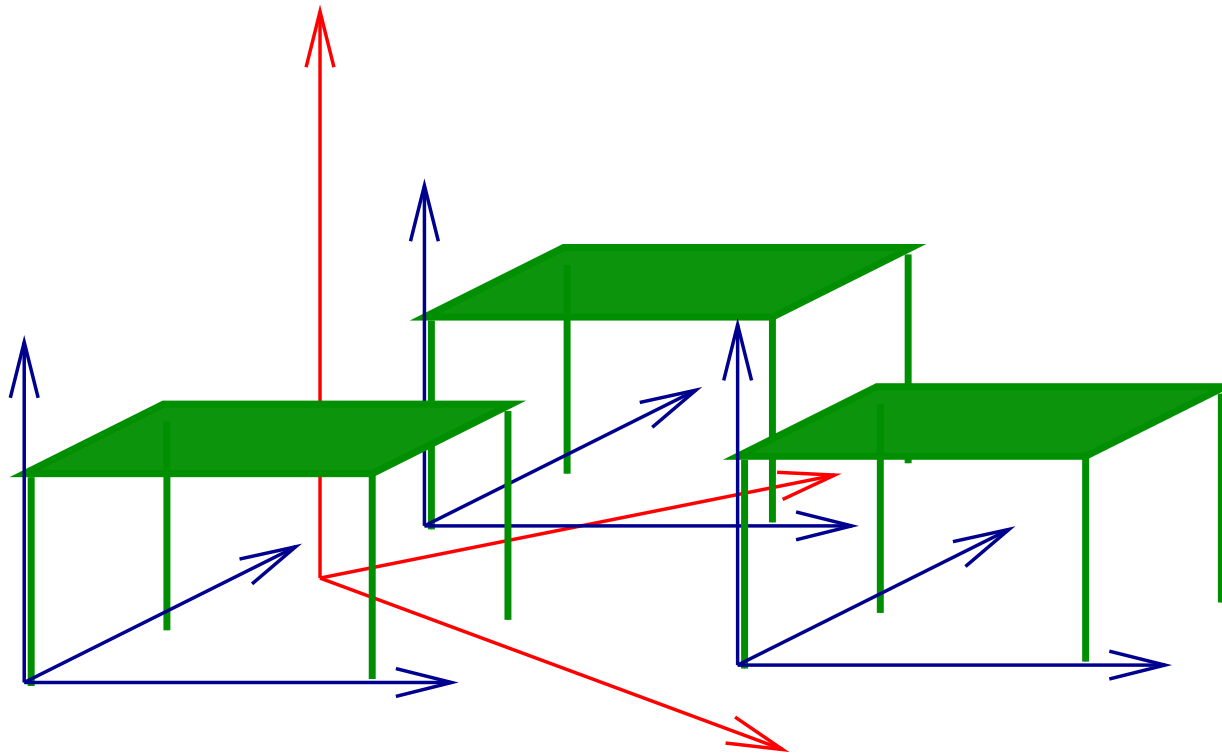
A Single Table

- A table defined in its own coordinate system.



Many Tables in a Room

- Place many tables in a world coordinate system.

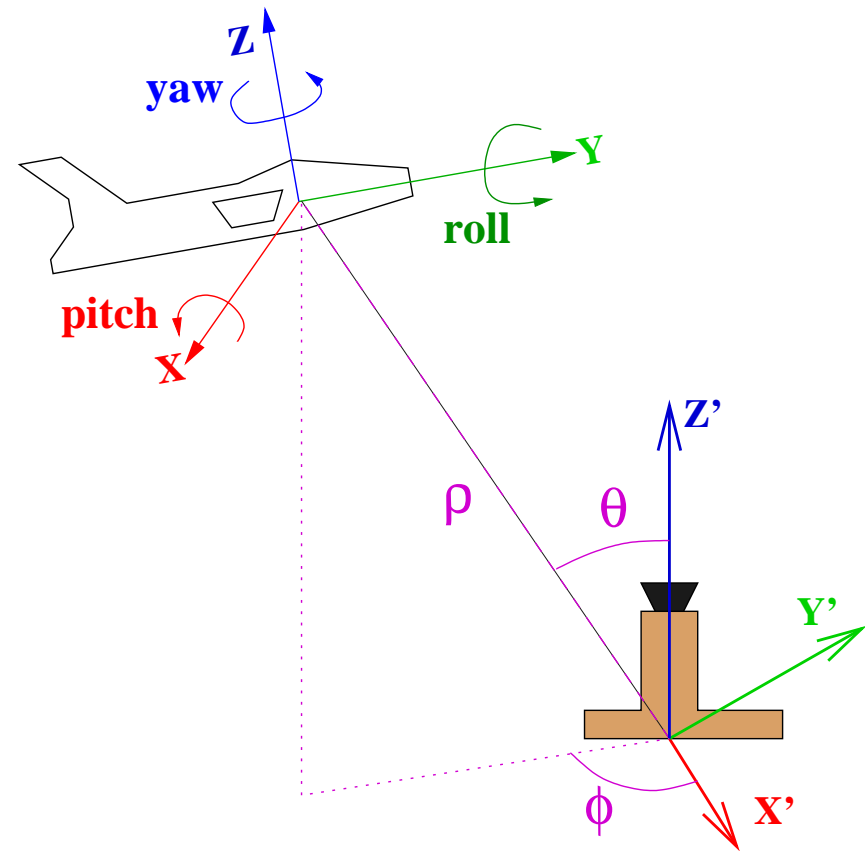


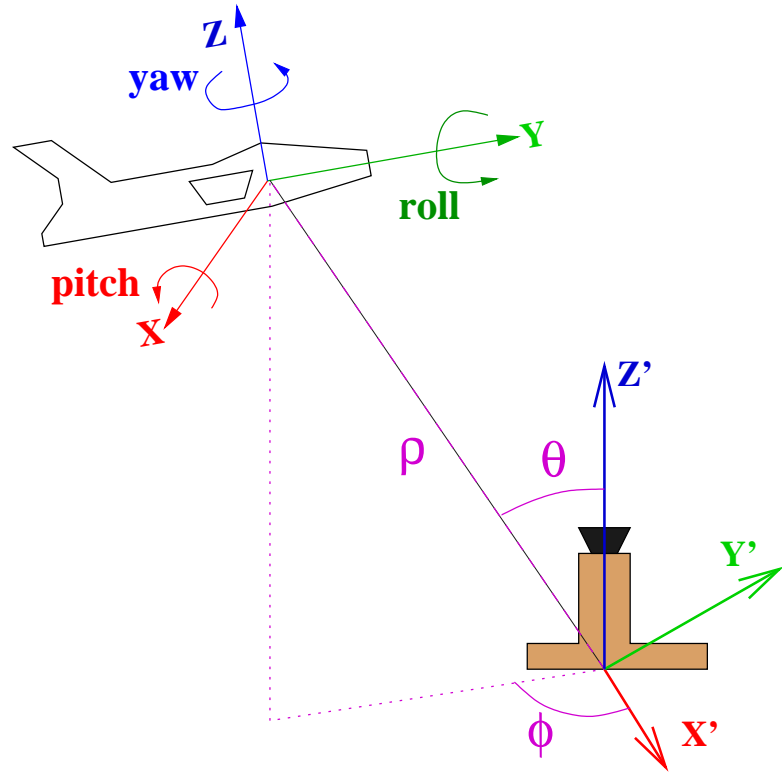
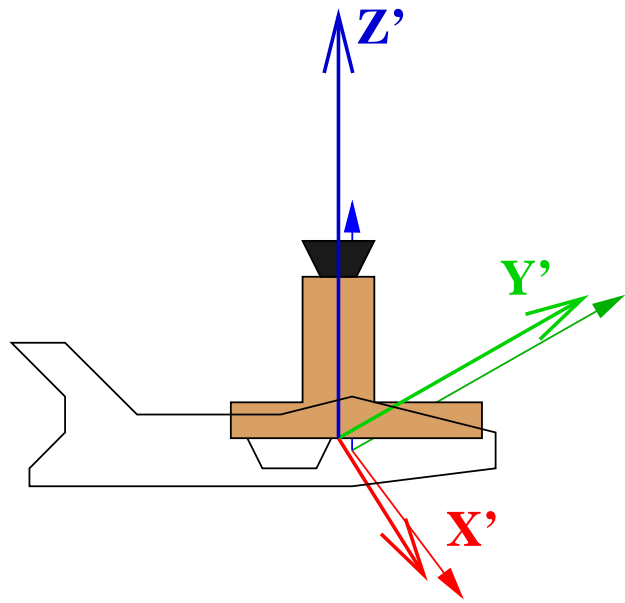
Modelling

- Goal: Transform object coordinates to world coordinates.
- Method: Place ORC fame in the world coordinate frame.
- A single transformation matrix or **modelling matrix** with translation, rotation, scaling.
- A unit cube at origin can generate any cuboid using translation/rotation/scaling.
- Different objects have different modelling matrices.

Example: Aircraft in a Polar World

- WC frame on ground, ORC frame on the aircraft.
- Controllers think in polar coordinates for position and roll-pitch-yaw for orientation.
- What are the modelling steps?





