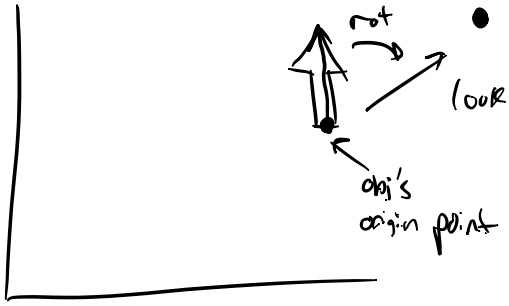
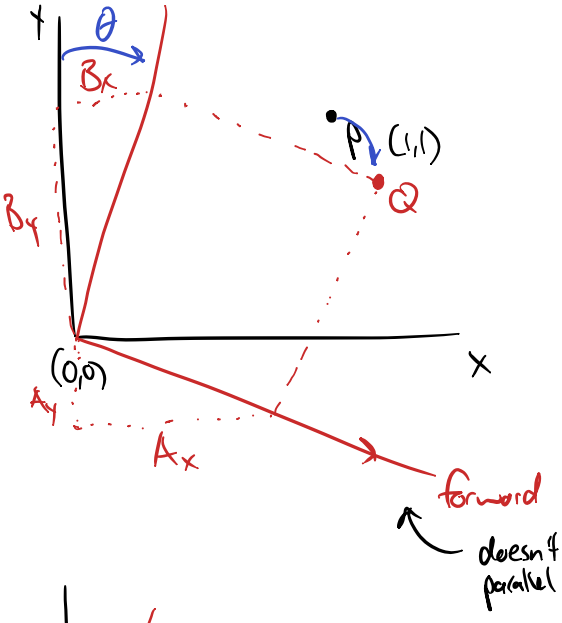
camera transformation

Camera Control

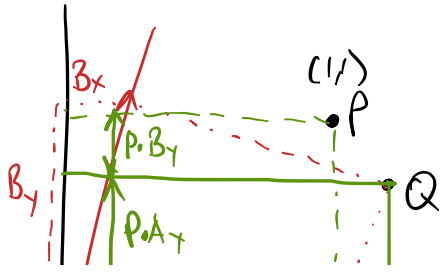
- represent camera as obj in world and have camera move around world using transformation

Rotating Space



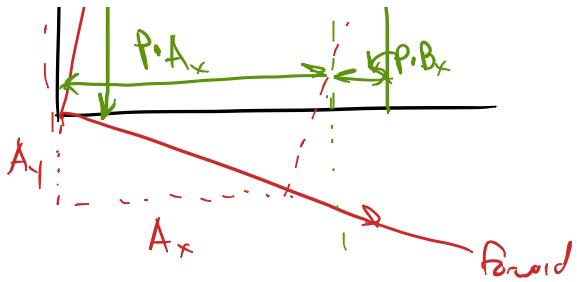


- rot P to Q
- not going to use angles
- designate 1 axis as "forward" and make unit vec
- vec ⊥ to forward is y-axis
- in rot space Q still exist at (1,1) (axis def by red vecs)



$$\left.\begin{aligned} Q_x &= A_x + B_x \\ Q_y &= A_y + B_y \end{aligned}\right\} \text{ for } P = (1,1)$$

scale vals if P not (1,1):



$$Q_x = P_x A_x + P_y B_x$$
$$Q_y = P_x A_y + P_y B_y$$

can create mapping for rotation by easily declaring a forward direction

Representing as a Matrix

$$\begin{bmatrix} P_x & P_y \end{bmatrix} \begin{bmatrix} A_x & A_y \\ B_x & B_y \end{bmatrix}$$

If we want to add in a translation

$$\begin{bmatrix} P_x & P_y & 1 \end{bmatrix} \begin{bmatrix} A_x & A_y & 0 \\ B_x & B_y & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

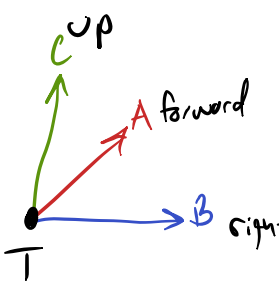
A = forward vector
B = orthogonal vector
T = translation

need these 3 things, now how matrix in 2D that will transform points for rot/translate

In 3D add z component:

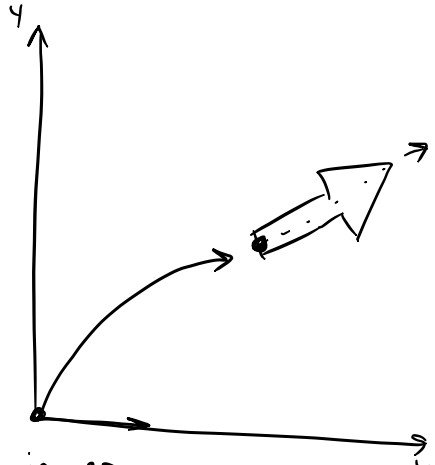
$$\begin{bmatrix} P_x & P_y & P_z & 1 \end{bmatrix} \begin{bmatrix} A_x & A_y & A_z & 0 \\ B_x & B_y & B_z & 0 \\ C_x & C_y & C_z & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

now we need an additional orthogonal vector C, just get that using A×B



"point at" matrix

"point at" matrix can position and rotate obj in world space



in 2D ex look down x axis
in 3D looks down z axis

"point at" transformation goes from origin point to some loc w/ some rot

Key: w/ a camera we need to do the exact opposite and transform all points to the (0,0) looking along an axis

We want inverse of "point at" matrix

"Point at"

$$\begin{bmatrix} A_x & A_y & A_z & 0 \\ B_x & B_y & B_z & 0 \\ C_x & C_y & C_z & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

get inverse →

"look at"

$$\begin{bmatrix} A_x & B_x & C_x & 0 \\ A_y & B_y & C_y & 0 \\ A_z & B_z & C_z & 0 \\ -T\cdot A & -T\cdot B & -T\cdot C & 1 \end{bmatrix}$$

dot prod of vecs get scalar

objects coords in word space times "look at" = obj's coords in view space