
3D transformations

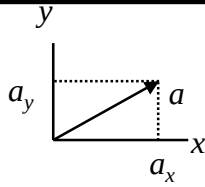
© 1999, Denis Zorin

Uses of transformations

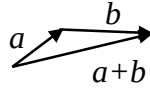
- **Modeling:** position and resize parts of a complex model;
- **Viewing:** define and position the virtual camera
- **Animation:** define how objects move/change with time

© 1999, Denis Zorin

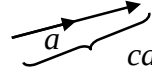
Review of vectors



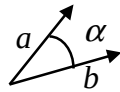
Addition:



Multiplication by a number c:



Scalar (dot) product
(a number):



$$2D: (a,b) = a_x b_x + a_y b_y$$

$$3D: (a,b) = a_x b_x + a_y b_y + a_z b_z$$

$$(a,b) = |a||b|\cos\alpha \quad (a,b) = (b,a)$$

$$(a, k_1 b + k_2 c) = k_1 (a,b) + k_2 (a,c)$$

Cross product
(a vector):

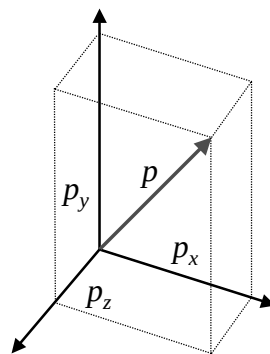
$$3D: a \times b = \det \begin{pmatrix} \mathbf{e}_x & \mathbf{e}_y & \mathbf{e}_z \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{pmatrix} \quad |a \times b| = |a||b|\sin\alpha$$

$$a \times b = -b \times a$$

$$a \times (k_1 b + k_2 c) = k_1 a \times b + k_2 a \times c$$

© 1999, Denis Zorin

Vectors in 3D



$$p = p_x e_x + p_y e_y + p_z e_z$$

unit basis vectors

© 1999, Denis Zorin

Homogeneous coordinates

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:

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

© 1999, Denis Zorin

Points vs. vectors

- a translated point is a different point
- a translated vector is the same vector
- in homogeneous coordinates:
 - points have last component nonzero
 - vectors have last component zero

© 1999, Denis Zorin

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

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

© 1999, Denis Zorin

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!

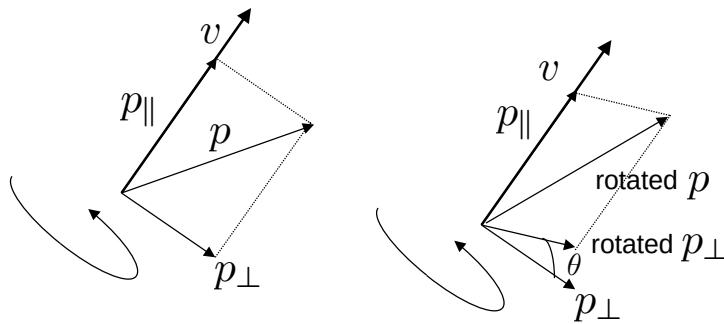
around Z axis:

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

© 1999, Denis Zorin

General rotations

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



Only the component perpendicular to axis changes

© 1999, Denis Zorin

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:

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

© 1999, Denis Zorin

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 :

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

© 1999, Denis Zorin

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**

© 1999, Denis Zorin

Transformation order

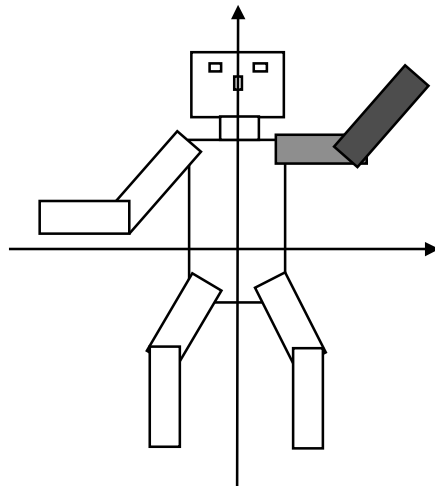
- When we write transformations using standard math notation, the closest transformation to the point is applied first:

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

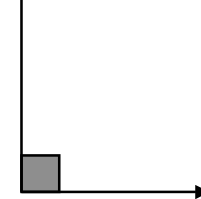
- first, the object is scaled, then rotated, then translated
- This is the most common transformation order for an object (scale-rotate-translate)