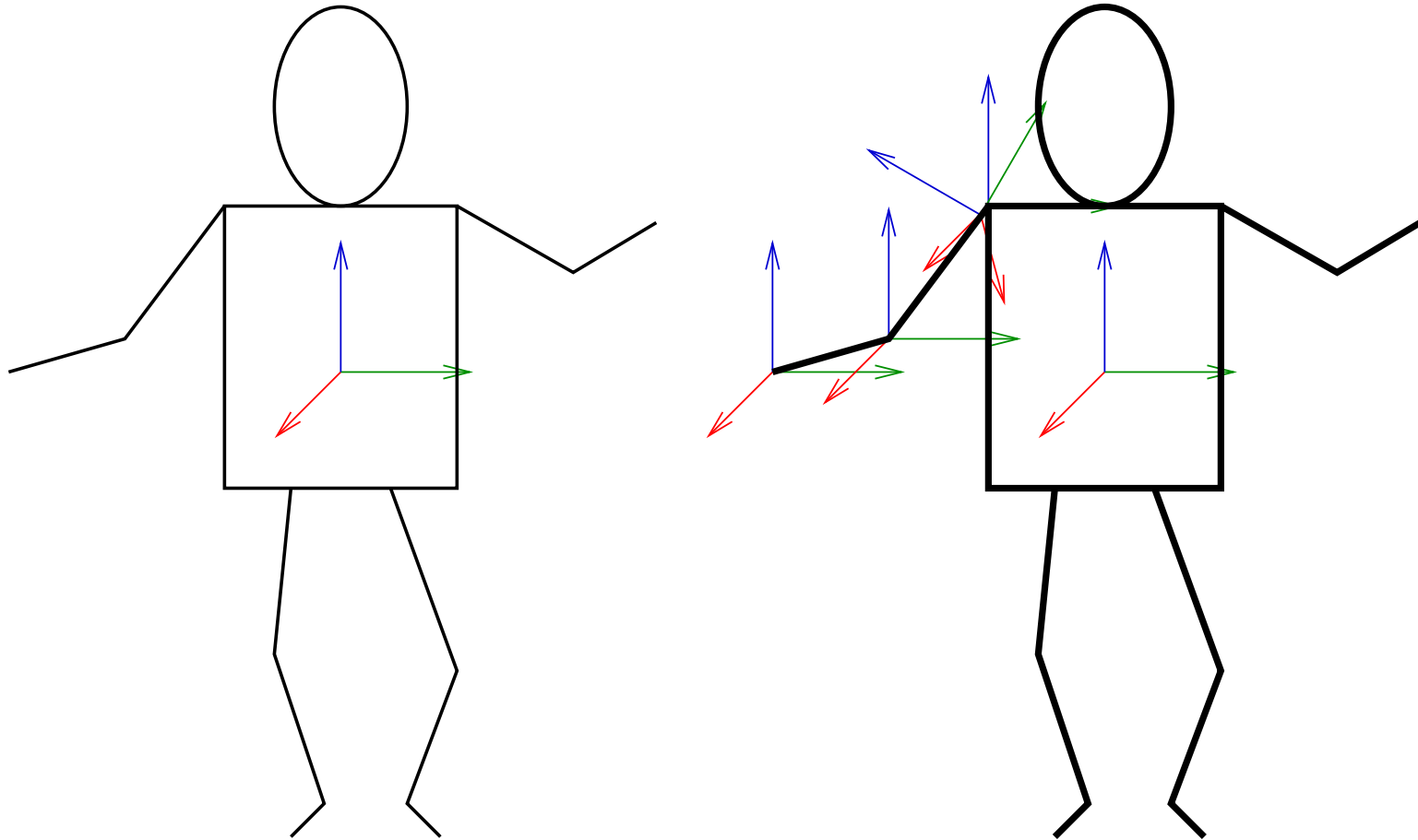
Aircraft in a Polar World

- Start with both axes aligned.
- Translate to the location given by (ρ, θ, ϕ) .
- Apply yaw, pitch, and roll in that order.
- Coordinate axes undergoing transformation!
- Net effect: $\mathbf{T}(\rho, \theta, \phi) \mathbf{R}_z(\mathbf{y}) \mathbf{R}_x(\mathbf{p}) \mathbf{R}_y(\mathbf{r})$.
- $\mathbf{T}(\rho, \theta, \phi) = \mathbf{R}_z(-\phi) \mathbf{R}_x(\theta) \mathbf{T}(\mathbf{0}, \mathbf{0}, \rho) \mathbf{R}_x(-\theta) \mathbf{R}_z(\phi)$.

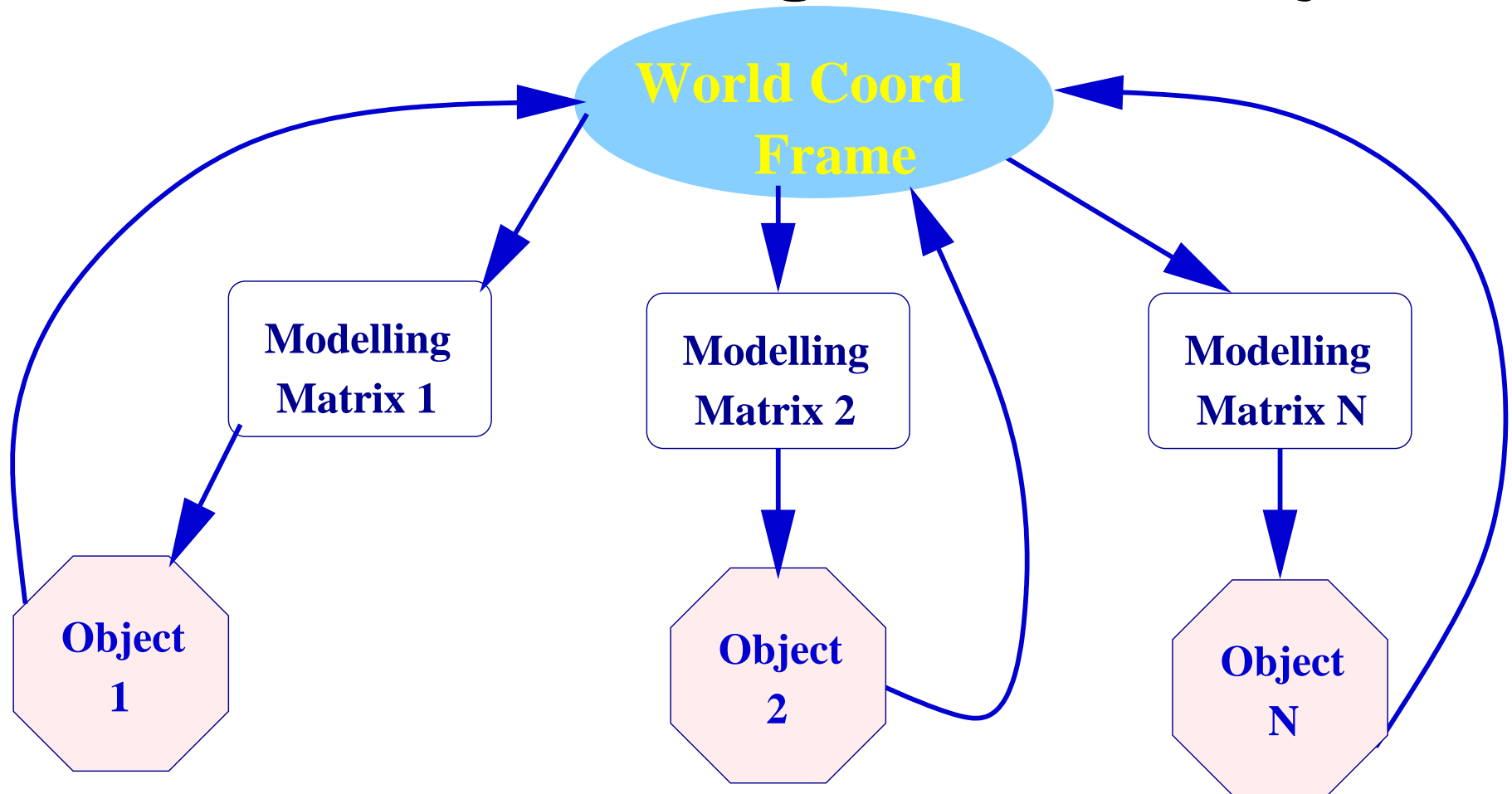
Hierarchy of Transformations

- A hierarchy of transformations needed to setup the world and the camera.
- A humanoid robot could have a coordinate frame on its body, another one on the shoulder, a third on the shoulder that moves with the upper arm, a fourth on the elbow, a fifth on the elbow that moves with the forearm, etc.
- Remember the wheel with an ant moving on its spoke!
- $M = T_1 T_2 T_3 \dots$ captures the composite transform as a shift in coordinate frames.

