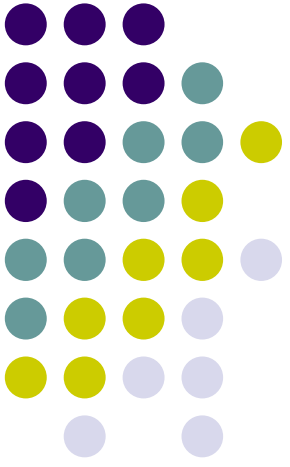
Computer Graphics (CS 4731)
Cameras

Joshua Cuneo

Computer Science Dept.

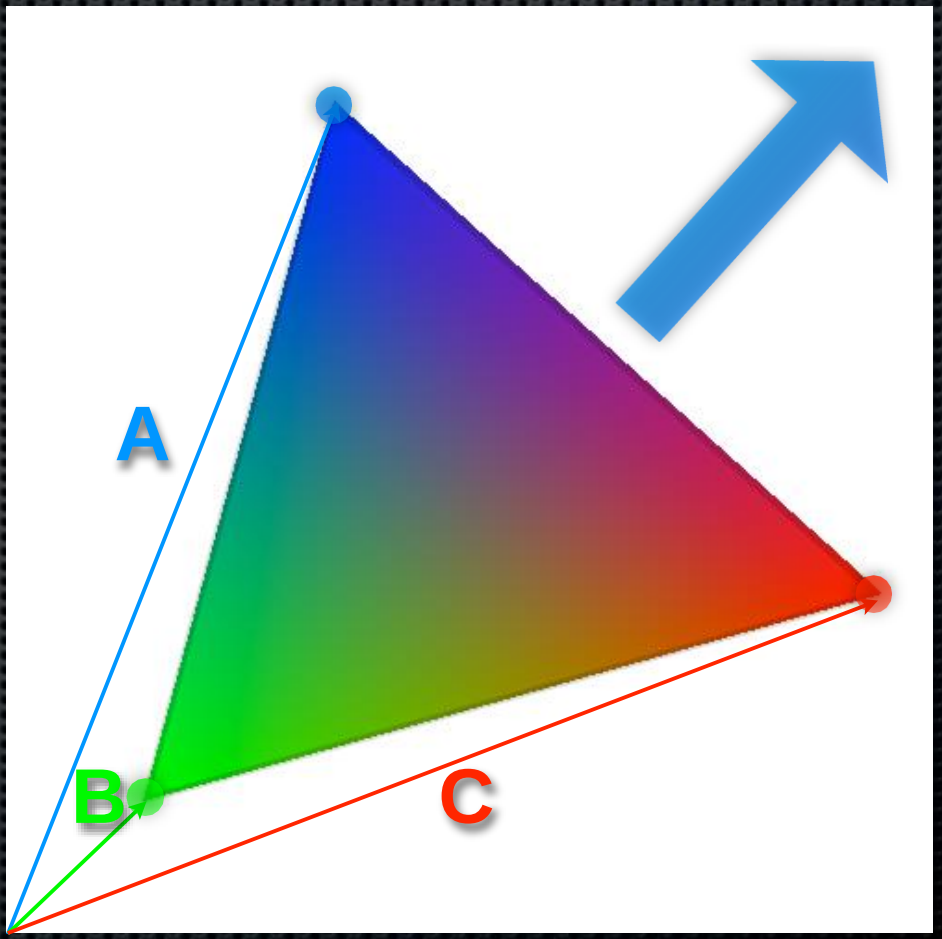
Worcester Polytechnic Institute (WPI)



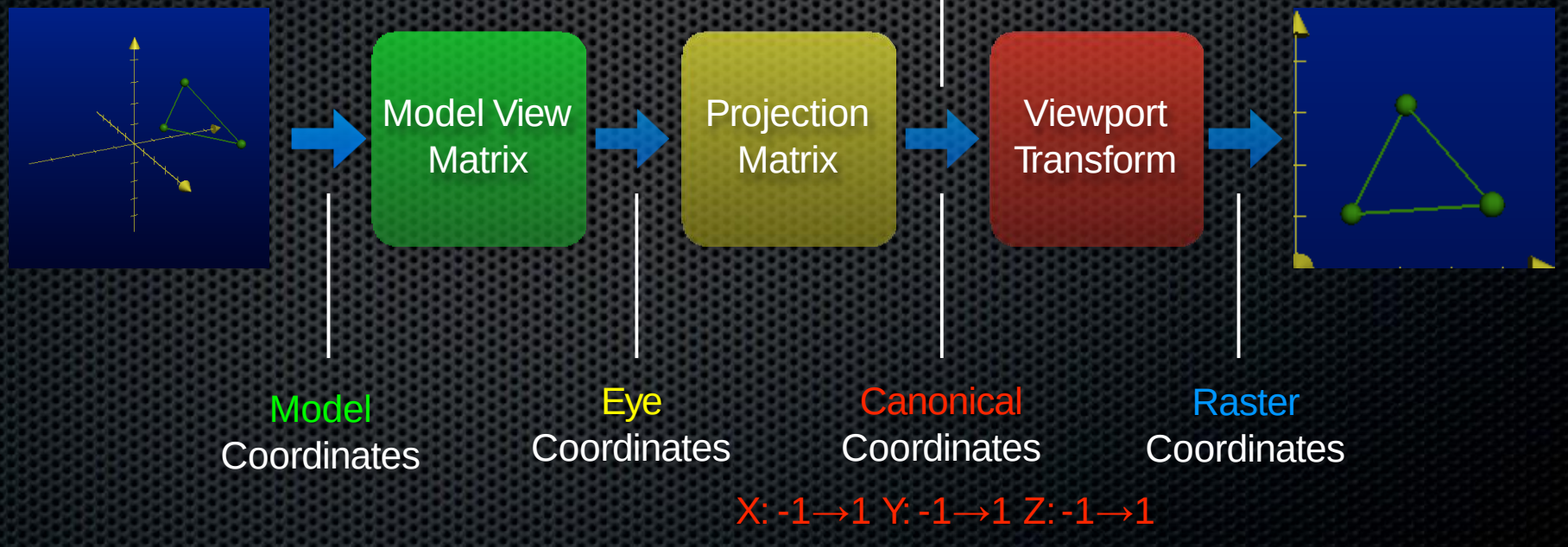
Slides Courtesy of Emmanuel Agu (WPI) and Matt Stoker (Utah)

What Can We Do So Far?

