CAP4730: Computational Structures in Computer Graphics

3D Transformations

Outline

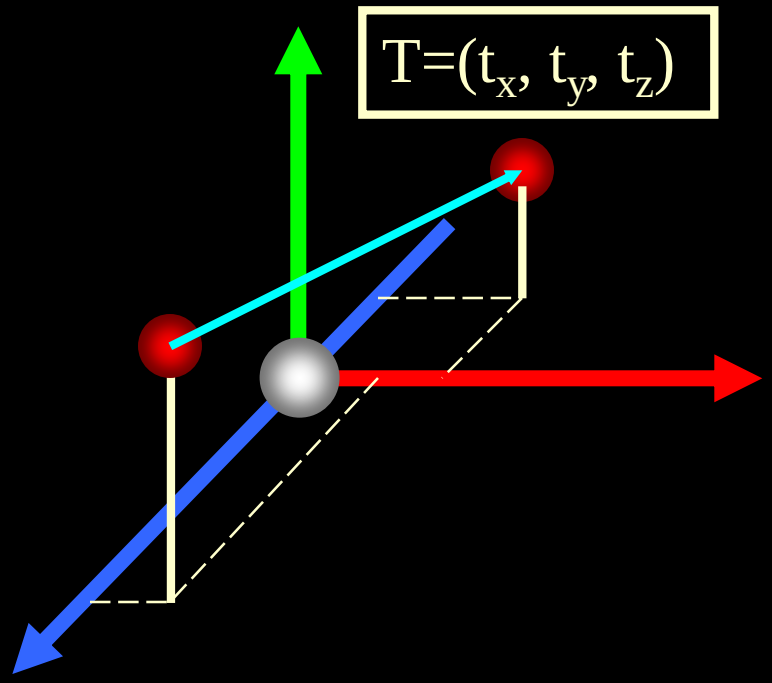
- 3D World
- What we are trying to do
- Translate
- Scale
- Rotate

Transformations in 3D!

- Remembering 2D transformations -> 3x3 matrices, take a wild guess what happens to 3D transformations.

$$T(t_x,t_y)=\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}t_x\\t_y\end{bmatrix}=\begin{bmatrix}1&0&t_x\\0&1&t_y\\0&0&1\end{bmatrix}$$

$$T(t_x,t_y,t_z)=\begin{bmatrix}x\\y\\z\end{bmatrix}+\begin{bmatrix}t_x\\t_y\\t_z\end{bmatrix}=\begin{bmatrix}1&0&0&t_x\\0&1&0&t_y\\0&0&1&t_z\\0&0&0&1\end{bmatrix}$$

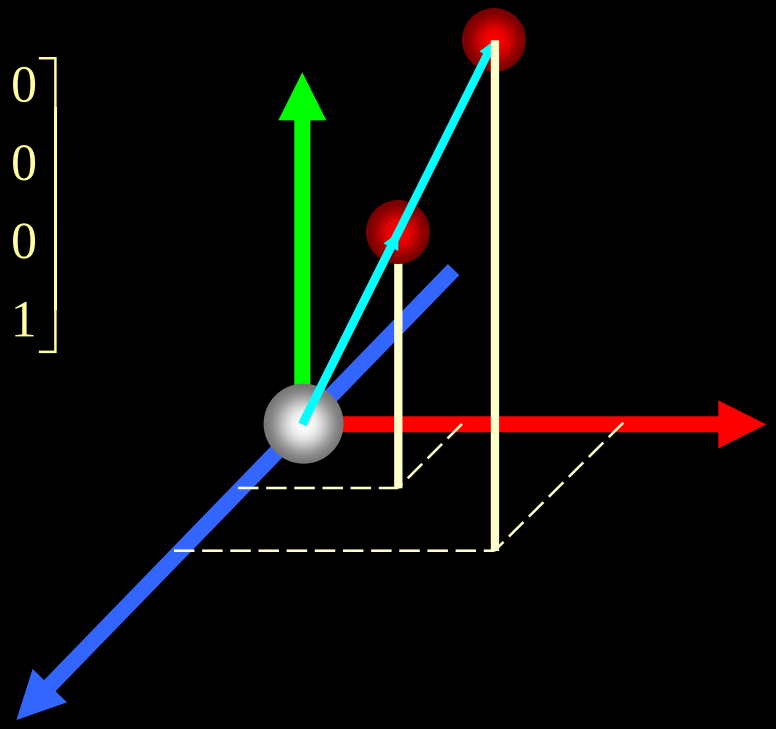


Scale, 3D Style

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$S=(s_x, s_y, s_z)$

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



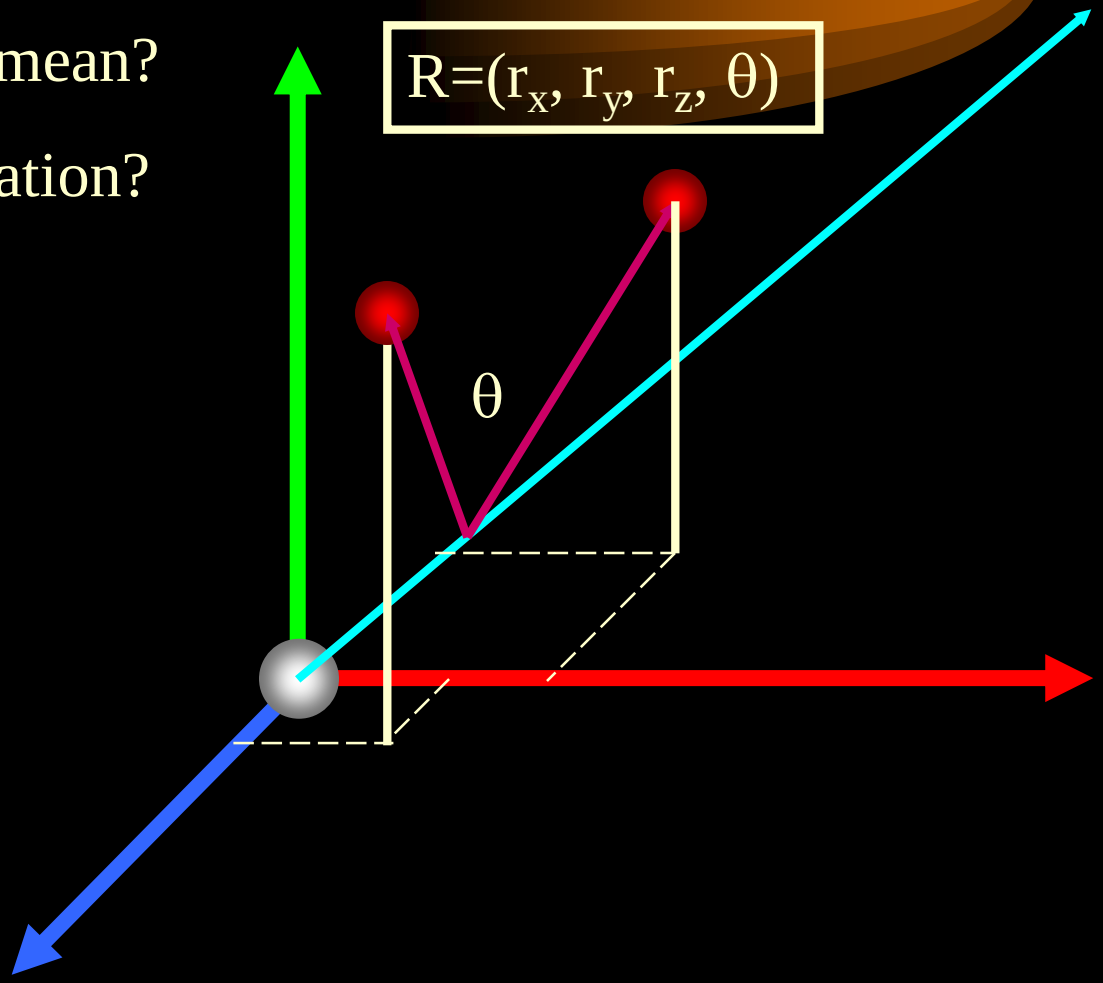
Rotations, in 3D no less!

What does a rotation in 3D mean?

Q: How do we specify a rotation?

A: We give a vector to rotate about, and a theta that describes how much we rotate.

Q: Since 2D is sort of like a special case of 3D, what is the vector we've been rotating about in 2D?

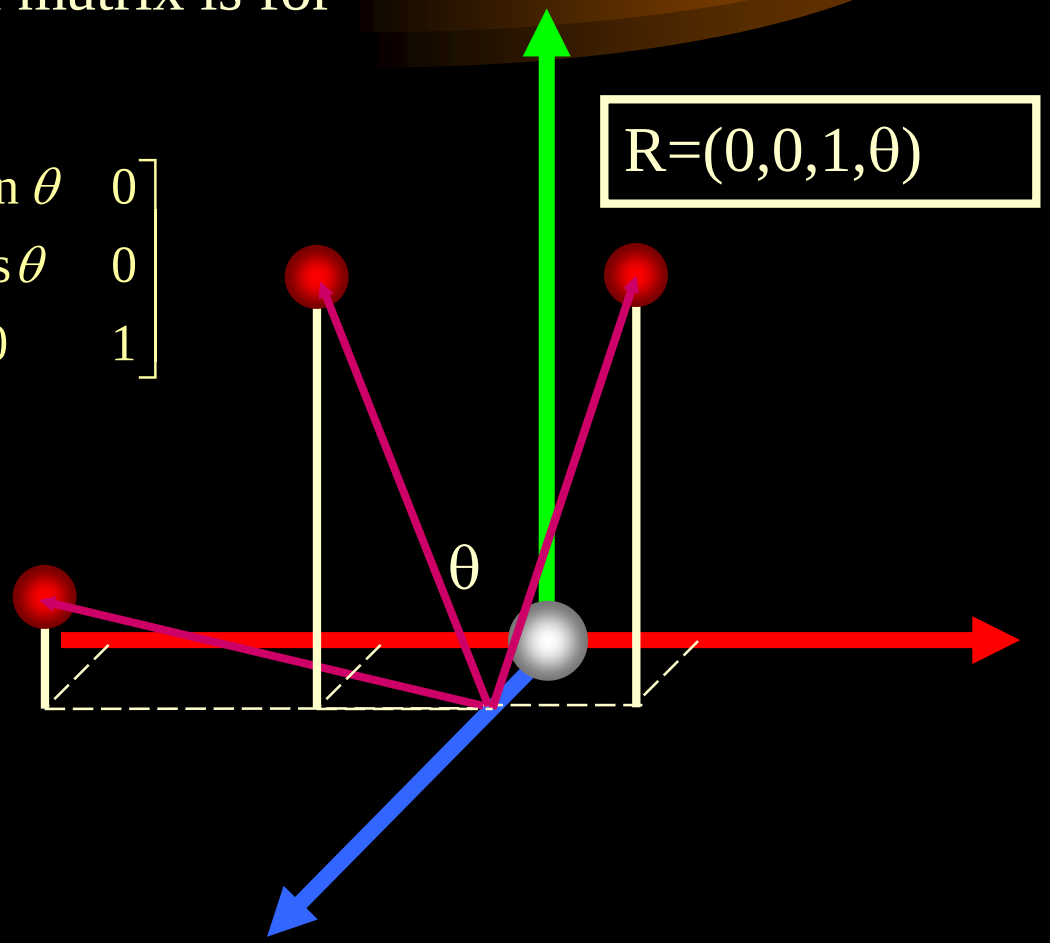


Rotations about the Z axis

What do you think the rotation matrix is for rotations about the z axis?

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

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

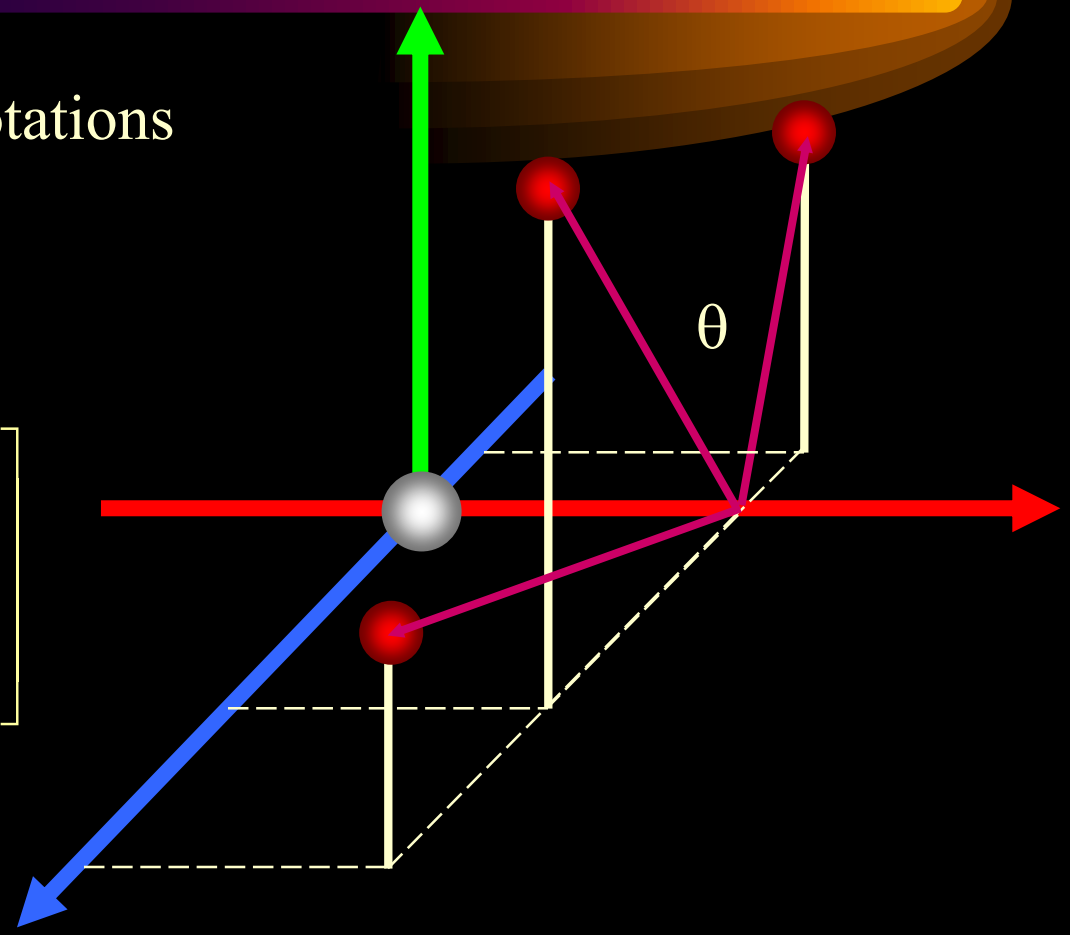


Rotations about the X axis

Let's look at the other axis rotations

R=(1,0,0,θ)