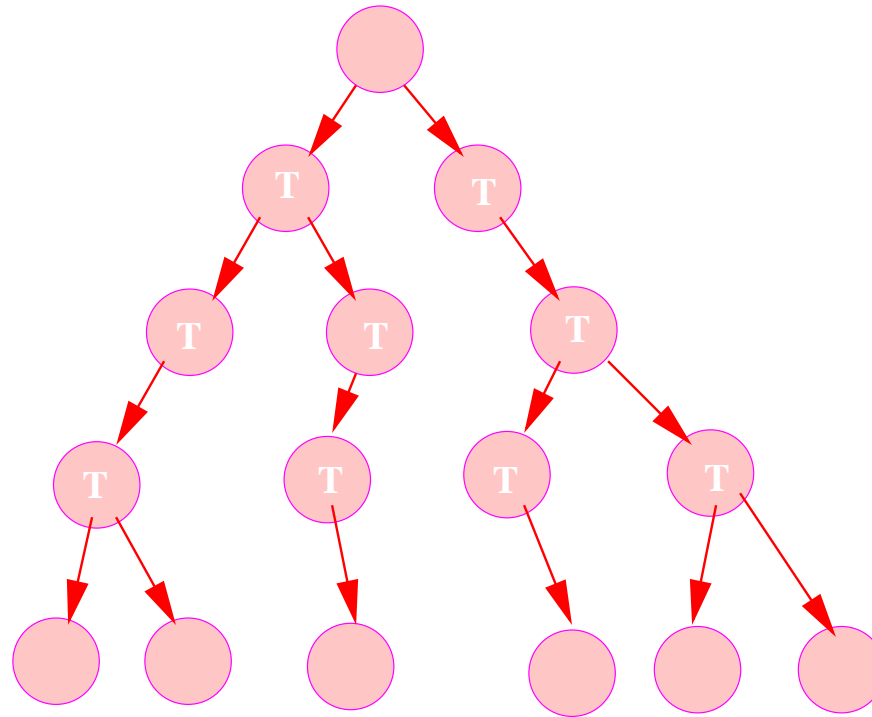
Humanoid Robot



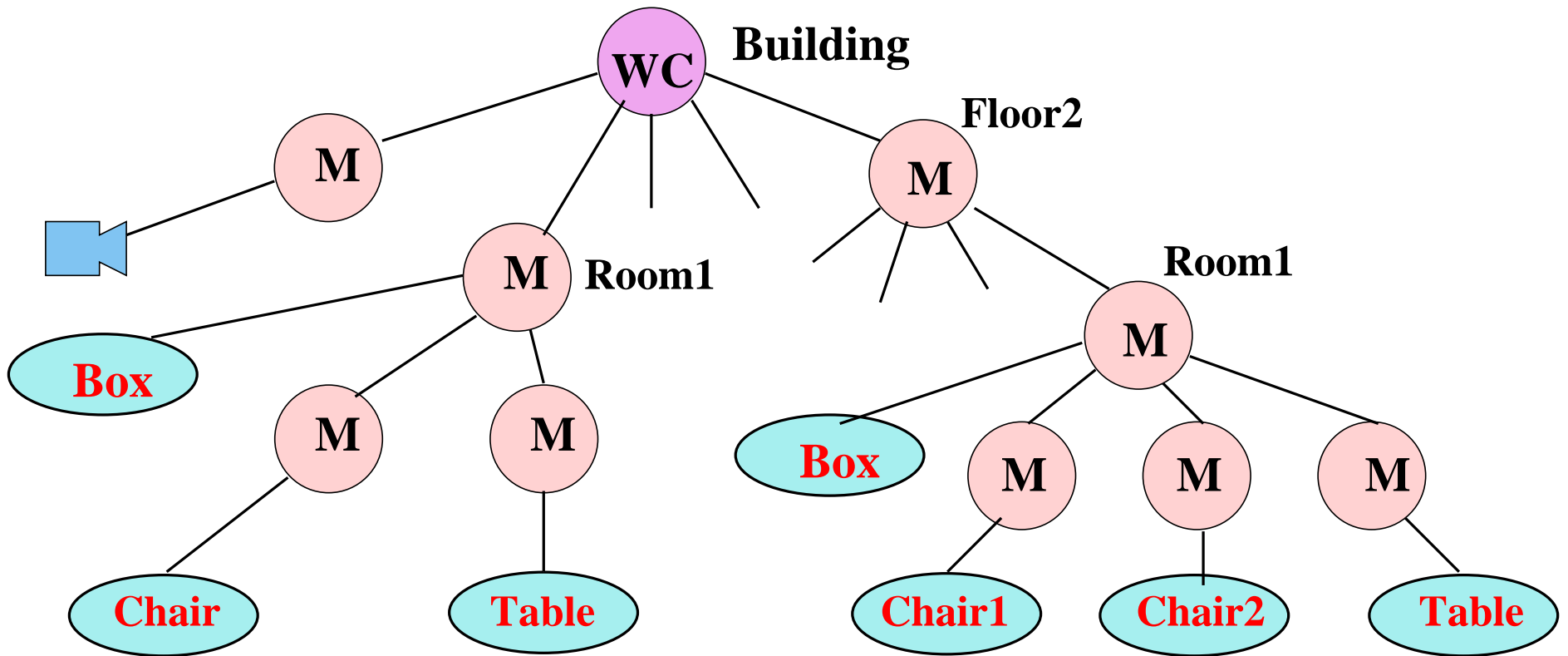
Modelling Different Objects



Scene Graph



- Objects organized hierarchically with transforms.



Modelling in OpenGL

- OpenGL has a single ModelView matrix M that transforms points given by `glVertex`.
- Right multiplication or Mp is applied.
- `glTranslate`, `glRotate` also right-multiply M .
- `glTranslate` command is same as: $M = M T$.
Similarly for others.
- Modelling commands are: `glTranslate`, `glRotate`, `glScale`, ...

- $M = T_1 R_2 S_3 R_4$ translates to the program fragment:

```
glTranslate()  
glRotate()  
glScale()  
glRotate()
```

Placing the Aircraft in OpenGL

```
// You are at the World Coordinate frame
glRotate(- $\phi$ , 0, 0, 1); //  $\mathbf{R}_Z(-\phi)$ 
glRotate( $\theta$ , 0, 1, 0); //  $\mathbf{R}_Y(\theta)$ 
glTranslate(0, 0,  $\rho$ ); //  $\mathbf{T}(\mathbf{0}, \mathbf{0}, \rho)$ 
glRotate(- $\theta$ , 0, 1, 0); //  $\mathbf{R}_Y(-\theta)$ 
glRotate( $\phi$ , 0, 0, 1); //  $\mathbf{R}_Z(\phi)$ 
glRotate(yaw, 0, 0, 1); //  $R_Z(y)$ 
glRotate(pitch, 1, 0, 0); //  $R_X(p)$ 
glRotate(roll, 0, 1, 0); //  $R_Y(r)$ 

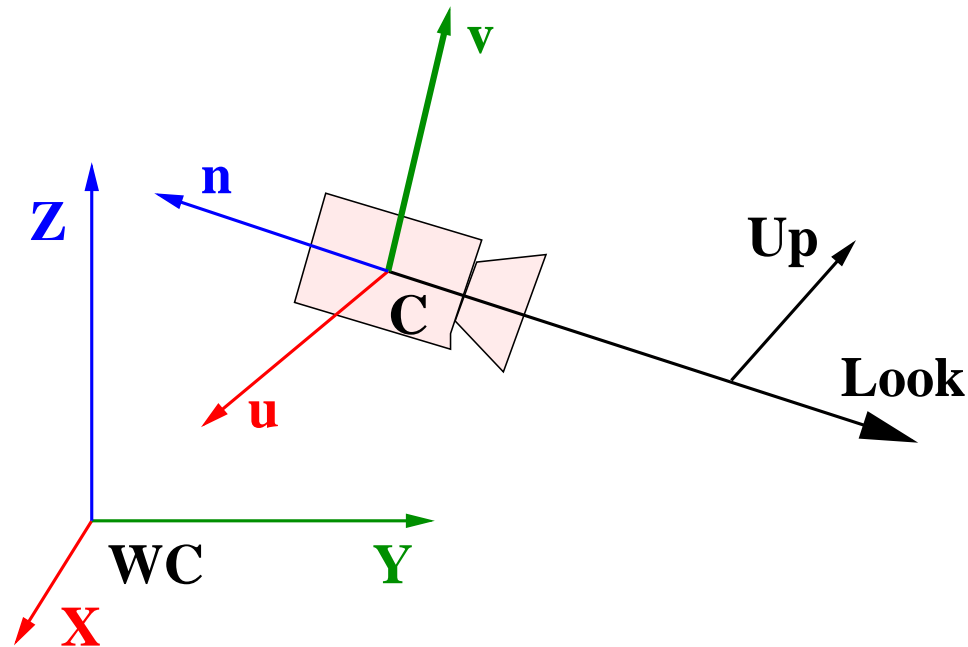
// Start drawing aircraft parts
glBegin(GL_TRIANGLES);
... // Draw aircraft triangle by triangle!
```

View Orientation or Viewing

- Placing the camera in the world and orienting it right.
- Has 6 degrees of freedom: 3 for position and 3 for orientation.
- Goal: Transform points expressed in WC to VRC.
- Viewing Transformation can be specified in many ways.
- Commonly using: **Camera location**, **Look point**, and **Up direction**.

Viewing Specification

- Camera center specified in world coordinates.
- Look point or vector and Up vector known in WC.



Transformation Steps

How do we align WC to VRC?

- Translate to $\mathbf{C} = (x, y, z)$.
- Rotate to align Z-axis to -(Look Vector).
- Rotate to align Y-axis to Up.
- Translation is easy. How do we get the rotation matrix?
- Remember columns of the matrix give directions **to** which the axes rotate!!

Rotation

- Let $\bar{\mathbf{l}} = \bar{\mathbf{L}}/|\bar{\mathbf{L}}|$ and $\bar{\mathbf{t}} = \bar{\mathbf{U}}/|\bar{\mathbf{U}}|$ be the unit vectors in those directions.
- Third column of the matrix: $\bar{\mathbf{n}} = -\bar{\mathbf{l}}$.
- Up vector needn't be orthogonal to the look vector. The $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$ vectors define the Up plane.
- First column: $\bar{\mathbf{u}} = \bar{\mathbf{t}} \times \bar{\mathbf{n}}/|\bar{\mathbf{t}} \times \bar{\mathbf{n}}|$
- Second column: $\bar{\mathbf{v}} = \bar{\mathbf{n}} \times \bar{\mathbf{u}}$.

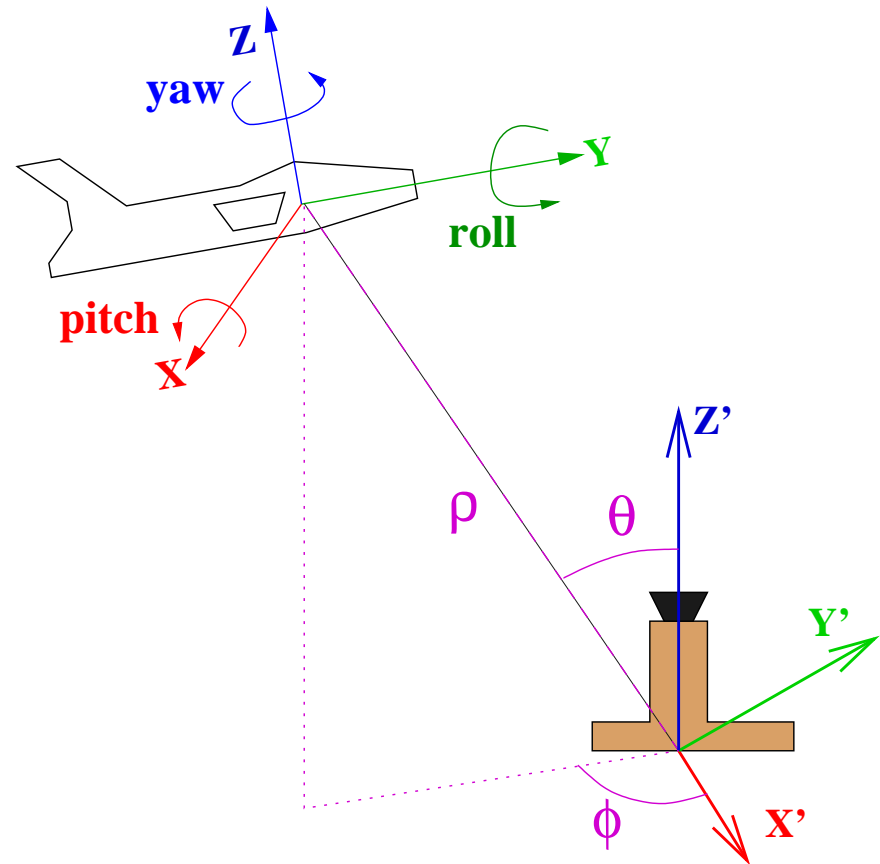
View Orientation Transformation

$$\mathbf{V}' = \begin{bmatrix} & x \\ \mathbf{I} & y \\ & z \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} & & & 0 \\ \bar{\mathbf{u}} & \bar{\mathbf{v}} & \bar{\mathbf{n}} & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & x \\ \bar{\mathbf{u}} & \bar{\mathbf{v}} & \bar{\mathbf{n}} & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- We have achieved: $\mathbf{P}_{WC} = \mathbf{V} \mathbf{P}_{VRC}$.
- But, we needed the reverse!
- Viewing transform: $\mathbf{V} = \mathbf{V}'^{-1} = \mathbf{R}^\top \mathbf{T}(-\mathbf{C})$.