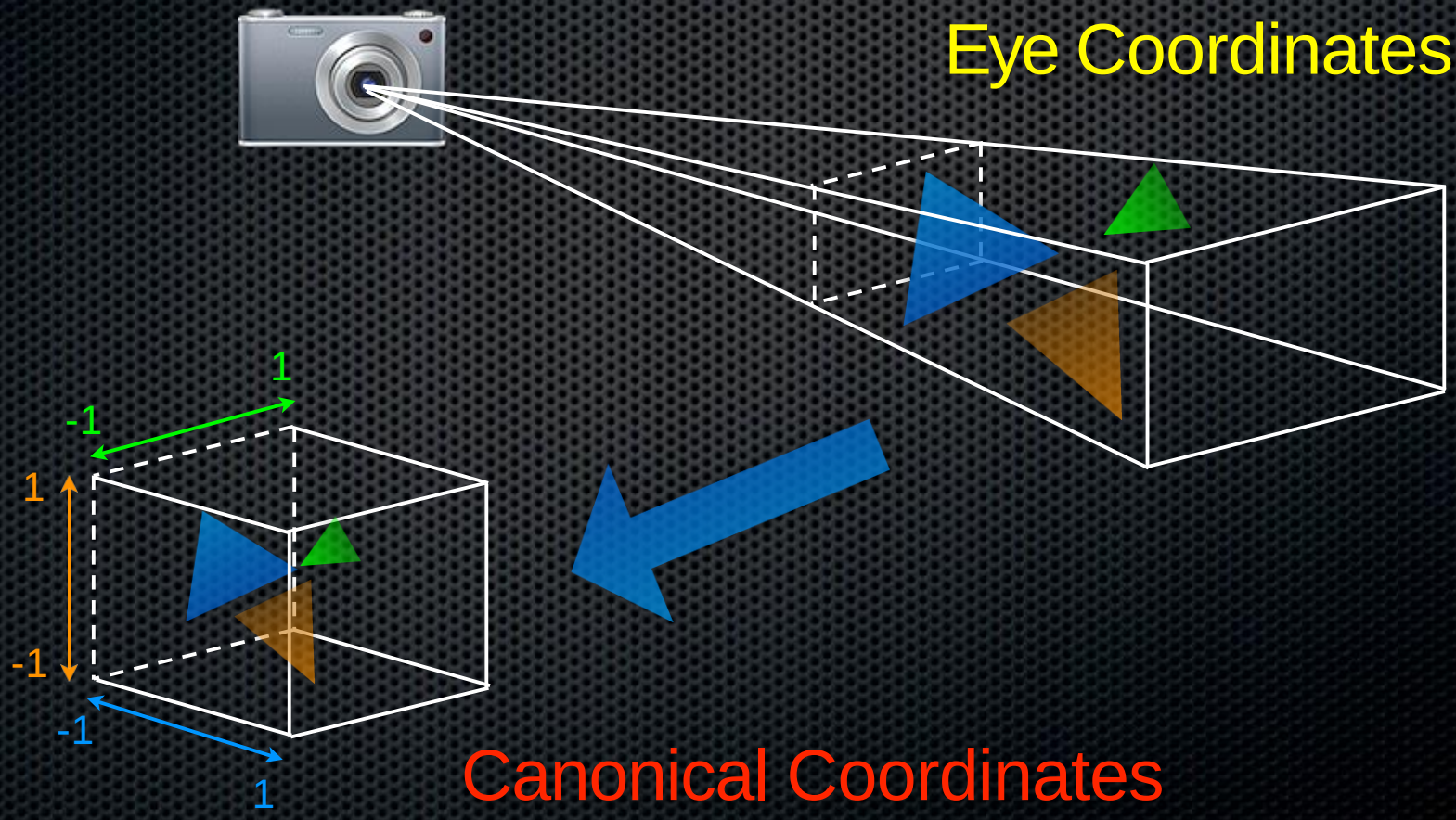
- Create and save rasters
- Draw triangles using interpolated colors
- Transform 3D input geometry



Typical Matrices



Projection Matrix

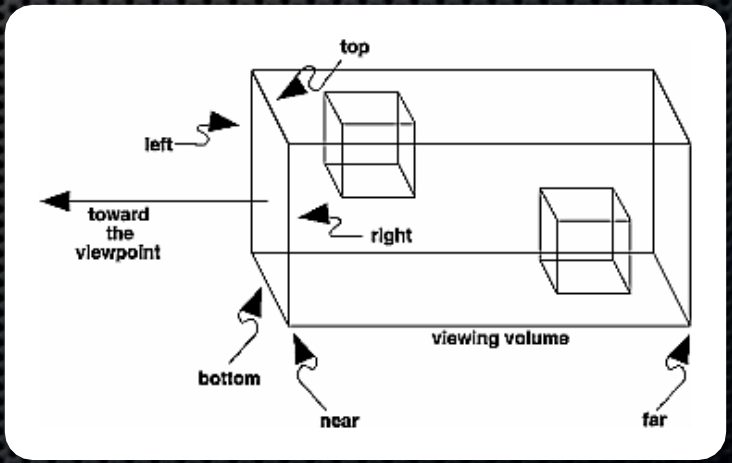


Projection - Orthographic

Combination Scale & Translation

ortho(l,r,b,t,n,f)

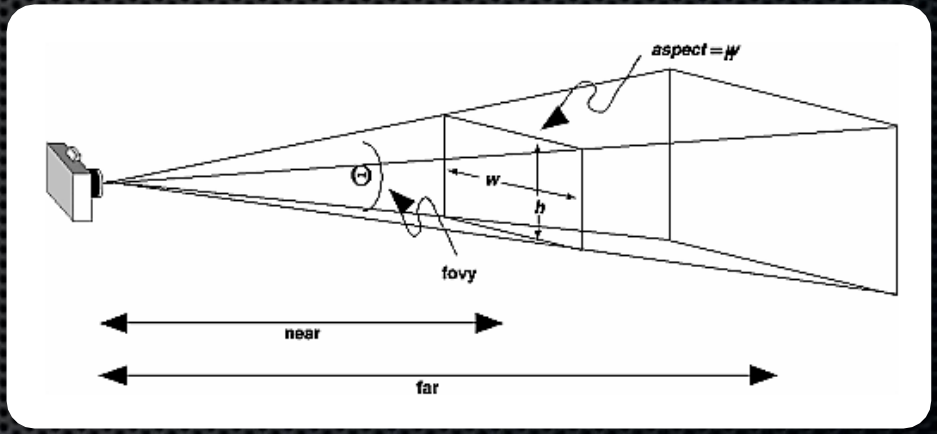
$$R = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } R^{-1} = \begin{bmatrix} \frac{r-l}{2} & 0 & 0 & \frac{r+l}{2} \\ 0 & \frac{t-b}{2} & 0 & \frac{t+b}{2} \\ 0 & 0 & \frac{f-n}{-2} & \frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Projection - Perspective

frustum(l,r,b,t,n,f)