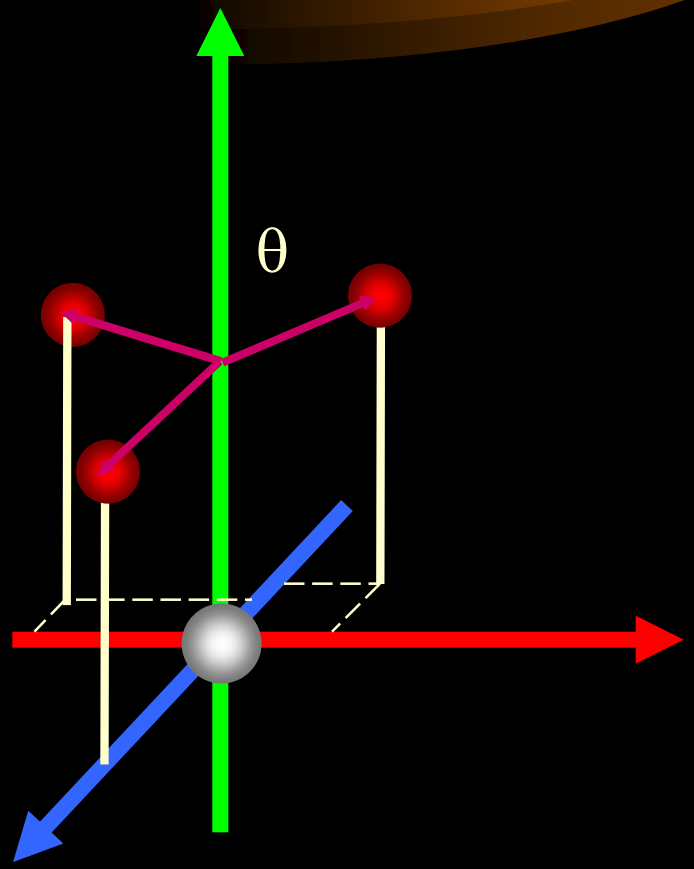
$$R(1,0,0,\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Rotations about the Y axis

R=(0,1,0,θ)

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



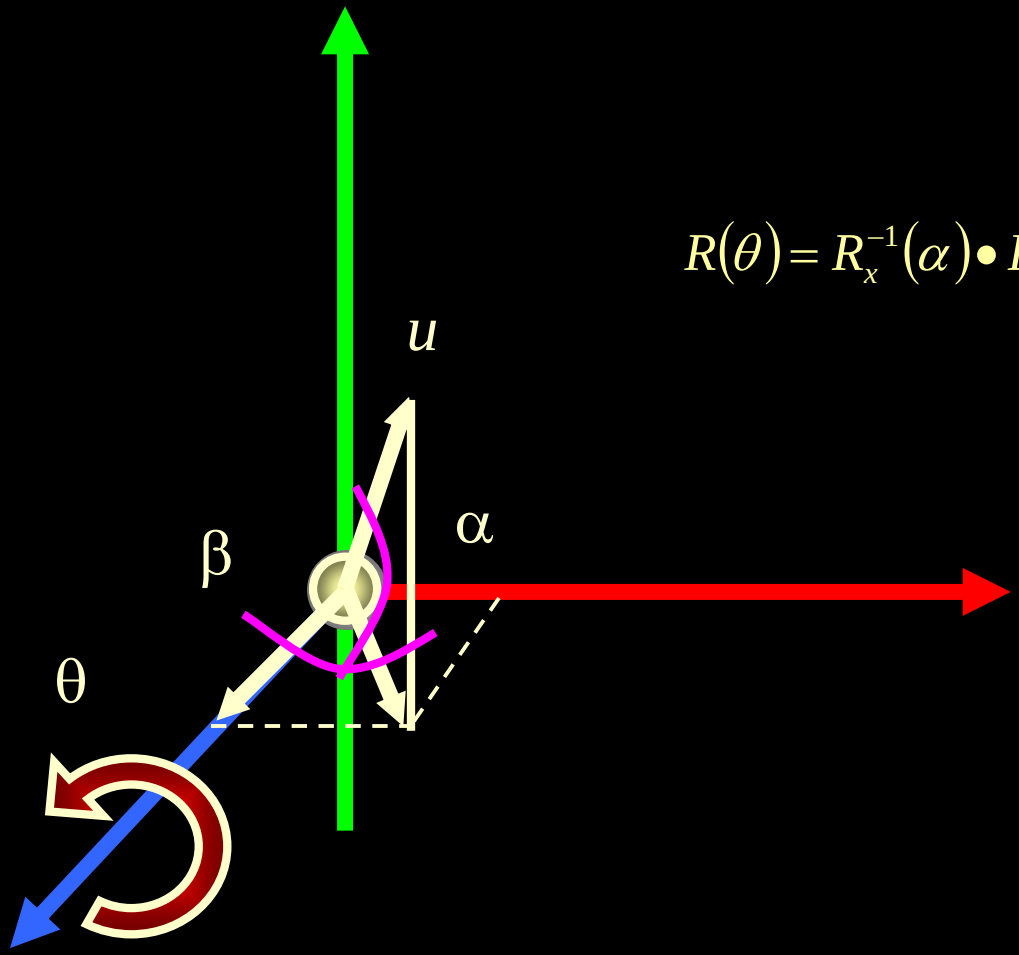
Rotations for an arbitrary axis

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

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

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

Rotations for an arbitrary axis



$$R(\theta) = R_x^{-1}(\alpha) \bullet R_y^{-1}(\beta) \bullet R_z(\theta) \bullet R_y(\beta) \bullet R_x(\alpha)$$

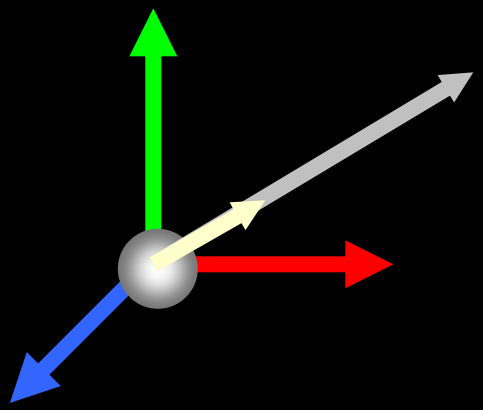
Steps:

1. Normalize vector *u*
2. Compute α
3. Compute β
4. Create rotation matrix

Vector Normalization

- Given a vector ***v***, we want to create a *unit vector* that has a magnitude of 1 and has the same *direction* as ***v***. Let's do an example.

$$Normalized\_V = \frac{V}{|V|}$$



Computing the Rotation Matrix

- 1. Normalize *u*
- 2. Compute $R_x(\alpha)$
- 3. Compute $R_y(\beta)$
- 4. Generate Rotation Matrix

$$u = (a, b, c)$$

$$d = \sqrt{b^2 + c^2}$$

$$u_z = (0,0,1)$$

$$\cos\alpha = \frac{u' \cdot u_z}{\left\|u'\right\| \cdot u_z} = \frac{c}{d}$$

$$\sin\alpha = \frac{b}{d}$$

$$\cos\beta = d$$

$$\sin\beta = -a$$

Rotation Matrix

$$R(\theta) = R_x^{-1}(\alpha) \bullet R_y^{-1}(\beta) \bullet R_z(\theta) \bullet R_y(\beta) \bullet R_x(\alpha)$$

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Applying 3D Transformations

P’=TRTP

Let’s compute M

$$P' = T(t_x, t_y, t_z)R(\theta)T(-t_x, -t_y, -t_z)P$$

$$P' = MP$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Homogenous Coordinates

- We need to do something to the vertices
- By increasing the dimensionality of the problem we can transform the addition component of Translation into multiplication.

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x_h \\ y_h \\ h \end{bmatrix} = \begin{bmatrix} x = \frac{x_h}{h} \\ y = \frac{y_h}{h} \\ \frac{h}{h} \end{bmatrix} Ex. \begin{bmatrix} 4 \\ 2 \end{bmatrix} \rightarrow \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}, Ex. \begin{bmatrix} 3 \\ 7 \end{bmatrix} \rightarrow \begin{bmatrix} 6 \\ 14 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 = \frac{6}{2} \\ 7 = \frac{14}{2} \\ \frac{2}{2} \end{bmatrix}$$

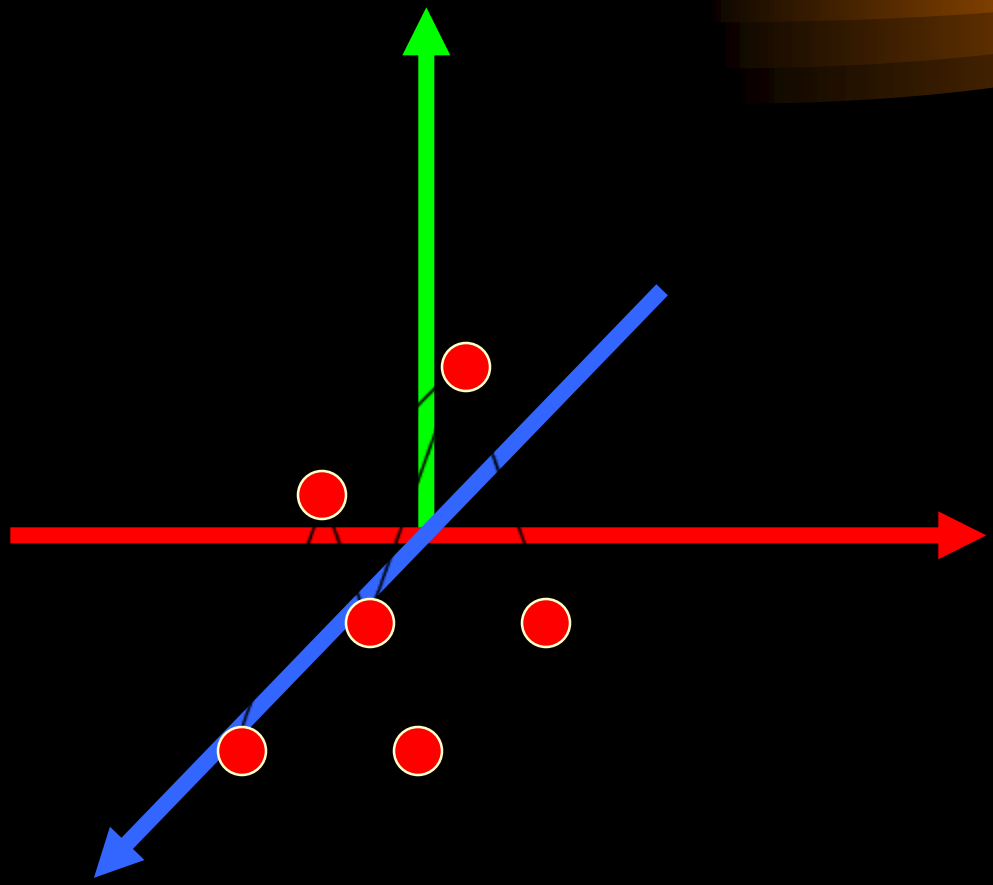
Homogenous Coordinates

- Homogenous Coordinates – embed 3D transforms into 4D
- All transformations can be expressed as matrix multiplications.
- Inverses and combination easier
- Equivalence of vectors (4 2 1 1)=(8 4 2 2)
- What this means programatically

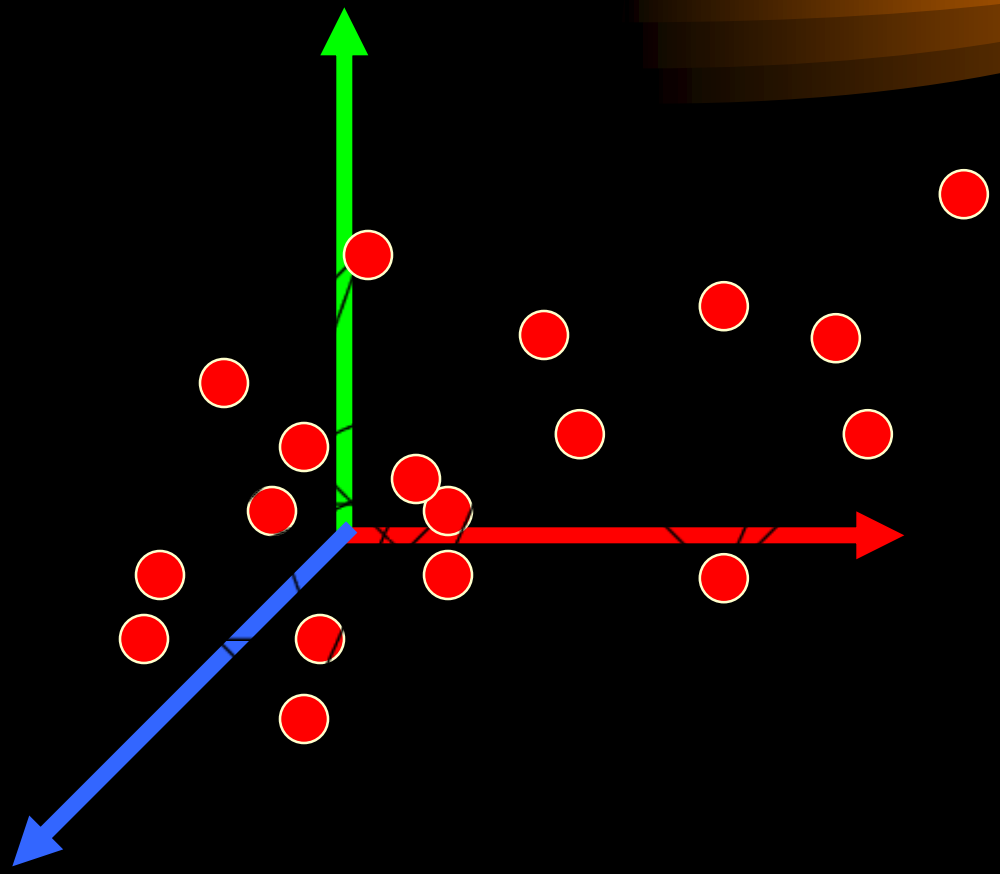
The Question

- Given a 3D point, an eye position, a camera position, and a display plane, what is the resulting pixel position?
- Now extend this for a group of three points
- Then apply what you know about scan conversion.

