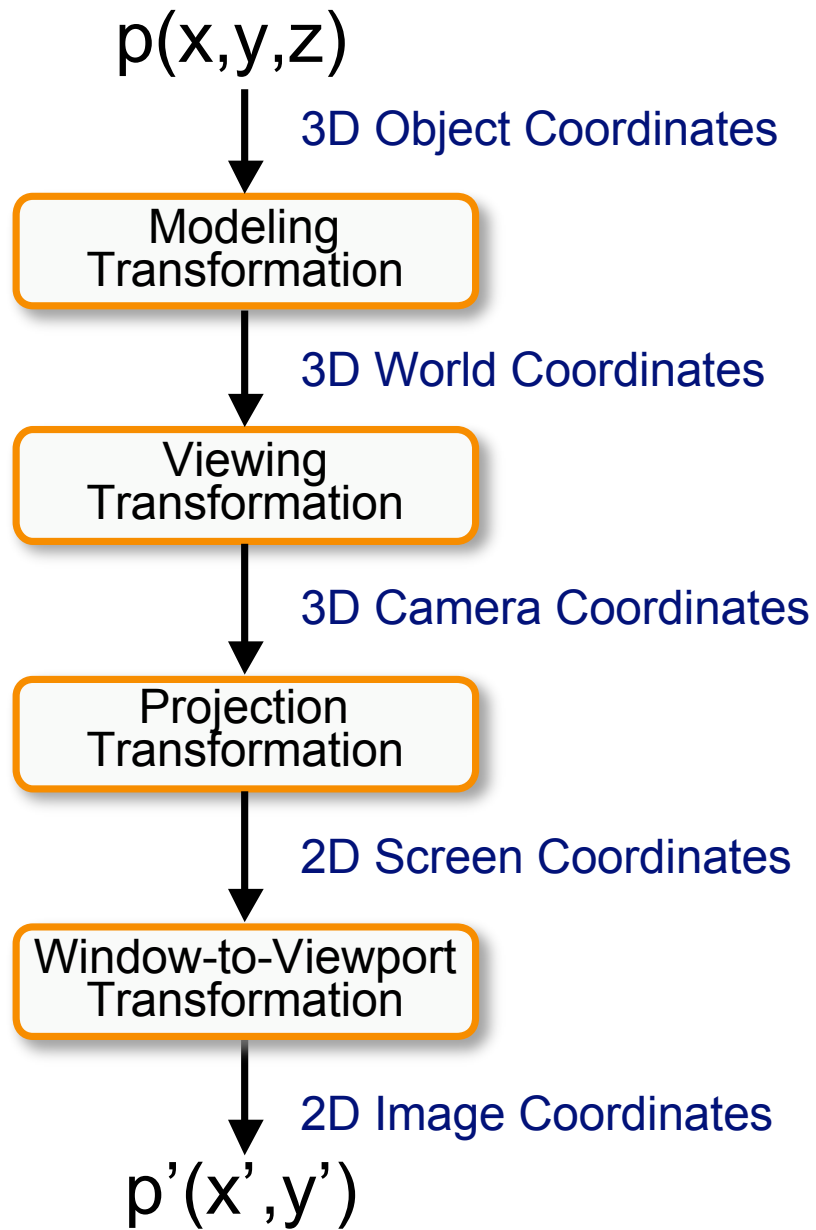


Lecture 3

Transformations
Homogeneous Coordinates

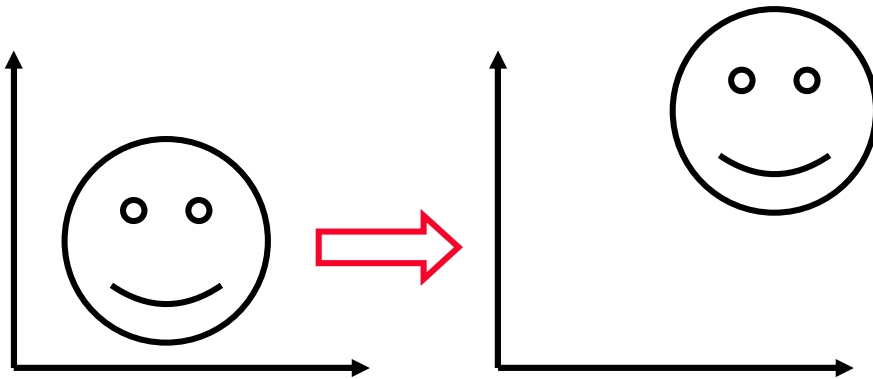
Announcements

- 4th of July
- Hw 1
 - Submission - readme + files + snapshot
 - Late policy - 3 days notice
 - Mailing List
 - Auditors
- Implicit Equations - point placement

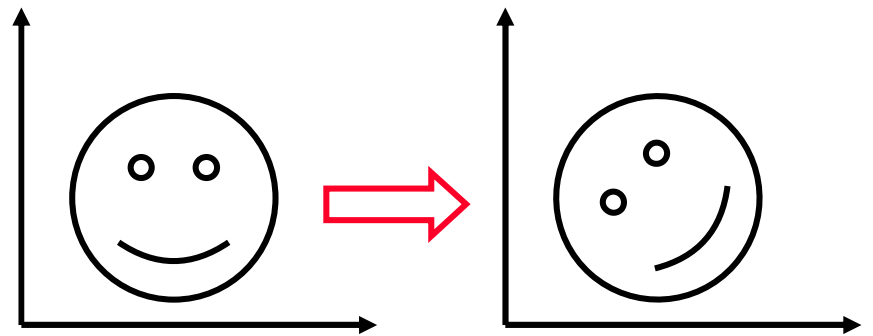


Transformations

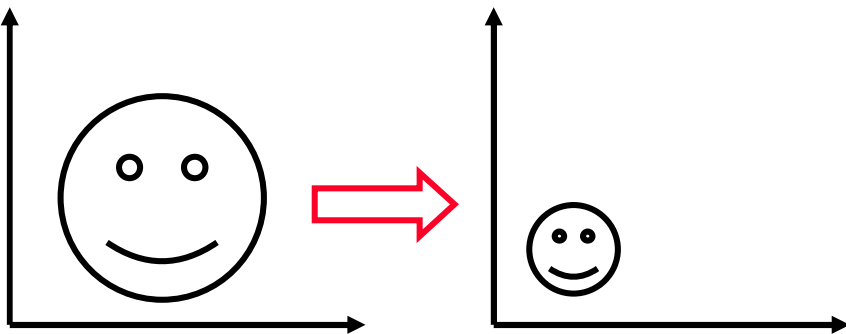
Examples of transformations:



