

The OpenGL Camera.

OpenGL defines a **FIXED STANDARD** “camera” looks toward the $-z$ (**MINUS ZEE**) axis and takes a fixed, standard picture.

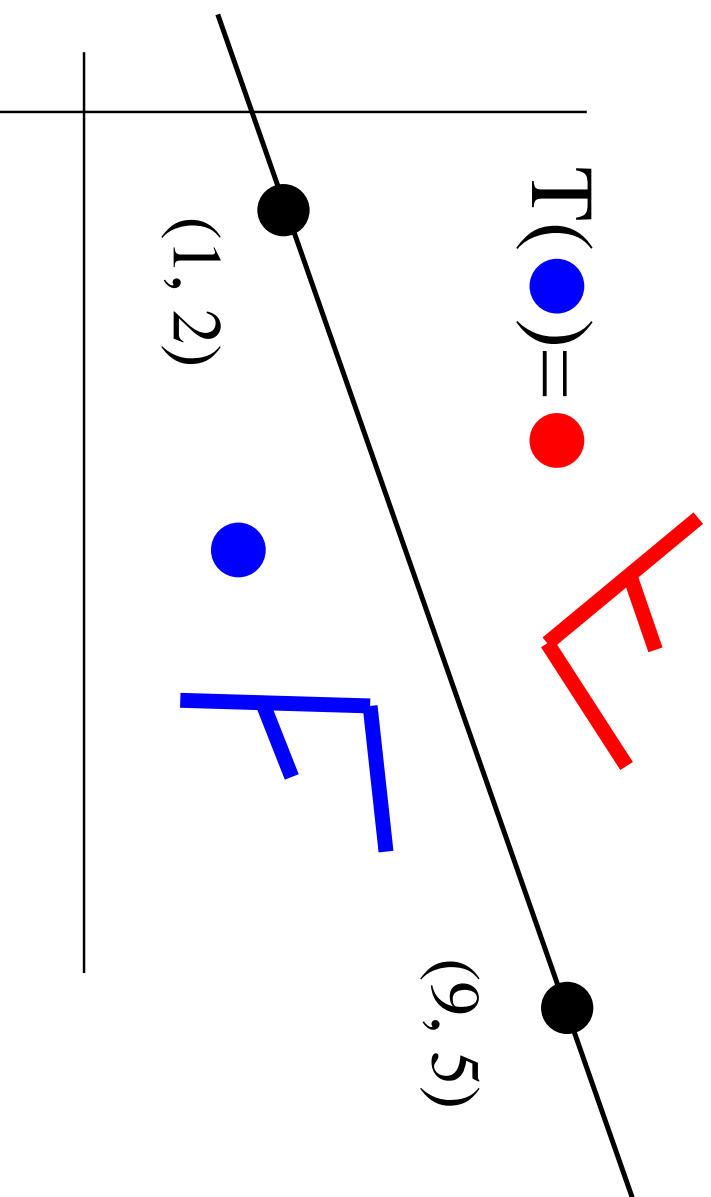
Suppose you want a picture taken with a differently shaped, directed and positioned camera. If T transforms the standard GL camera into the camera you want to use, then T^{-1} is used for GL viewing.

T^{-1} transforms objects and your custom camera into objects and the OpenGL standard camera. The transformation preserves relationships between the objects and the camera so the standard picture of the transformed objects is the **SAME** as the custom camera picture of the original objects.

Using T^{-1} is like bringing somebody to a photography studio.

Kind of problem this topic will help solve:

In the plane, suppose you are given an oblique line by means of two points, and that line doesn't necessarily go through the origin. How can you make the computer do a transformation that **reflects** objects across this line?



Dot product and cosines.

