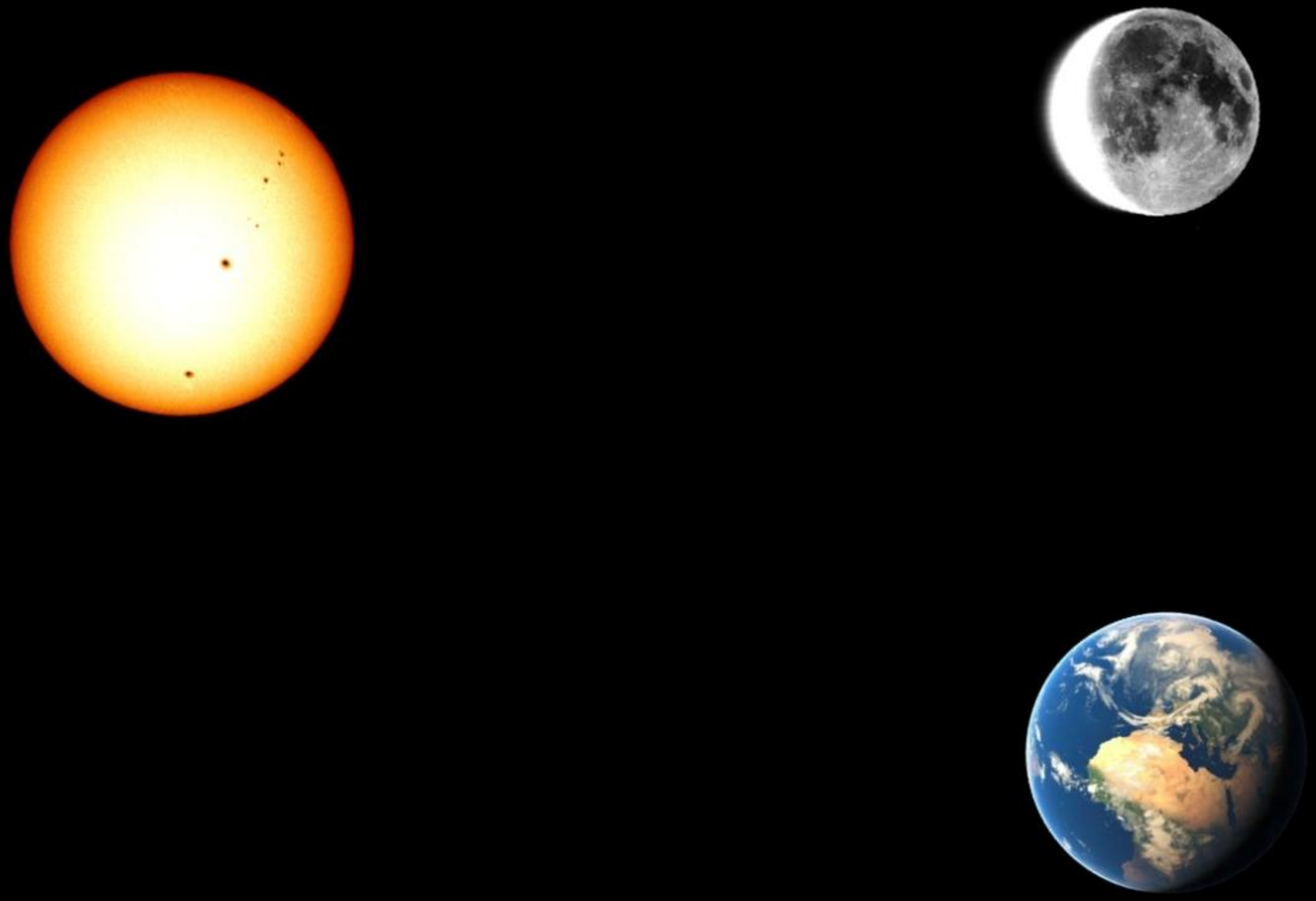
Animation (Hierarchies)