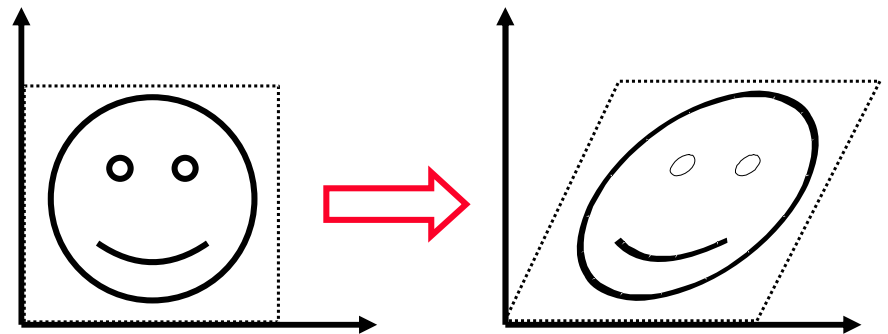
translation



rotation



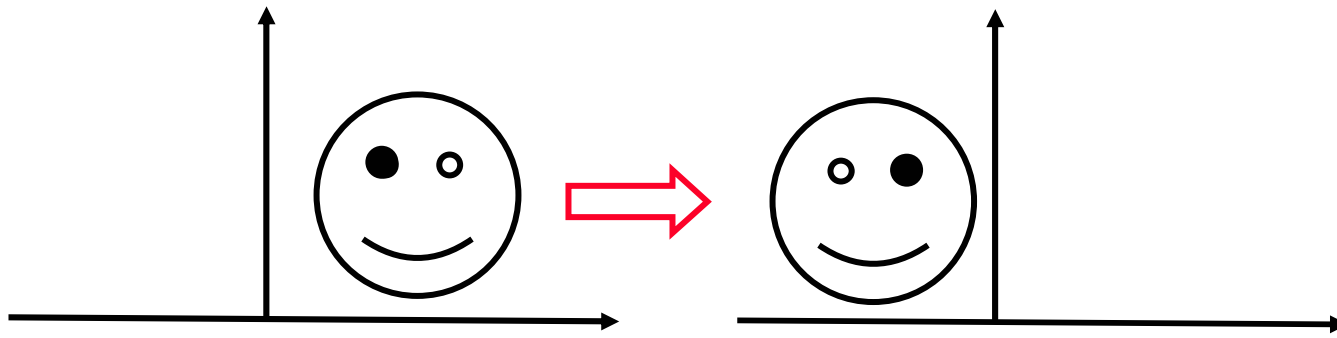
scaling



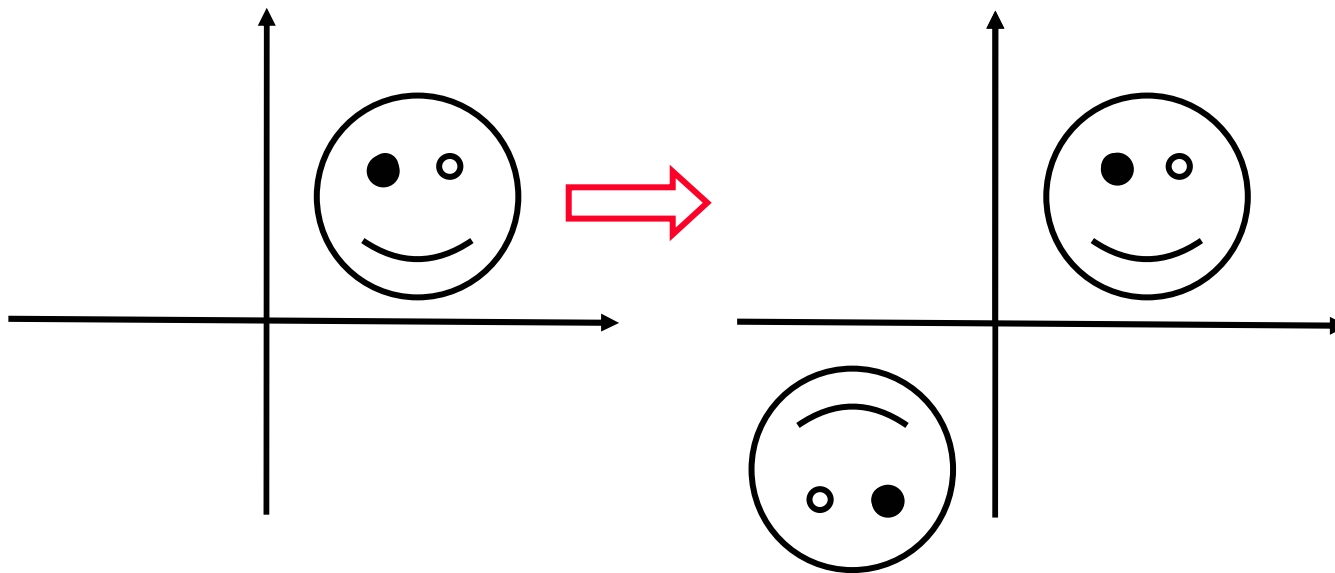
shear

Transformations

More examples:



reflection with respect to the y-axis



reflection with respect to the origin

Transformations

- **Linear Transformations**

$$T(\alpha p + \beta q) = \alpha T(p) + \beta T(q)$$

$$\forall \alpha, \beta$$

Takes straight lines to straight lines

Can be written as a matrix multiplication

e.g. rotation, reflection, scale, shear

Translation ??

Transformations

- Affine Transformations

$$T(\alpha p + \beta q) = \alpha T(p) + \beta T(q)$$

$$\alpha + \beta = 1$$

Preserve parallelism

$$p(t) = p + vt \quad T(p(t)) = T(p) + tT(v)$$

e.g. rotation, reflection, scale, translation, shear

Non-affine example -
perspective projection

Transformations

- **Rigid Body Transformations**

Preserve length and angles

a.k.a Orthographic

e.g. rotation, reflection, translation

- **Projection Transformations**

e.g. perspective, orthogonal

Transformations and matrices

Any affine transformation can be written as

$$\begin{pmatrix} \mathbf{x}' \\ \mathbf{y}' \end{pmatrix} = \begin{pmatrix} \mathbf{a}_{11} & \mathbf{a}_{12} \\ \mathbf{a}_{21} & \mathbf{a}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} + \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix} \quad \mathbf{p}' = \mathbf{A}\mathbf{p} \quad \text{w/ fixed origin!}$$

Linear transformation + translation

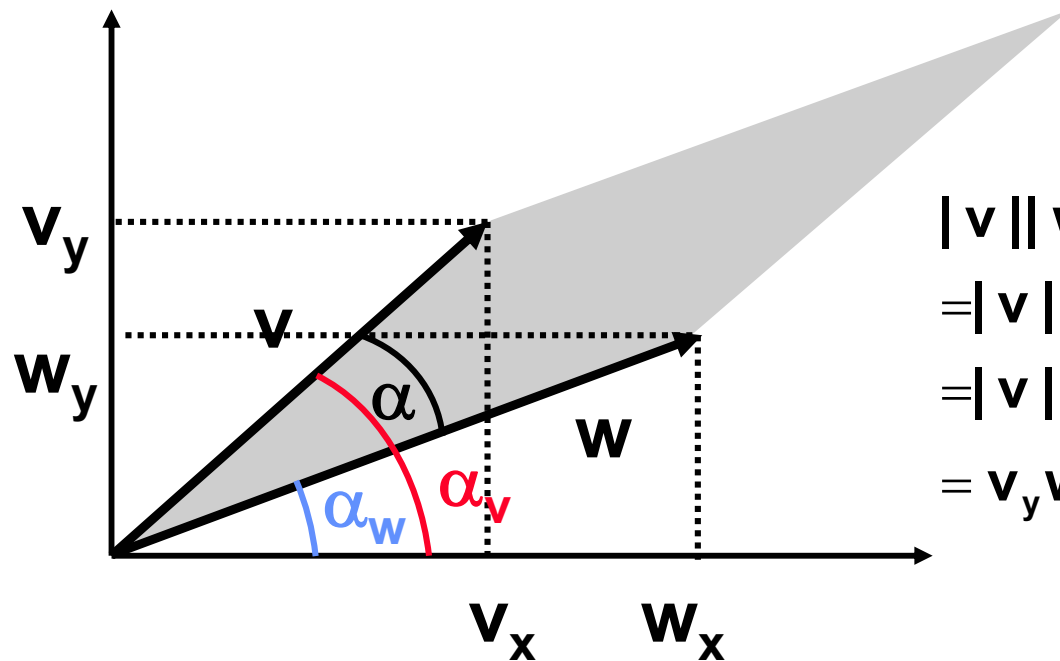
Images of basis vectors under transformation matrices

$$\begin{aligned} \mathbf{e}_x &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} & \text{(column form of writing vectors)} \\ \mathbf{e}_y &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned}$$
$$\mathbf{A}\mathbf{e}_x = \begin{pmatrix} \mathbf{a}_{11} & \mathbf{a}_{12} \\ \mathbf{a}_{21} & \mathbf{a}_{22} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{a}_{11} \\ \mathbf{a}_{21} \end{pmatrix} \quad \mathbf{A}\mathbf{e}_y = \begin{pmatrix} \mathbf{a}_{12} \\ \mathbf{a}_{22} \end{pmatrix}$$

Determinant

Area of a parallelogram

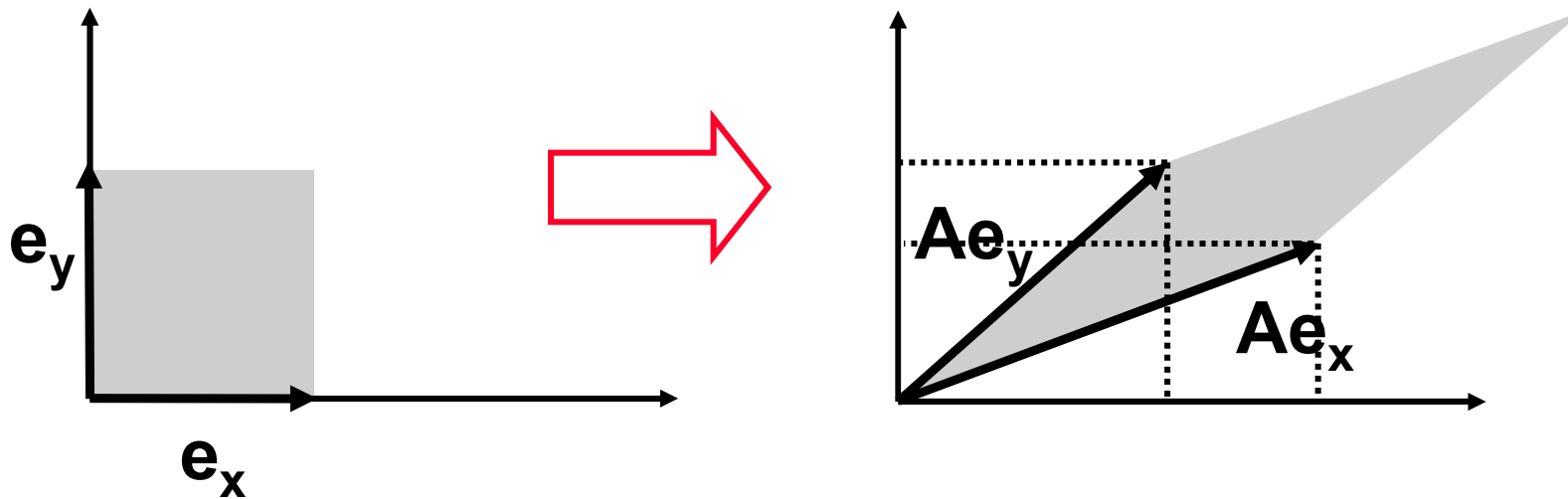
Area of the shaded parallelogram:
area = $|\mathbf{v}| |\mathbf{w}| \sin \alpha$



$$\begin{aligned} |\mathbf{v}| |\mathbf{w}| \sin \alpha &= |\mathbf{v}| |\mathbf{w}| \sin(\alpha_v - \alpha_w) \\ &= |\mathbf{v}| |\mathbf{w}| (\sin \alpha_v \cos \alpha_w - \cos \alpha_v \sin \alpha_w) \\ &= |\mathbf{v}| |\mathbf{w}| (v_y w_x / |\mathbf{v}| |\mathbf{w}| - v_x w_y / |\mathbf{v}| |\mathbf{w}|) \\ &= v_y w_x - v_x w_y \end{aligned}$$

