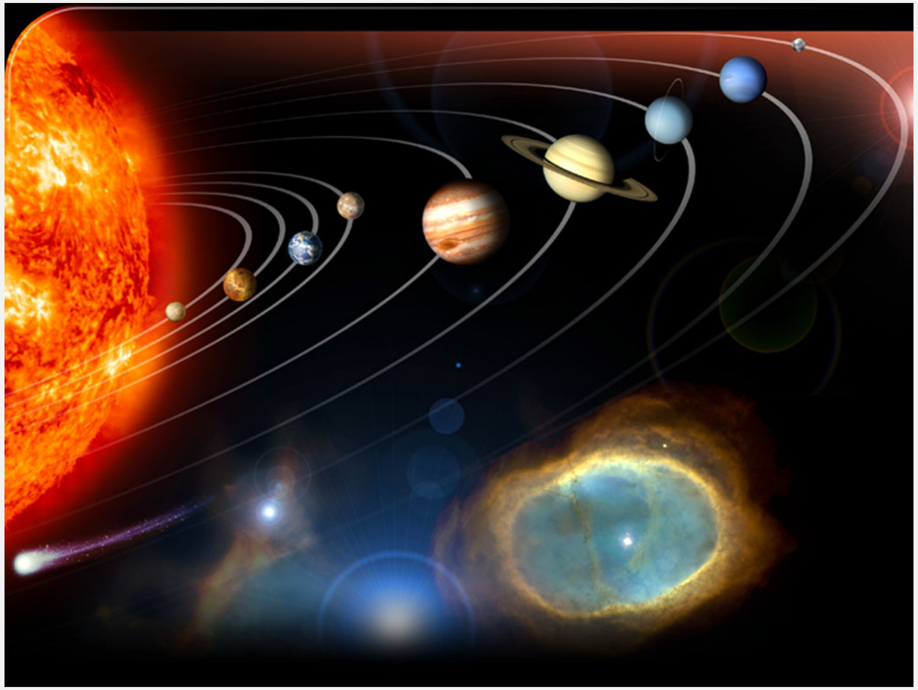
Advanced Transformation

Transformations for Hierarchical Objects

Solar System

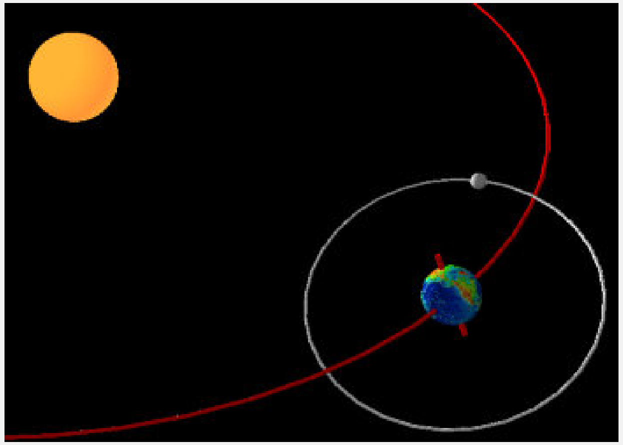
- An example of hierarchical objects
  - How can we design transformations for hierarhical objects



(video, youtube)

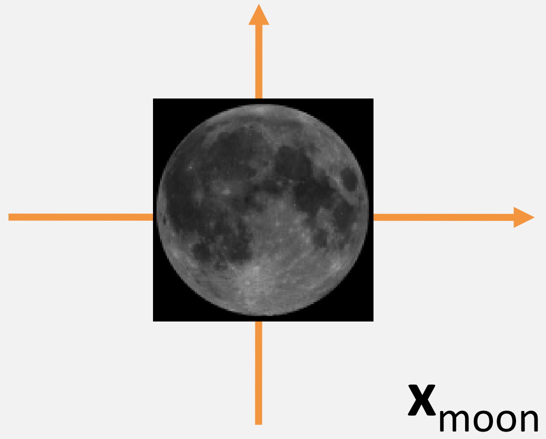
Sun – Earth – Moon

- Sun
- Earth
  - rotating itself
  - orbiting around the sun
- Moon
  - rotating itself
  - orbiting around the earth