Dot product algebraically (and computationally):

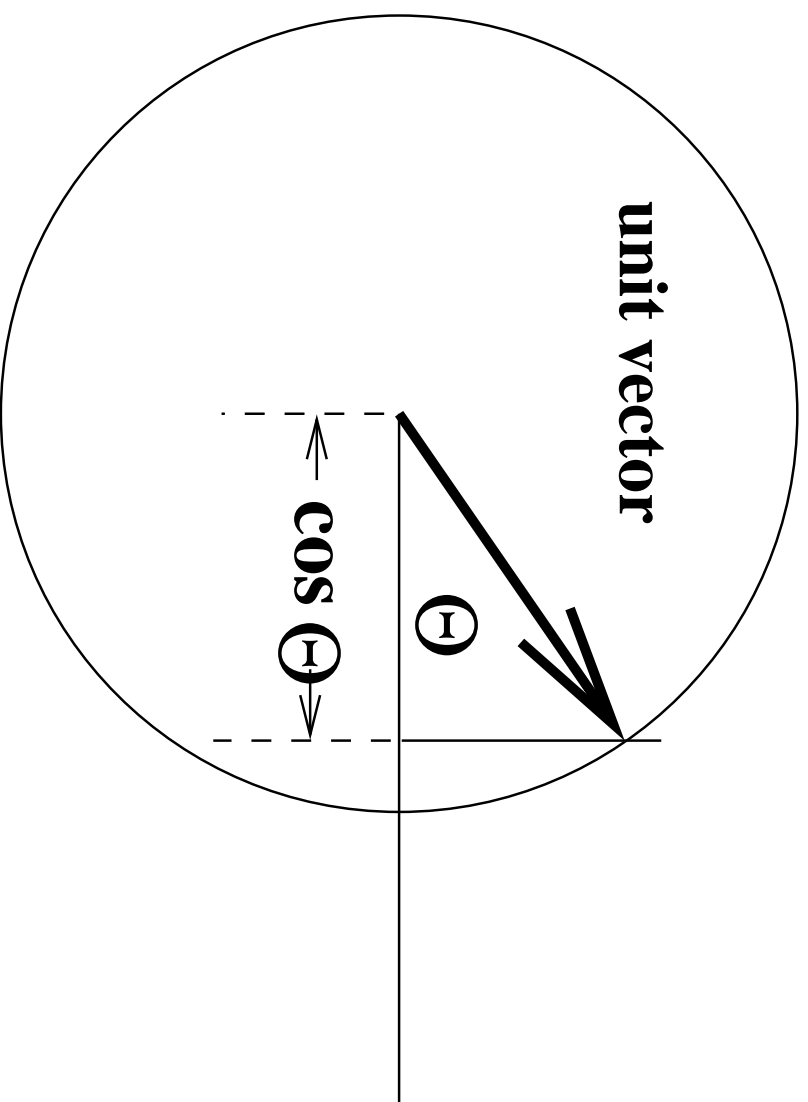
$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = x_1 x_2 + y_1 y_2 + z_1 z_2; \quad \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = x_1 x_2 + y_1 y_2$$

So, when $\mathbf{V} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$, $\mathbf{V} \cdot \mathbf{V} = x^2 + y^2 + z^2 = \text{Length}^2(\mathbf{V}) = \|\mathbf{V}\|^2$. (This is the 3-d version of the Pythagorean Theorem.)

So, $\|\mathbf{V}\| = \sqrt{\mathbf{V} \cdot \mathbf{V}}$.

Geometrically,

$$\mathbf{V} \cdot \mathbf{W} = \|\mathbf{V}\| \|\mathbf{W}\| \cos \Theta \quad (\text{cosine of angle between } \mathbf{V} \text{ and } \mathbf{W})$$



Ordinary coordinate unit vectors are mutually orthogonal and length 1 or **normal** also, so they are **orthonormal**.

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 1^2 + 0^2 + 0^2 = 1$$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = 1 * 0 + 0 * 1 + 0 * 0 = 0$$

They are also orthonormal *after rotation*: Consider rotated \mathbf{U}_x and \mathbf{U}_y :

$$\begin{aligned} & \begin{bmatrix} \cos(\Theta) \\ \sin(\Theta) \end{bmatrix} \cdot \begin{bmatrix} -\sin(\Theta) \\ \cos(\Theta) \end{bmatrix} = 0 \text{ and } \begin{bmatrix} -\sin(\Theta) \\ \cos(\Theta) \end{bmatrix} \cdot \begin{bmatrix} -\sin(\Theta) \\ \cos(\Theta) \end{bmatrix} = \\ & (-\sin \Theta)^2 + \cos^2 \Theta = \sin^2 \Theta + \cos^2 \Theta = 1 \end{aligned}$$