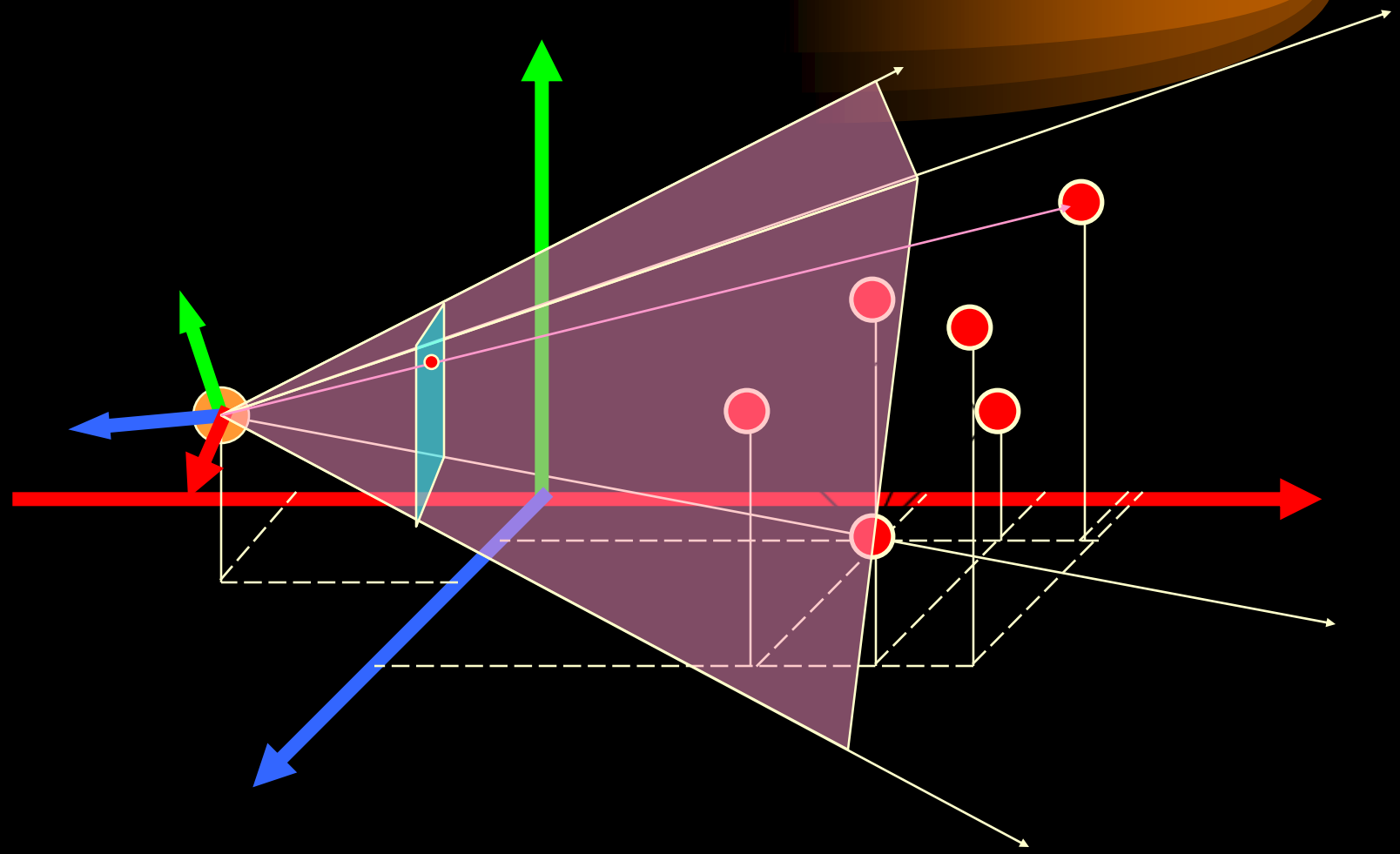
Different Phases: Model Definition



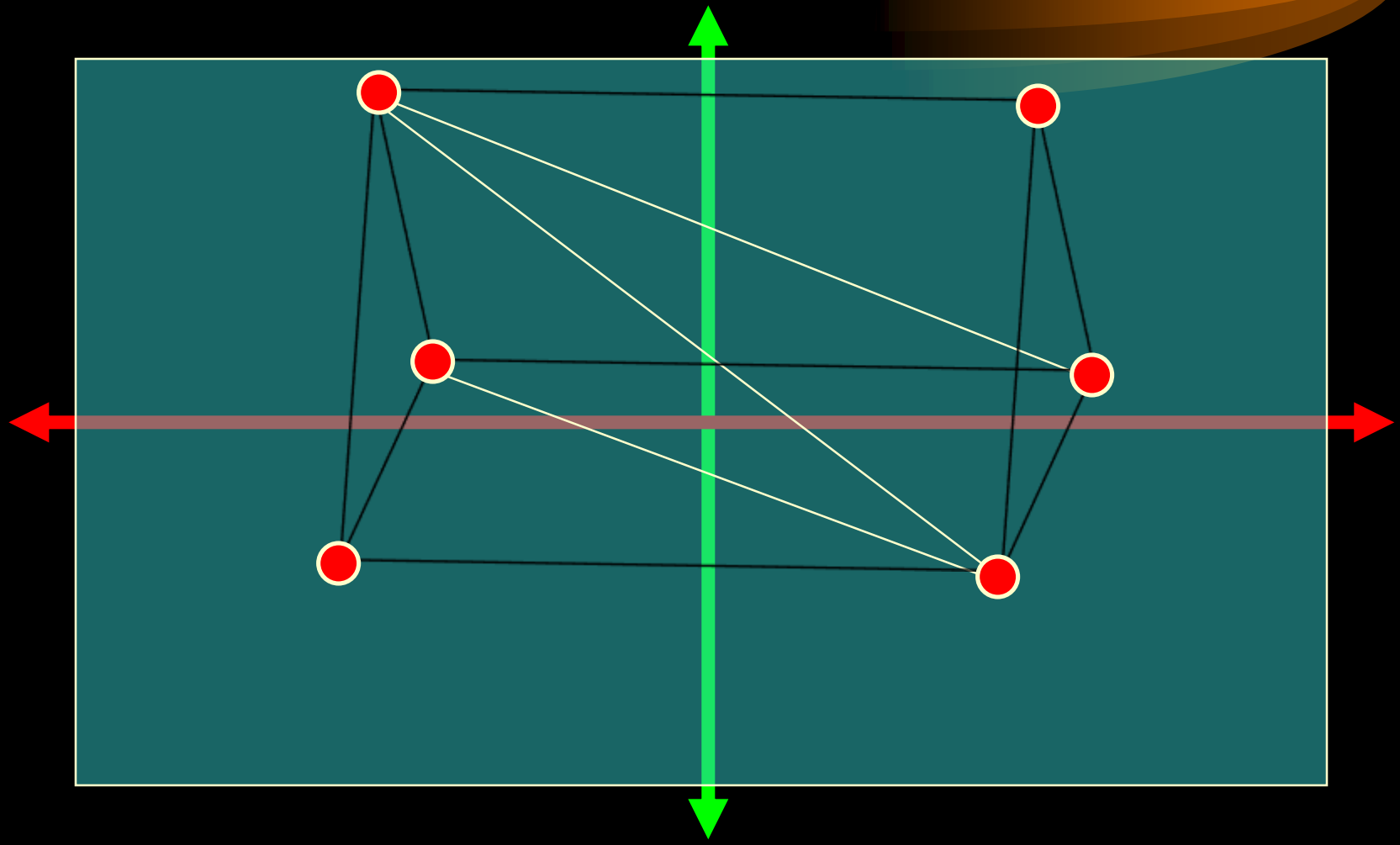
Different Phases: Transformations



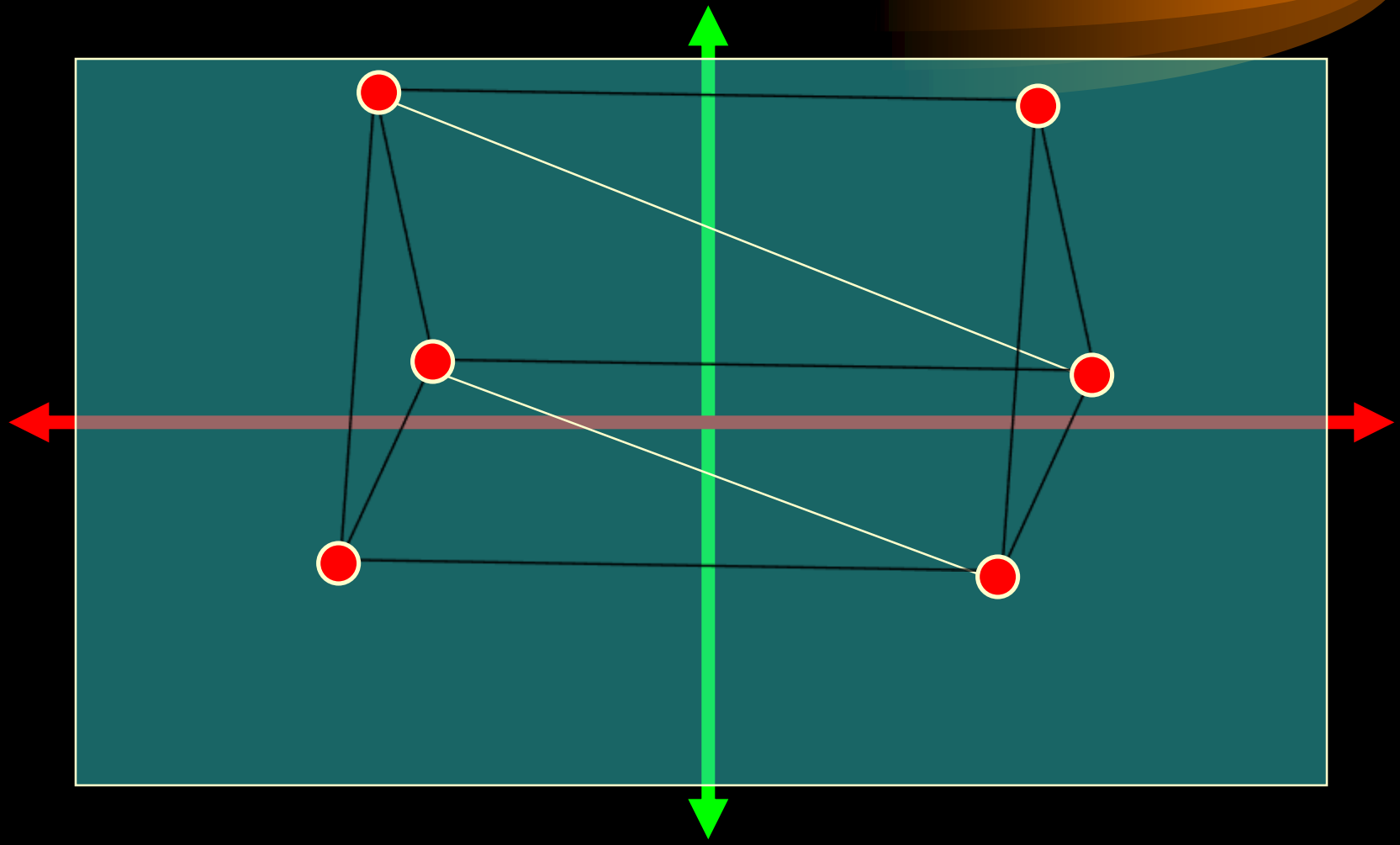
Different Phases: Projection



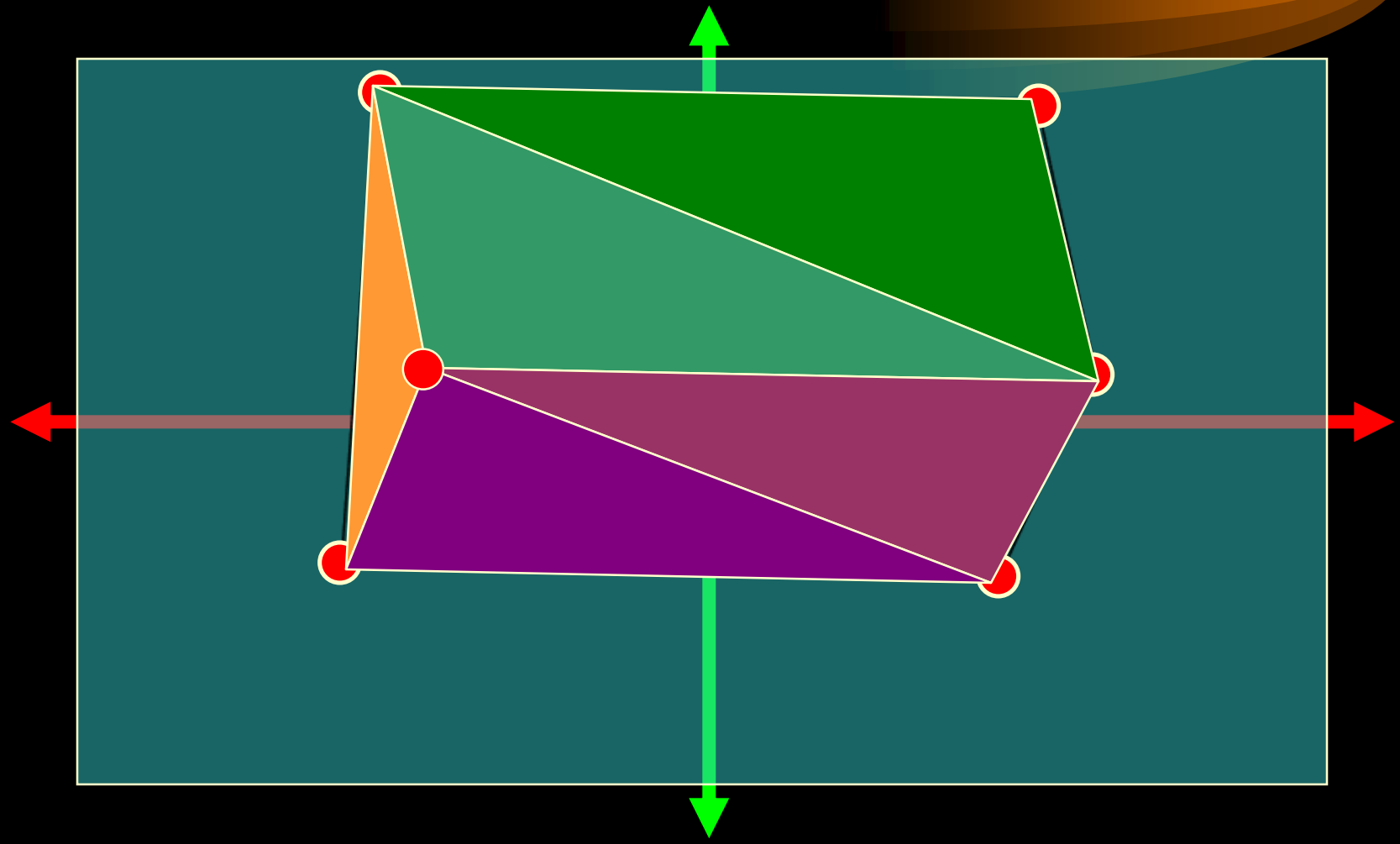
Different Phases: Projection



Different Phases: Rasterization



Different Phases: Scan Conversion