Viewing from an Aircraft in Polar World

- Need to give the pilot's view from aircraft.
- What are the viewing steps?



Aircraft in Polar World: Viewing

- Start with both axes aligned.
- Translate to the location given by (ρ, θ, ϕ) .
- Apply yaw, pitch, and roll in that order.
- Coordinate axes undergoing transformation!
- Net effect: $\mathbf{T}(\rho, \theta, \phi) \mathbf{R}_z(\mathbf{y}) \mathbf{R}_x(\mathbf{p}) \mathbf{R}_y(\mathbf{r})$.
- But, we want camera coordinates from world coordinates!
- Viewing transform: $\mathbf{R}_y(-\mathbf{r}) \mathbf{R}_x(-\mathbf{p}) \mathbf{R}_z(-\mathbf{y}) \mathbf{T}^{-1}(\rho, \theta, \phi)$.
- Inverse of modelling or placing aircraft in WC.

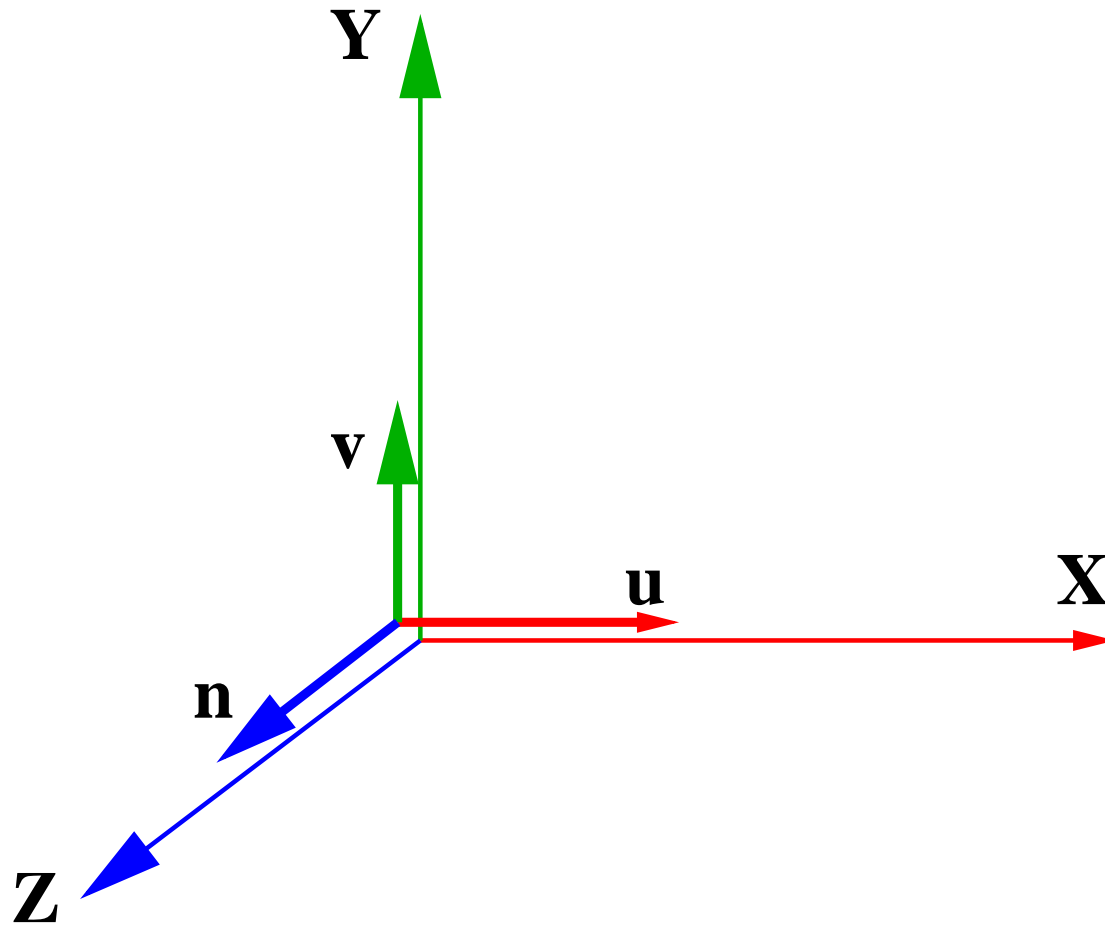
Modelling and Viewing in OpenGL

- Modelling and Viewing are not truly independent.
- What ultimately matters is only the **relative geometry** between the camera and the object(s).
- OpenGL uses only a single *ModelView* matrix to represent the relative geometry.
- Other graphics toolkits may use separate matrices for conceptual clarity.

Setting up Objects and Camera

- First arrange all objects in WC. (Modelling).
- Next place the camera in WC. (Viewing)
- Thus, viewing commands are issued first, followed by modelling commands!
- In OpenGL, commands for both are same:
glRotate(), *glTranslate()*, *glScale()*.
- Camera at origin, looking along $-Z$ with Y axis as the up vector on start when ModelView matrix is identity.

Identity as ModelView Matrix

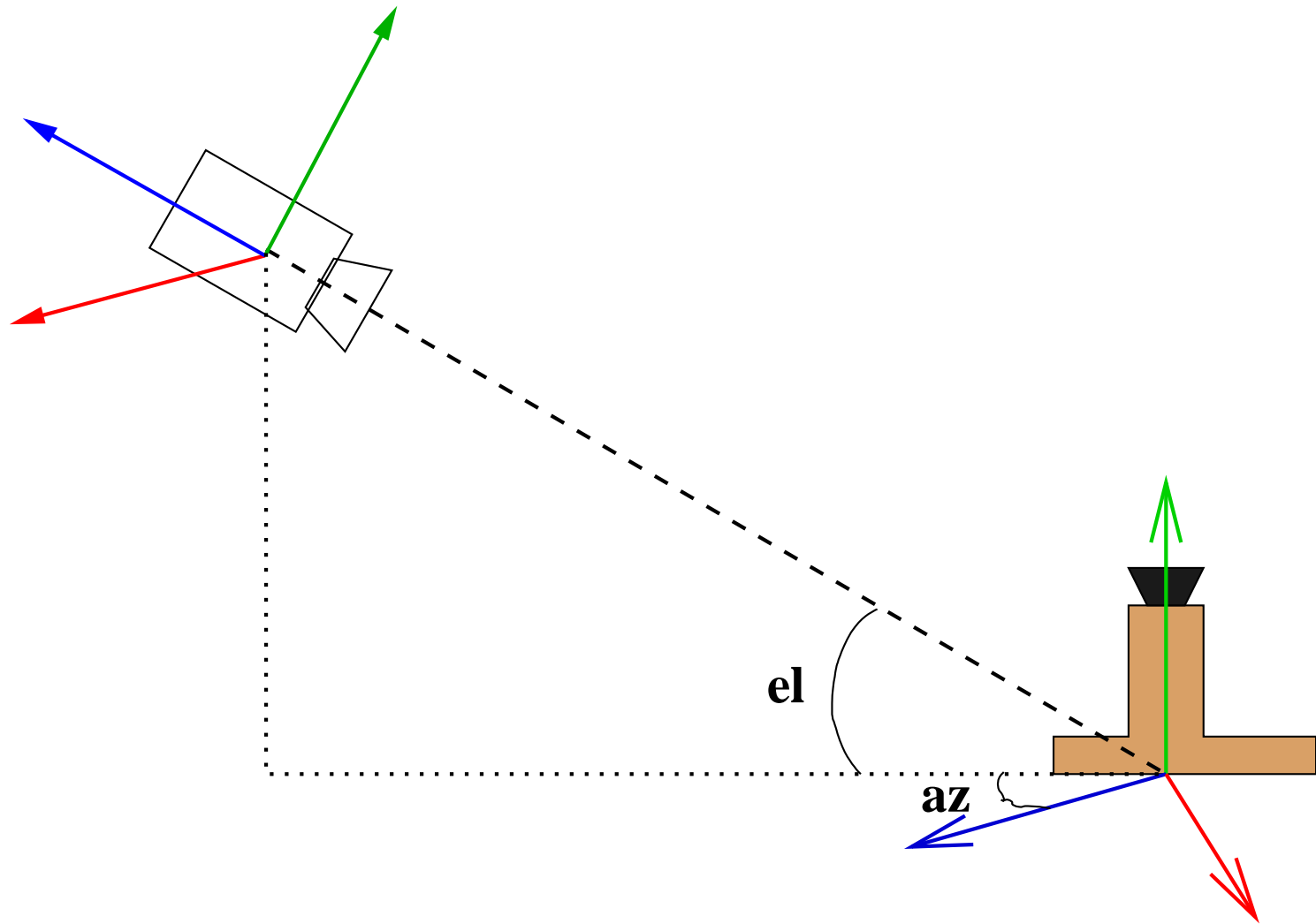


Structure of an OpenGL Program

```
// Camera is given by Pos & Orientation in WC.  
glLoadIdentity();           // Default starting point  
glRotate(-Orient);         // Orient to be parallel to WC!  
glTranslate(-Pos);        // Move WC away from VRC  
// WC is set. Model each object with it as reference.  
glPushMatrix();           // Save for later use  
glRotate(r1);              // Start transformation1  
glTranslate(t1);           // and so on  
... Object Geometry ...   // Actual polygons/objects  
glPopMatrix()              // End of object1. Go to next
```