Determinant

Determinant of a transformation matrix = area of the image of a unit square.



$$\text{area} = \det \begin{pmatrix} \mathbf{a}_{11} & \mathbf{a}_{12} \\ \mathbf{a}_{21} & \mathbf{a}_{22} \end{pmatrix} = \mathbf{a}_{11}\mathbf{a}_{22} - \mathbf{a}_{12}\mathbf{a}_{21}$$

Some Transformation Matrices

- Scale $\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$ uniform vs. differential
- Shear $\begin{pmatrix} 1 & \tan \phi \\ 0 & 1 \end{pmatrix}$ (vert, cw) $\begin{pmatrix} 1 & 0 \\ \tan \phi & 1 \end{pmatrix}$ (hor, ccw)
- Rotation $\begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix}$ about the origin, angle ccw
- Reflection $\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$ $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$

Decomposition of 2D Transforms

Any 2D transform can be decomposed into the product of rotation, a scale and another rotation (up to a sign change = reflection)

$$M = R_2 S R_1$$

Rotation matrices are orthonormal!

SVD Decomposition!

(For symmetric = Eigen value decomposition)

Inverses of Transformation Matrices

Geometric inversion instead of algebraic

- Inverse of scale \Rightarrow scale by $(1/s_x, 1/s_y)$
- Inverse of rotation \Rightarrow rotation by transpose
- Inverse of translation \Rightarrow translation in opposite direction

If a series of transforms = $M = M_1 M_2 \dots M_n$

$$M^{-1} = M_n^{-1} M_{n-1}^{-1} \dots M_1^{-1}$$

Any transformation matrix $\Rightarrow R_1 S R_2$

$$\text{inverse} \Rightarrow R_2^{-1} S^{-1} R_1^{-1}$$

Homogeneous coordinates

Problem

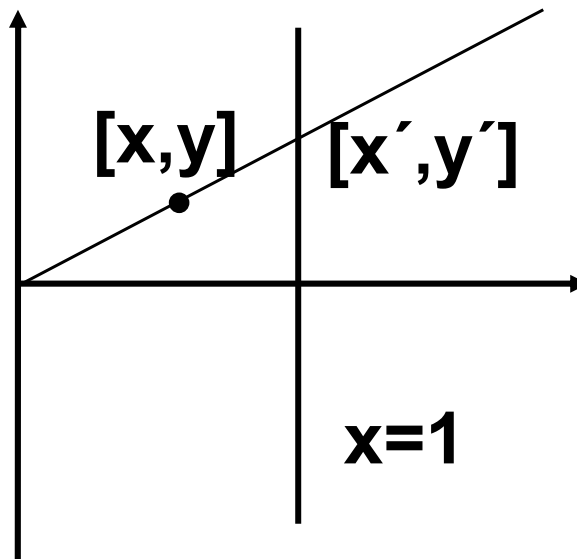
For affine transformations :

- Just a 2x2 matrix not enough.
- Need a translation vector

For projection transformations

- Cannot even write as $Ax+b$!

e.g.
perspective
projection



$$\begin{aligned}x' &= 1 \\ y' &= y/x\end{aligned}$$

equations not linear!

Homogeneous Coordinates in 2D

- 2D point \Rightarrow 3D point $\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

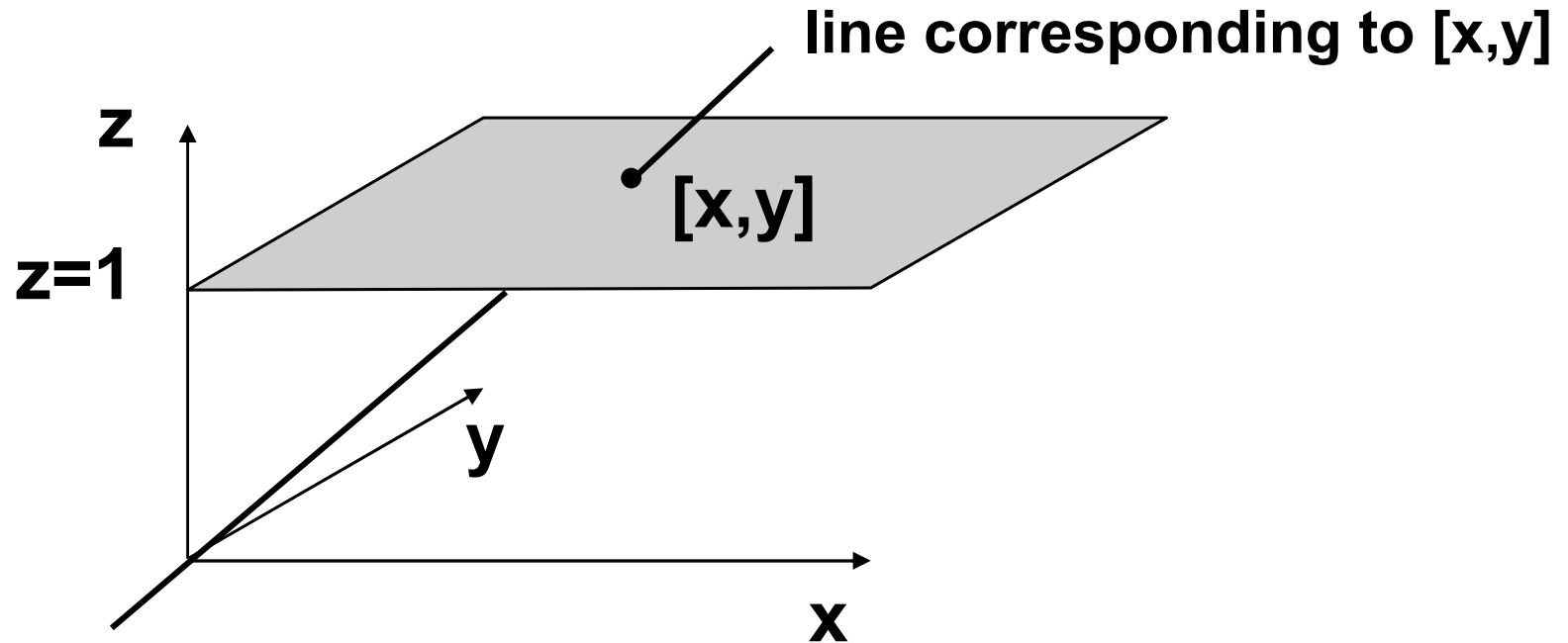
- 3D \Rightarrow 2D $\begin{pmatrix} x \\ y \\ w \end{pmatrix} \rightarrow \begin{pmatrix} x/w \\ y/w \end{pmatrix}$ if w non-zero,
else not a point

Geometric : Project to the plane at
construction $z=1$ from origin

- Each 2D point - is a line in 3D.

Each point on the line = $[kx, ky, k]$ for some k

Homogeneous coordinates



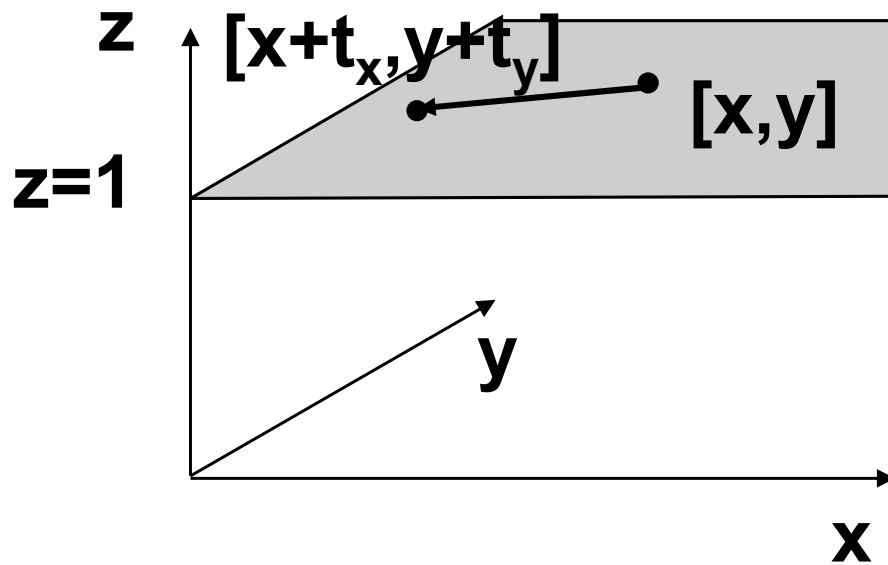
From homogeneous to 2d: $[x,y,w]$ becomes $[x/w,y/w]$
From 2d to homogeneous: $[x,y]$ becomes $[kx,ky,k]$
(can pick any nonzero k !)