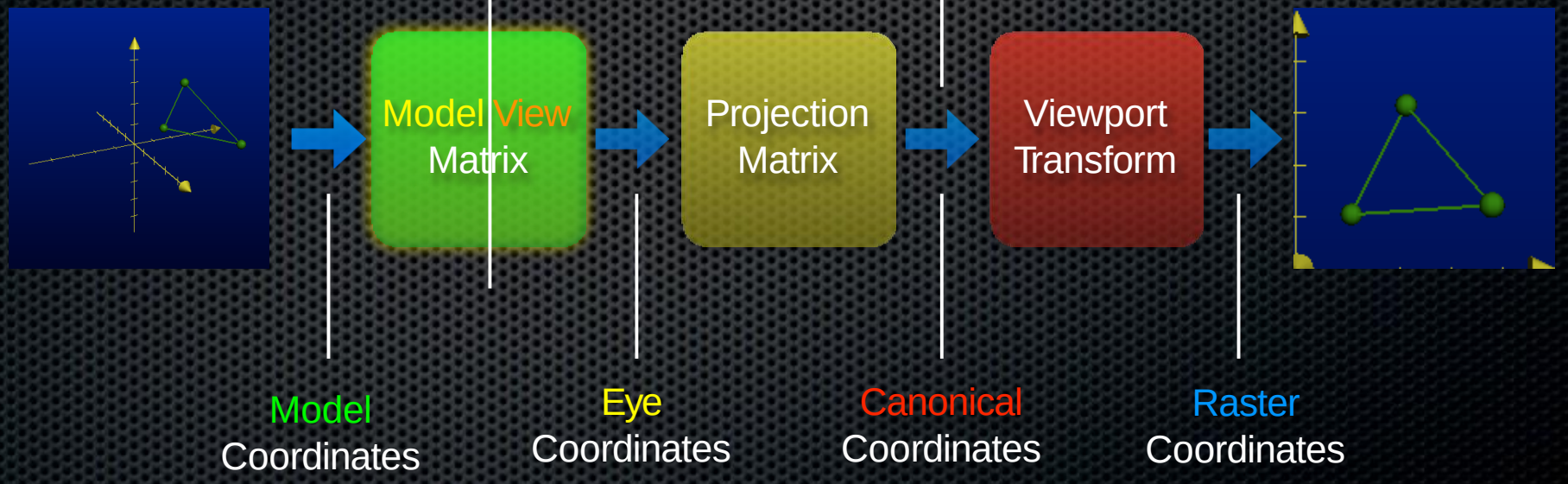
$$R = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \text{and } R^{-1} = \begin{bmatrix} \frac{r-l}{2n} & 0 & 0 & \frac{r+l}{2n} \\ 0 & \frac{t-b}{2n} & 0 & \frac{t+b}{2n} \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{-(f-n)}{2fn} & \frac{f+n}{2fn} \end{bmatrix}$$



v = fovY
a = aspect
n = near
f = far

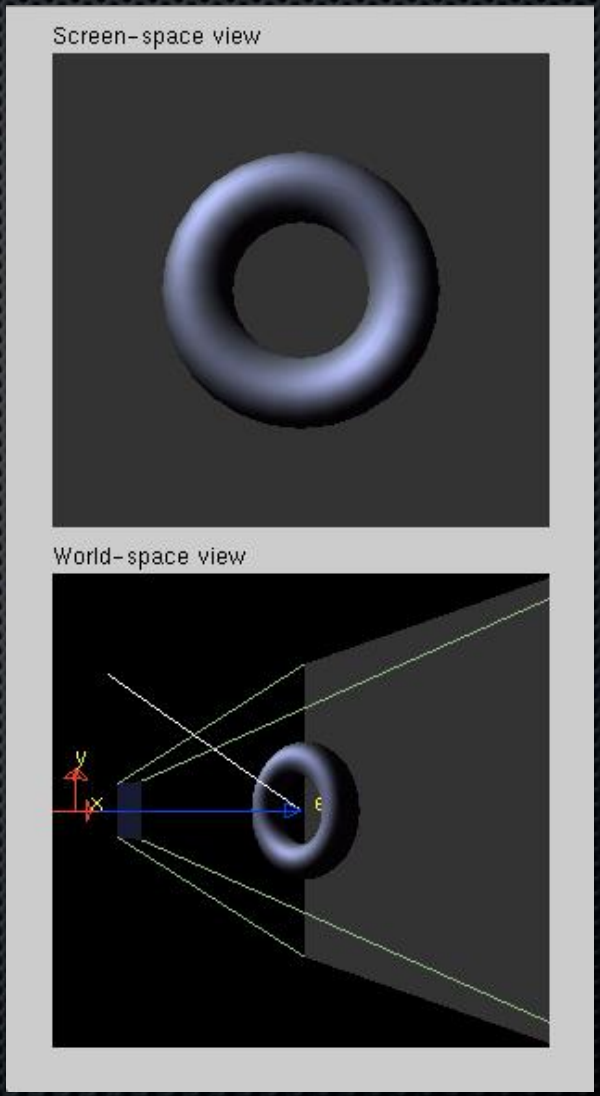
$$\begin{bmatrix} a/\tan(v/2) & 0 & 0 & 0 \\ 0 & 1/\tan(v/2) & 0 & 0 \\ 0 & 0 & -(f+n)/(f-n) & -2nf/(f-n) \end{bmatrix}$$

Typical Matrices



Eye Coordinates

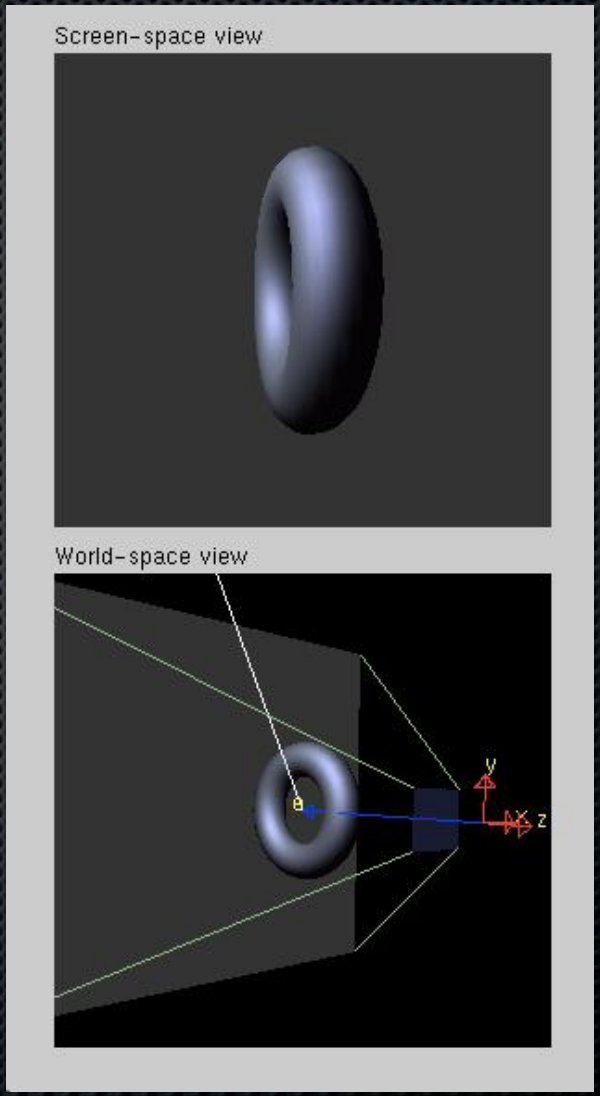
- User positioned at (0,0,0)
- +X axis to the user's right
- +Y axis points up
- User looks down -Z axis by the right hand rule