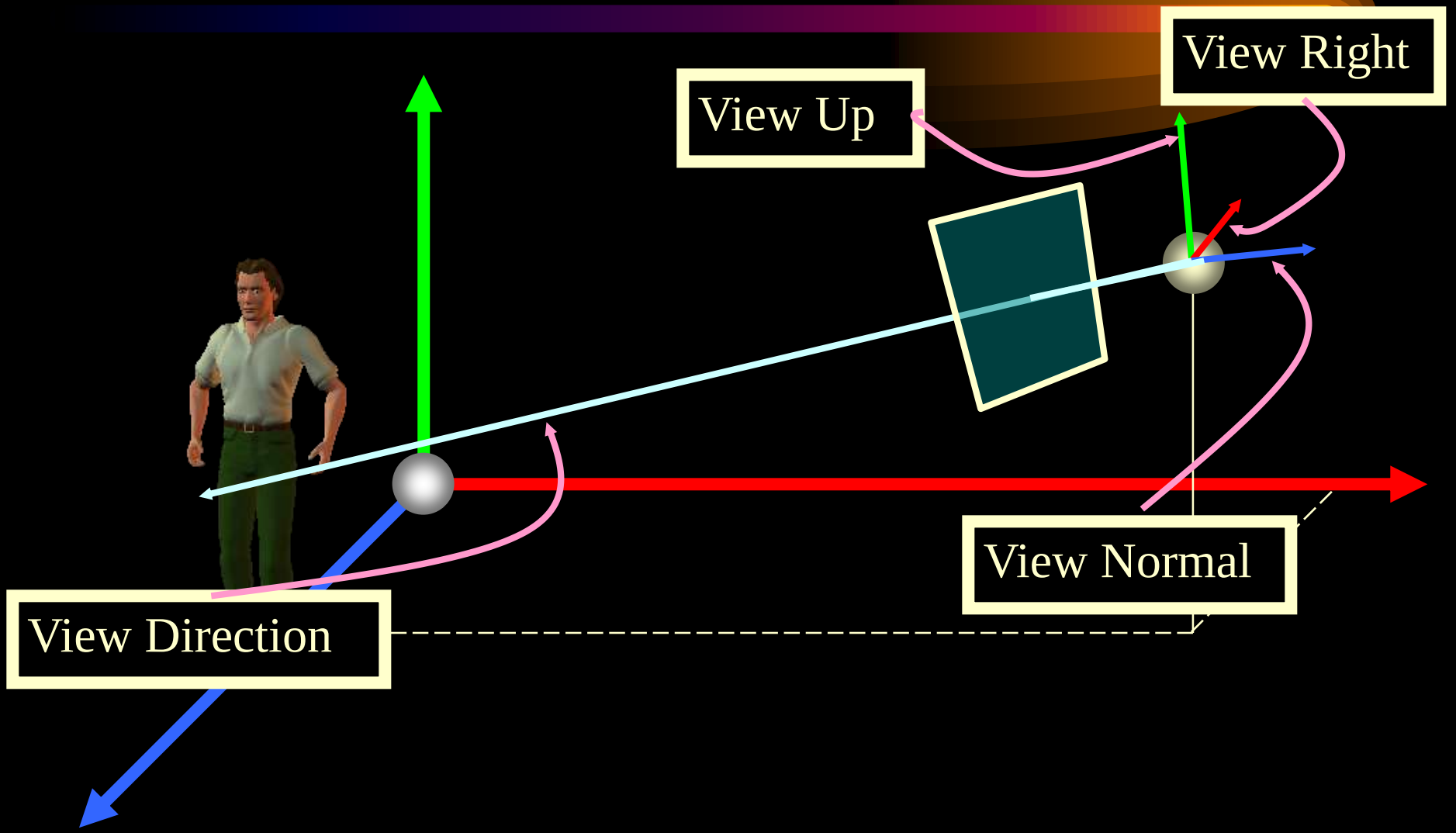


What are the steps needed?

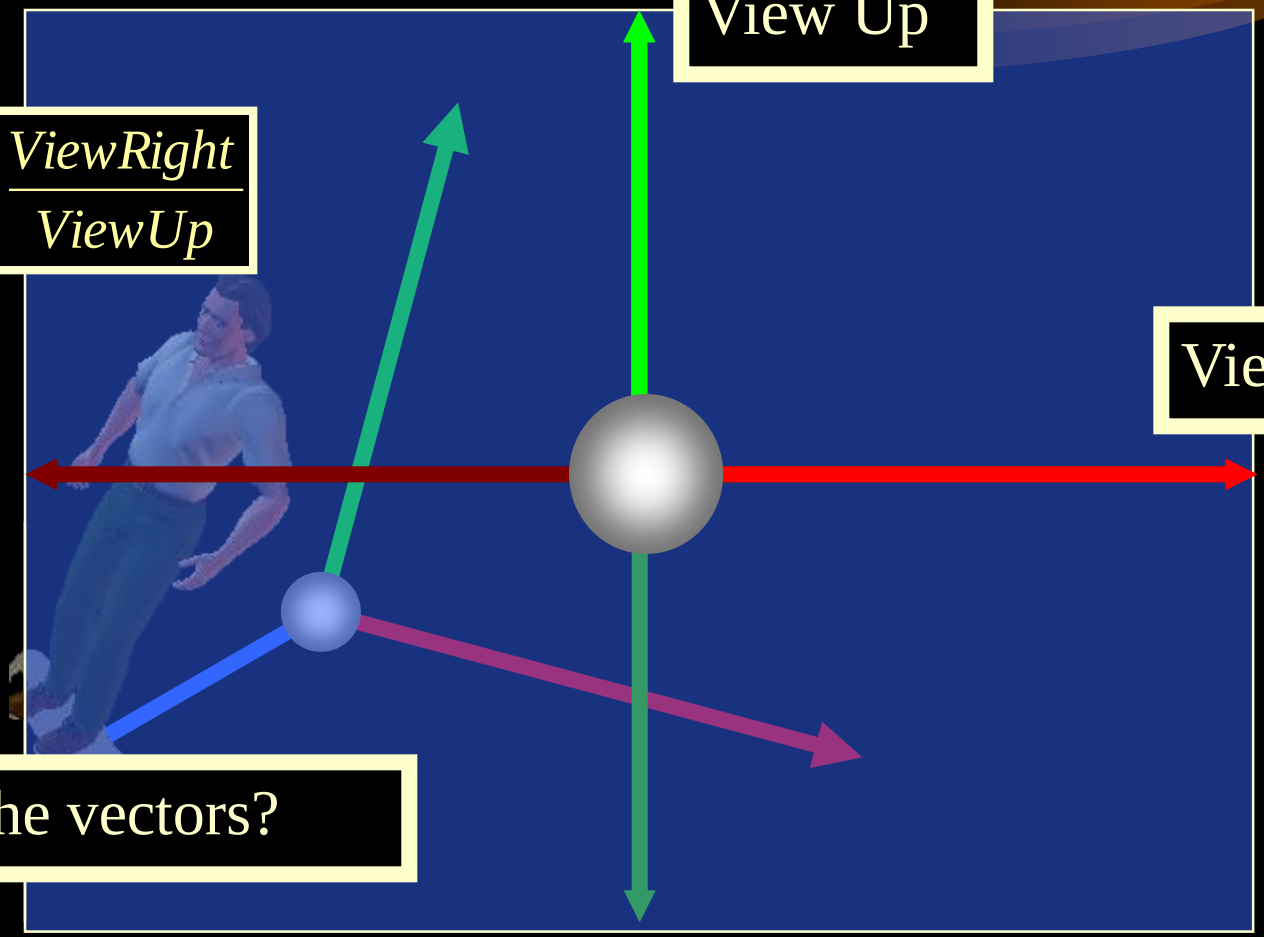
Let's Examine the Camera

- If I gave you a world, and said I want to "render" it from another viewpoint, what information do I have to give you?
  - Position
  - Which way we are looking
  - Which way is "up"
  - Aspect Ratio
  - Field of View
  - Near and Far