Homogeneous transformations

Any linear transformation can be written in matrix form in homogeneous coordinates.

Example 1: translations

$[x,y]$ in hom. coords is $[x,y,1]$



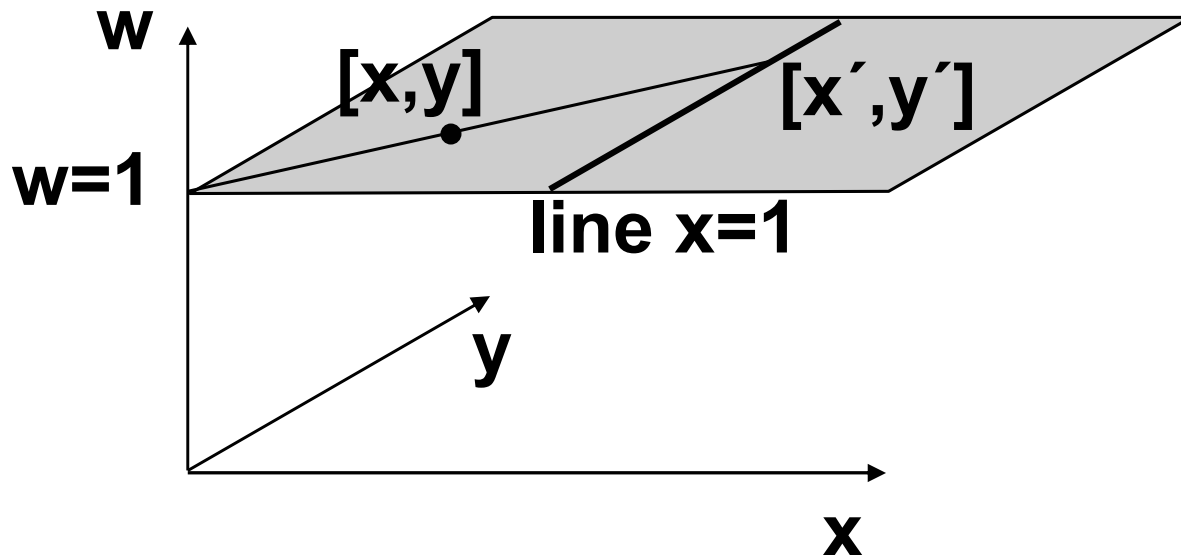
$$\begin{aligned}x' &= x+t_x = x+ t_x \cdot 1 \\y' &= y+t_y = y+t_y \cdot 1 \\w' &= 1\end{aligned}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous transformations

Example 2: perspective projection

$x' = 1$ Can multiply all three components
 $y' = y/x$ by the same number -- the 2D point
 $w' = 1$ won't change! Multiply by x .



$$\begin{aligned}x' &= x \\y' &= y \\w' &= x\end{aligned}$$

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

Homogeneous Transformations

Example 3: Rotation about a point p in 2D.

- Translate by $-p = T(-p)$
- Rotate $R(q)$
- Translate by $p = T(p)$
- Transformation matrix $M = T(p)R(q)T(-p)$

Matrices of basic transformations

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ rotation} \quad \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \text{ translation}$$

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ scaling} \quad \begin{bmatrix} 1 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ skew}$$

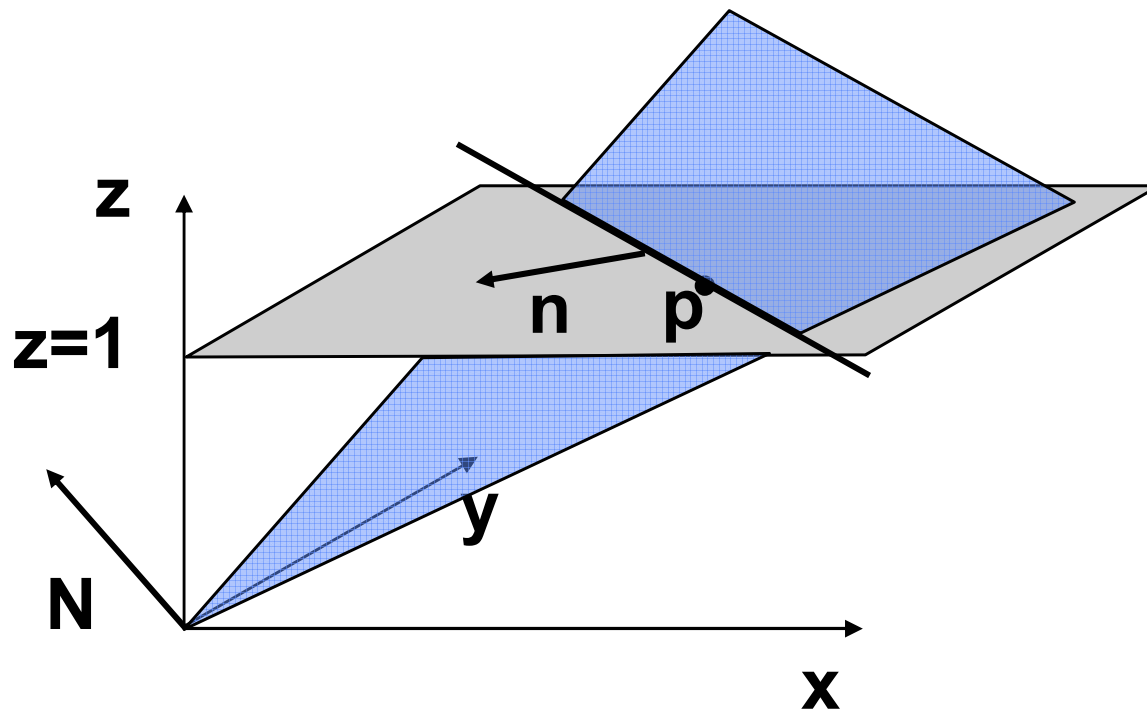
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \text{ general affine transform}$$

Homogeneous line equation

Implicit line equation in 2D: $(n \cdot (q-p)) = 0$,

n = 2D vector, p = 2D point on the line.

Goal: rewrite in homogeneous coordinates.



2D point corresponds to a 3D line through origin;
2D line corresponds to a plane through origin

In other words, the 2D line is intersection of a plane through origin with the plane $z=1$.

Homogeneous line equation

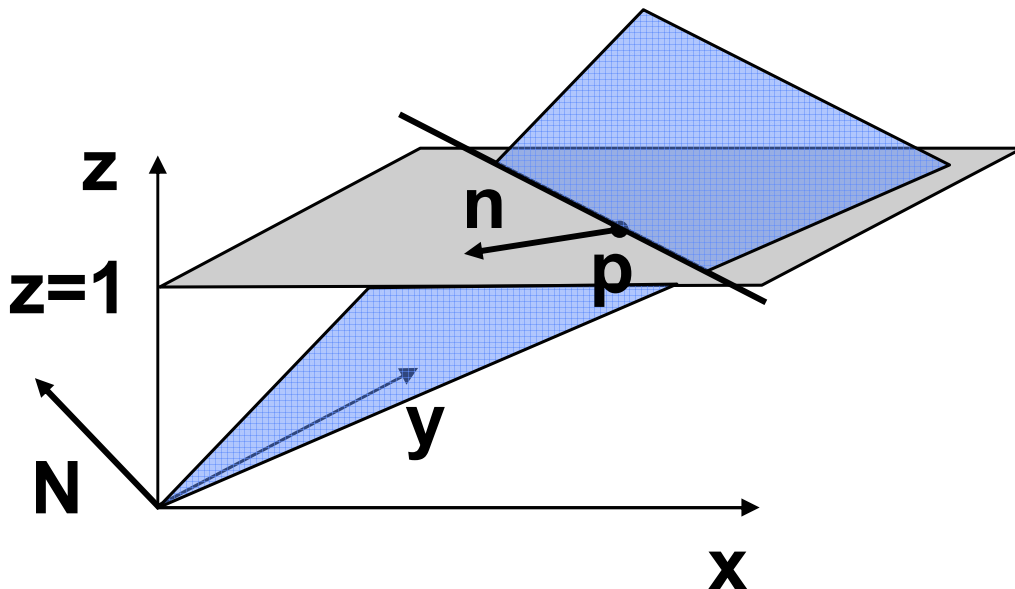
Rewrite the line equation:

$$(n, q - p) = n_x x + n_y y + (n, -p) = ([n_x, n_y, -(n, p)], [x, y, 1]) = (N, \bar{q})$$

where $N=[n_x, n_y, -(n, p)]$ is the normal to the plane corresponding to the line, and \bar{q} is the homogeneous form of $q=[x, y]$: $\bar{q}=[x, y, 1]$