Eye Coordinates

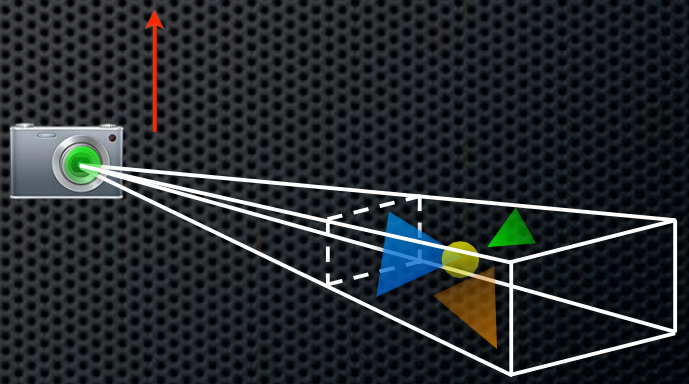
- User positioned at (0,0,0)
- **+**X axis to the user's right
- **+**Y axis points up
- User looks down **-**Z axis by the right hand rule

The world literally must revolve around you!



Better Definition

- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be “up”
- User looks down vector between eye point and spot point

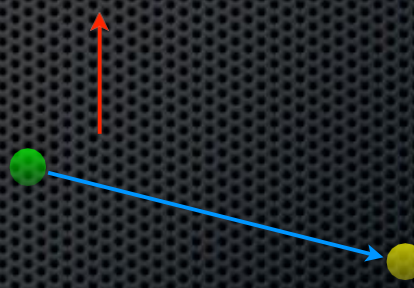


Better Definition

- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be “up”
- User looks down vector between eye point and spot point

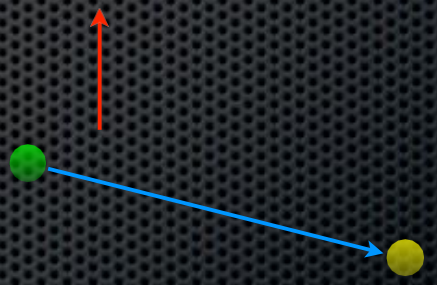
Better Definition

- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be "up"
- User looks down vector between eye point and spot point



Better Definition

- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be “up”
- User looks down look vector



Better Definition

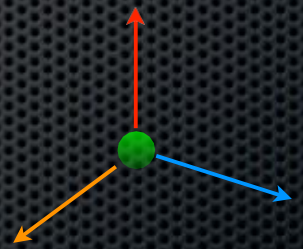
- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be “up”
- User looks down look vector

Better Definition

- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be “up”
- User looks down look vector
- There exists a vector describing the direction “right”

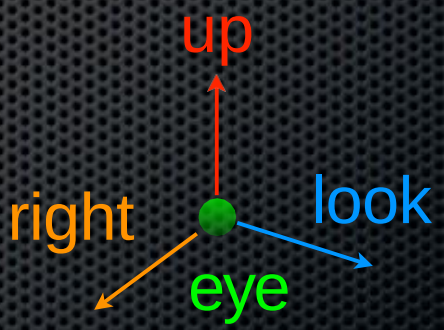
Better Definition

- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be “up”
- User looks down look vector
- There exists a vector describing the direction “right”

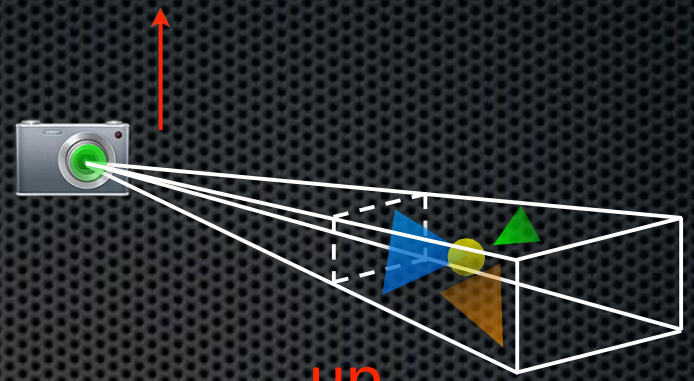


Better Definition

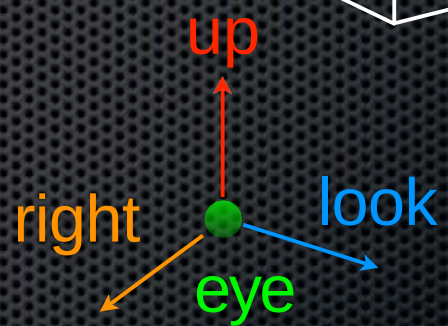
- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be “up”
- User looks down look vector
- There exists a vector describing the direction “right”



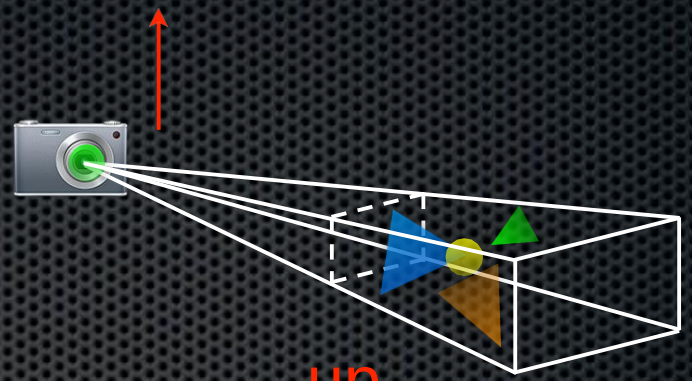
Camera Matrix



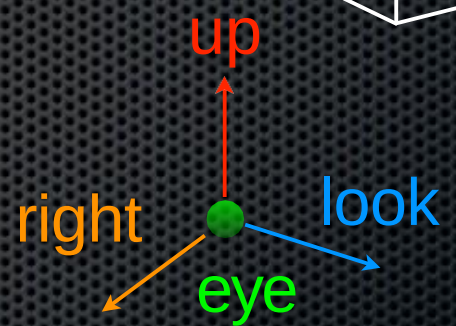
- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be “up”
- User looks down look vector
- There exists a vector describing the direction “right”