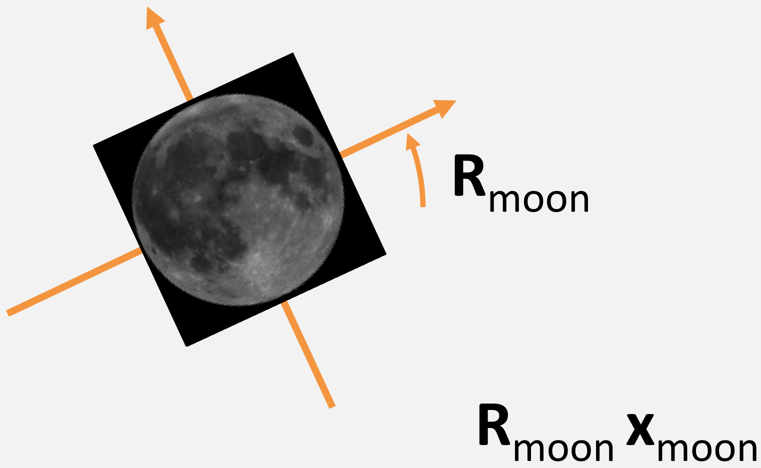
Hierarchical Transformation

- Moon
  - Modeling the moon



Hierarchical Transformation

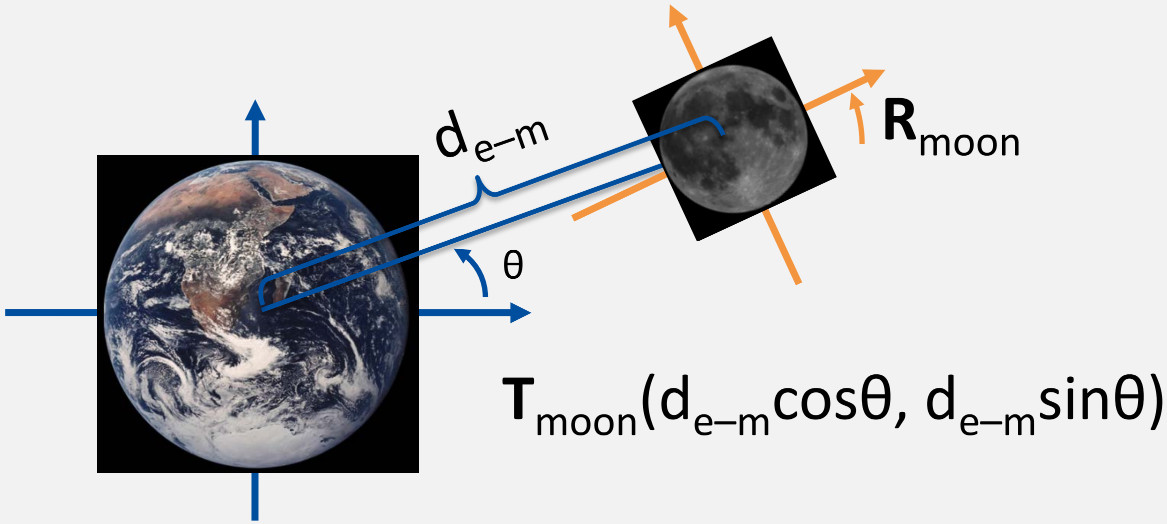
- Moon
  - Modeling the moon
  - Rotating itself



Hierarchical Transformation

- Earth – Moon
  - The moon is orbiting around the earth

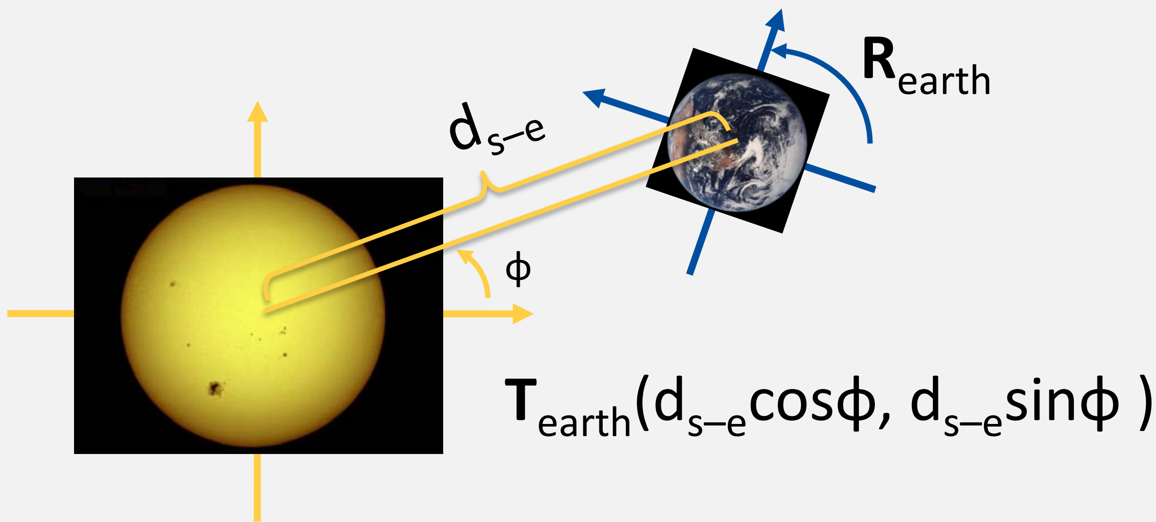
$$\mathbf{x}_{earth}=\mathbf{T}_{moon}(d_{e\text{–}m}\cos\theta,\ d_{e\text{–}m}\sin\theta)\mathbf{R}_{moon}\ \mathbf{x}_{moon}$$