Homogeneous form
of the line equation:

$$(N \cdot \bar{q}) = 0$$



Homogeneous coordinates **in 3D!**

regular 3D point to homogeneous:

$$\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \longrightarrow \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

homogeneous point to regular 3D:

$$\begin{pmatrix} p_x \\ p_y \\ p_z \\ p_w \end{pmatrix} \longrightarrow \begin{pmatrix} p_x/p_w \\ p_y/p_w \\ p_z/p_w \end{pmatrix}$$

Translation and scaling

Similar to 2D; translation by a vector

$$t = [t_x, t_y, t_z] \quad \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Nonuniform scaling in
three directions


$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotations around coord axes

angle θ , around X axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

around Y axis:

$$\begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


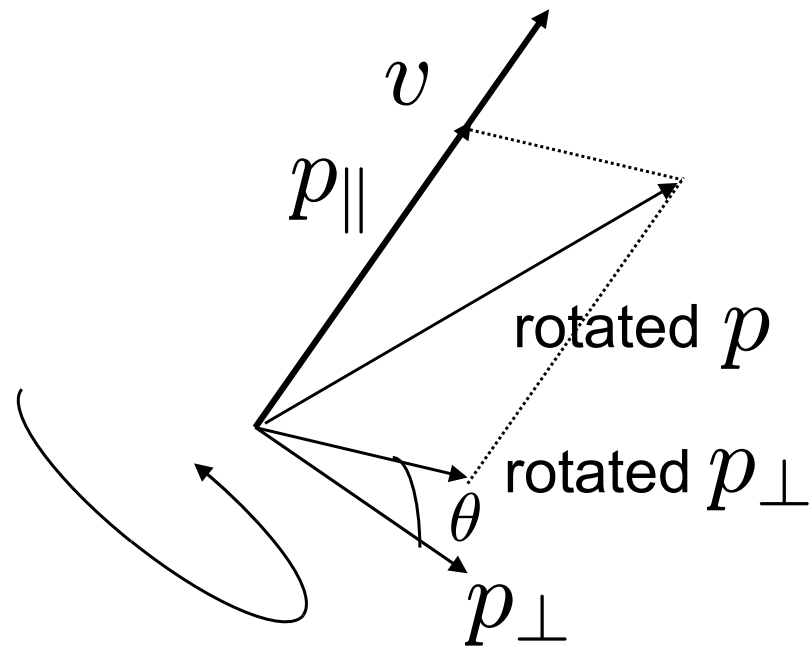
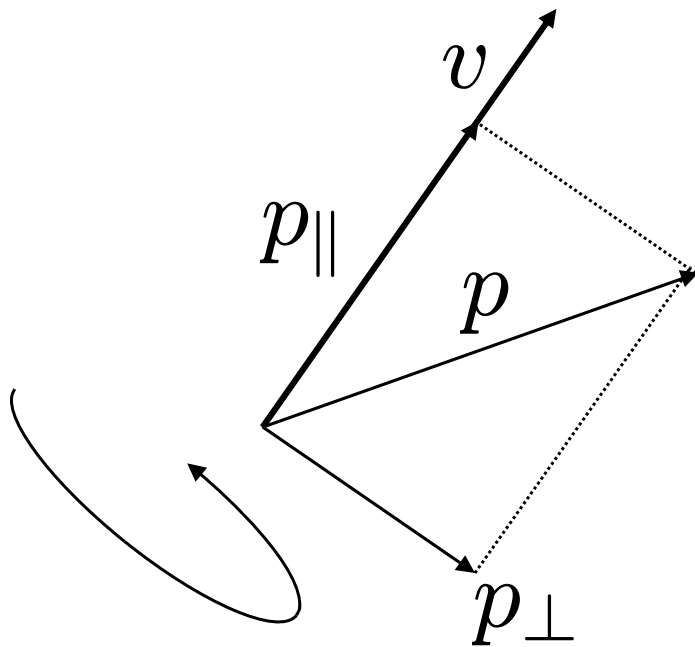
note where the minus is!

around Z axis:

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

General rotations

Given an axis (a unit vector) and an angle, find the matrix



Only the component perpendicular to axis changes

General rotations

(rotated vectors are denoted with ')

project p on v : $p_{\parallel} = (p, v)v$

the rest of p is
the other component: $p_{\perp} = p - (p, v)v$

rotate perp. component: $p'_{\perp} = p_{\perp} \cos \theta + (v \times p_{\perp}) \sin \theta$

add back two components: $p' = p'_{\perp} + p_{\parallel}$

Combine everything, using $v \times p_{\perp} = v \times p$ to simplify:

$$p' = \cos \theta p + (1 - \cos \theta)(p, v)v + \sin \theta(v \times p)$$

General rotations

How do we write all this using matrices?

$$p' = \cos \theta p + (1 - \cos \theta)(p, v)v + \sin \theta(v \times p)$$

$$(p, v)v = \begin{bmatrix} v_x v_x p_x + v_x v_y p_y + v_x v_z p_z \\ v_y v_x p_x + v_y v_y p_y + v_y v_z p_z \\ v_z v_x p_x + v_z v_y p_y + v_z v_z p_z \end{bmatrix} = \begin{bmatrix} v_x v_x & v_x v_y & v_x v_z \\ v_y v_x & v_y v_y & v_y v_z \\ v_z v_x & v_z v_y & v_z v_z \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

$$(v \times p) = \begin{bmatrix} -v_z p_y + v_y p_z \\ v_z p_x - v_x p_z \\ -v_y p_x + v_x p_y \end{bmatrix} = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

Final result, the matrix for a general rotation around a by angle θ :

$$\cos \theta \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + (1 - \cos \theta) \begin{bmatrix} v_x v_x & v_x v_y & v_x v_z \\ v_y v_x & v_y v_y & v_y v_z \\ v_z v_x & v_z v_y & v_z v_z \end{bmatrix} + \sin \theta \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}$$

Homogeneous Coordinates: Points vs Directions

Remember :

- Points have location - no direction or size
(w.r.t some origin)
- Vectors have size and direction but no location

Homogeneous Coordinates: Points vs Directions

- Directions, offset vectors - we don't want them to move with translation.
- $\langle x, y, 0 \rangle$ or $\langle x, y, z, 0 \rangle$ refers to directions

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

- “ideal point”, “point at infinity”
- also useful in perspective transformations

Some properties

- Difference of two points = vector

$$(x_1, y_1, z_1, l) - (x_2, y_2, z_2, l) = (x_1 - x_2, y_1 - y_2, z_1 - z_2, 0)$$

- Point + vector = point
- vector + vector = vector
- only meaningful to scale a vector

Transformation of Normals

- Normals don't get transformed the way you would expect them to.
- A normal vector is not defined between two points.
- Instead, it is a vector perpendicular to a vector defined between two points (the tangent)

$$\mathbf{n}^T \mathbf{t} = 0$$

Transformation of Normals

Let $t_M = Mt$, $n_N = Nn$. Find N s.t. $n_N^T t_M = 0$

$$n^T t = 0$$

$$n^T t = n^T I t = n^T M^{-1} M t$$

$$= (n^T M^{-1})(M t)$$

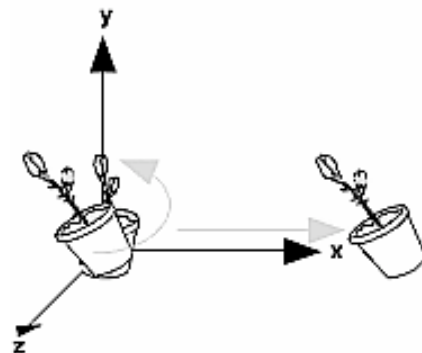
$$= (n^T M^{-1}) t_M$$

$$n^T N^T = n^T M^{-1}$$

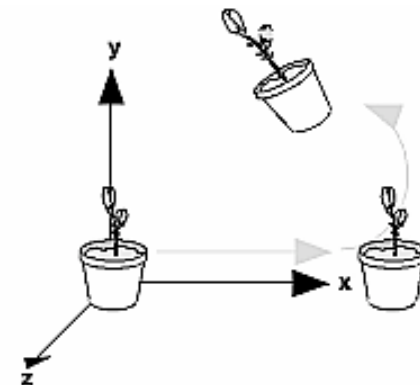
$$N = (M^{-1})^T$$

Composition of transformations

- **Order matters!** (rotation * translation \neq translation * rotation)
- **Composition of transformations = matrix multiplication:**
if T is a rotation and S is a scaling, then applying scaling first and rotation second is the same as applying transformation given by the matrix TS (note the order).
- **Reversing the order does not work in most cases**



