Translation

$v = \begin{pmatrix} x \\ y \end{pmatrix}$ $\vec{d} = \begin{pmatrix} d_x \\ d_y \end{pmatrix}$ $v + \vec{d} = \begin{pmatrix} x+d_x \\ y+d_y \end{pmatrix}$

homogenous coordinates $v = v' = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ $(x, y, 1)$ for a point

$(x, y, 0)$ for a direction

transition matrix: $\begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+d_x \\ y+d_y \\ 1 \end{pmatrix} = \begin{pmatrix} x+d_x \\ y+d_y \end{pmatrix}$

direction does not affect the translation $\Rightarrow \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} = \begin{pmatrix} x+d_x \\ y+d_y \\ 0 \end{pmatrix}$

Homogeneous 2D > Affine.

$\left(\begin{array}{cc|c} s_x & 0 & 0 \\ 0 & s_y & 0 \\ \hline 0 & 0 & 1 \end{array}\right)$ $\left(\begin{array}{cc|c} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ \hline 0 & 0 & 1 \end{array}\right)$ $\left(\begin{array}{cc|c} 1 & m & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array}\right)$

composition of transformation

$\begin{pmatrix} s_x & & \\ & s_y & \\ & & 1 \end{pmatrix}\begin{pmatrix} 1 & & d_x \\ & 1 & d_y \\ & & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ <= a combination of transformation

by the associate rule of matrix,

we could put matrices together.

$M_1 M_2 M_3 \cdots M_0 = (M_2) M_0$

all these are applied in order from right to left. e.g. $S_4 R_3 T_2 R_1 V$

Though these matrices could be combined, we still could not change the order of translation applied.

$\begin{pmatrix} s_x & & \\ & s_y & \\ & & 1 \end{pmatrix}\begin{pmatrix} 1 & & d_x \\ & 1 & d_y \\ & & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & & \\ & s_y & \\ & & 1 \end{pmatrix}\begin{pmatrix} x+d_x \\ y+d_y \\ 1 \end{pmatrix} = \begin{pmatrix} s_x(x+d_x) \\ s_y(y+d_y) \\ 1 \end{pmatrix}$

This order MATTER!!!

for a left-hand system, the transition is the same.

if we're appling a rotation on the system we could ---

1) directly apply a rotation on the point itself

2) apply a rotation to every later movements.

Basic vertexs $\hat{i}$ $\hat{j}$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\begin{pmatrix} i_1 & j_1 \\ i_2 & j_2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} i_1 x + i_2 y \\ j_1 x + j_2 y \end{pmatrix}, \text{ where } i_1^2 + i_2^2 = j_1^2 + j_2^2 = 1$$

↑ modified coordinate system.

then, we can let $\sin\theta = i_1$, so $\cos\theta = i_2$,

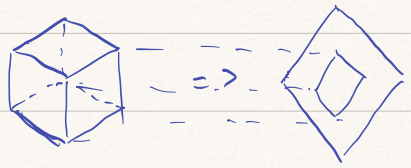
$\sin\alpha = j_1$, $\cos\alpha = j_2$, and $\alpha = \theta + 90°$ by def

moving the shape is the moving a collection of vertices.

so for rotation

Projection

orthogonal projection have: 1) parallel projection lines



2) projection lines that are propendict to the image plane.

perspective projection: closer is bigger, further away is smaller

openGL does not have a 2D square bound, so we use a 3D cube space instead.

$NDC = P \cdot V$

projection matrics

world vector

normalized

$[-1, 1] = P \cdot (1200, 500)$

glOrtho() is a function compute projected matrix.

we have to provide the viewing volum of all six faces.

(left, right, bottom, top, near, rear)
x y z

openGL is a global state machine, so we just have to call glOrtho() once. glMatrixMode let us modify the projection matrix : glMatrixMode(GL_PROJECTION)

given current matrix -> glLoadIdentity()

load with identity

glOrtho(L, R, B, T, N, R)

multiply the current matrix with ortho one. It is a transform