Camera Matrix

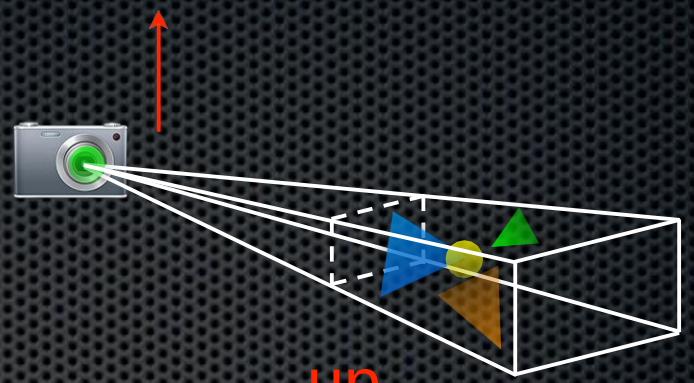


- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be “up”
- User looks down look vector
- There exists a vector describing the direction “right”

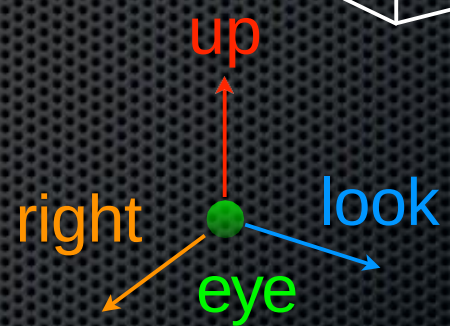


look = spot - eye

Camera Matrix



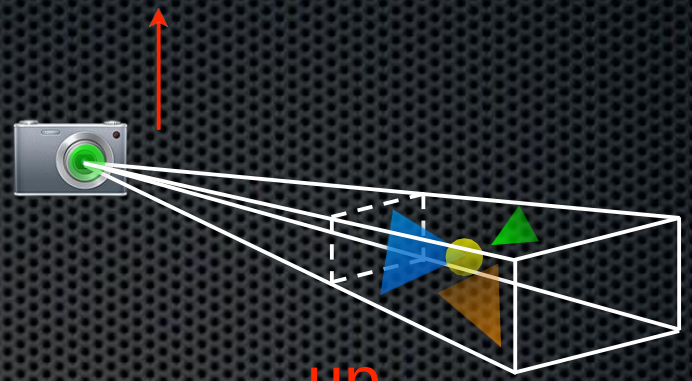
- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be “up”
- User looks down look vector
- There exists a vector describing the direction “right”



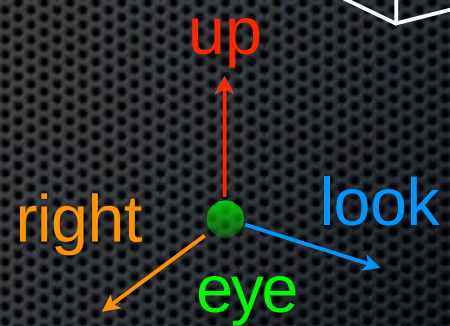
look = spot - eye

right = look x up

Camera Matrix



- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be "up"
- User looks down look vector
- There exists a vector describing the direction "right"

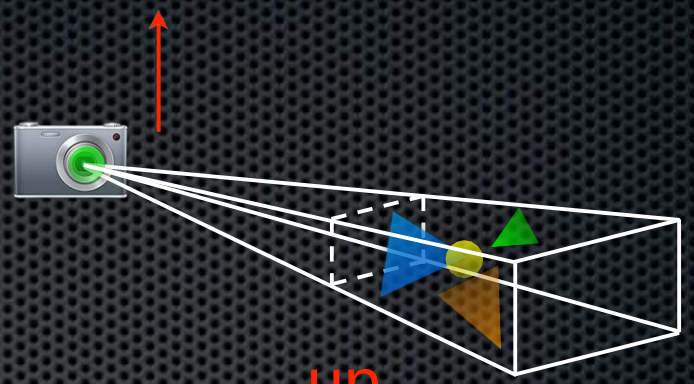


look = spot - eye

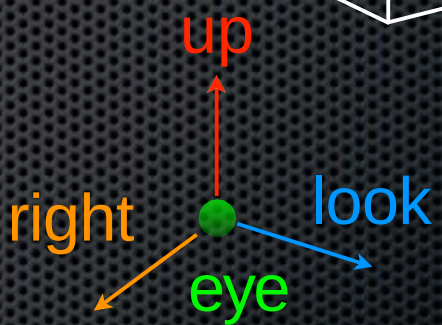
right = look x up

up = right x look

Camera Matrix



- User positioned at (eye_X,eye_Y,eye_Z)
- User looks at ($spot_X$, $spot_Y$, $spot_Z$)
- Any direction can be "up"
- User looks down look vector
- There exists a vector describing the direction "right"



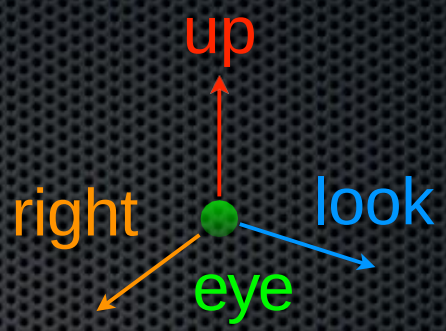
look = spot - eye

right = look x up

up = right x look

look, right, up are normalized

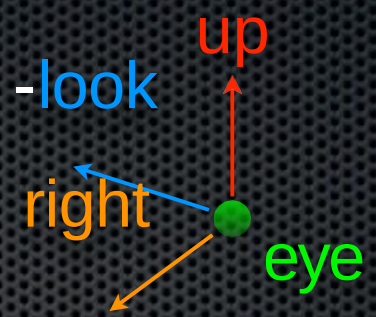
Camera Matrix



Change of Coordinate System Matrix

$$\begin{bmatrix} right_x & right_y & right_z & 0 \\ up_x & up_y & up_z & 0 \\ look_x & look_y & look_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