Polar View

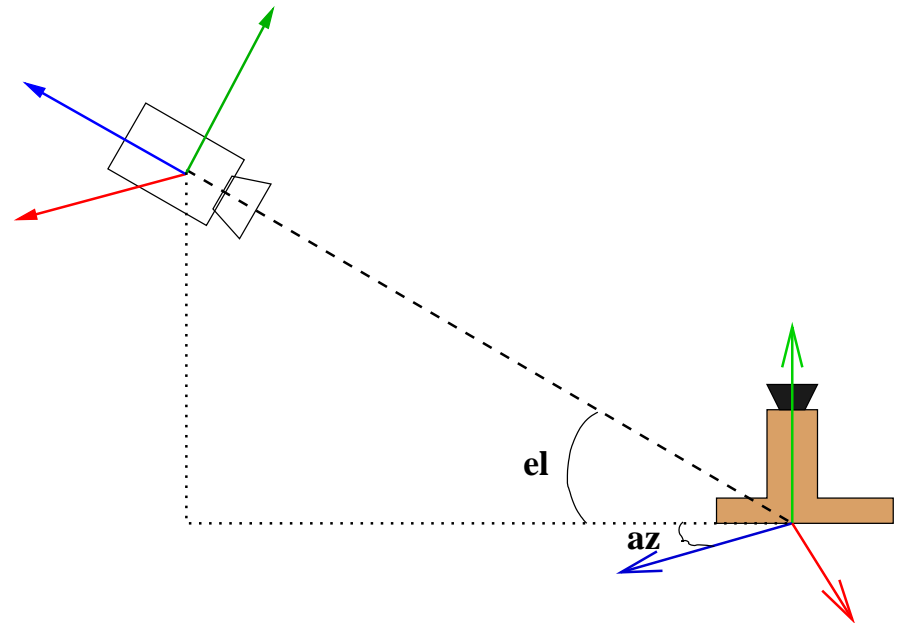
- A camera always looks at an object. Its position is specified using polar coordinates.
- Distance d , azimuth az measured from Z-axis $([0, 2\pi])$, twist tw about camera's own axis $([0, 2\pi])$, and elevation el measured from the horizontal (X-Z) plane $([-\frac{\pi}{2}, +\frac{\pi}{2}])$.
- How do we align WC to VRC? (We want the inverse!)



Polar View: OpenGL code

- (i) Rotate by $-az$ about Y
- (ii) Rotate by $-el$ about X
- (iii) Rotate by tw about Z
- (iv) Translate by d along Z

```
glTranslate(0, 0, -d);  
glRotate(-tw, 0, 0, 1);  
glRotate(el, 1, 0, 0);  
glRotate(az, 0, 1, 0);
```



Modelling & Viewing: Summary

- Place objects in the world coordinate frame
- Place camera in the world coordinate frame
- Can compute object points in camera coordinate frame
- $P_{VRC} = V \cdot M \cdot P_{ORC}$

Projections

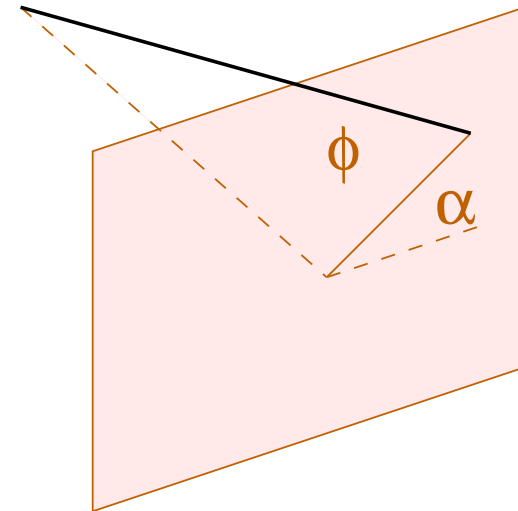
- Projection involves *projectors* starting from 3D points and hitting the 2D *projection plane*, forming the *image* of the point.
- Two types of projections.
- **Parallel projection**: Projectors are parallel to each other, all have the same *direction of projection*.
- **Perspective projection**: All projects pass through a point in space called the *centre of projection (COP)*.

Parallel Projections

- **Orthographic:** Projection plane is perpendicular to the direction of projection.
 - If direction of projection parallel to the axes:
plan, elevation, side elevation.
 - If DoP $(\pm 1, \pm 1, \pm 1)$: *isometric* projection.
- **Oblique:** Otherwise.
 - *Cabinet* when projectors make 45 degrees with the projection plane.
 - *Cavalier* when they make $\arctan(2)$ degrees with the projection plane.

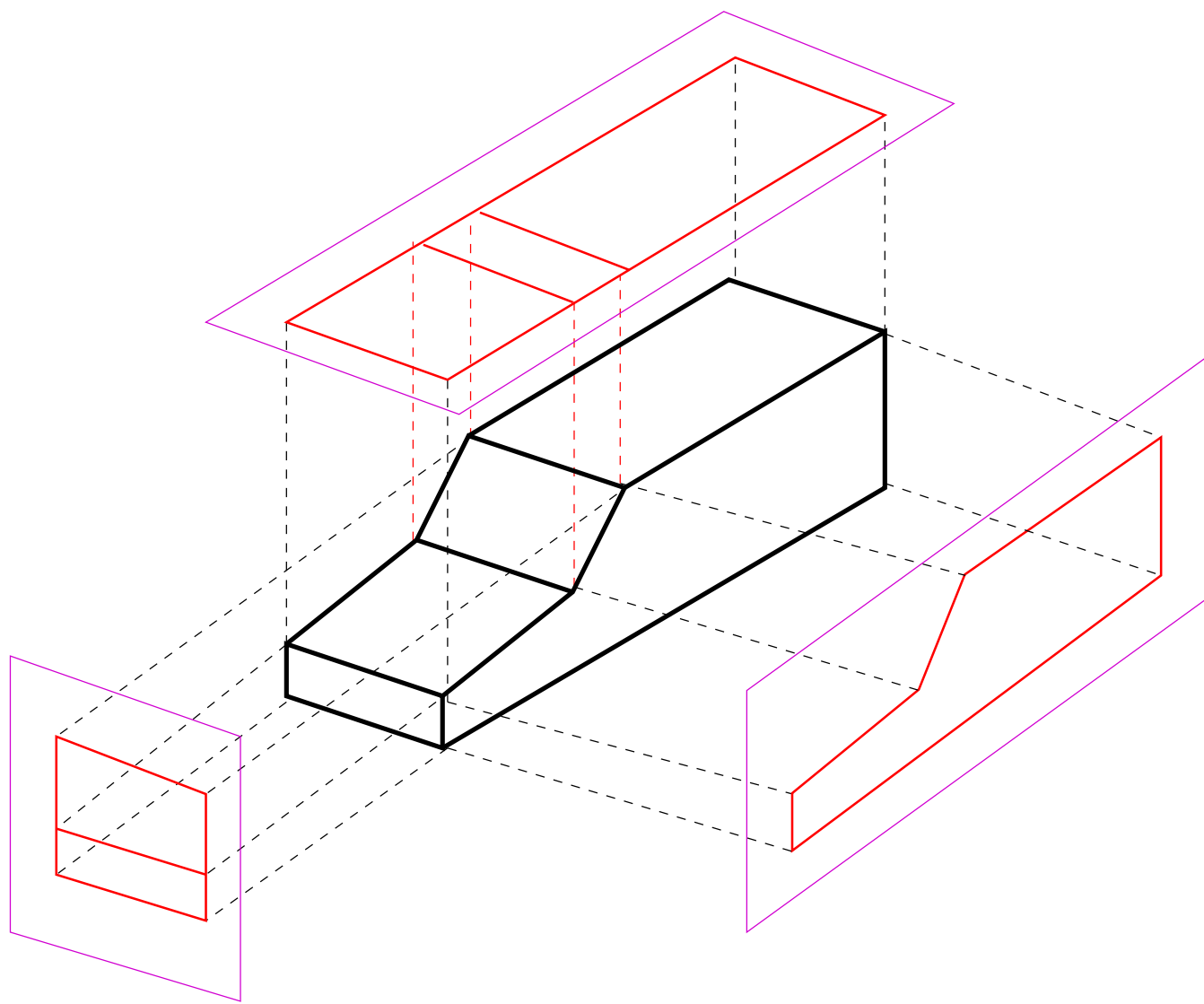
Oblique Projections

- Projector angle ϕ and angle with horizontal α .
- Cabinet: Length along the depth axis preserved.
- Cavalier: Length along depth axis halved. More realistic.
- α is between 30 and 45 degrees.
- Orthographic when $\phi = 90$.



Orthographic Projections

- Lengths parallel to the projection plane are preserved.
- Only direction of projection matters; distance from the point to the projection plane doesn't.
- Good approximation for a camera with a long focal length. (Orthographic with uniform scaling).
- Plan, elevation, side views etc.