Rotate then Translate



Translate then Rotate

Transformation order

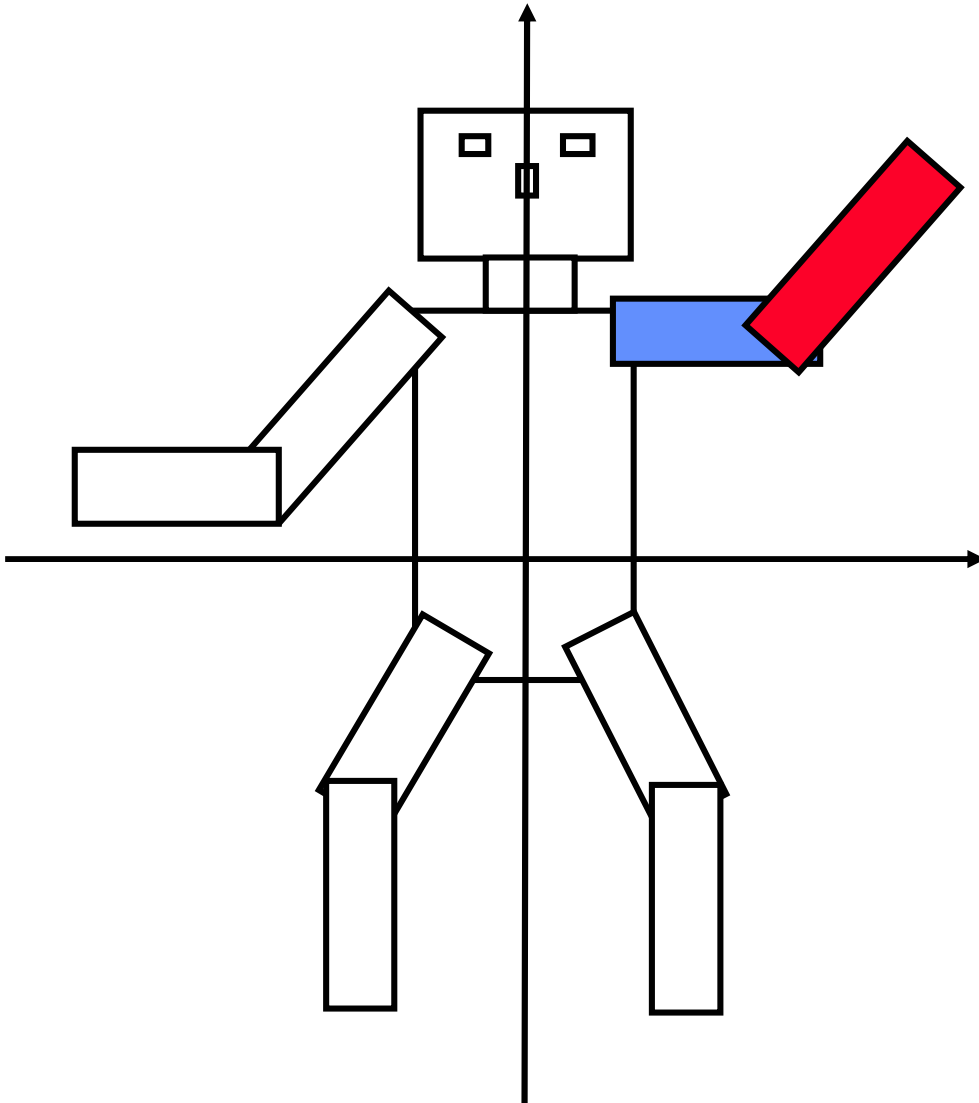
- When we write transformations using standard math notation, the closest transformation to the point is applied first:

$$T R S p = T(R(Sp))$$

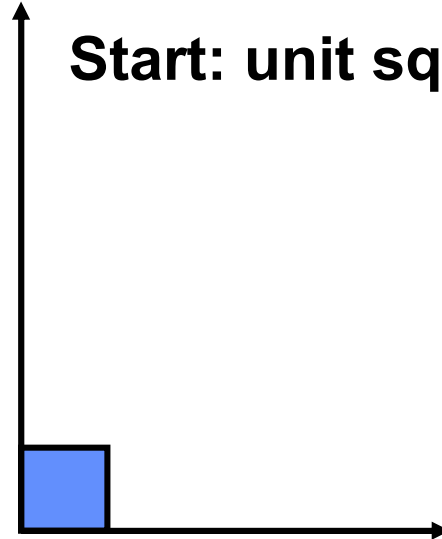
- first, the object is scaled, then rotated, then translated
- This is the most common transformation order for an object (scale-rotate-translate)

Hierarchical Transformations

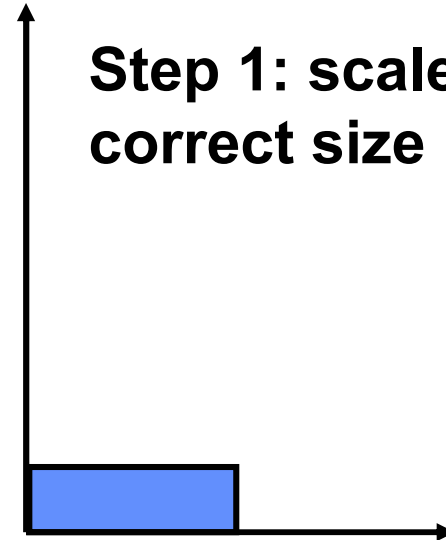
Building the arm



Start: unit square

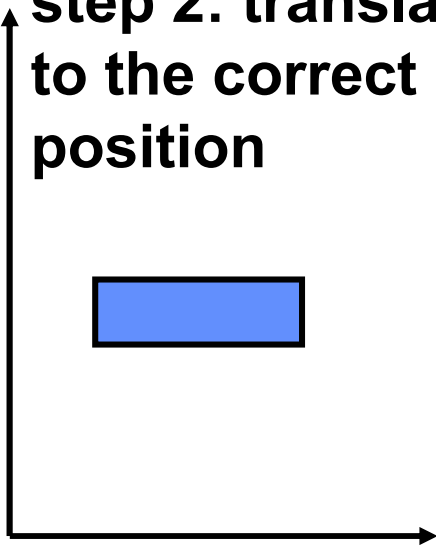


Step 1: scale to the correct size

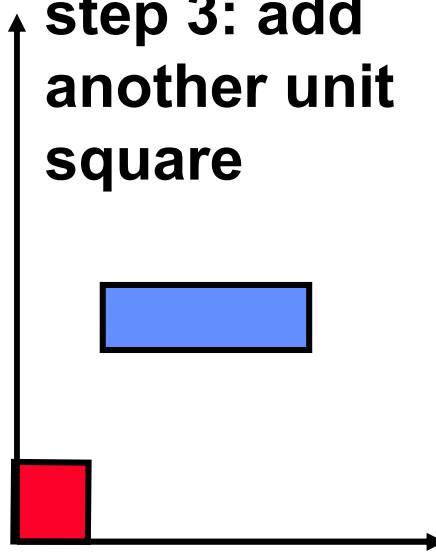


Building the arm

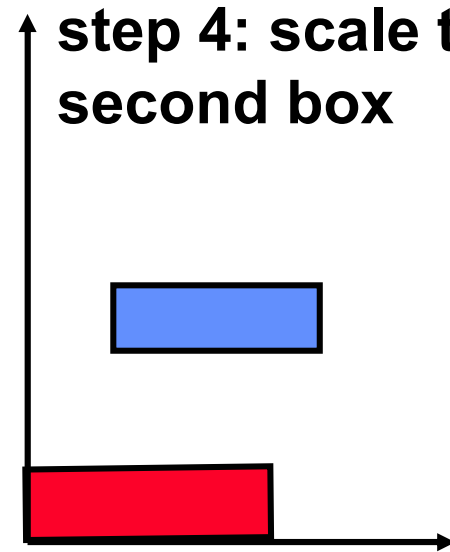
step 2: translate to the correct position