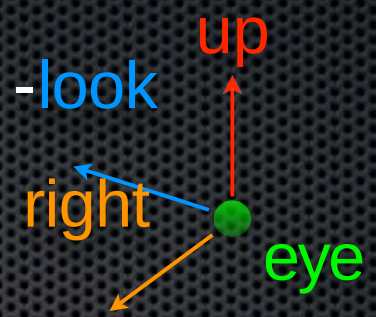
Camera Matrix



Change of Coordinate System Matrix

$$\begin{bmatrix} right_x & right_y & right_z & 0 \\ up_x & up_y & up_z & 0 \\ -look_x & -look_y & -look_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera Matrix



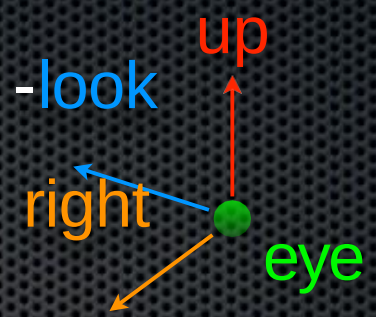
Change of Coordinate System Matrix

$$\begin{bmatrix} right_x & right_y & right_z & 0 \\ up_x & up_y & up_z & 0 \\ -look_x & -look_y & -look_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translate Camera to Origin

Translate($-eye_x$, $-eye_y$, $-eye_z$)

Camera Matrix



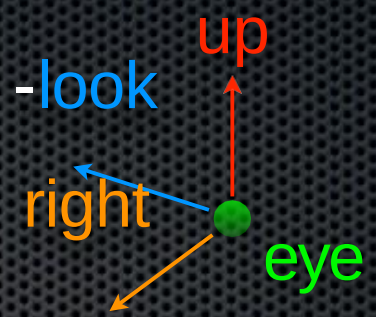
Change of Coordinate System Matrix

$$B = \begin{bmatrix} right_x & right_y & right_z & 0 \\ up_x & up_y & up_z & 0 \\ -look_x & -look_y & -look_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translate Camera to Origin

$$T = \text{Translate}(-eye_x, -eye_y, -eye_z)$$

Camera Matrix



Change of Coordinate System Matrix

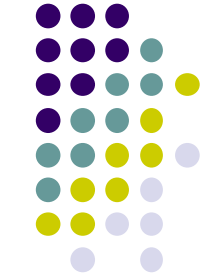
$$B = \begin{bmatrix} right_x & right_y & right_z & 0 \\ up_x & up_y & up_z & 0 \\ -look_x & -look_y & -look_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translate Camera to Origin

$$T = \text{Translate}(-eye_x, -eye_y, -eye_z)$$

$$M = BT$$

The LookAt Function



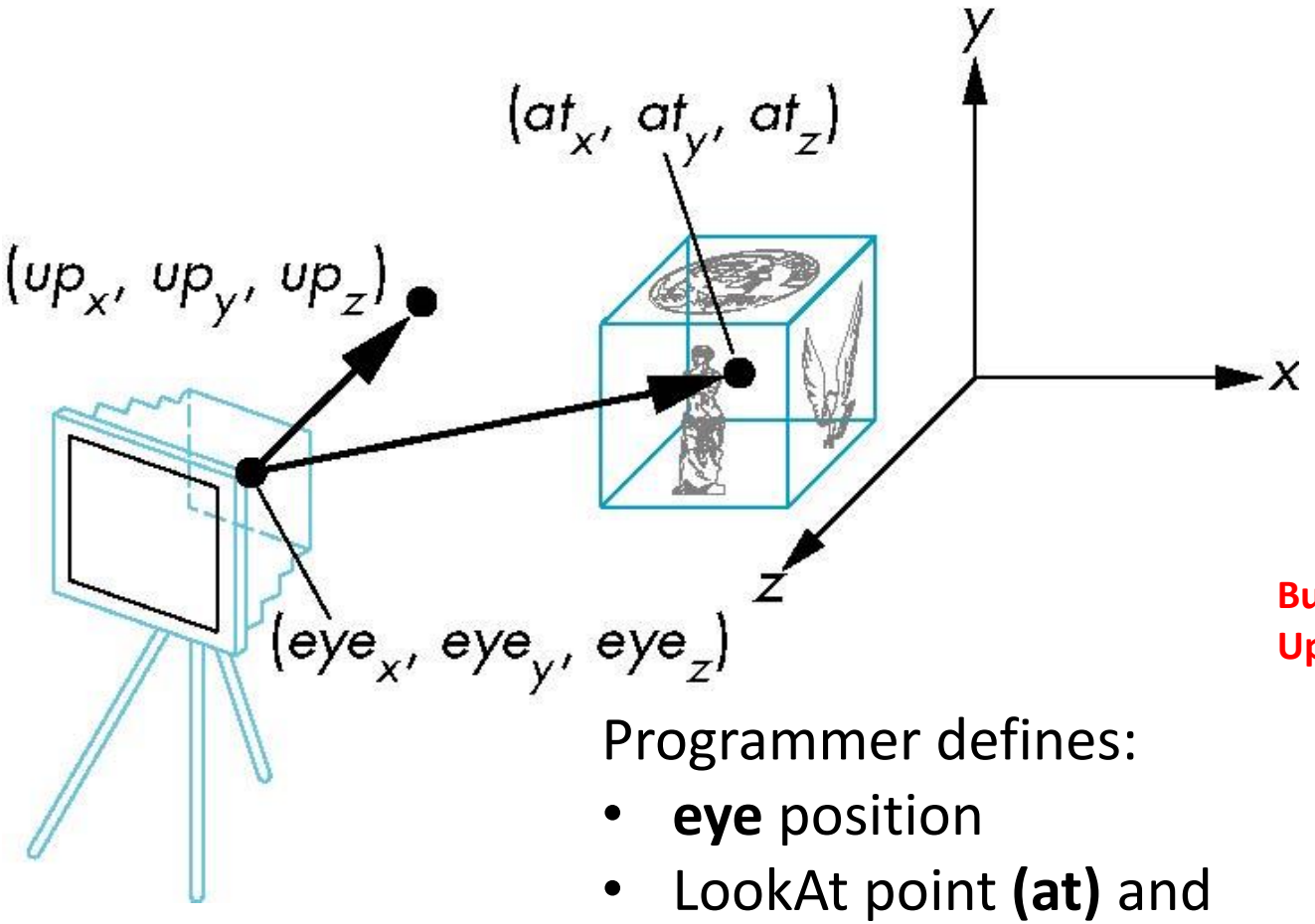
- Sets camera position, transforms object distances to camera frame

```
mat4 mv = gl.lookAt(vec3 eye, vec3 at, vec3 up);
```

Builds 4x4 matrix for positioning, orienting Camera and puts it into variable **mv**

