3D transformations and hierarchical modelling

Dr Nicolas Holzschuch
University of Cape Town
e-mail: holzschu@cs.uct.ac.za

Map of the lecture

- Homogeneous coordinates in 3D
- Geometric transformations in 3D
  - translations, rotations, scaling,…
- Hierarchical modelling:
  - the need for hierarchical modelling
  - how to do it?

Homogeneous coordinates in 3D

- In order to model all transformations as matrices:
  - introduce a fourth coordinate, w $\longrightarrow$ $\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$
  - two vectors are equal if: $x/w = x'/w'$, $y/w = y'/w'$ and $z/w=z'/w'$
- All transformations are 4x4 matrices

Translations in 3D

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{cases} x' = x + wt_x \\ y' = y + wt_y \\ z' = z + wt_z \\ w' = \quad w \end{cases}$$

Scaling in 3D

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{cases} x' = s_x x \\ y' = s_y y \\ z' = s_z z \\ w' = w \end{cases}$$

Rotations in 3D

- One rotation: one axis and one angle
- Matrix depends on both axis and angle
  - direct expression possible, from axis and angle, using cross-products
- Rotations about axis have simple expression
  - other rotations express as composition of these rotations

Rotation around x-axis

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

x-axis is unmodified

Sanity check: a rotation of $\pi/2$ should change y in z, and z in -y

$$R_x(\tfrac{\pi}{2}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation around y-axis

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

y-axis is unmodified

Sanity check: a rotation of $\pi/2$ should change z in x, and x in $-z$

$$R_y(\tfrac{\pi}{2}) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about z-axis

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

z-axis is unmodified

Sanity check: a rotation of $\pi/2$ should change x in y, and y in $-x$

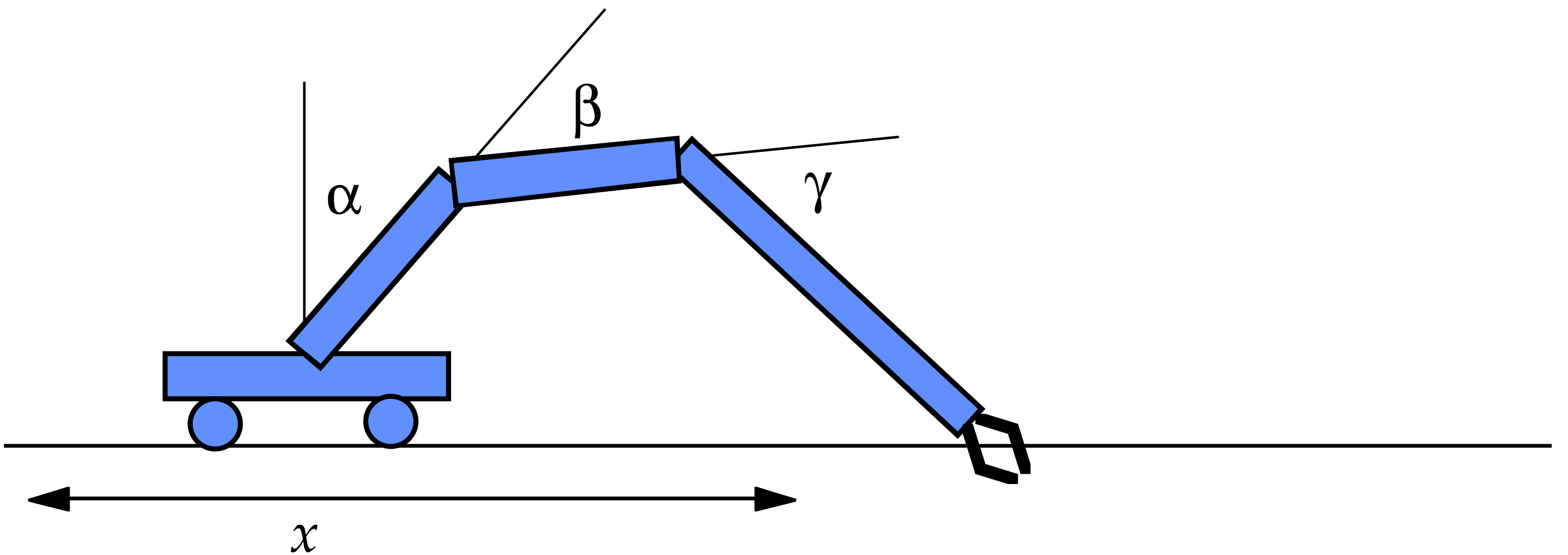
$$R_z(\tfrac{\pi}{2}) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Any transformation in 3D

- All transformations in 3D can be expressed as combinations of translations, rotations, scaling
  - expressed using matrix multiplication
- Transformations can be expressed as 4x4 matrices

Defining complex objects

Our problem:



Defining complex objects

- Object defined as a combination of smaller objects:
  - robot, car, tire
- Ensure a consistent behaviour:
  - the object stays connected
  - If I move the hand, the arm follows
- Use "natural" parameters: x,α,β,γ

How to do this?