Orthographic Projection Equation

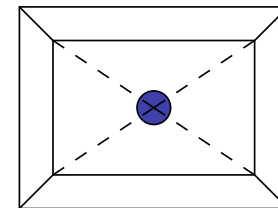
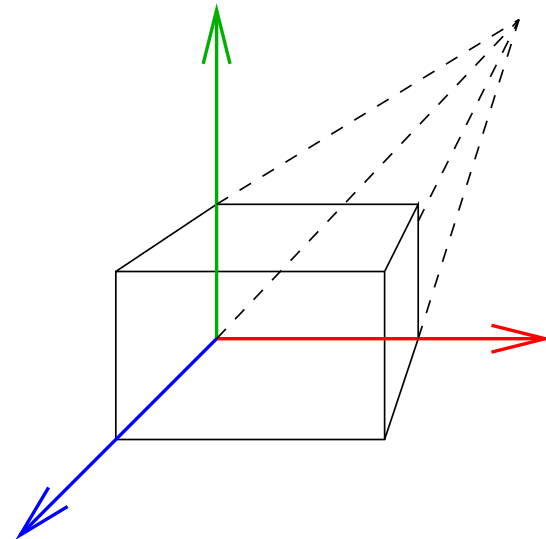
- Can be expressed as a matrix equation:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- If uniform scaling is involved, the top two 1's should be the scale factor.

Perspective Projections

- Can be characterized by the number of **vanishing points**. (projections of points at infinity).
- Depends on the number of axes the projection plane intersects.
- 1-point, 2-point, and 3-point perspective projections.



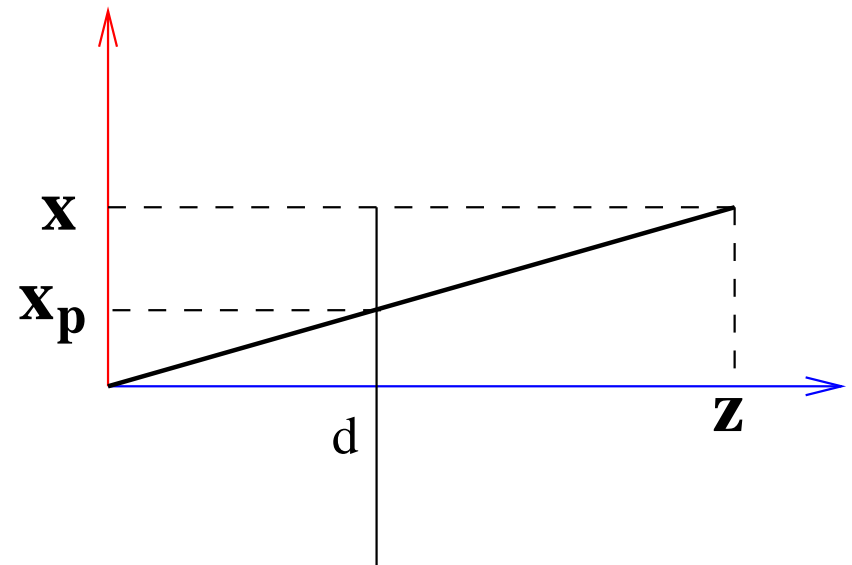
Geometry of Perspective Projection

- $\frac{x_p}{d} = \frac{x}{z}, \quad \frac{y_p}{d} = \frac{y}{z}, \quad z = d.$

- In matrix form,

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- Coordinates scaled down proportional to the depth or z values.

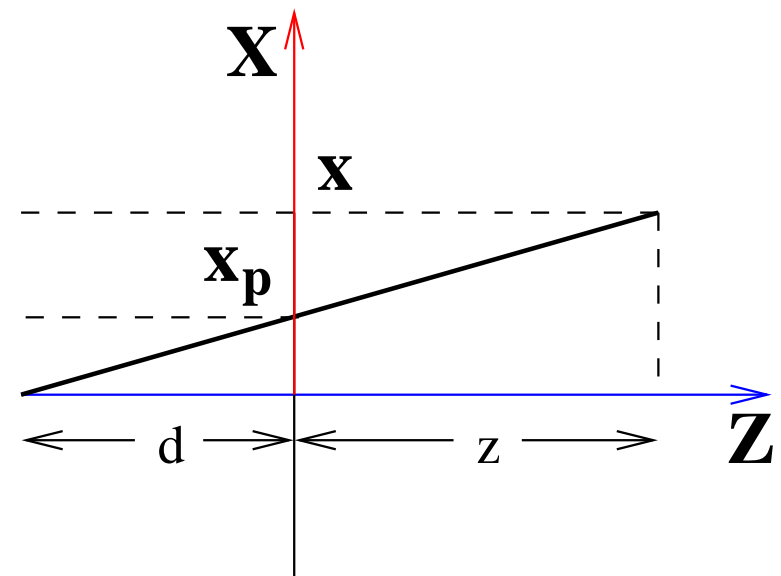


Another View

- Shift origin to lie on the projection plane, CoP at $(0, 0, -d)$, we get the matrix:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- Orthographic Projection matrix is a special case when $d \rightarrow \infty$.