The LookAt Function

```
LookAt(eye, at, up)
```



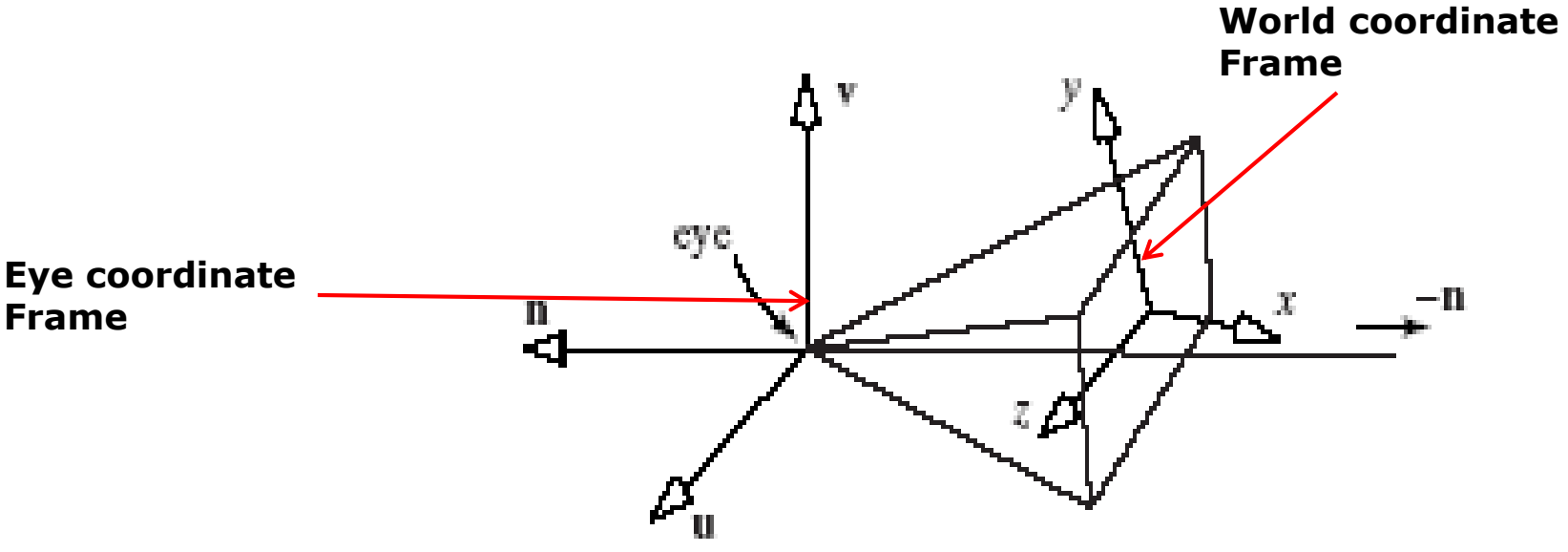
But Why do we set Up direction?

Programmer defines:

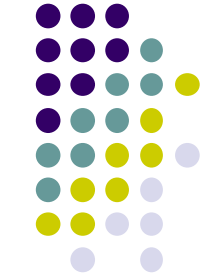
- **eye** position
- LookAt point **(at)** and
- **Up** vector (***Up*** direction usually (0,1,0))

What does LookAt do?

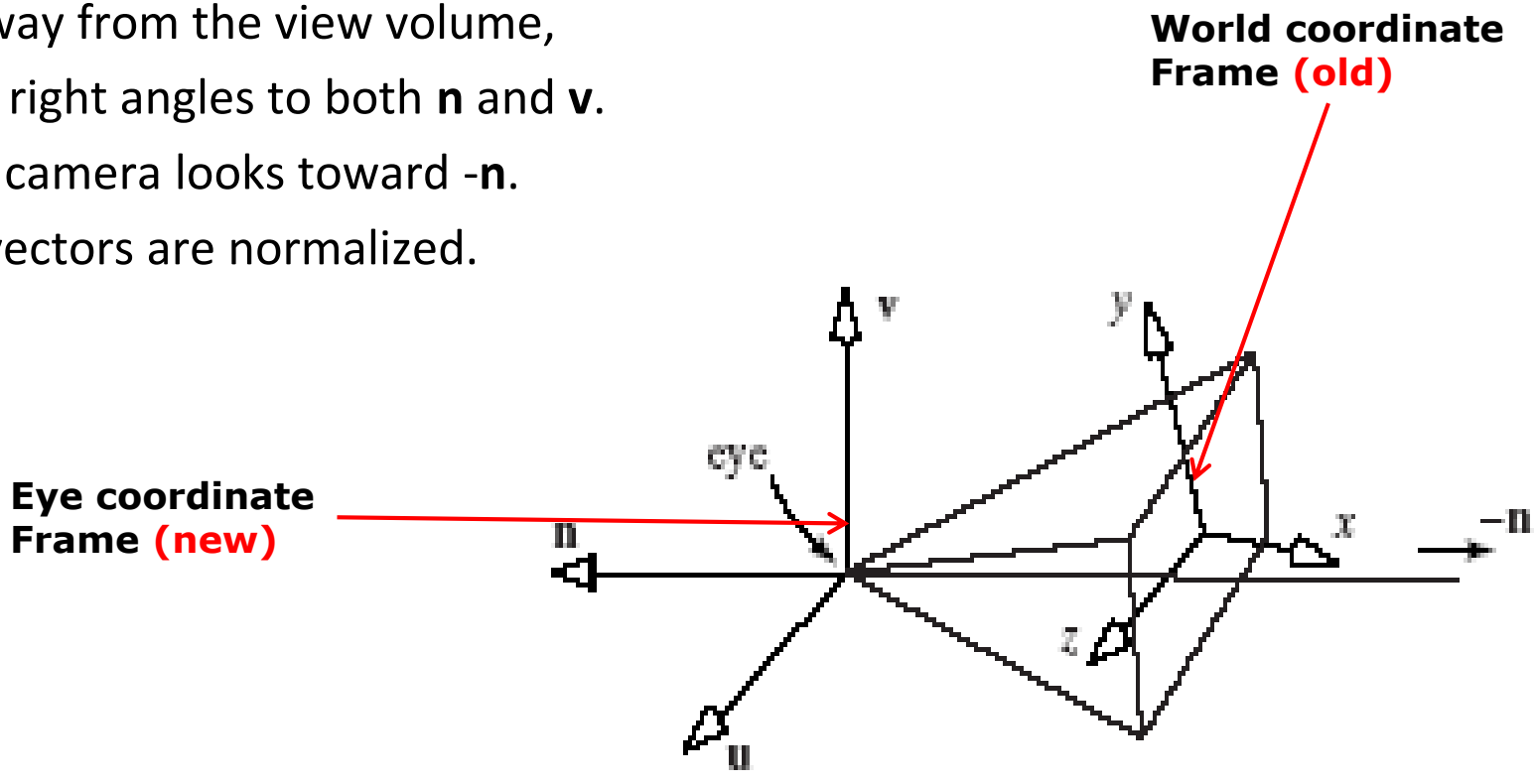
- Programmer defines eye, lookAt and Up
- **LookAt method:**
  - Forms new axes (u, v, n) at camera
  - Transform objects from world to eye camera frame