Rachel McDonnell
Associate Professor in Creative Technologies
ramcdonn@tcd.ie

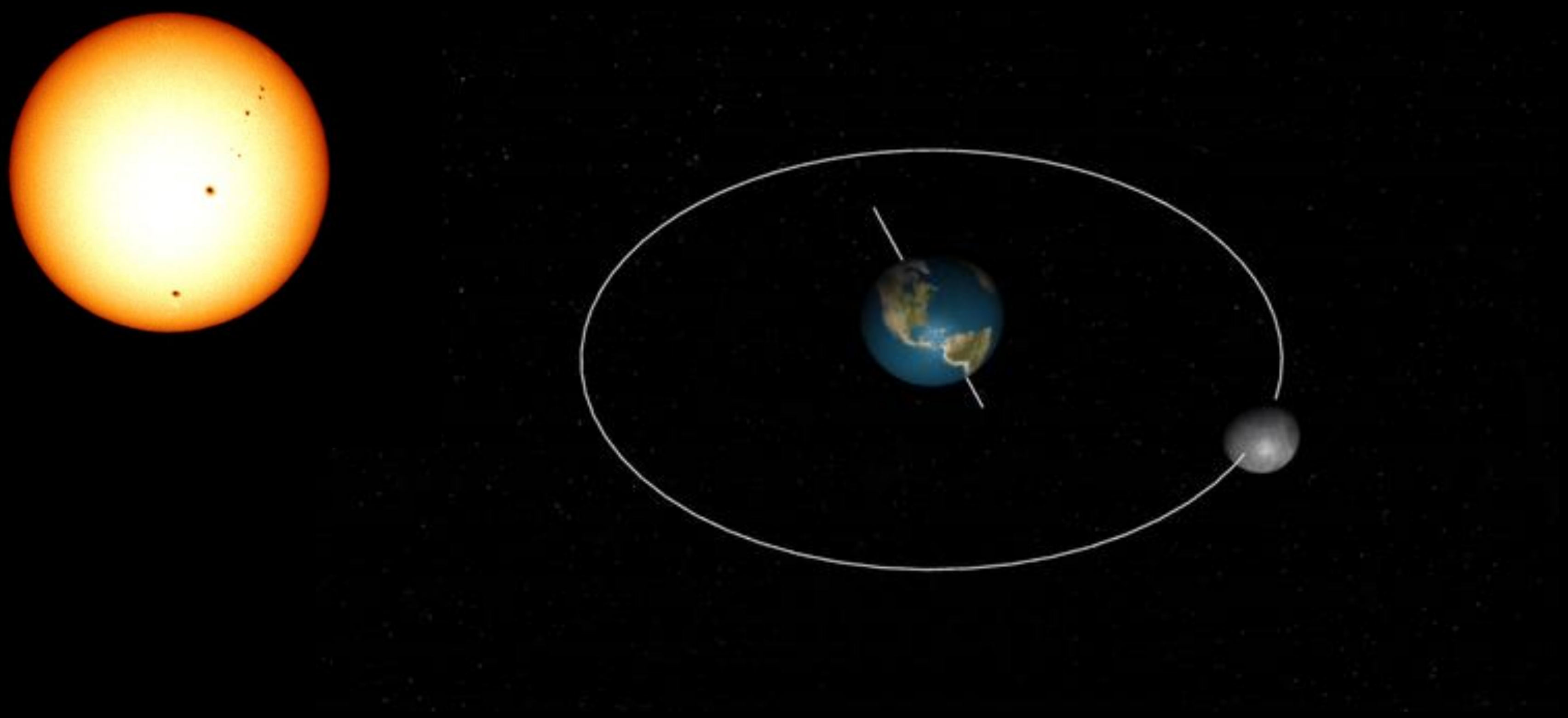
Course www: **Blackboard**

Credits: Some notes taken from Prof. Jeff Chastine

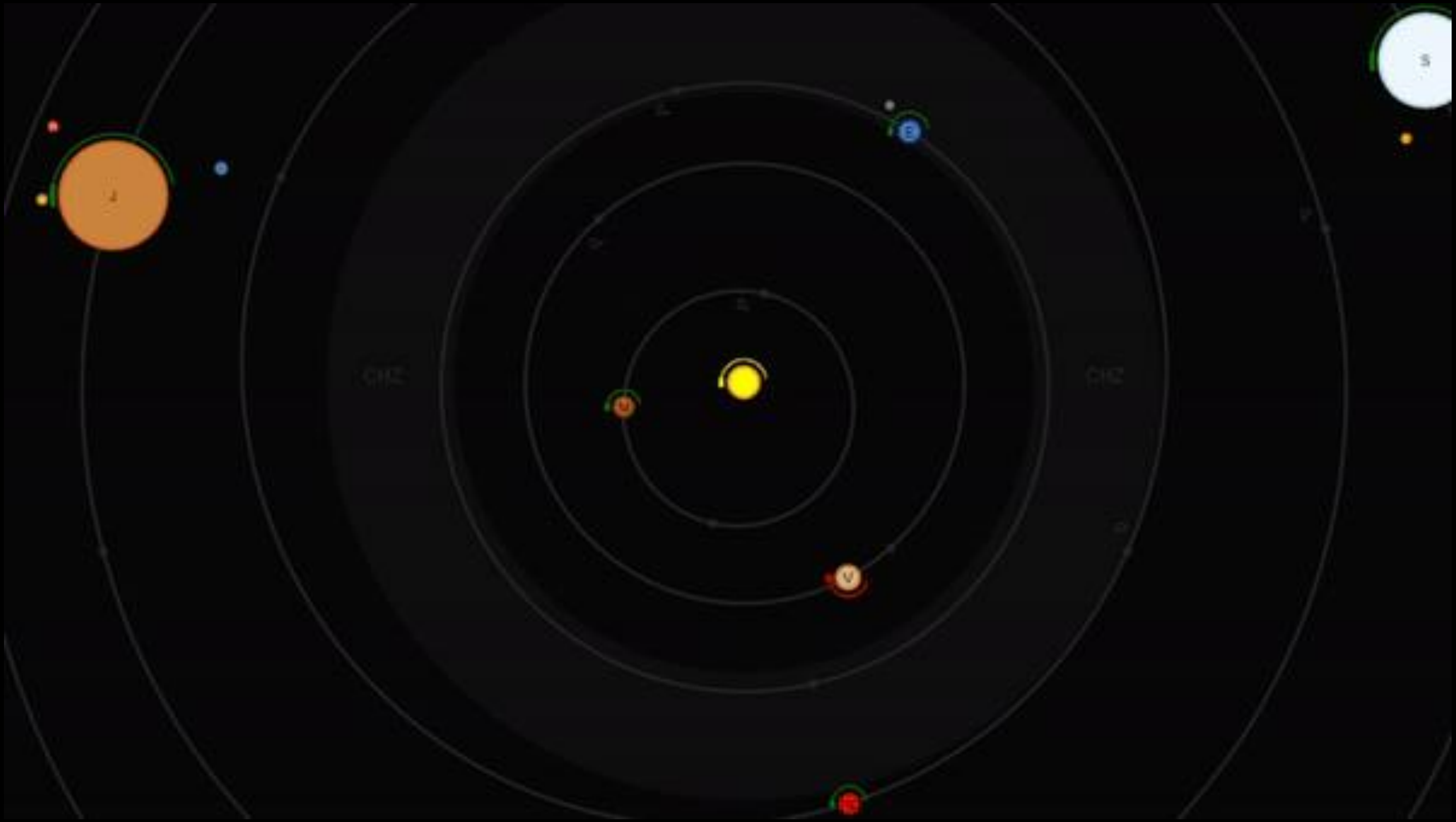
Relative Motion



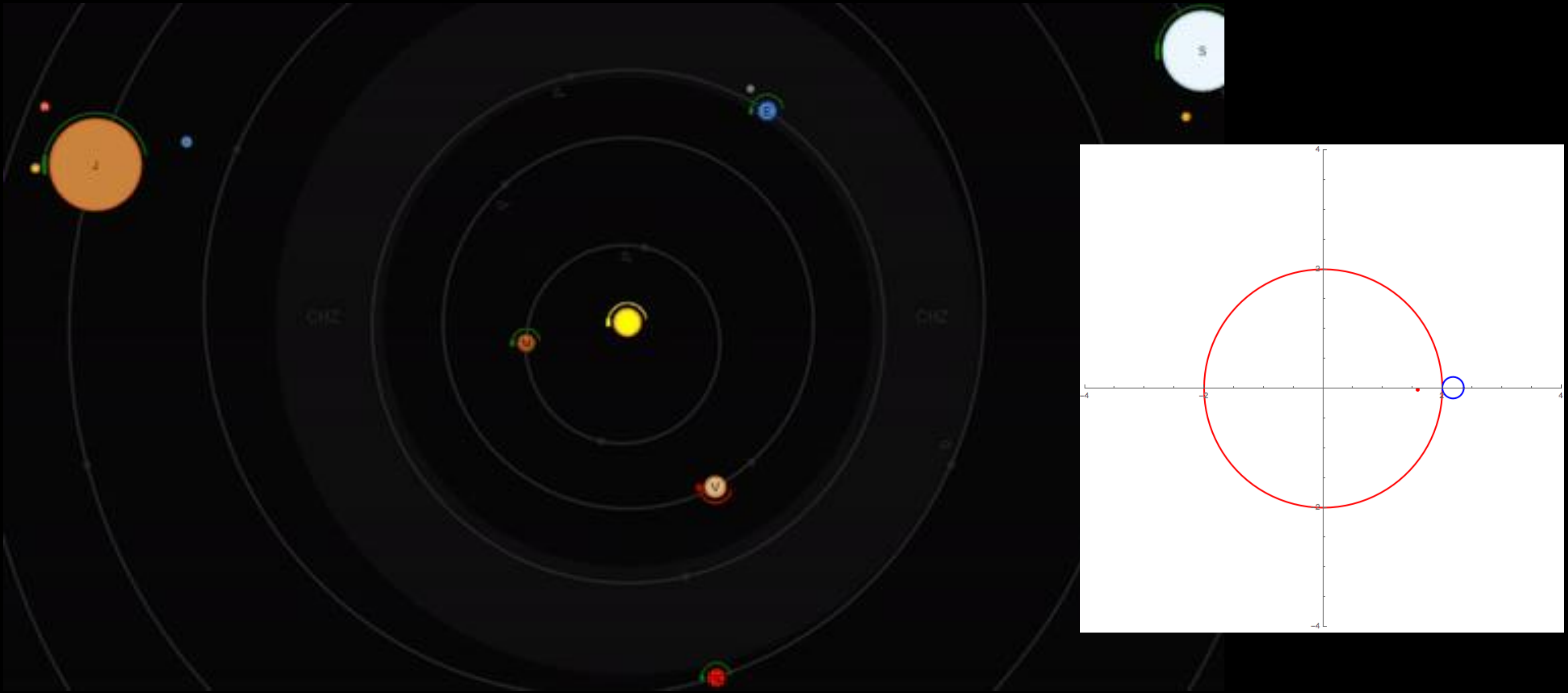
Relative Motion



Relative Motion



Relative Motion



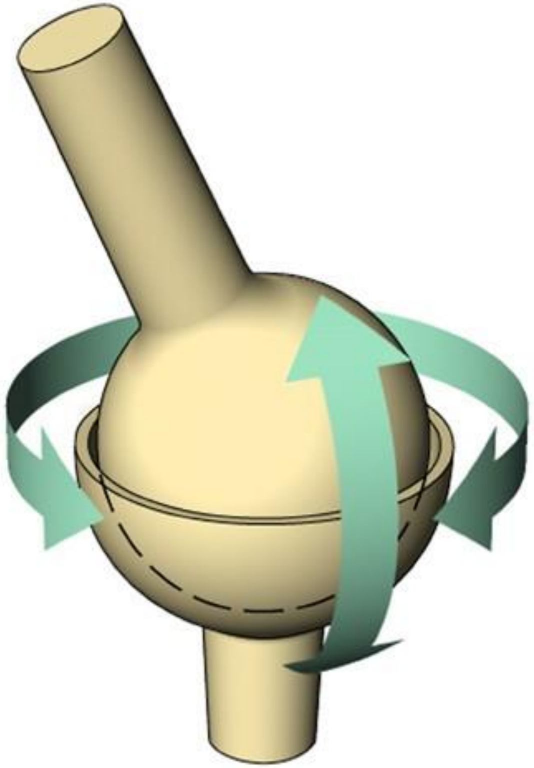