$$x_p = \frac{xd}{z+d} = \frac{x}{1+\frac{z}{d}}, \quad z_p = 0$$

Projections: Summary

Perspective

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$d < \infty$$

Orthographic

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

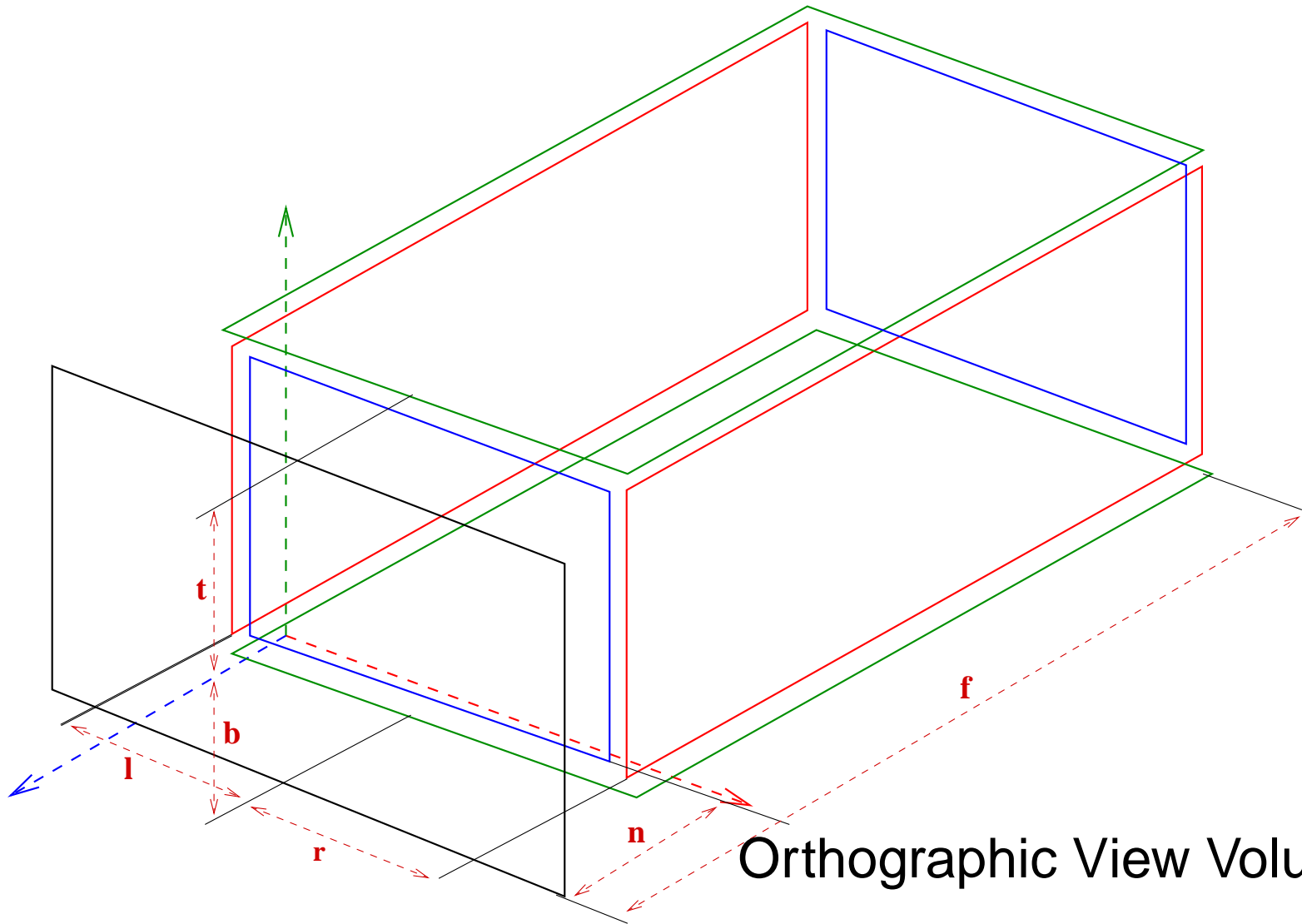
$$d = \infty$$

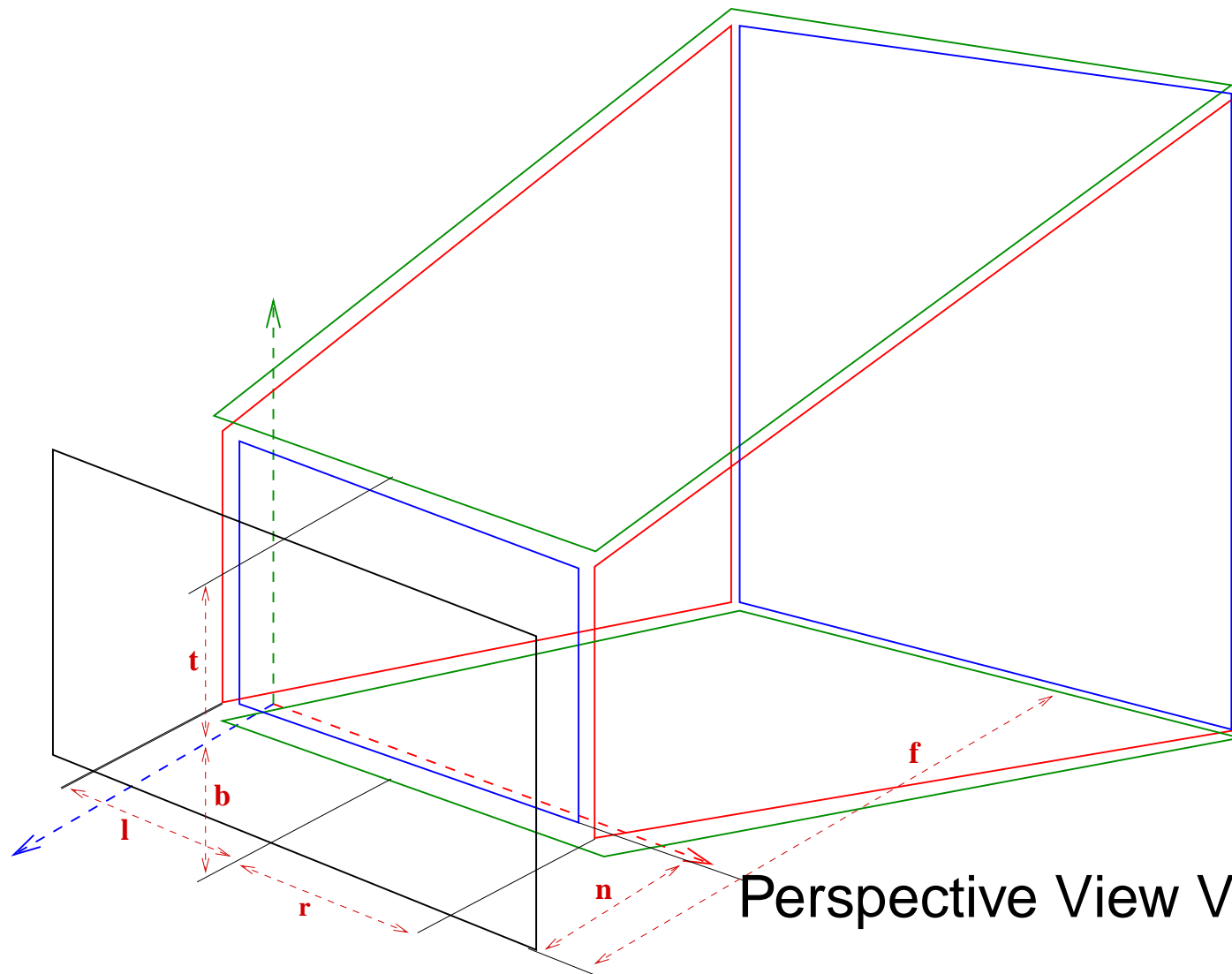
Volume of Visibility

- Cameras have finite fields of view in horizontal and vertical directions.
- What is the shape of its visible space?
- A cylinder for orthographic projections and a cone starting from the CoP for perspective may seem natural.
- Mathematics is difficult for cones; rectangular structures are easier!
- **View Volume:** The volume of potentially visible space.

View Volume

- View volume is a cube for orthographic cameras and a (truncated) pyramid for perspective projections.
- 4 planes (**left, right, top, bottom**) define the view volume.
- Graphics cameras use 2 additional planes to limit visibility: **near & far!**
- Planes are specified in VRC; they move with the camera.





Perspective View Volume

What About the Focal Length?

- An ideal pin-hole camera has the whole world in focus.
- Finite focal-length lenses introduce the effect of focus in real cameras.
- Even for them, the **depth of field** (region in focus) increases as the f-stop increases or the aperture gets smaller.
- Computer Graphics simulates ideal pin-hole cameras.
- Depth of field can be simulated by intentional blurring.

View Volume Specification

- View volume is specified by 6 planes:
left, right, top, bottom, near, far.
- All values are given in camera coordinates.
But, near, far values are positive.
- Needn't be symmetric!
- Can image the world sector by sector using both orthographic and perspective cameras.

Alternate Specification

- Symmetric view volumes: horizontal and vertical fields of view θ_h , θ_v

- For symmetric perspective view volumes:

$$\tan \frac{\theta_h}{2} = \dots?$$

$$\tan \frac{\theta_v}{2} = \dots?$$

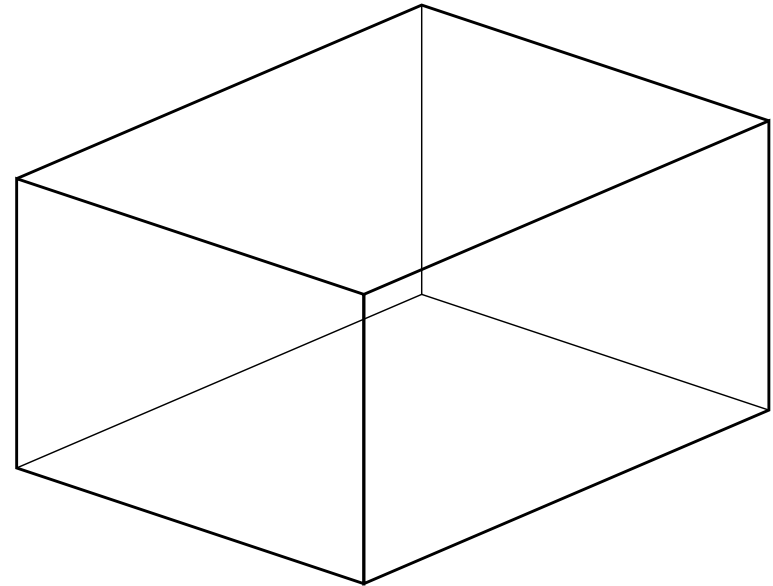
- Aspect ratio: **top / right**.

Canonical View Volume

- Projection is not performed right away; instead, map the view volume to a cube of fixed dimensions, called the **canonical view volume** or a standard view volume
- This is achieved by the **normalizing transformation**.
- Why?
 - Easier to eliminate objects outside the view volume.
 - Orthographic & perspective aren't different.
 - The z -coordinates not thrown away. (Used later!)

Canonical View Volume: Dimensions

- PHIGS: $-1 \leq x \leq 1$,
 $-1 \leq y \leq 1$, $0 \leq z \leq -1$.
- OpenGL: $-1 \leq x \leq 1$,
 $-1 \leq y \leq 1$, $-1 \leq z \leq 1$.
- Canonical view volume is the Orthographic View Volume (with scaling along x and y).



Orthographic Normalizing Matrix

- Lengths (**right - left**), (**top - bottom**) and (**far - near**) scaled to 2.
- Shift origin so as to range from -1 to +1.
- Matrix??

- Matrix:

$$\begin{bmatrix} \frac{2}{\text{right}-\text{left}} & 0 & 0 & -\frac{\text{right}+\text{left}}{\text{right}-\text{left}} \\ 0 & \frac{2}{\text{top}-\text{bottom}} & 0 & -\frac{\text{top}+\text{bottom}}{\text{top}-\text{bottom}} \\ 0 & 0 & \frac{2}{\text{far}-\text{near}} & \frac{\text{far}+\text{near}}{\text{far}-\text{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

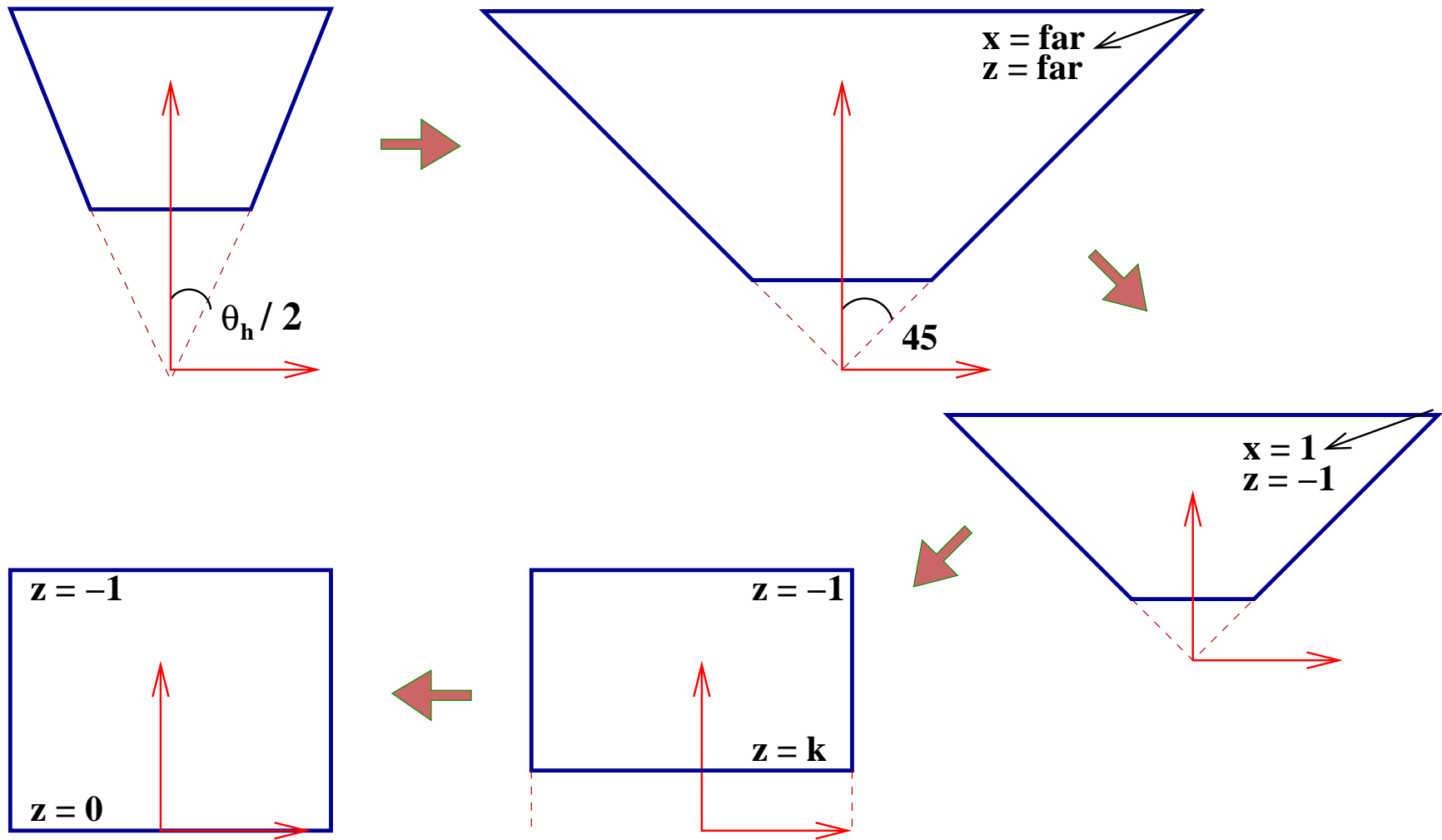
- $(0, 0, -n)$ maps to $+1$ and $(0, 0, -f)$ maps to -1 .
- Drop z and use the (x, y) coordinates as the (normalized) window coordinates.

