programming transformations

- what are convenient coordinate systems to work with
  - how do describe where the objects are relative to each other
  - how to get everything ready for the camera
- how to deal with this in an openGL program

world frame

- (rhon) world frame $\vec{\mathbf{w}}^t$
- never changes
- coordinates of a point wrt this frame are called world coordinates.

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

object frame

- we wish to describe the geometry of an object without thinking about its placement in the world.
- we associate a rhon frame with the object $\vec{\mathbf{o}}^t$
- we describe our geometry using coordinates wrt $\vec{\mathbf{o}}^t$.
  - called object coordinates

$$\begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

- example: canonical cube

object and world relationship

- relationship between world and object is expressed as

$$\vec{\mathbf{o}}^t = \vec{\mathbf{w}}^t O$$

  where O is a RB matrix.
- with the above understanding, in the computer program we store only O
- we can place and move the object by changing $\vec{\mathbf{o}}^t$.
  - we update $\vec{\mathbf{o}}^t$ by updating O
- in general we will have many objects, each with its own associated matrix
- in our code, we will store these matrices in `g_objectRbt[i]`

eye frame

- to create picture, we need a point of view.
- position of each object in picture is based on its relationship to eye
  - its coordinates relative to the eye's frame
- so we have an eye frame

- think of this frame as x=right arm, y=up, -z=forward
- eye coordinates:

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

- relationship between world and eye is expressed as

$$\vec{\mathbf{e}}^t = \vec{\mathbf{w}}^t E$$

where E is a RB matrix.
- any frame could act as the eye. in the code, we will have a special frame named `g_skyRbt` which will be the default eye.

to render

- a point can be expressed with object coords, world coords, and eye coords.

$$\tilde{p} = \vec{\mathbf{o}}^t \mathbf{c} = \vec{\mathbf{w}}^t O \mathbf{c} = \vec{\mathbf{e}}^t E^{-1} O \mathbf{c}$$

- it makes sense for our renderer to use eye coordinates.
- computed as

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = E^{-1} O \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

- in our code we will store object coords in the VBO, pass $E^{-1}O$ to the vertex shader, as a uniform variable and do this multiplication in the vertex shader.

moving an object wrt a

- we move an object by transforming $\vec{\mathbf{o}^{\mathbf{t}}}$
- this is implemented by updating O.
- Let us say we wish to apply some transformation M, (say translate in first axis) to an object frame $\vec{\mathbf{o}}^t$ with respect to some frame $\vec{\mathbf{a}}^t = \vec{\mathbf{w}}^t A$

$$\begin{aligned} & \vec{\mathbf{o}}^t \\ = \; & \vec{\mathbf{w}}^t O \\ = \; & \vec{\mathbf{a}}^t A^{-1} O \\ \Rightarrow \; & \vec{\mathbf{a}}^t M A^{-1} O \\ = \; & \vec{\mathbf{w}}^t A M A^{-1} O \end{aligned}$$

- in code : $O \leftarrow AMA^{-1}O$.
  - we will call this `doMtoOwrtA(M,O,A)`.

non useful $\vec{\mathbf{a}}^t$

- suppose we use $\vec{\mathbf{o}}^t$ as the auxiliary frame
- this of course simplifies to

$$\begin{aligned} & \vec{\mathbf{o}}^t \\ = \; & \vec{\mathbf{w}}^t O \\ \Rightarrow \; & \vec{\mathbf{w}}^t O M \end{aligned}$$

- problem: directions don't match what i see on the screen so hard to control (see demo)
- suppose we use $\vec{\mathbf{e}}^t$ as the auxiliary frame
- object orbits around the eye (see demo).
- suppose we use $\vec{\mathbf{w}}^t$ as the auxiliary frame
- this of course simplifies to

$$\begin{aligned} & \vec{\mathbf{o}}^t \\ = & \vec{\mathbf{w}}^t O \\ \Rightarrow & \vec{\mathbf{w}}^t M O \end{aligned}$$

- but now we have two problems

useful $\vec{\mathbf{a}}^t$

- we want a new frame that has the origin of the object but directions of the eye.
  - see fig
- let us factor our matrices as

$$\begin{aligned} O &= (O)_T(O)_R \\ E &= (E)_T(E)_R \end{aligned}$$

- desired auxiliary frame should be

$$\vec{\mathbf{a}}^t = \vec{\mathbf{w}}^t (O)_T (E)_R$$

  - read right to left
- so $A = (O)_T(E)_R$.
- in the spec, if our object is a cube and the eye the "sky camera" we will call this auxiliary; frame "cube-sky"

useful a for moving eye

- eye frame can be moved just like an object frame $E \leftarrow AMA^{-1}E$.
- to get "intuitive" directions, we may need to negate some of the signs.
- to have eye orbit around object $A = (O)_T(E)_R$.
- to have eye orbit around center of room $A = (E)_R$.
  - in spec we call this world-sky
- to have egomotion, we can choose $\vec{\mathbf{a}}^t = \vec{\mathbf{e}}^t$, giving us $A = E$.
- there may be other useful ways to control the eye depending on the context.

later

- later on we will come back to scales and hierarchies.