Hierarchical Transformation

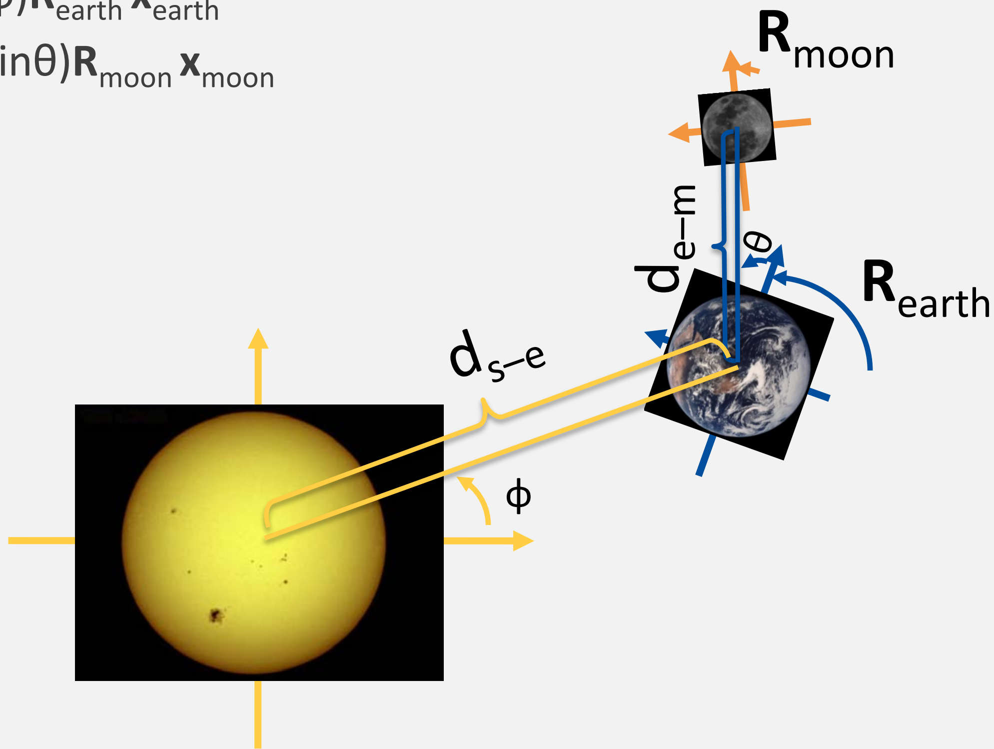
- Sun – Earth
  - The earth is orbiting around the sun

$$\mathbf{x}_{sun} = \mathbf{T}_{earth}(d_{s-e}\cos\phi,\ d_{s-e}\sin\phi)\mathbf{R}_{earth}\ \mathbf{x}_{earth}$$



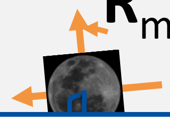
Hierarchical Transformation

- Sun – Earth – Moon
  - $\mathbf{x}_{sun}=\mathbf{T}_{earth}(d_{s-e}\cos\phi,\ d_{s-e}\sin\phi)\mathbf{R}_{earth}\ \mathbf{x}_{earth}$
  - $\mathbf{x}_{earth}=\mathbf{T}_{moon}(d_{e-m}\cos\theta,\ d_{e-m}\sin\theta)\mathbf{R}_{moon}\ \mathbf{x}_{moon}$



Hierarchical Transformation

- Sun – Earth – Moon
  - $\mathbf{x}_{sun}=\mathbf{T}_{earth}(d_{s-e}\cos\phi,\ d_{s-e}\sin\phi)\mathbf{R}_{earth}\ \mathbf{x}_{earth}$
  - $\mathbf{x}_{earth}=\mathbf{T}_{moon}(d_{e-m}\cos\theta,\ d_{e-m}\sin\theta)\mathbf{R}_{moon}\ \mathbf{x}_{moon}$