This makes it easy to figure out the inverse of R_Θ

$$\begin{aligned} & \begin{bmatrix} \begin{bmatrix} \cos \Theta & -\sin \Theta \\ \sin \Theta & \cos \Theta \end{bmatrix} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} \cos(\Theta) \\ \sin(\Theta) \end{bmatrix} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} \cos(\Theta) \\ -\sin(\Theta) \end{bmatrix} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} \sin(\Theta) \\ \cos(\Theta) \end{bmatrix} \end{bmatrix} \\ & = \begin{bmatrix} \mathbf{U}_x \cdot \mathbf{U}_x & \mathbf{U}_x \cdot \mathbf{U}_y \\ \mathbf{U}_y \cdot \mathbf{U}_x & \mathbf{U}_y \cdot \mathbf{U}_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

The same is true in 3-d.

The same is true for 3-d rotation matrices (built out of **direction cosines**)

$$\begin{bmatrix} \begin{bmatrix} c_{1x} & c_{1y} & c_{1z} \end{bmatrix} \\ \begin{bmatrix} c_{2x} & c_{2y} & c_{2z} \end{bmatrix} \\ \begin{bmatrix} c_{3x} & c_{3y} & c_{3z} \end{bmatrix} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} c_{1x} \\ c_{1y} \\ c_{1z} \end{bmatrix} \\ \begin{bmatrix} c_{2x} \\ c_{2y} \\ c_{2z} \end{bmatrix} \\ \begin{bmatrix} c_{3x} \\ c_{3y} \\ c_{3z} \end{bmatrix} \end{bmatrix} = I$$

Each column $\begin{bmatrix} c_{ix} \\ c_{iy} \\ c_{iz} \end{bmatrix}$ is the coordinates of a unit vector \mathbf{W}_i ; these are the cosines of the angles \mathbf{W}_i makes with the x -axis, y -axis and z -axis respectively.

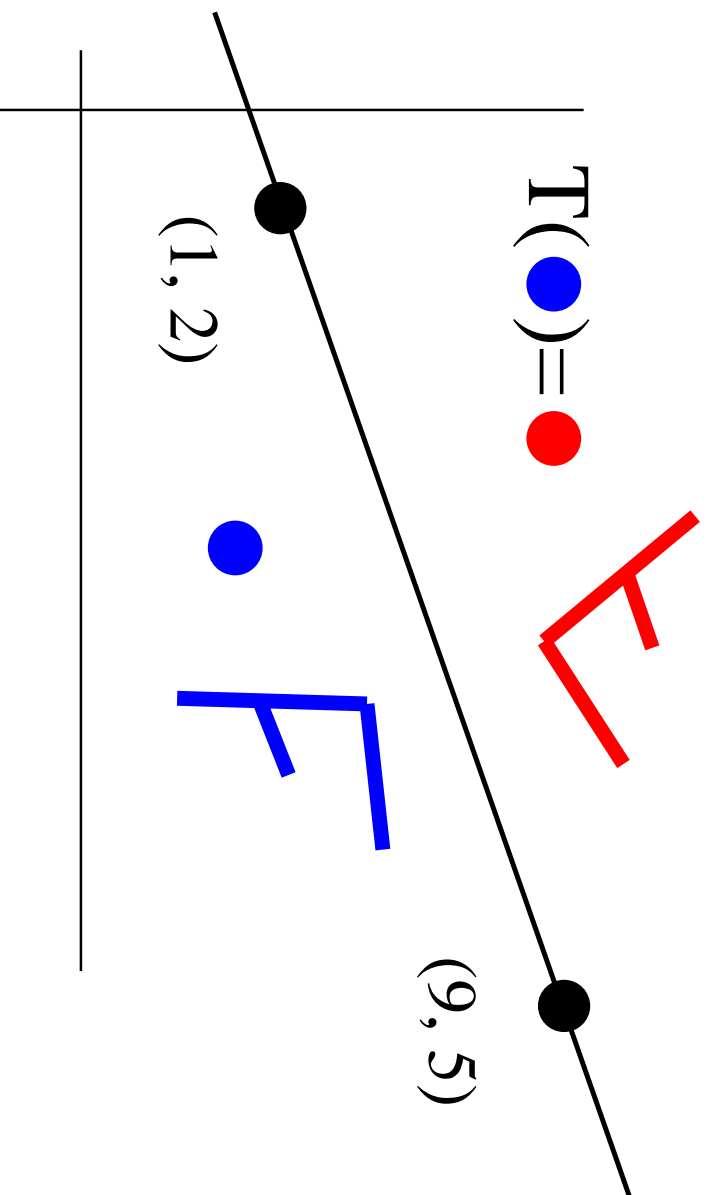
Only for **VERY SPECIAL MATRICES** does the inverse equal the **transpose**!