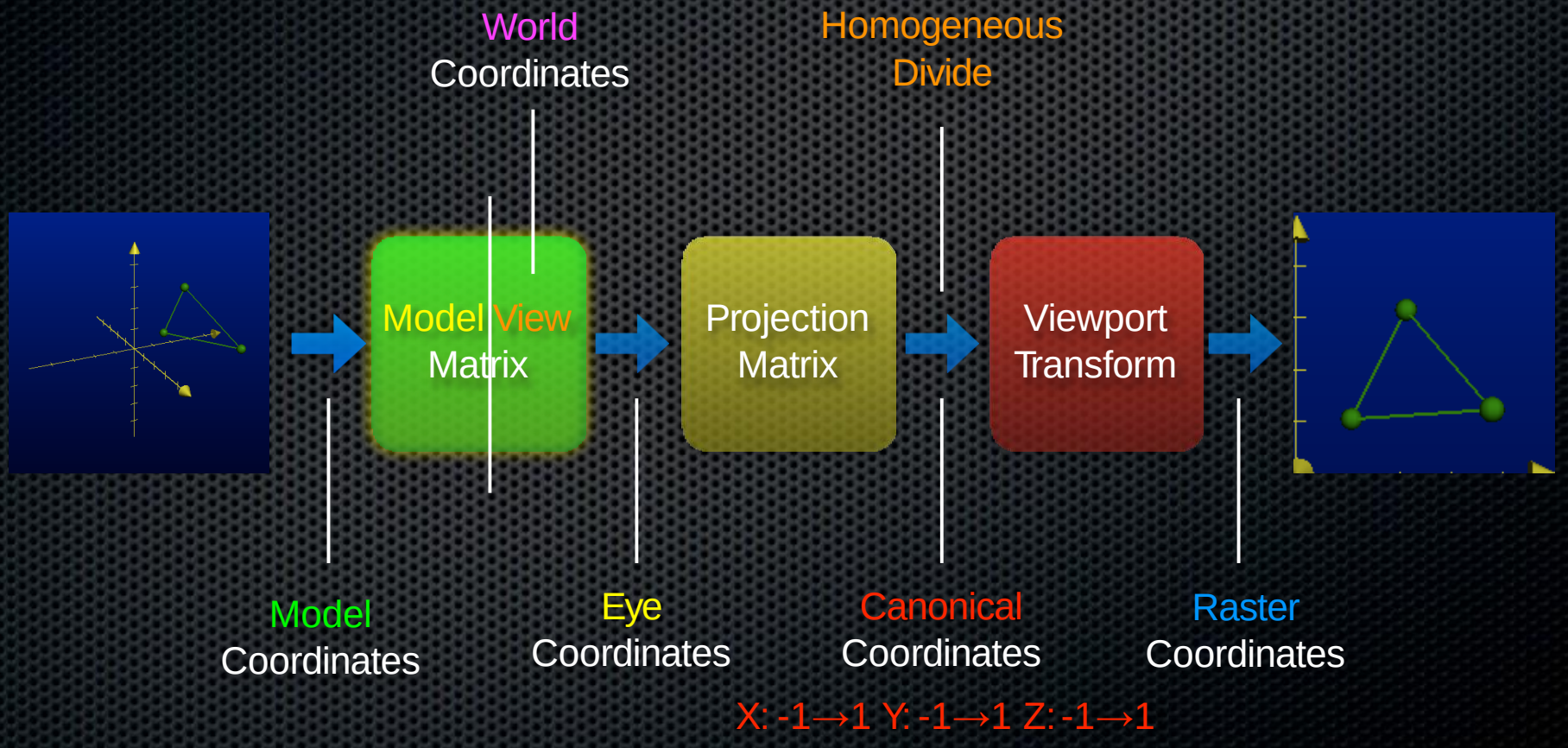
Camera with Arbitrary Orientation and Position



- Define new axes (u, v, n) at eye
  - **v** points vertically upward,
  - **n** away from the view volume,
  - **u** at right angles to both **n** and **v**.
  - The camera looks toward -**n**.
  - All vectors are normalized.



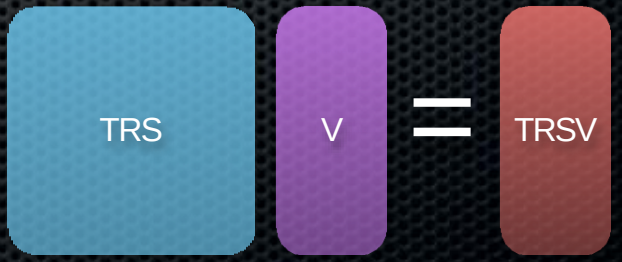
Typical Matrices



Either This
10,000 Times

Notice the ordering!

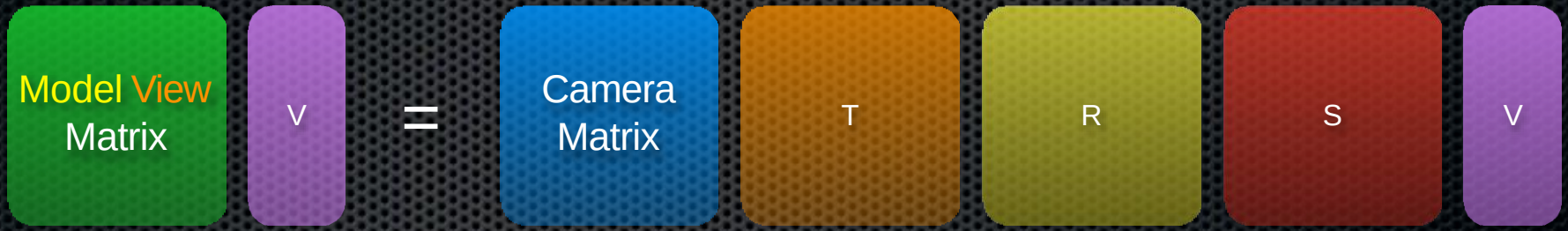
And This
10,000 Times



Model View Matrix Parts

Camera matrix is concatenated **first**

Model View Matrix Parts



Camera matrix is concatenated **first**