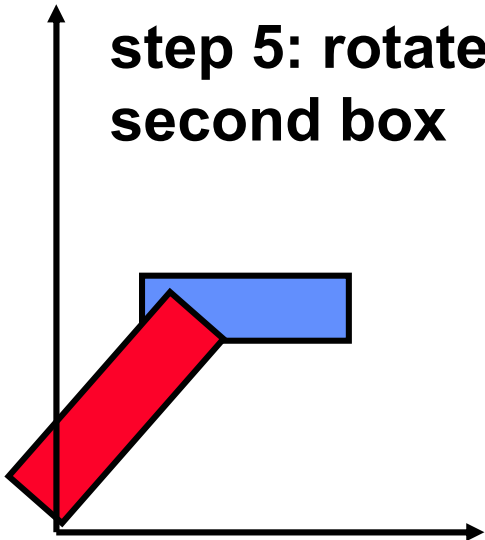
step 3: add another unit square



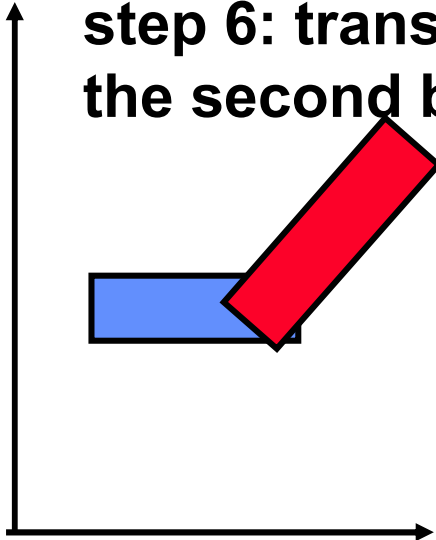
step 4: scale the second box



step 5: rotate the second box



step 6: translate the second box



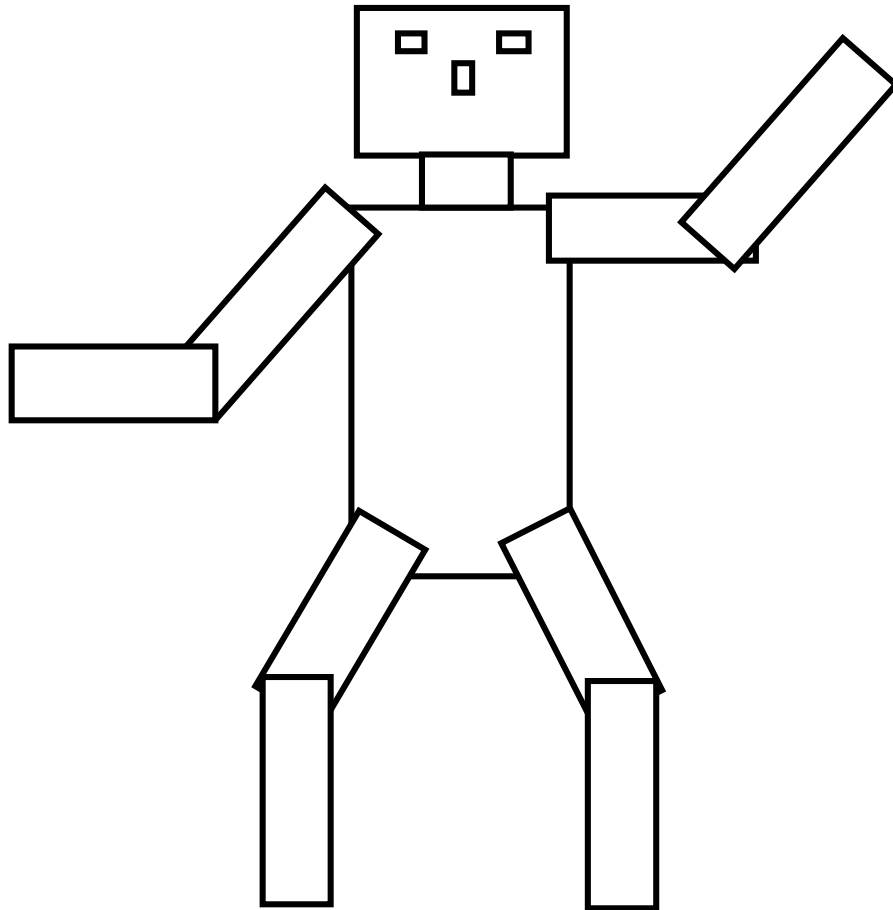
Hierarchical transformations

- **Positioning each part of a complex object separately is difficult**
- **If we want to move whole complex objects consisting of many parts or complex parts of an object (for example, the arm of a robot) then we would have to modify transformations for each part**
- **solution: build objects hierarchically**

Complex models

- can be built in a simple, modular fashion
- can be stored efficiently
- can be updated simply

Hierarchical transformations



Idea: group parts hierarchically,
associate transforms with each
group.

whole robot = head + body +
legs + arms

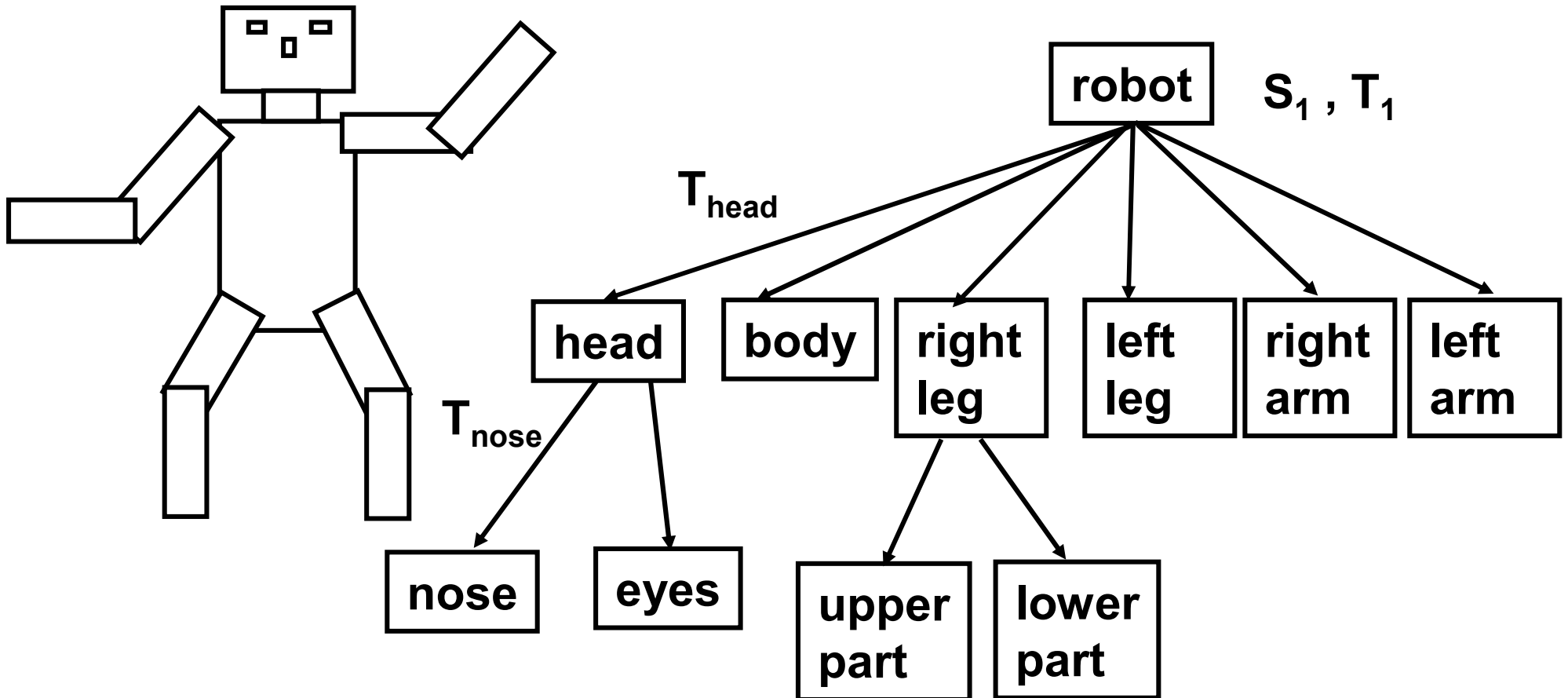
leg = upper part + lower part

head = neck + eyes + ...

Hierarchical transformations

- Hierarchical representation of an object is a tree.
- The non-leaf nodes are groups of objects.
- The leaf nodes are primitives (e.g. polygons)
- Transformations are assigned to each node, and represent the relative transform of the group or primitive with respect to the parent group
- As the tree is traversed, the transformations are combined into one

Hierarchical transformations



Transformation stack

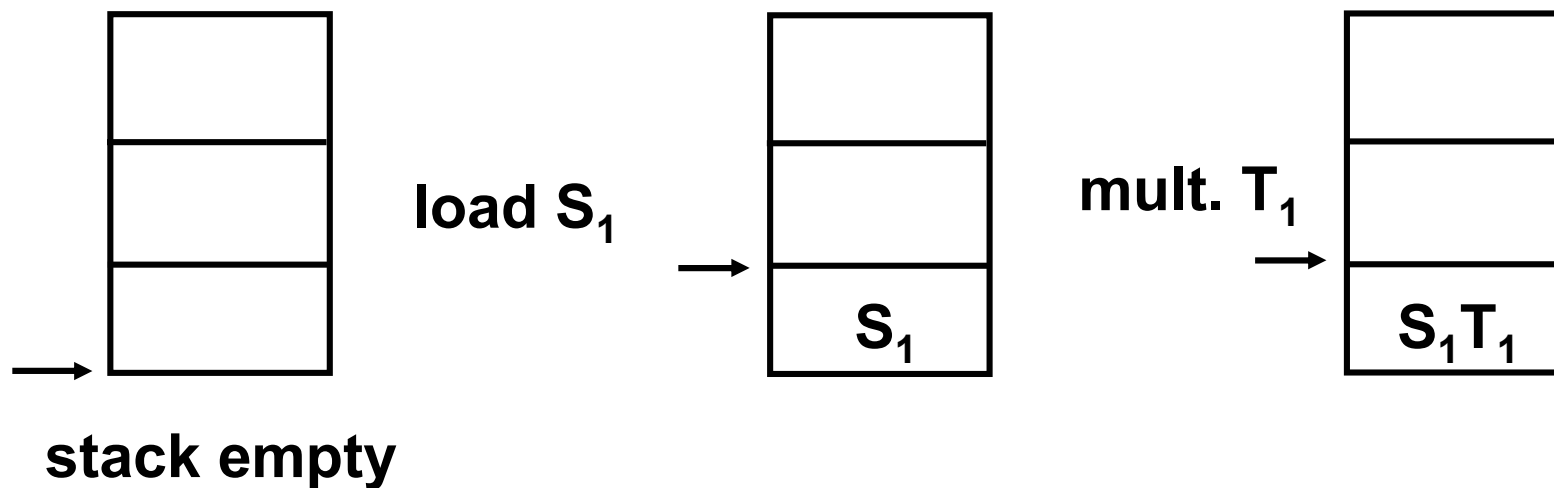
To keep track of the current transformation, the transformation stack is maintained.