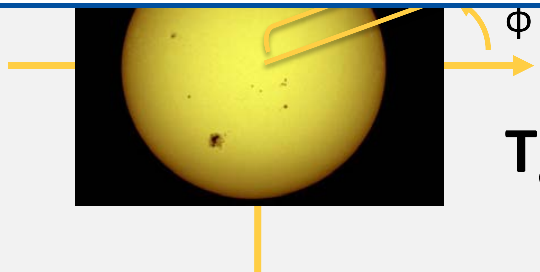


Transformation of the Earth w.r.t. the frame of the Sun

$$\mathbf{x}_{sun}=\mathbf{T}_{earth}(d_{s-e}\cos\phi,\ d_{s-e}\sin\phi)\mathbf{R}_{earth}\ \mathbf{x}_{earth}$$

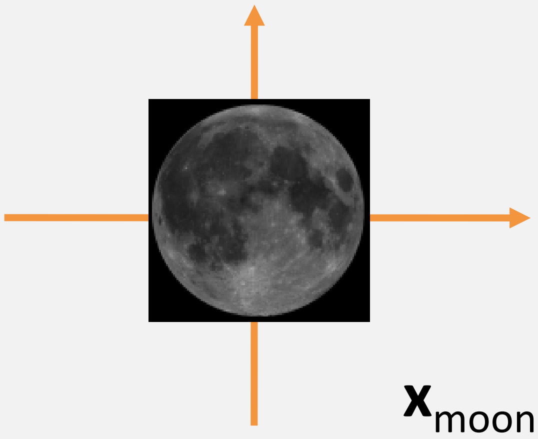
Transformation of the Moon w.r.t. the frame of the Sun

$$\mathbf{x}_{sun}=\mathbf{T}_{earth}(d_{s-e}\cos\phi,\ d_{s-e}\sin\phi)\mathbf{R}_{earth}\mathbf{T}_{moon}(d_{e-m}\cos\theta,\ d_{e-m}\sin\theta)\mathbf{R}_{moon}\ \mathbf{x}_{moon}$$



OpenGL ES 1.x Codes – Hierarchical Transformation

- Moon
  - Modeling the moon

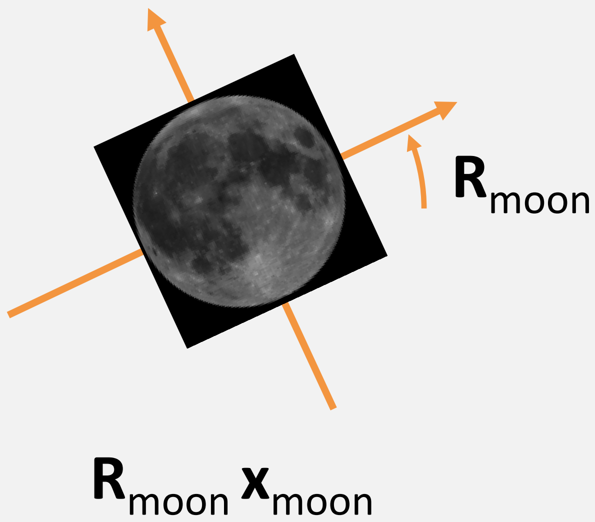



```
void draw_moon()
{
  glEnableClientState(GL_VERTEX_ARRAY);
  glEnableClientState(GL_COLOR_ARRAY);
  glVertexPointer(…);
  glColorPointer(…);
  glDrawElements(…);
  glDiableClientState(GL_VERTEX_ARRAY);
  glDiableClientState(GL_COLOR_ARRAY);
}
```

OpenGL ES 1.x Codes – Hierarchical Transformation

- Moon
  - Modeling the moon
  - Rotating itself

```
void draw_earth_system()
{
  // ...
  glRotatef(…);     // rotating the Moon
  draw_moon();
}

void draw_moon()   { // … glDrawElements(); … }
```

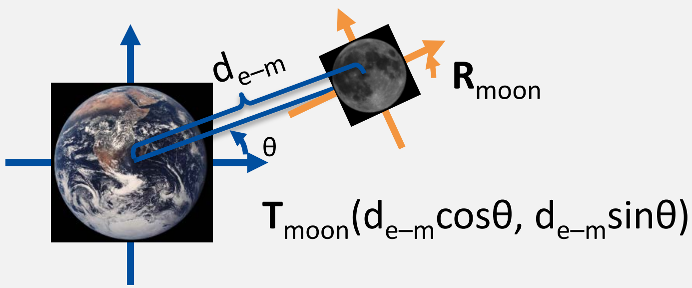


OpenGL ES 1.x Codes – Hierarchical Transformation

- Earth – Moon
  - The moon is orbiting around the earth

```
void draw_earth_system()
{
  draw_earth();

  glTranslatef(…); // orbiting around the Earth
  glRotatef(…);    // rotating the Moon
  draw_moon();
}

void draw_moon()   { // … glDrawElements(); … }
void draw_earth()  { // … glDrawElements(); … }
```

$$\mathbf{x}_{earth}=\mathbf{T}_{moon}(d_{e-m}\cos\theta,\ d_{e-m}\sin\theta)\mathbf{R}_{moon}\ \mathbf{x}_{moon}$$