Camera



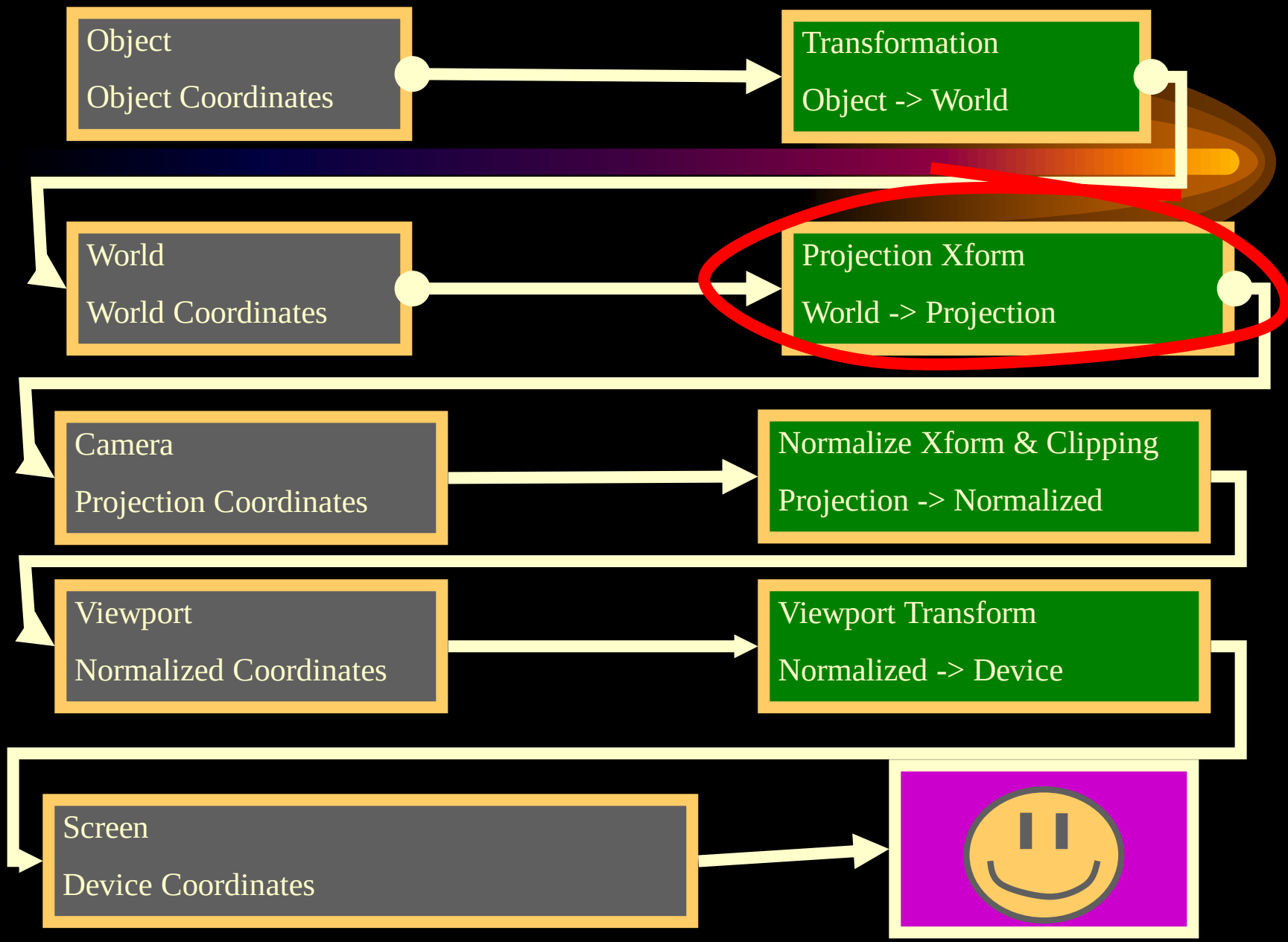
Camera



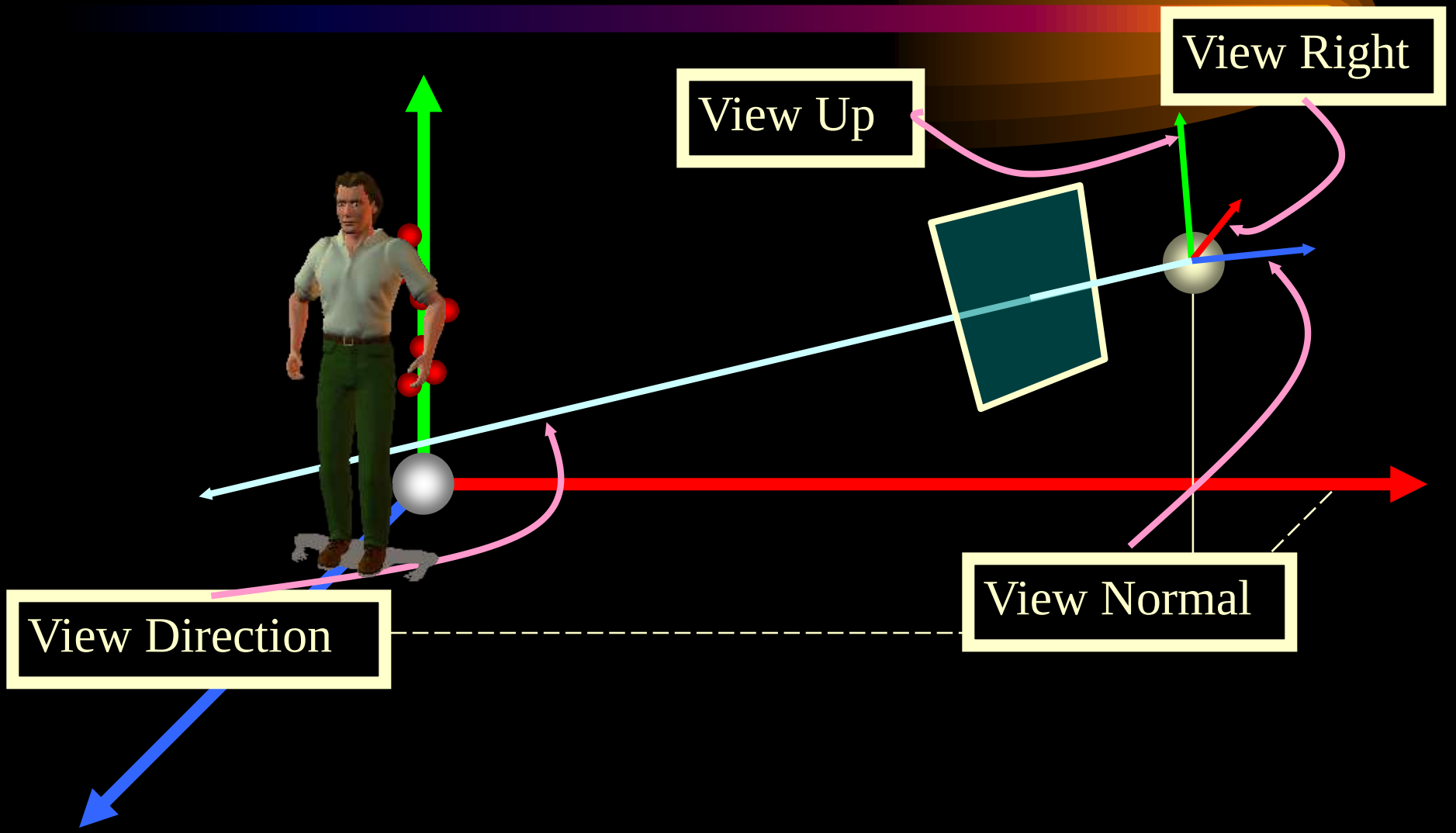
$$AspectRatio = \frac{ViewRight}{ViewUp}$$

What are the vectors?

Graphics Pipeline So Far



Transformation World->Camera



Transformation World->Camera

$$T = \begin{bmatrix} 1 & 0 & 0 & -camera_x \\ 0 & 1 & 0 & -camera_y \\ 0 & 0 & 1 & -camera_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

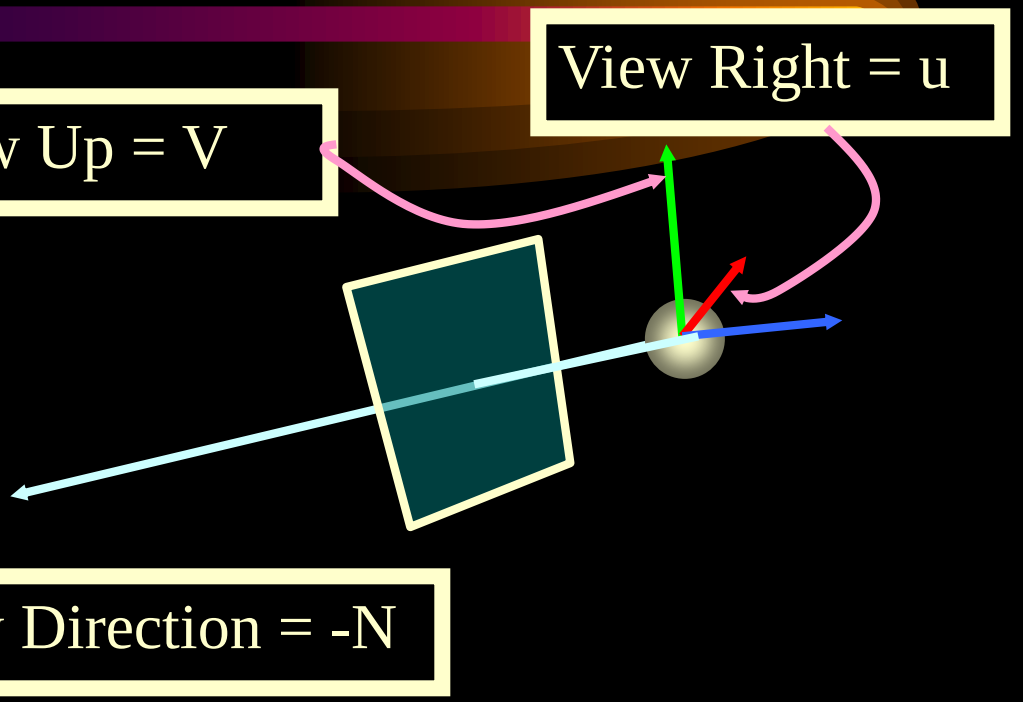
$$n = \frac{N}{|N|} = (n_1, n_2, n_3)$$

$$u = \frac{V \times N}{|V \times N|} = (u_1, u_2, u_3)$$

$$v = n \times u = (v_1, v_2, v_3)$$

$$R = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{WC \to VC} = R \bullet T$$



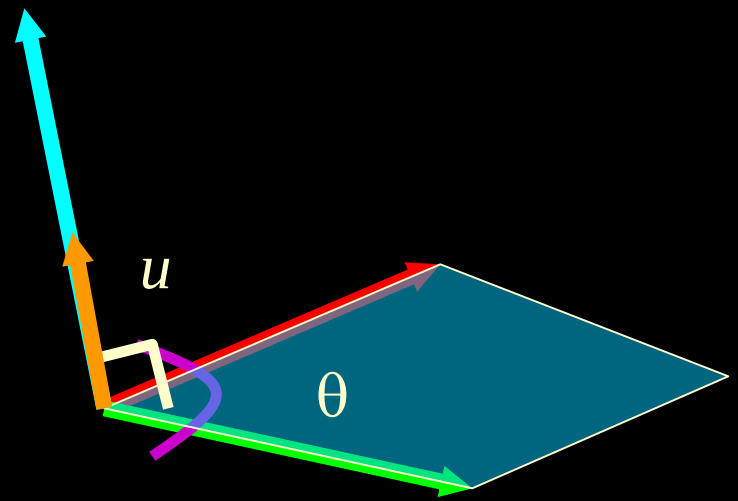
Cross Products

Given two vectors, the cross product returns a vector that is perpendicular to the plane of the two vectors and with magnitude equal to the area of the parallelogram formed by the two vectors.

$$A \times B = u|A||B|\sin\theta$$

$$A \times B = (A_y B_z - A_z B_y, A_z B_x - A_x B_z, A_x B_y - A_y B_x)$$

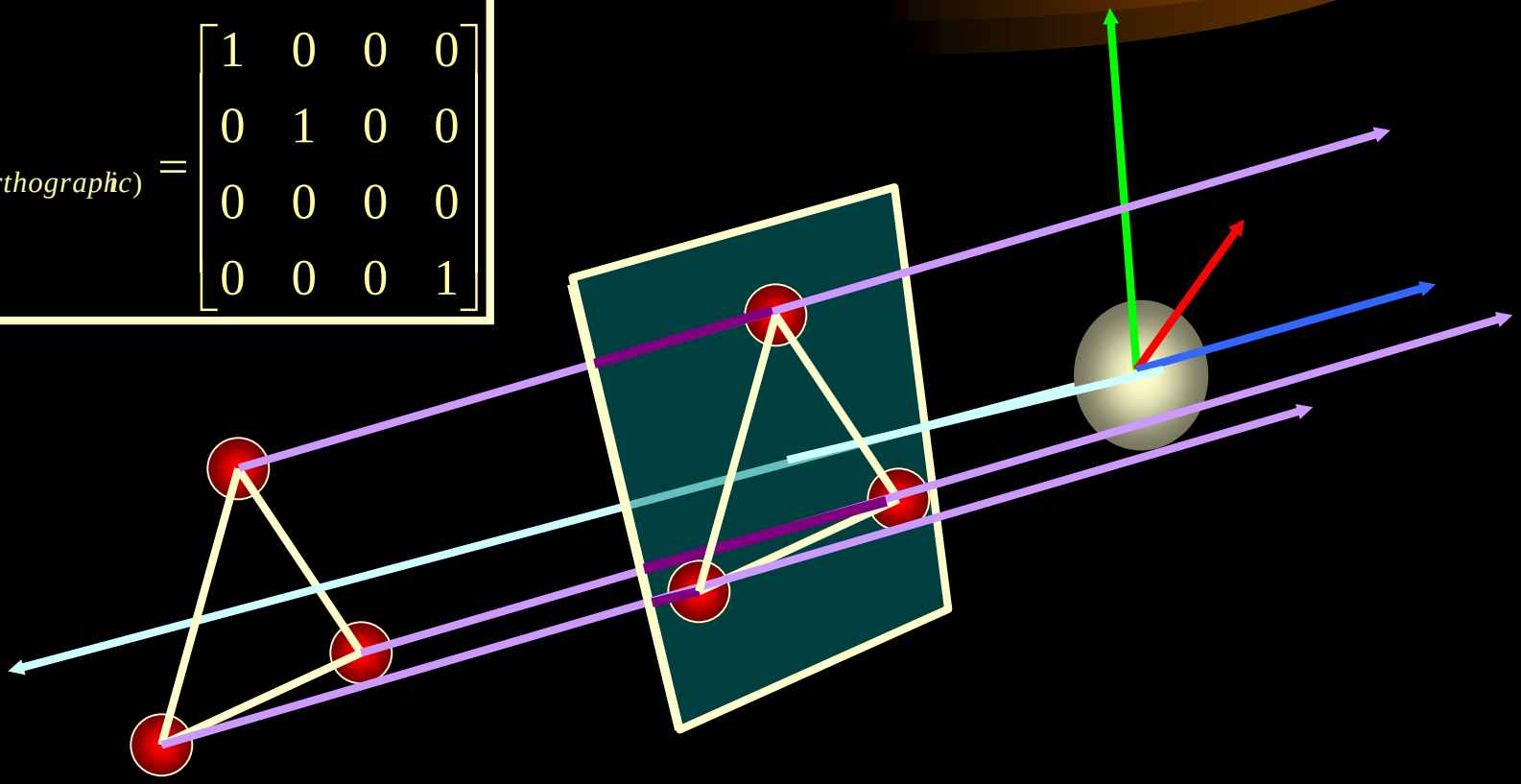
$$A \times B = \begin{bmatrix} u_x & u_y & u_x \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{bmatrix}$$



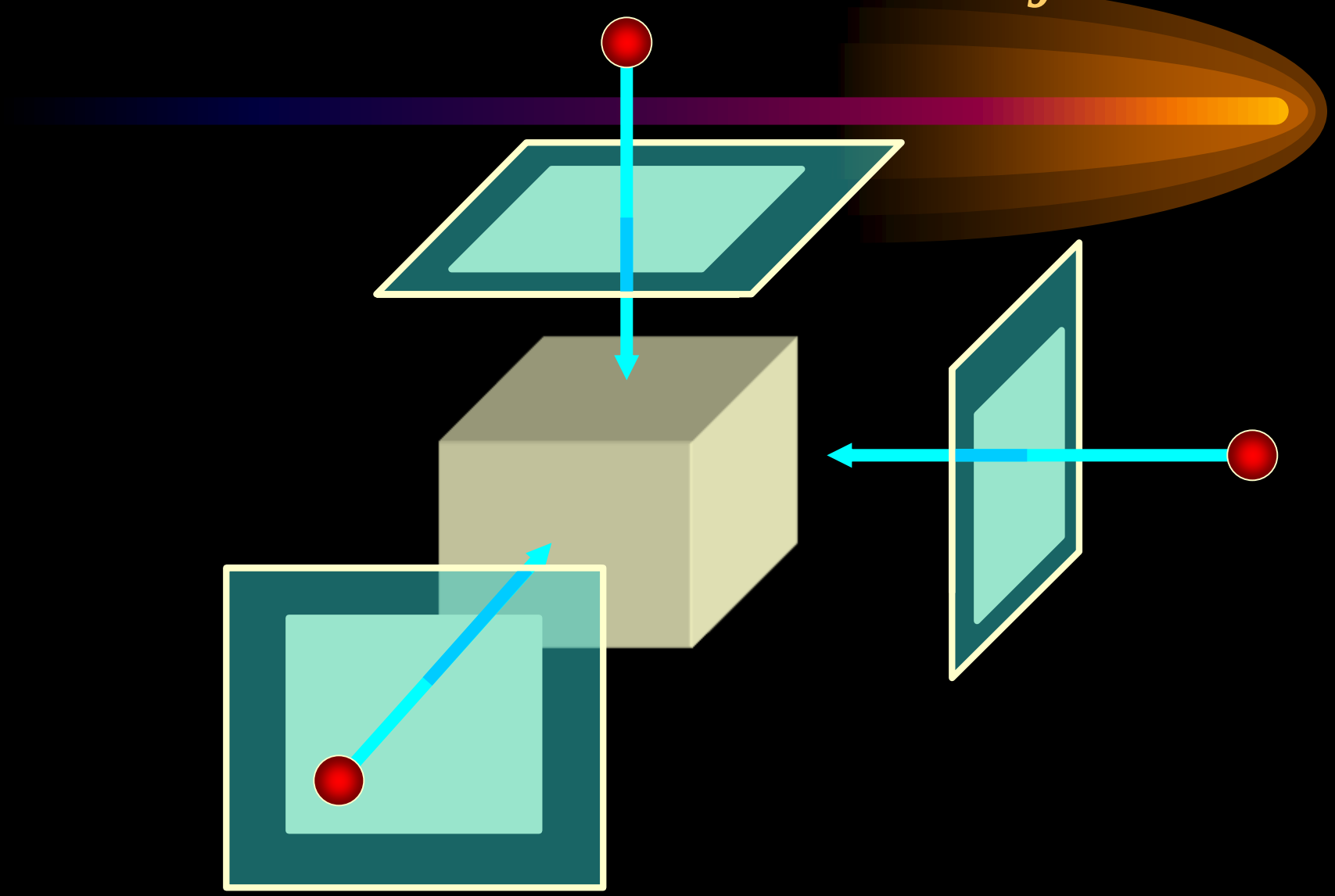
Parallel Projections (known aliases): Orthographic or Isometric Projection

$$V_P' = P_{Parallel} MV$$

$$P_{Parallel(Orthographic)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Parallel Projection