For example, it's false for translations with homogeneous coordinates.

Also, only in 2-d to successive rotations **commute**. In 3-d $R_x R_y \neq R_y R_x$.

Everybody please **TRY THIS**. Define 3 axes by drawing lines on paper and placing a pencil perpendicular to them. Try rotating a non-symmetric object 90° around the x – *axis* and then 90° around the y – *axis*; see how it ends up. Then, starting with same initial position, try the same rotations in the other order.

In the plane, suppose you are given an oblique line by means of two points, and that line doesn't necessarily go through the origin. How can you make the computer do a transformation that **reflects** objects across this line?



Strategy idea: Reflection through the x -axis is easy: Use $x' = x$ and $y' = -y$. We can **conjugate** x -axis reflection with transformations that move the x -axis into the oblique line. Detailed steps:

1. Form $T_{(1,2)}$ that translates the origin to $(1, 2)$:

$$T_{(1,2)} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and inverse } T_{(1,2)}^{-1} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Form R_{Θ} that rotates the x -axis into the *direction* of our reflection line. R_{Θ}^{-1} is it's transpose.

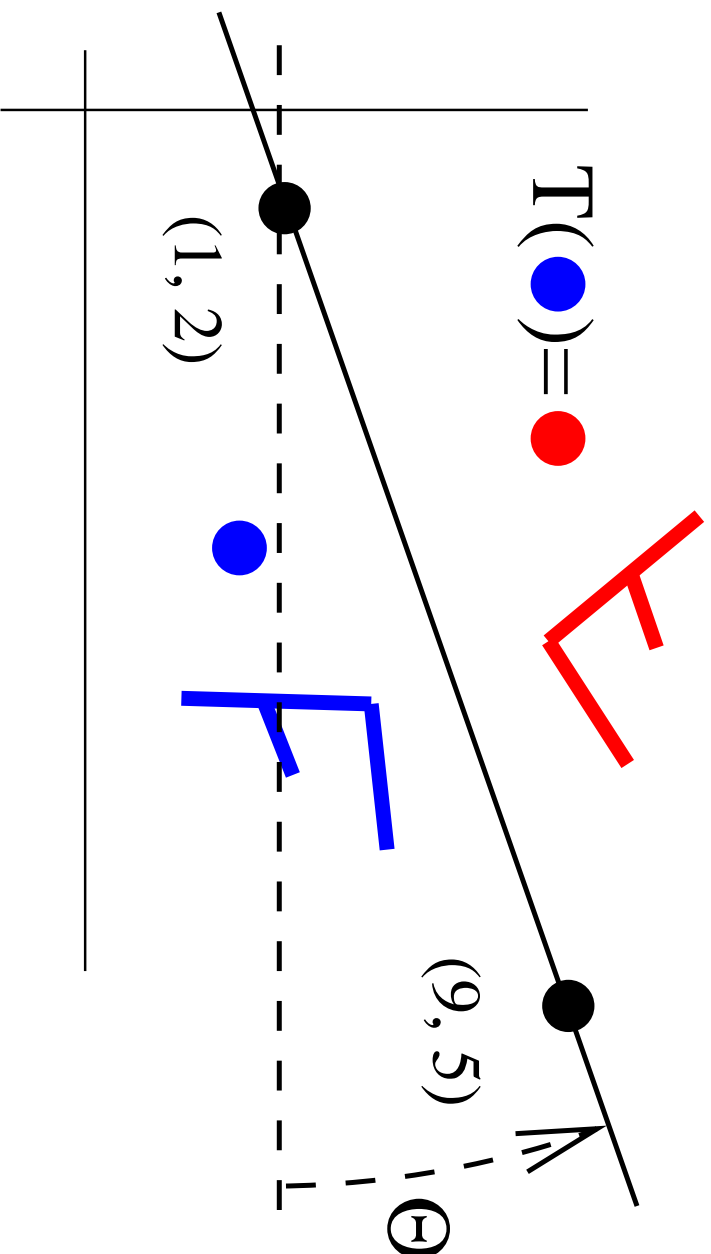
It's a mystery how to build it, to solve later.