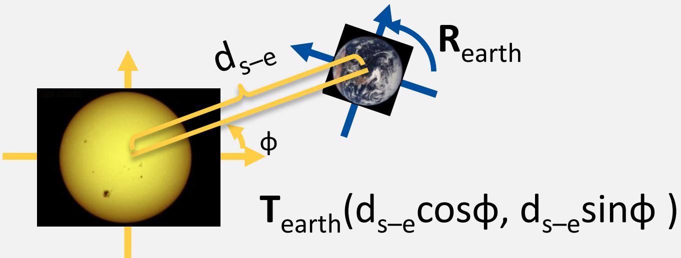


OpenGL ES 1.x Codes – Hierarchical Transformation

- Sun – Earth
  - The earth is orbiting around the sun

$$\boxed{\mathbf{x}_{sun}=\mathbf{T}_{earth}(d_{s\text{–}e}\cos\phi,\ d_{s\text{–}e}\sin\phi)\mathbf{R}_{earth}\ \mathbf{x}_{earth}}$$

```
void draw_sun_system()
{
  draw_sun();

  glTranslatef(…); // orbiting around the Sun
  glRotatef(…);    // rotating the Earth system
  draw_earth_system();
}

void draw_moon()   { // … glDrawElements(); … }
void draw_earth()  { // … glDrawElements(); … }
void draw_sun()    { // … glDrawElements(); … }
```

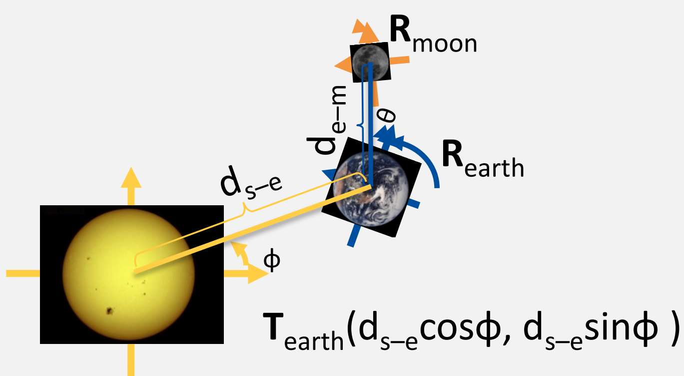


OpenGL ES 1.x Codes – Hierarchical Transformation

- Sun – Earth – Moon

$$\mathbf{x}_{earth} = \mathbf{T}_{moon}(d_{e-m}\cos\theta, d_{e-m}\sin\theta)\mathbf{R}_{moon}\ \mathbf{x}_{moon}$$

$$\mathbf{x}_{sun} = \mathbf{T}_{earth}(d_{s-e}\cos\phi, d_{s-e}\sin\phi)\mathbf{R}_{earth}\ \mathbf{x}_{earth}$$



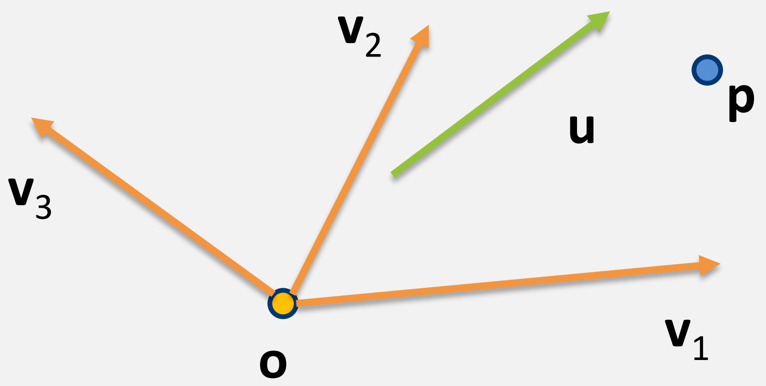

```
void draw_sun_system()
{
  draw_sun();

  glTranslatef(…); // orbiting around the Sun
  glRotatef(…);    // rotating the Earth system
  draw_earth_system();
}

void draw_earth_system()
{
  draw_earth();

  glTranslatef(…); // orbiting around the Earth
  glRotatef(…);    // rotating the Moon
  draw_moon();
}

void draw_moon()   { // … glDrawElements(); … }
void draw_earth()  { // … glDrawElements(); … }
void draw_sun()    { // … glDrawElements(); … }
```