© 1999, Denis Zorin

Building the arm



Start: unit square

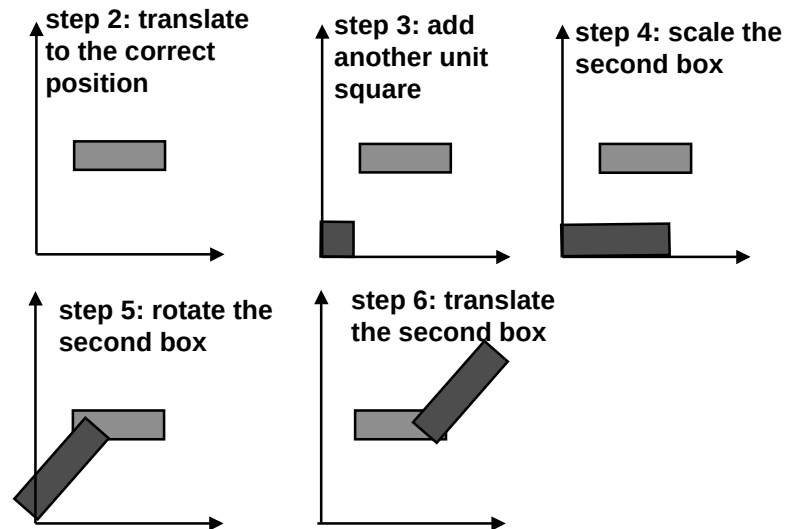


Step 1: scale to the correct size



© 1999, Denis Zorin

Building the arm



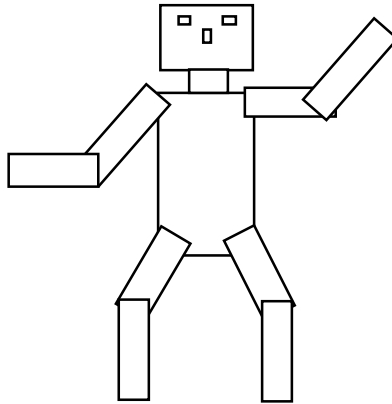
© 1999, Denis Zorin

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

© 1999, Denis Zorin

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 + ...

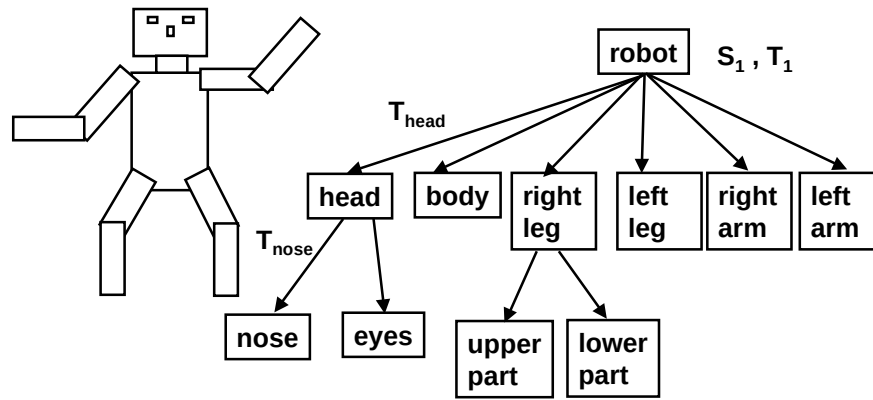
© 1999, Denis Zorin

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

© 1999, Denis Zorin

Hierarchical transformations



© 1999, Denis Zorin

Transformation stack

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

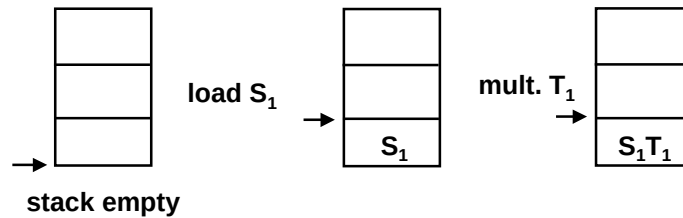
Basic operations on the stack:

- 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

© 1999, Denis Zorin

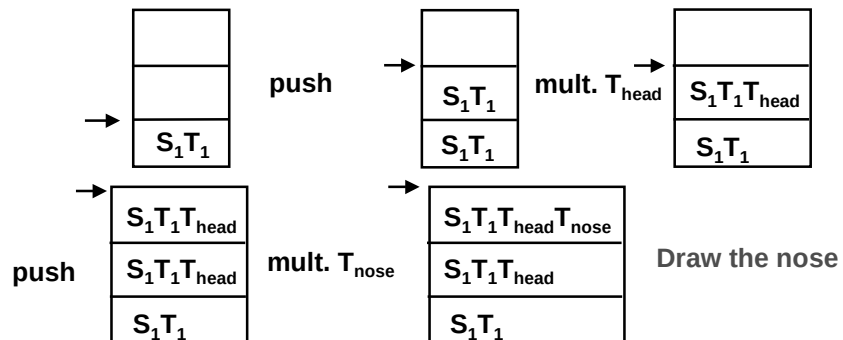
Transformation stack example

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:



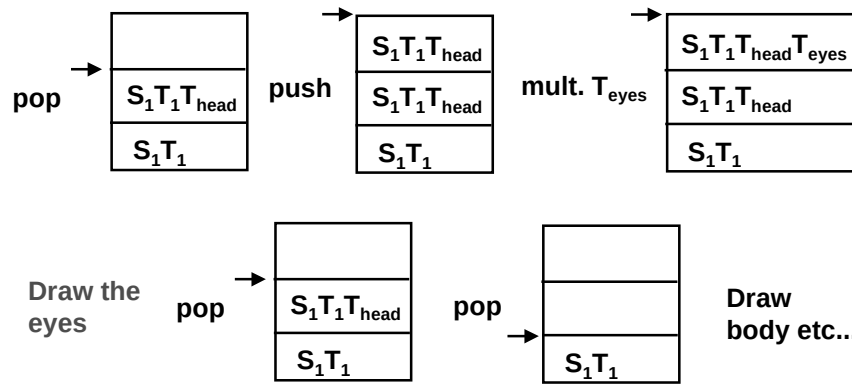
© 1999, Denis Zorin

Transformation stack example



© 1999, Denis Zorin

Transformation stack example



© 1999, Denis Zorin

Transformation stack example

Sequence of operations in the (pseudo)code:

```
load  $S_1$  ; mult  $T_1$ ;
push; mult.  $T_{head}$ ;
  push;
    mult  $T_{nose}$ ; draw nose;
  pop;
  push;
    mult.  $T_{eyes}$ ; draw eyes;
  pop;
pop;
```

© 1999, Denis Zorin

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  $R_{arm}(time)$   
    mult.  $R_{arm}$   
    ...  
}
```

© 1999, Denis Zorin