3. Express x -axis reflection by a matrix:

$$F_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4. Put it together:

$$\text{Answer} = T_{(1,2)} R_{\Theta} F_x R_{\Theta}^{-1} T_{(1,2)}^{-1}$$



Please analyze the

geometric meaning of the 5 transformations composed to form the Answer.

Now, let's solve the mystery.

1. Form a unit vector $\mathbf{U}_{x'}$ in the *direction* of our reflection line. That direction is the direction of vector $(\Delta x, \Delta y) = (9 - 1, 5 - 2) = (8, 3)$. (We leave this as a submystery!)
2. Rotate $\mathbf{U}_{x'}$ 90° CCW to form $\mathbf{U}_{y'}$.
3. Answers:

$$R_\Theta = \begin{bmatrix} \mathbf{U}_{x'} & \mathbf{U}_{y'} \\ \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix}$$

and R_Θ^{-1} is R_Θ^t .

Now the submystery: Form a unit vector $\mathbf{U}_{x'}$ in the *direction* of our reflection line. That direction is the direction of vector $(\Delta x, \Delta y) = (9 - 1, 5 - 2) = (8, 3)$.

$$\begin{aligned}\text{Length of } (\Delta x, \Delta y) &= \|(\Delta x, \Delta y)\| = \sqrt{(\Delta x)^2 + (\Delta y)^2} \\ &= \sqrt{(9 - 1)^2 + (5 - 2)^2} = \sqrt{(8)^2 + (3)^2} = \sqrt{64 + 9} = \sqrt{73}\end{aligned}$$

So,

$$\mathbf{U}_{x'} = \frac{1}{\sqrt{73}} \begin{bmatrix} 8 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{8}{\sqrt{73}} \\ \frac{3}{\sqrt{73}} \end{bmatrix}$$

$\mathbf{U}_{y'}$ is $\mathbf{U}_{x'}$ rotated 90° :

$$\mathbf{U}_{x'} = \begin{bmatrix} \frac{8}{\sqrt{73}} \\ \frac{3}{\sqrt{73}} \end{bmatrix} \text{ so } \mathbf{U}_{y'} = \begin{bmatrix} -\frac{3}{\sqrt{73}} \\ \frac{8}{\sqrt{73}} \end{bmatrix}$$

Now that the submystery is solved, the solution to the mystery is:

$$R_{\Theta} = \begin{bmatrix} \frac{8}{\sqrt{73}} & -\frac{3}{\sqrt{73}} & 0 \\ \frac{3}{\sqrt{73}} & \frac{8}{\sqrt{73}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$R_{\Theta}^{-1} = \begin{bmatrix} \frac{8}{\sqrt{73}} & \frac{3}{\sqrt{73}} & 0 \\ -\frac{3}{\sqrt{73}} & \frac{8}{\sqrt{73}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Putting it all together, Answer = $T_{(1,2)} R_{\Theta} F_x R_{\Theta}^{-1} T_{(1,2)}^{-1}$

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{8}{\sqrt{73}} & -\frac{3}{\sqrt{73}} & 0 \\ \frac{3}{\sqrt{73}} & \frac{8}{\sqrt{73}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{8}{\sqrt{73}} & \frac{3}{\sqrt{73}} & 0 \\ -\frac{3}{\sqrt{73}} & \frac{8}{\sqrt{73}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

(Translate (1,2) to origin)

(Rotate translated line into x -axis)

(Reflect)

(Rotate x -axis back to translated line)

(Translate origin to (1,2))