View Transformations

Coordinate System (Frame)

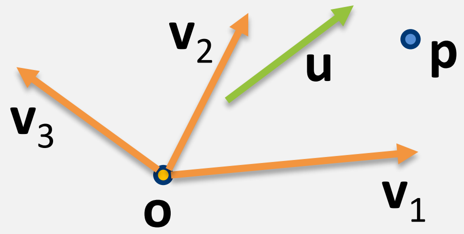
- A Coordinate system (or frame) consists of a set of basis vectors and an origin
  - A set of basis vectors: $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$
  - An origin: $\mathbf{o}$

- How to representing a vector **u** and a point **p** w.r.t. a given coordinate system?



Coordinate System (Frame)

- Consider a set of basis vectors and an origin
  - $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$
  - $\mathbf{o}$
- A vector **u** is written
  - $\mathbf{u} = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \ldots + \alpha_n\mathbf{v}_n + 0 \cdot \mathbf{o}$
  - The list of scalars, $\{\alpha_1, \alpha_2, \ldots. \alpha_n, 0\}$ is the representation of **u** w.r.t. the given coordinate system
- A point **p** is written
  - $\mathbf{p} = \beta_1\mathbf{v}_1 + \beta_2\mathbf{v}_2 + \ldots + \beta_n\mathbf{v}_n + 1 \cdot \mathbf{o}$
  - The list of scalars, $\{\beta_1, \beta_2, \ldots. \beta_n, 1\}$ is the representation of **p** w.r.t . the given coordinate system

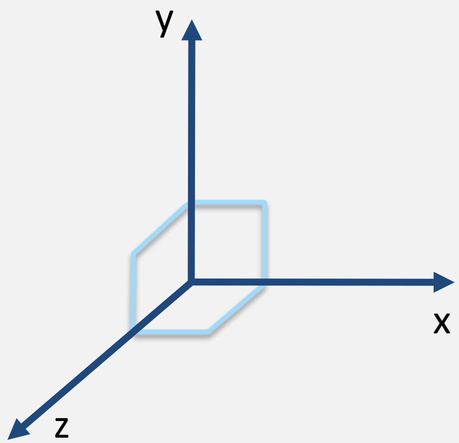


$$\boldsymbol{u} = [\alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_n]^T = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$$

$$\boldsymbol{p} = [\beta_1 \quad \beta_2 \quad \cdots \quad \beta_n]^T = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}$$

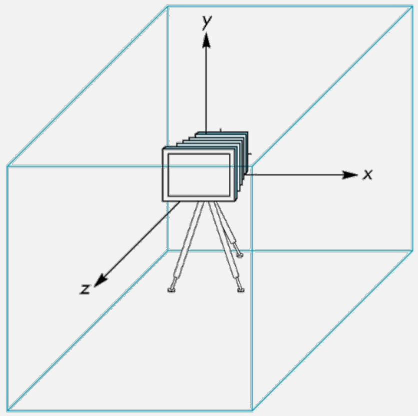
Coordinate System (Frame)

- In OpenGL ES, we just care about ***orthonormal*** frames
  - ***Ortho*** means that the basis vectors are orthogonal to each other
    - x-axis ⊥ y-axis ⊥ z-axis
  - ***normal*** means that the length of each basis vector is 1
    - The unit length of each axis is equal to 1



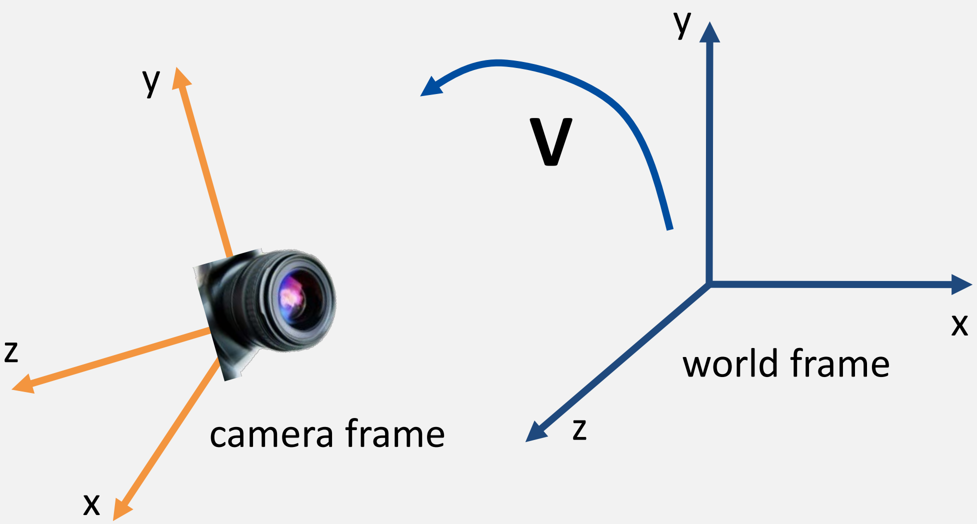
Inside of gluLookAt()