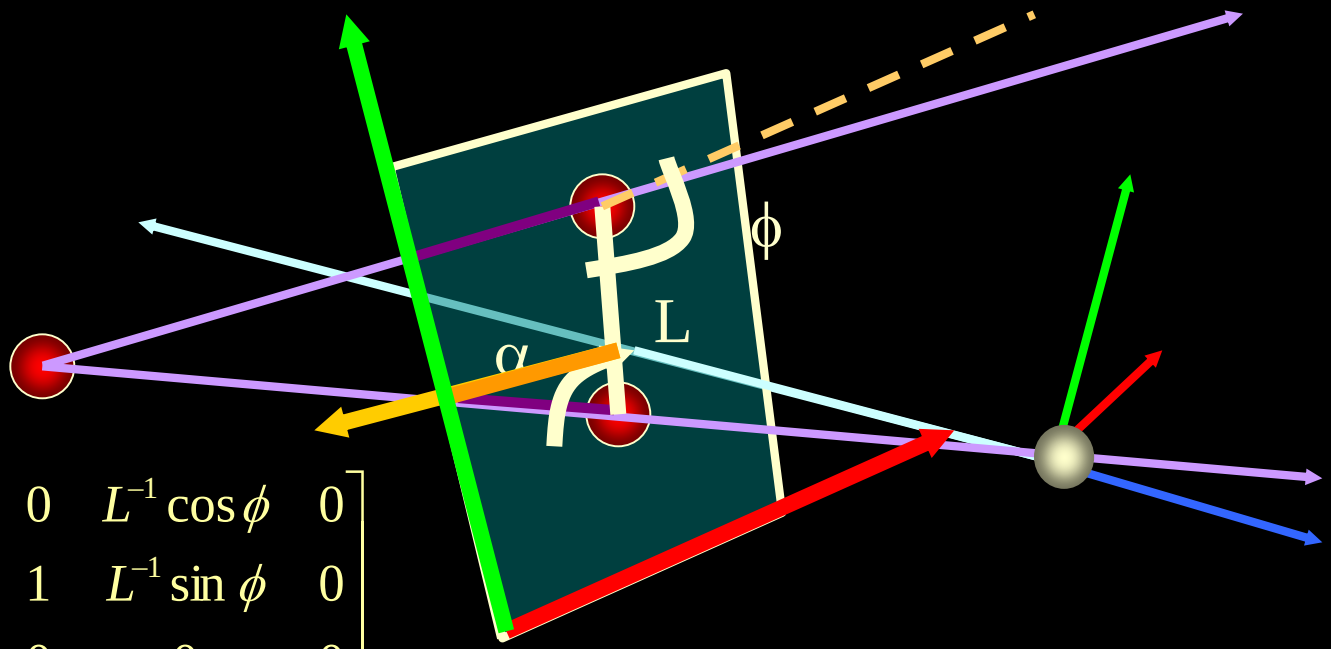


Parallel Projections (known aliases): Oblique Projection

$$V_P' = P_{Parallel} M V_{VC}$$

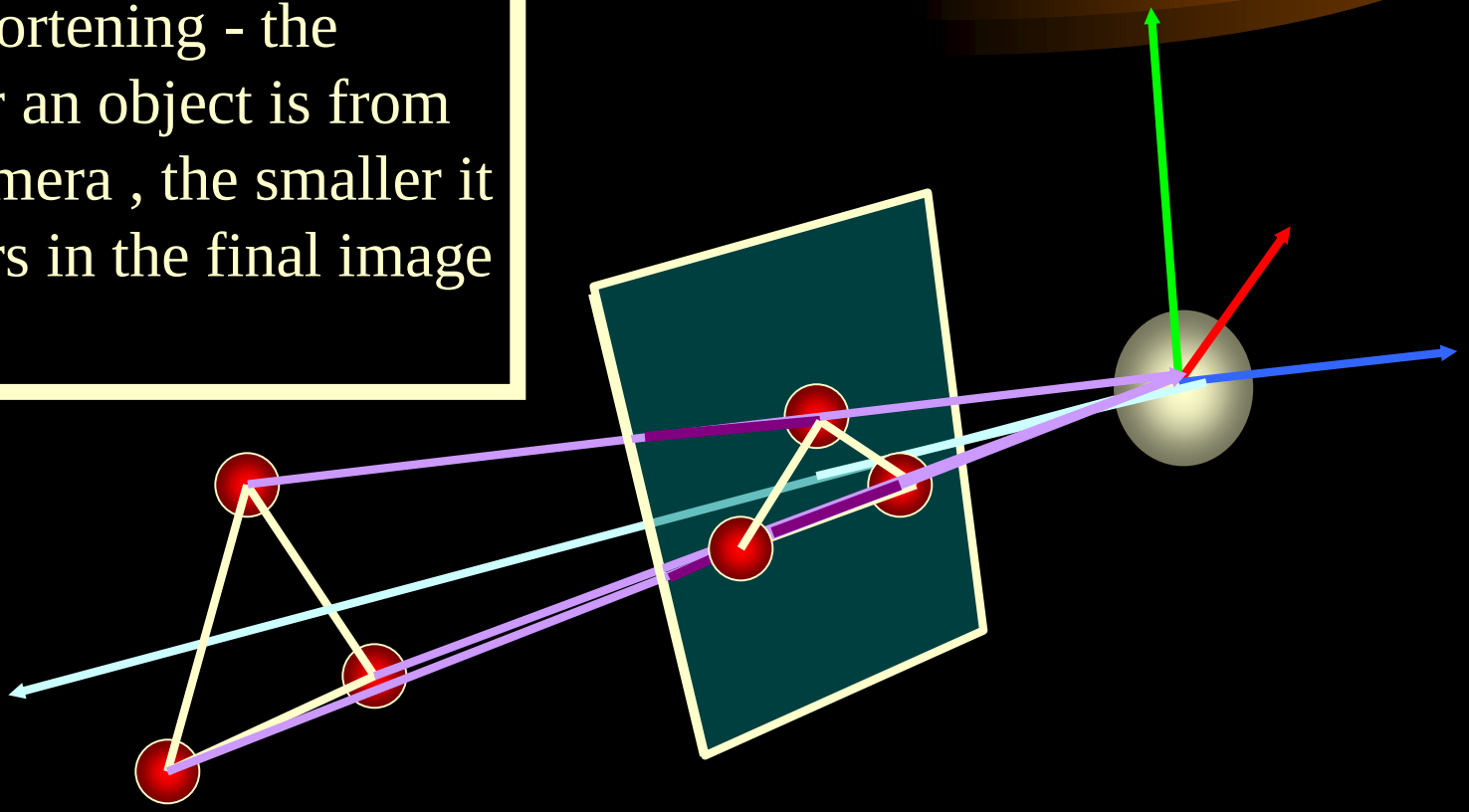
$$L^{-1} = \tan^{-1} \alpha$$

$$P_{Parallel(Oblique)} = \begin{bmatrix} 1 & 0 & L^{-1}\cos\phi & 0 \\ 0 & 1 & L^{-1}\sin\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

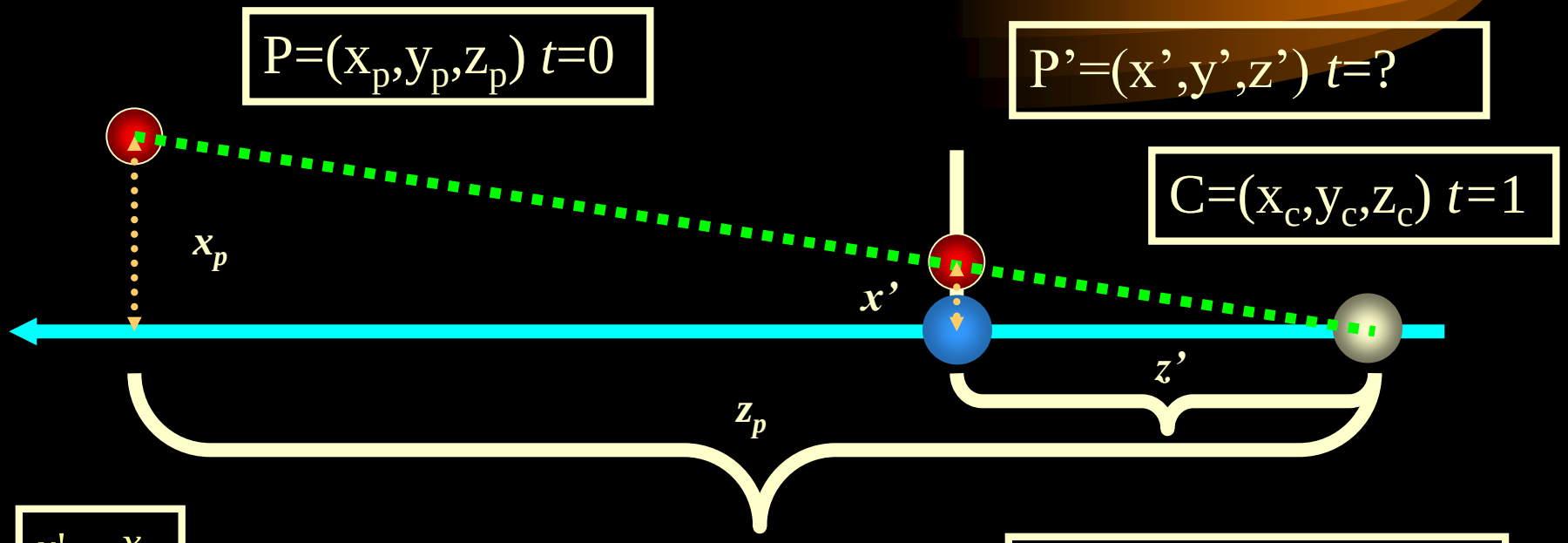


Projections

foreshortening - the farther an object is from the camera , the smaller it appears in the final image



Perspective Projection Side View



$$\frac{x'}{z'} = \frac{x_p}{z_p}$$

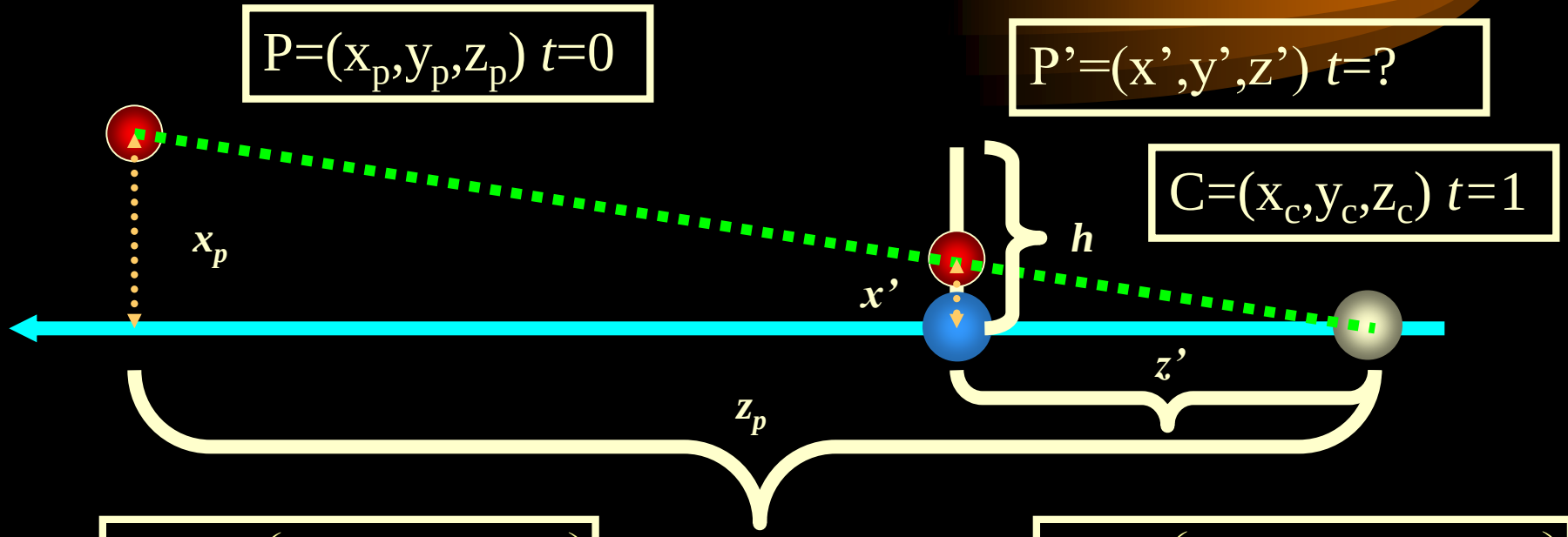
$$\frac{y'}{z'} = \frac{y_p}{z_p}$$

$$z' = z'$$

$$w = \frac{z_p}{z'};\quad \left(x' = x_p \frac{z'}{z_p} = \frac{x_p}{w},\quad y' = y_p \frac{z'}{z_p} = \frac{y_p}{w},\quad z' = z_p \frac{z'}{z_p} \right)$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z'} & 0 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

Perspective Projection Side View



$$w = \frac{z_p}{z'};\quad \begin{pmatrix} x' = x_p \dfrac{z'}{z_p} = \dfrac{x_p}{w} \\ y' = y_p \dfrac{z'}{z_p} = \dfrac{y_p}{w} \\ z' = z_p \dfrac{z'}{z_p} \end{pmatrix}$$

Scale by h

$$w = z_p \quad \begin{pmatrix} x' = x_p \dfrac{z'}{z_p} = x_p \dfrac{z'}{hz_p} \\ y' = y_p \dfrac{z'}{z_p} = y_p \dfrac{z'}{hz_p} \\ z' = \dfrac{fz_p}{f - z'} - \dfrac{z'\,f}{f - z'} \end{pmatrix}$$

Perspective Divide

$$w = z_p \left(\begin{array}{l} x' = x_p \dfrac{z'}{z_p} = x_p \dfrac{z'}{hz_p} \\ y' = y_p \dfrac{z'}{z_p} = y_p \dfrac{z'}{hz_p} \\ z' = \dfrac{fz_p}{f - z'} - \dfrac{z'\, f}{f - z'} \end{array} \right)$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} \dfrac{z'}{h_{horiztonal}} & 0 & 0 & 0 \\ 0 & \dfrac{z'}{h_{vertical}} & 0 & 0 \\ 0 & 0 & \dfrac{far}{far - near} & \dfrac{-near * far}{far - near} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

Foreshortening - look at the x,y, and w values, and how they depend on how far away the object is.