- Easier to specify the position of the wheel with respect to the car
- Easier to specify the position of the bolts on the wheel with respect to the wheel
- We don't use absolute coordinates in life

Relative coordinates

- Use relative coordinates:
  - specify the position of the forearm with respect to the arm
- Using concatenation of transformations:
  - translate to the arm position
  - draw the arm
  - translate to the forearm position *relative to the arm*
  - draw the forearm

Concatenation of transformations

- Sometimes I want to go back to the origin:
  - I finished drawing the hand, now it’s the other arm
  - better specify the position of the other arm with respect to the body (instead of the arm)
- I need the possibility to go back

Transformations stack

- Keep current transformation information
  - initially = **M**, from model to viewport
  - **M′=MT** (translation by x)
  - draw robot body
  - **M″=M′T1** (translation to center of 1st wheel)
  - draw first wheel as circle of center (0,0)
  - return to **M′**
  - **M‴=M′T2** (translation to center of 2nd wheel)
  - draw second wheel

Stack in graphics libraries

- OpenGL:
  - `popmatrix(), pushmatrix()`
- SPHIGS:
  - `openStructure(), closeStructure()`
- Postscript:
  - `gsave, grestore`

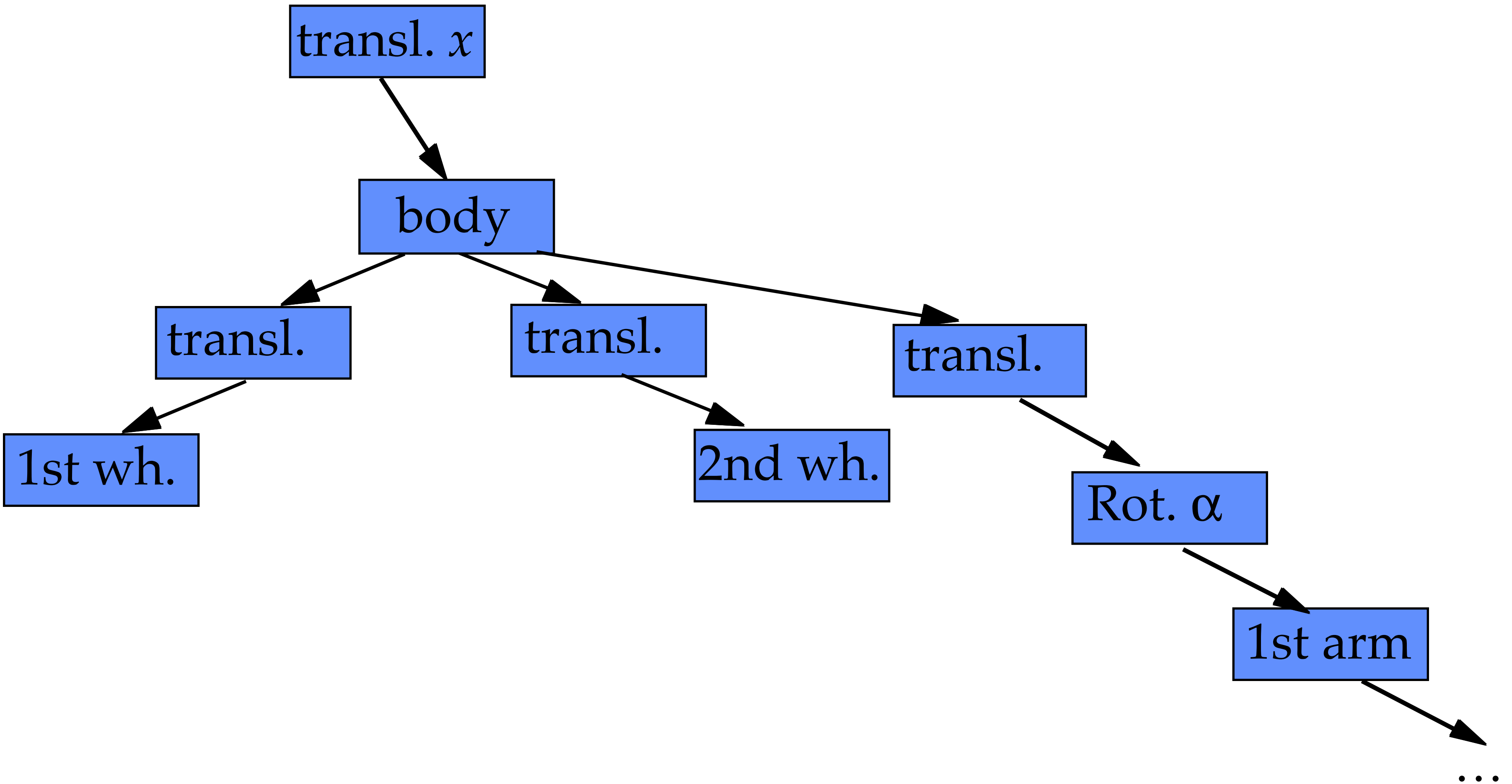
Sample implementation

- Set transformation as projection matrix
- translate by x (concatenate translation matrix with transformation matrix)
- draw car body
- save transformation matrix
  - translate+rotate
  - draw first wheel
- restore transformation matrix
- save transformation matrix
  - translate+rotate
  - draw second wheel
- restore transformation matrix

Hierarchical definition

- How to make sure you're having the right transformation?
- How to know it's time to go back to previous transformation?
- Define your object hierarchically
- Drawing = traversal of the tree

Object defined hierarchically



Object hierarchy: conclusion

- Define your object as a tree:
  - specify parts position *relative* to others
  - use a transformation stack
- Interests:
  - easy variation of parameters
  - objects are re-usable
    - one procedure for all four wheels
  - ensured consistency