- Initially, OpenGL ES camera coordinate is as follows
  - Center of projection (COP) is placed in the origin
  - Right direction is the positive direction of x-axis
  - Up direction is the positive direction of y-axis
  - Viewing direction is the negative direction of z-axis



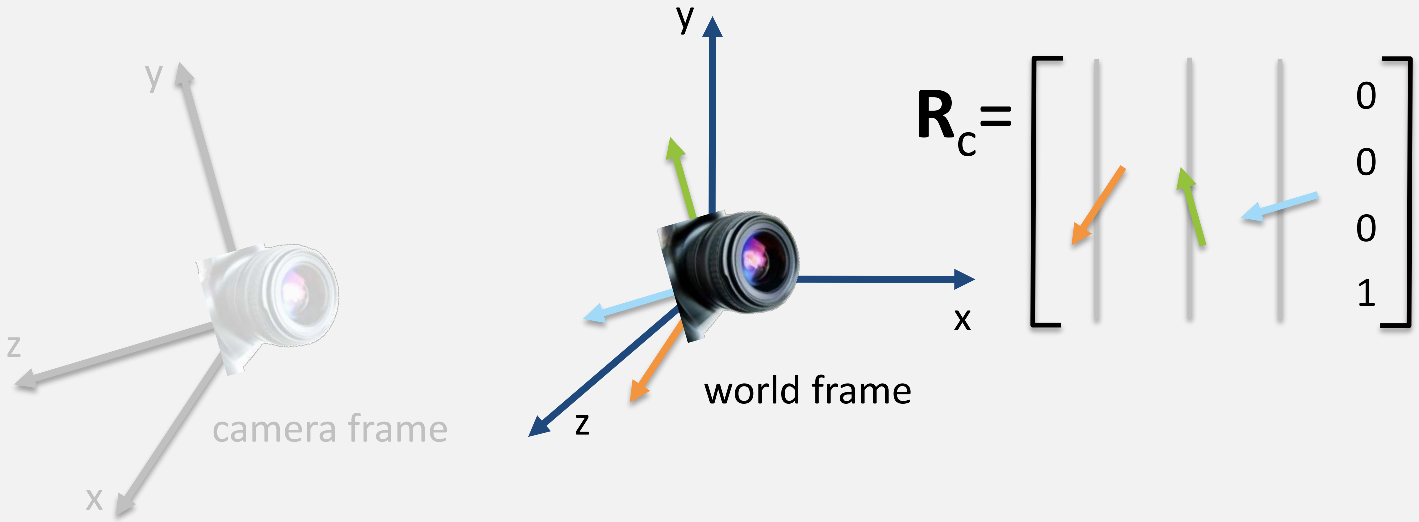
Inside of gluLookAt()

- When we apply gluLookAt()
  - Rotate the camera frame on the world frame: $\mathbf{R}_c$
  - Translate the camera frame on the world frame: $\mathbf{T}_c$
  - Therefore, $\mathbf{V} = \mathbf{T}_c\mathbf{R}_c$ and gluLookAt() generates $\mathbf{V}^{-1} = \mathbf{R}_c^{-1}\mathbf{T}_c^{-1}$



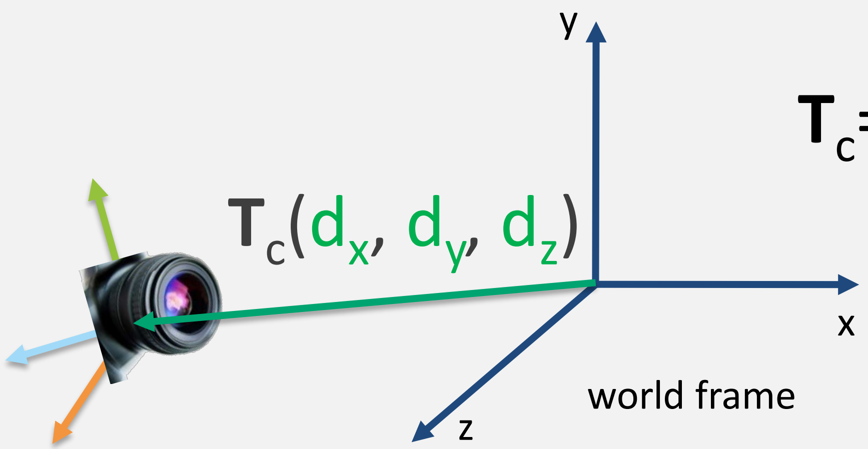
Inside of gluLookAt()