Modelview Matrix - describes how to move the world->camera coordinate system

Perspective Matrix - describes the camera you are viewing the world with.

Let's closely examine

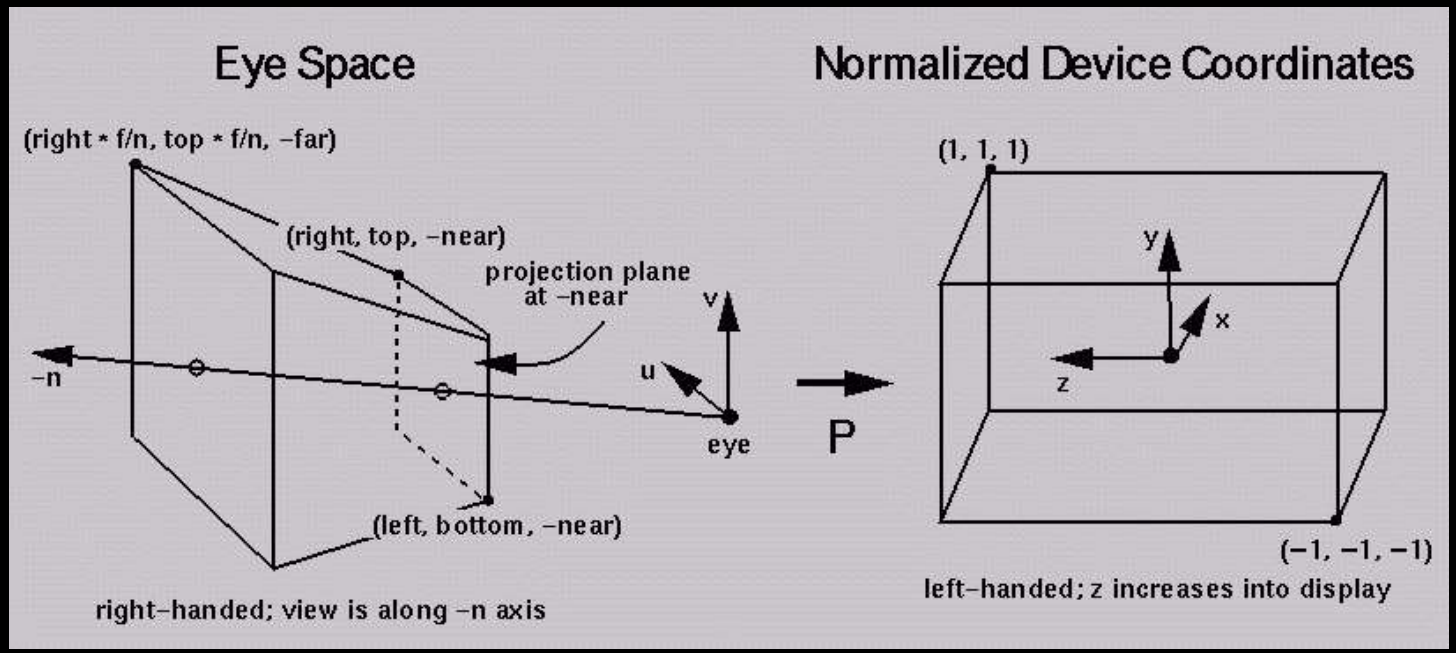
$$P_{Perspective\_Matrix} = \begin{bmatrix} \frac{z'}{h_{horiztonal}} & 0 & 0 & 0 \\ 0 & \frac{z'}{h_{vertical}} & 0 & 0 \\ 0 & 0 & \frac{far}{far - near} & \frac{-near * far}{far - near} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\frac{z'}{h_{horiztonal}} = \frac{2 * z'}{x_{\max} - x_{\min}}$$

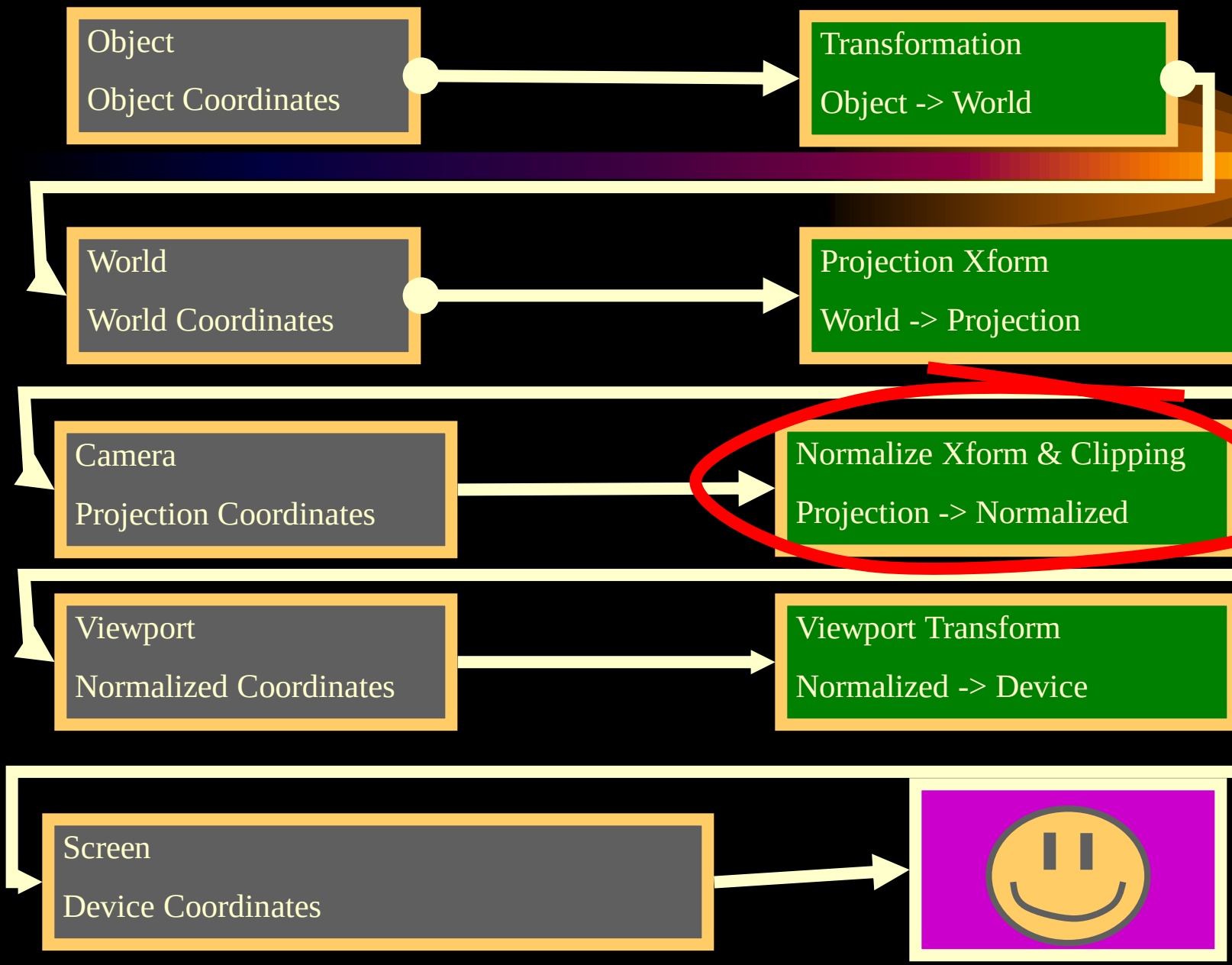
$$\frac{z'}{h_{vertical}} = \frac{2 * z'}{y_{\max} - y_{\min}}$$

What the Perspective Matrix means

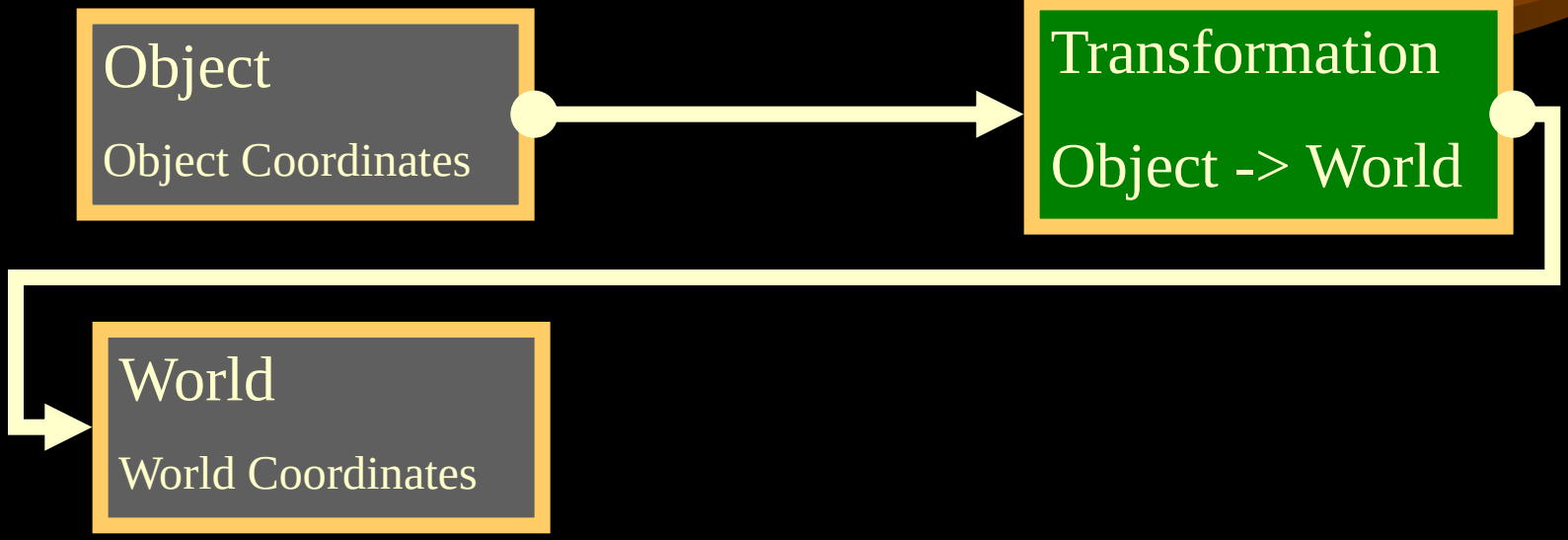
Note: Normalized Device Coordinates are a LEFT-HANDED Coordinate system



Graphics Pipeline So Far

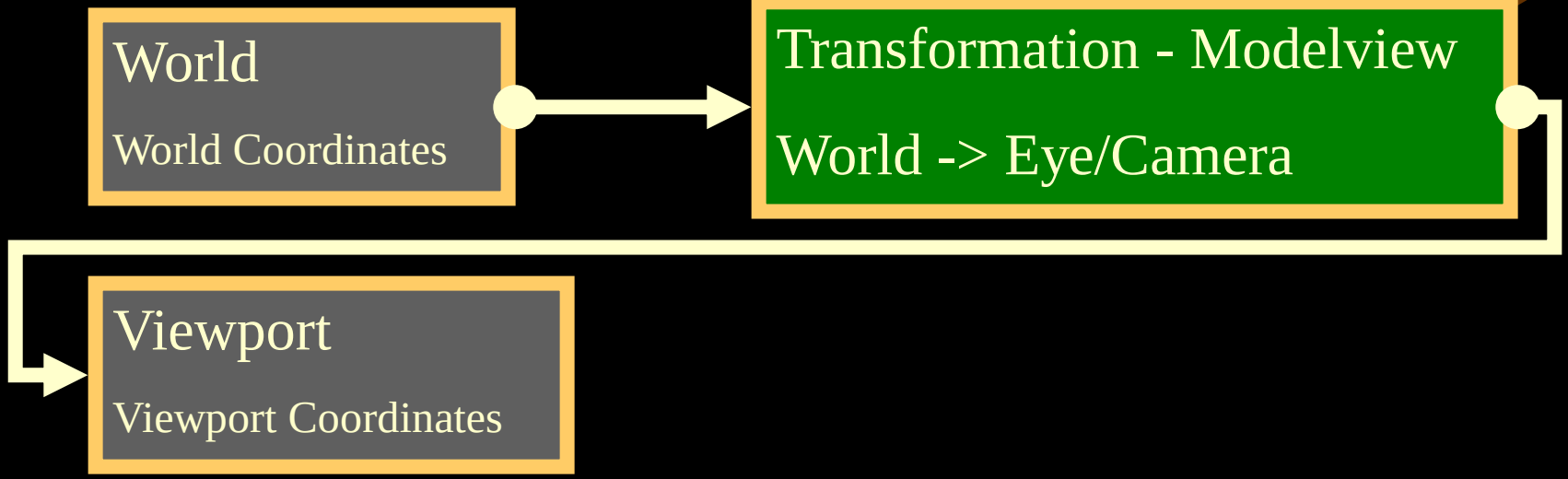


What happens to an object...



$$\begin{bmatrix} x'_{world\_coordinates} \\ y'_{world\_coordinates} \\ z'_{world\_coordinates} \\ 1 \end{bmatrix} = M_{Object->World} \begin{bmatrix} x_{object\_coordinates} \\ y_{object\_coordinates} \\ z_{object\_coordinates} \\ 1 \end{bmatrix}$$

What happens to an object...



$$\begin{bmatrix} x'_{view\_coordinates} \\ y'_{view\_coordinates} \\ z'_{view\_coordinates} \\ 1 \end{bmatrix} = M_{World->View} \begin{bmatrix} x'_{world\_coordinates} \\ y'_{world\_coordinates} \\ z'_{world\_coordinates} \\ 1 \end{bmatrix}$$

$$M_{World->View} = M_{Modelview}$$

What happens to an object...