Basic operations on the stack:

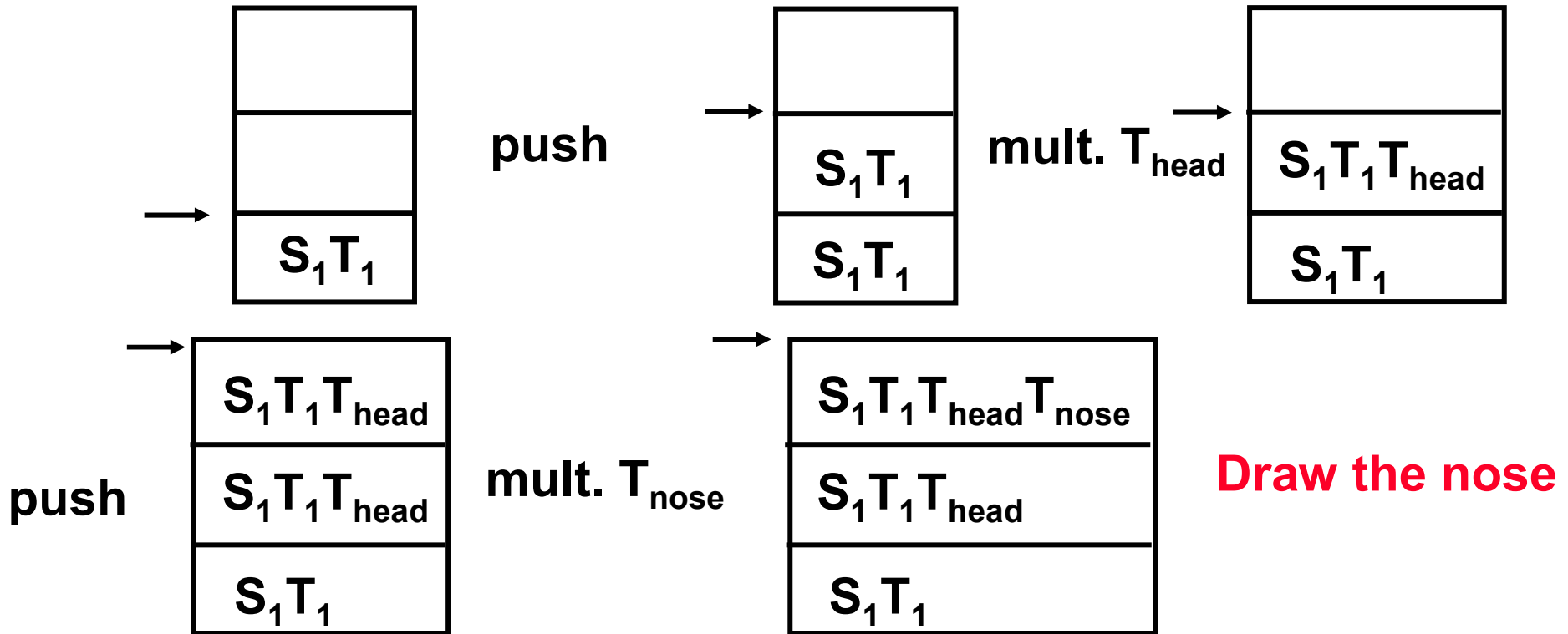
- **push: create a copy of the matrix on the top and put it on the top**
- **pop: remove the matrix on the top**
- **multiply: multiply the top by the given matrix**
- **load: replace the top matrix with a given matrix**

Transformation stack example

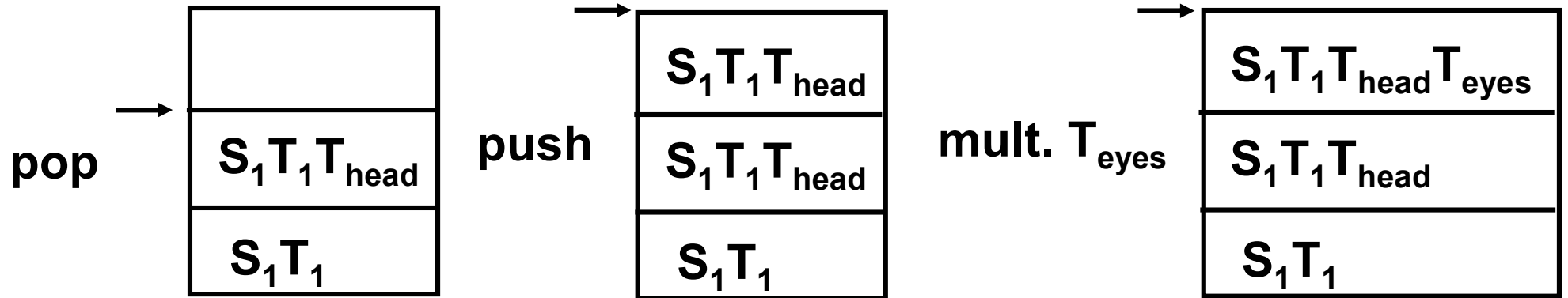
TO draw the robot, we use manipulations with the transform stack to get the correct transform for each part. For example, to draw the nose and the eyes:



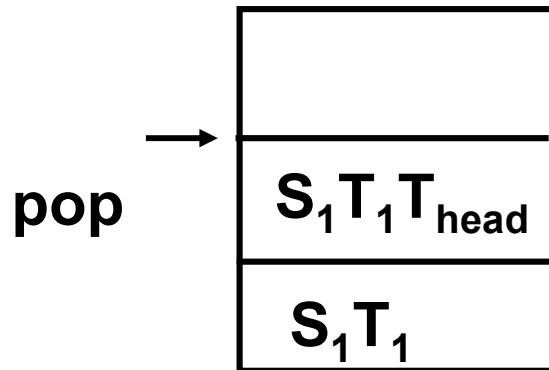
Transformation stack example



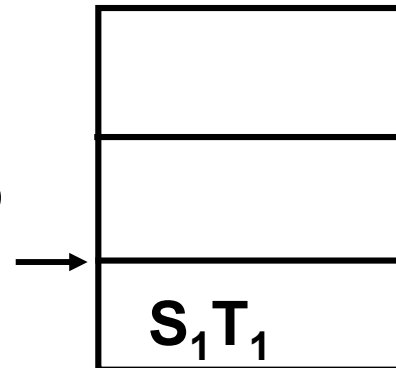
Transformation stack example



Draw the eyes



pop



Draw body etc...

Transformation stack example

Sequence of operations in the (pseudo)code:

```
load  $S_1$  ; mult  $T_1$ ;
```

```
push; mult.  $T_{\text{head}}$ ;
```

```
    push;
```

```
        mult  $T_{\text{nose}}$ ; draw nose;
```

```
    pop;
```

```
    push;
```

```
        mult.  $T_{\text{eyes}}$ ; draw eyes;
```

```
    pop;
```

```
pop;
```

Animation

The advantage of hierarchical transformations is that everything can be animated with little effort.

General idea: before doing a mult. or load, compute transform as a function of time.

```
time = 0;
main loop {
    draw(time);
    increment time;
}
```

```
draw( time ) {
    ...
    compute  $R_{\text{arm}}(\text{time})$ 
    mult.  $R_{\text{arm}}$ 
    ...
}
```

Some OpenGL

`glMatrixMode(GL_MODELVIEW)` Only 1 matrix!

`glLoadIdentity();`

`glMultMatrixf(N);`

Matrix = NM

`glMultMatrixf(M)`

Transformations applied
in opposite order

`glBegin`

...

`glEnd`

Some OpenGL

- `glLoadMatrix`
- `glMultMatrix`
- `glPushMatrix`
- `glPopMatrix`

- `glTranslate`
- `glRotate`
- `glScale`

Some Troubleshooting Tips

- all drawing in “display” function
- Matrices in column-major mode
- make sure current color is not the same as background
- Projection(perspective, ortho) - clipping planes
- viewpoint - default: at origin, looking down z.
- rotation axis
- homogeneous coordinates