- When we apply gluLookAt()
  - Rotate the camera frame on the world frame: $\mathbf{R}_c$
  - Translate the camera frame on the world frame: $\mathbf{T}_c$
  - Therefore, $\mathbf{V} = \mathbf{T}_c\mathbf{R}_c$ and gluLookAt() generates $\mathbf{V}^{-1} = \mathbf{R}_c^{-1}\mathbf{T}_c^{-1}$



Inside of gluLookAt()

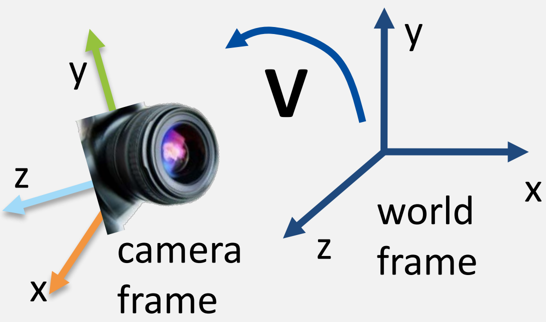
- When we apply gluLookAt()
  - Rotate the camera frame on the world frame: $\mathbf{R}_c$
  - Translate the camera frame on the world frame: $\mathbf{T}_c$
  - Therefore, $\mathbf{V} = \mathbf{T}_c\mathbf{R}_c$ and gluLookAt() generates $\mathbf{V}^{-1} = \mathbf{R}_c^{-1}\mathbf{T}_c^{-1}$



$$\mathbf{T}_c = \begin{bmatrix} 1 & & & dx \\ & 1 & & dy \\ & & 1 & dz \\ & & & 1 \end{bmatrix}$$

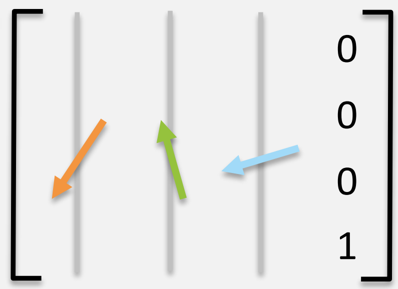
Inside of gluLookAt()

- When we apply gluLookAt()
  - Rotate the camera frame on the world frame: $\mathbf{R}_c$
  - Translate the camera frame on the world frame: $\mathbf{T}_c$
  - Therefore, $\mathbf{V} = \mathbf{T}_c\mathbf{R}_c$ and gluLookAt() generates $\mathbf{V}^{-1} = \mathbf{R}_c^{-1}\mathbf{T}_c^{-1}$



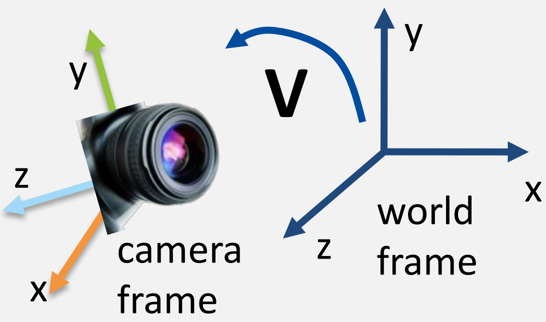
$$\mathbf{V}=\mathbf{T}_c\mathbf{R}_c$$

$$= \begin{bmatrix} 1 & & & dx \\ & 1 & & dy \\ & & 1 & dz \\ & & & 1 \end{bmatrix}$$



Inside of gluLookAt()

- When we apply gluLookAt()
  - Rotate the camera frame on the world frame: $\mathbf{R}_c$
  - Translate the camera frame on the world frame: $\mathbf{T}_c$
  - Therefore, $\mathbf{V} = \mathbf{T}_c\mathbf{R}_c$ and gluLookAt() generates $\mathbf{V}^{-1} = \mathbf{R}_c^{-1}\mathbf{T}_c^{-1}$



$$\mathbf{V}^{-1}=\mathbf{R}_c^{-1}\mathbf{T}_c^{-1}$$

$$= \begin{bmatrix} & & & \\ & & & \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & -dx \\ & 1 & & -dy \\ & & 1 & -dz \\ & & & 1 \end{bmatrix}$$