Perspective Normalizing Matrix

- General case: $(left, bottom, -near) \rightarrow (-1, -1, 1)$,
 $(l, t, -n) \rightarrow (-1, 1, 1)$, $(r, t, -n) \rightarrow (1, 1, 1)$, $(rf/n, bf/n, -f) \rightarrow (1, -1, -1)$, $(lf/n, tf/n, -f) \rightarrow (-1, 1, -1)$, etc.
- Each matching point gives 3 equations in x, y, z in terms of X, Y, Z and the 16 unknowns m_{ij} . (Only 15 unknowns upto scale!)
- Can solve for them given 5 point matchings. We have 8, hence easy solution!
- Let us look at the symmetric projecting matrix in detail.

Symmetric Perspective Proj Matrix

- More complicated than orthographic case, as a frustum has to be mapped to a cube. Do it in steps.
- First, scale the horizontal and vertical extents so that the vertical and horizontal fields of view are 90 degrees.
- A scaling transformation with $s_x = ??$, $s_y = ??$, $s_z = 1$
- View volume is almost right except for a uniform scale.
- Next, scale uniformly so that the far plane is at -1. We will also have $-1 \leq x, y \leq 1$ at the far plane after this.
- $s_x = s_y = s_z = ??$



- M_1 with $s_x = \cot \frac{\theta_h}{2}$, $s_y = \cot \frac{\theta_v}{2}$, $s_z = 1$
- M_2 with $s_x = s_y = s_z = \frac{1}{\text{far}}$

- $M_2 M_1 = \begin{bmatrix} \frac{\cot \theta_h/2}{\text{far}} & 0 & 0 & 0 \\ 0 & \frac{\cot \theta_v/2}{\text{far}} & 0 & 0 \\ 0 & 0 & \frac{1}{\text{far}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- The near plane is now at $\mathbf{k} = \frac{\text{near}}{\text{far}}$.
- View volume fits into the canonical view volume, but is still a frustum!

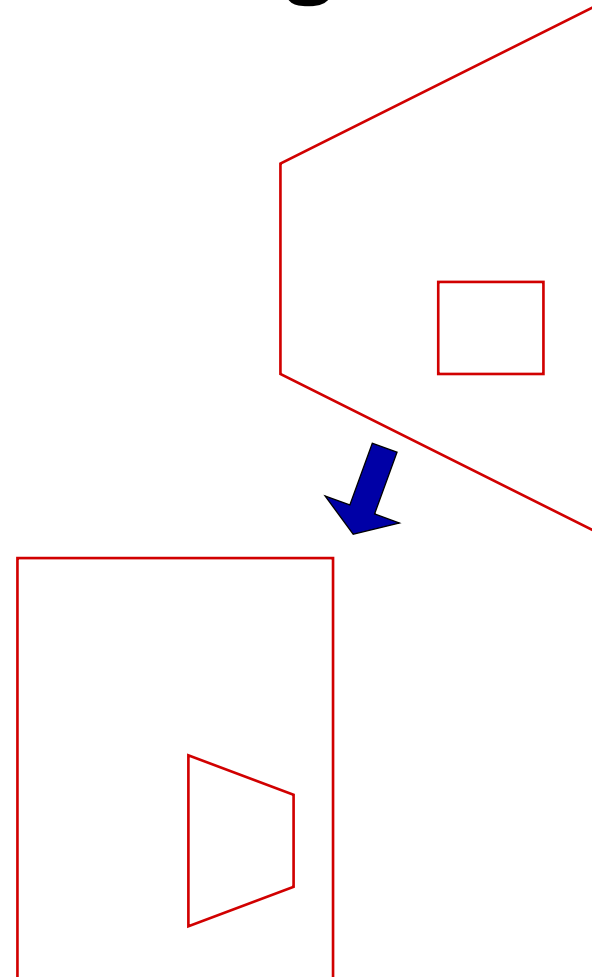
- Scale by to $-z$ to convert to a cube using a matrix with last row $[0 \ 0 \ -1 \ 0]$.
- Simultaneously send third component to range $0 \dots -1$.
 $z = -k$ maps to 0 and $z = -1$ maps to -1 .
- Scale z by $1 - k$ and add $\frac{k}{1-k}$ as the translation component.
- Divide the first and second components by the fourth component to get where the point projects to in the normalized image space.
- Keep third component for later use. Relative ordering needs to be preserved. The values are not important.

Perspective Normalizing Txform

- Matrix M_3 for this step:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1-k} & \frac{k}{1-k} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- $(x, y, -k)$ and $(x, y, -1)$ go to?
- Final matrix: $M = M_3M_2M_1$.
- Frustum becomes a cube!



- Above matrix converts z to the range $[0, -1]$.
The following matrix converts z to range $[+1, -1]$.

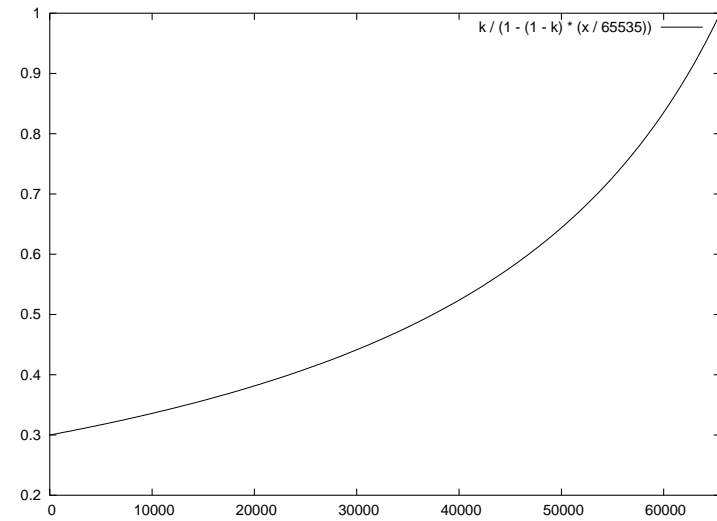
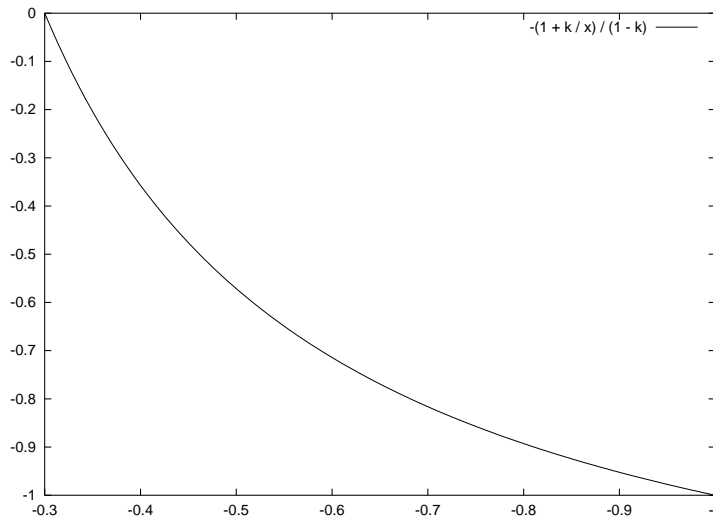
$$M'_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{2}{1-k} & \frac{1+k}{1-k} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Final 2D Coordinates

$$(u, v, d) \equiv \begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = M_3 M_2 M_1 \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{\text{VRC}}$$

- **Perspective division:** Divide x', y' coordinates by the w to get the normalized coordinates (u, v) . (z' maintains ordering and can be used without division.)
- The normalized d component has non-linear precision. Higher around the *near* plane and lower around the *far* plane due to the division by z .

- $d = -(1 + k/z)/(1 - k)$ and $z = k/(1 - (1 - k)d)$
- Effect of M_3 for $k = 0.3$: d vs z (left) and z vs d (right):



- The resolution of d is variable for z , not linear.

OpenGL Normalizing/Perspective Matrix

- The projection matrix in OpenGL is given by

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & A & 0 \\ 0 & \frac{2n}{t-b} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where,

$$A = \frac{r+l}{r-l}, B = \frac{t+b}{t-b}, C = \frac{f+n}{f-n}, D = \frac{2nf}{f-n}$$

Actual Projection

- We have already performed the perspective division.
- Projection involves simply dropping the z coordinate and scaling x - y to the viewport.
- Why go through with the z -coordinates?
- The ordering is preserved along the depth dimension.
 z values can be used for visibility determination.

Where is the Film?

- Turns out: **It does not matter.**
- A final scaling is in the viewport transformations.
- As long as the film is in front of the camera, we will see an upright image.
- Can consider the near plane as the film plane.

3D Module: Recapitulation

- 3D Graphics additionally involves projecting the 3D world to the 2D image plane of the camera.
- Compute the 3D world with respect to the camera. Or compute the relative geometry first.
- This involves a series of rigid transformations. For complex objects/environments, each object or its part is described in its own coordinate system.
- **Modelling** places these different objects in the world

coordinate system. This could involve a hierarchy of transforms for objects made up of complex parts.

- **View Orientation** computes the world in the VRC.
- Camera can be perspective or parallel (orthographic, oblique). 6 planes give the view volume and defines the camera.
- **View Mapping** involves mapping the world to a canonical view volume, which is an orthographic view volume. This **Normalizing Transformation** has different forms for parallel and perspective cameras.

- **Projection** and **Clipping** are easy to perform in the canonical view volume. An image with dimensions from -1 to +1 results.
- **Viewport** transformation is the final step, involving a 2D scaling and translation to map to window coordinates that can be used to address the frame buffer.
- Given a description of the 3D world primitives, project each point to 2D to get 2D primitives. These can be scan-converted using standard algorithms.