Viewport

Viewport Coordinates

Transformation - Projection

(Includes Perspective Divide)

Eye/Camera ->View Plane

Rasterization

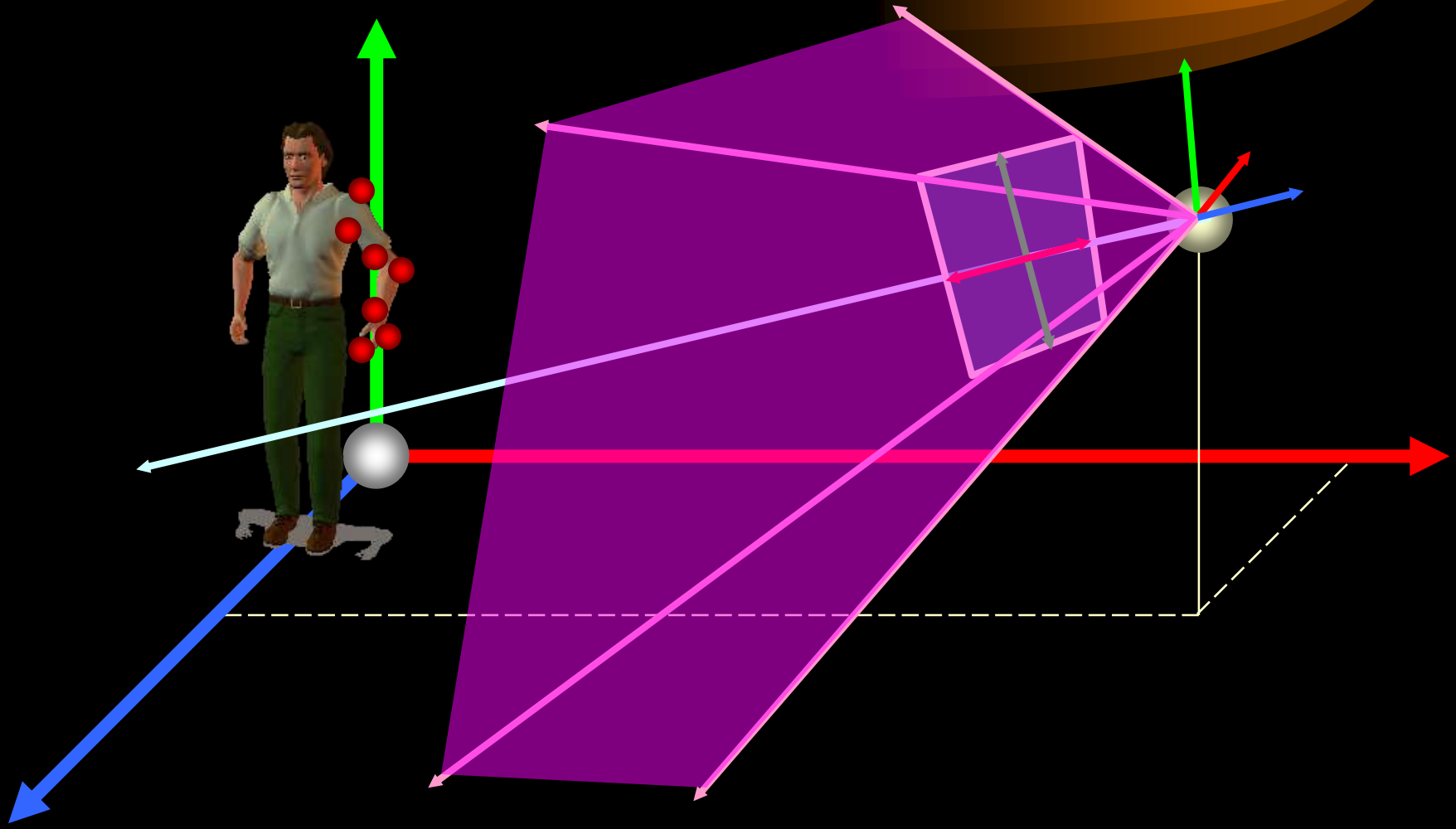
Scan Converting Triangles

$$\begin{bmatrix} x'_{normalized\_screen\_coordinates} \\ y'_{normalized\_screen\_coordinates} \\ z'_{normalized\_screen\_coordinates} \\ 1 \end{bmatrix} = M_{View->Screen} \begin{bmatrix} x'_{view\_coordinates} \\ y'_{view\_coordinates} \\ z'_{view\_coordinates} \\ 1 \end{bmatrix}$$

$$M_{View->Screen} = M_{Parallel\_Projection}$$

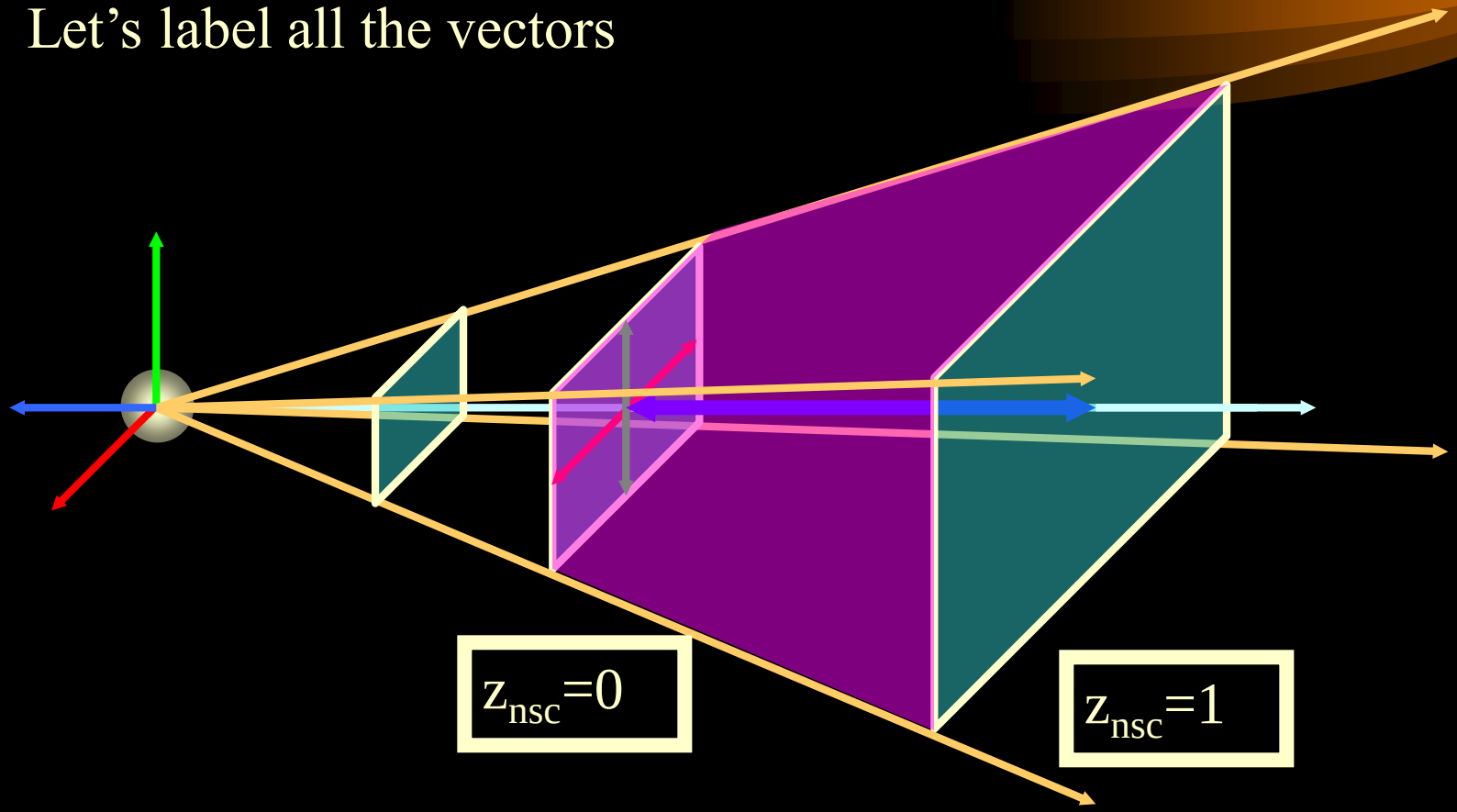
$$M_{View->Screen} = M_{Perspective\_Projection}$$

Normalized Screen Coordinates



Normalized Screen Coordinates

Let's label all the vectors

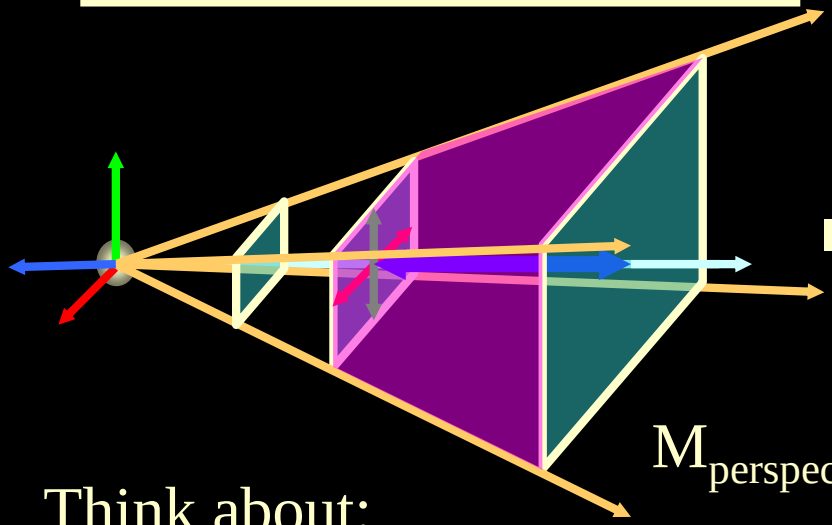


View Volume (View Frustum)

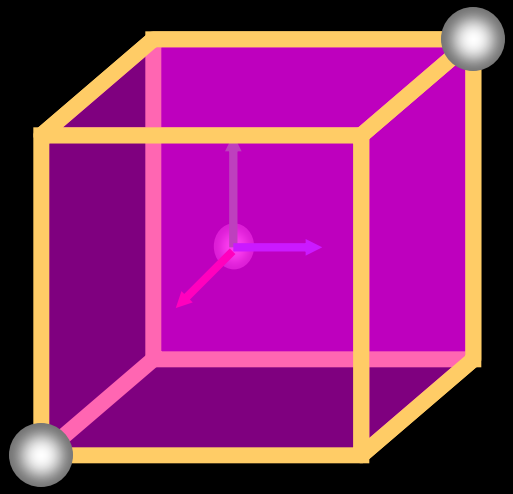
Usually:

View Plane = Near Plane

Far Plane



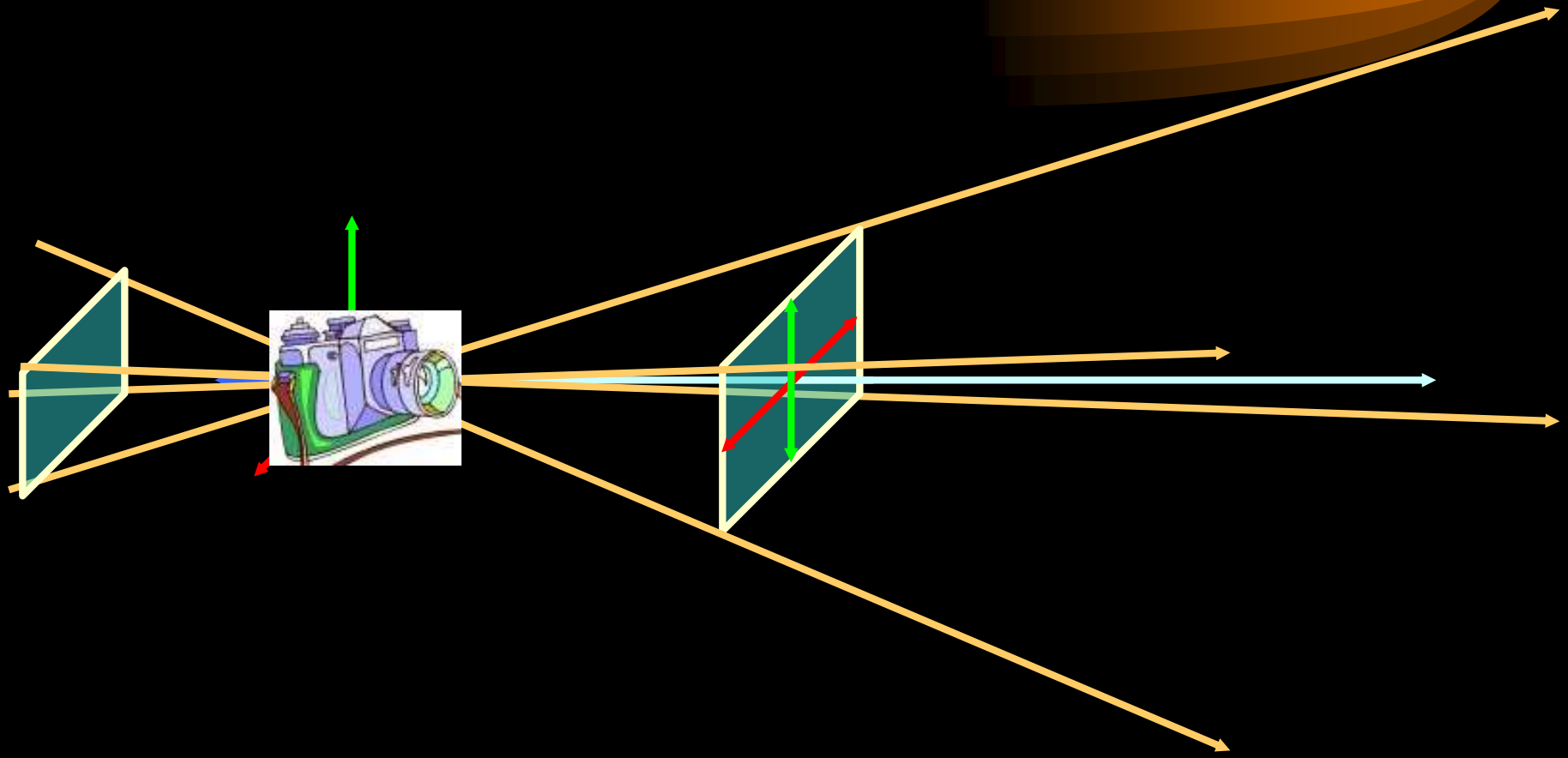
$M_{perspective\_Matrix}$



Think about:

Clipping 3D Triangles
View Frustum Culling

Comparison with a camera



Let's verbalize what's going on

Review:

- Pipeline
- Series of steps
- What we'll do next:
  - Hidden Surface Removal
  - Depth Buffers
  - Lighting
  - Shading
  - Blending (the elusive alpha)
  - Textures