Relative Motion

- describe a motion as relative to another one.
  - why?
    - simpler to understand and easy to model with math.

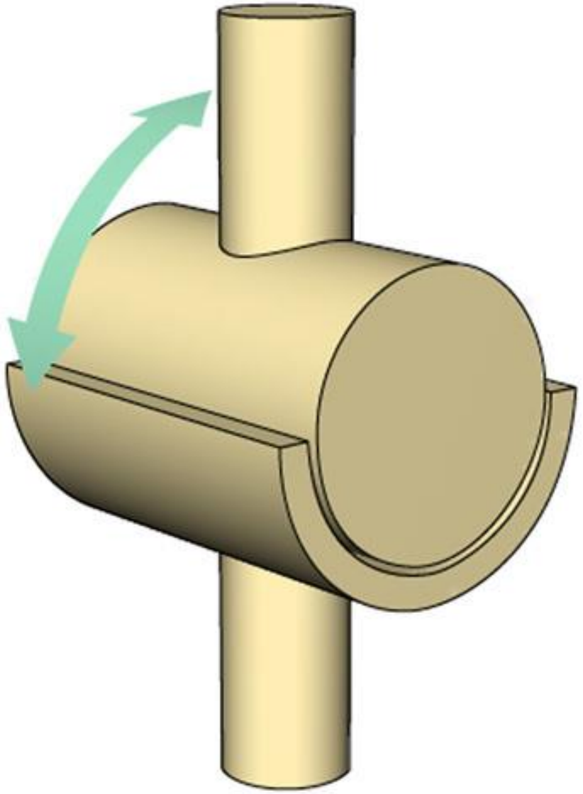
    → useful to animate specific elements (like a human character).
- when more than one is called a *motion hierarchy.*
- typically: components of a hierarchy represent objects that are physically connected or *linked*.
- we can induce constraints with hierarchies:
  - reduced motion's freedom.
- two approaches for animating figures defined by hierarchies: forward & inverse kinematics (future lectures).

Degrees of Freedom

rotational joints example:



3 degrees of freedom
pitch, roll and yaw



1 degrees of freedom

rotational and translational example:

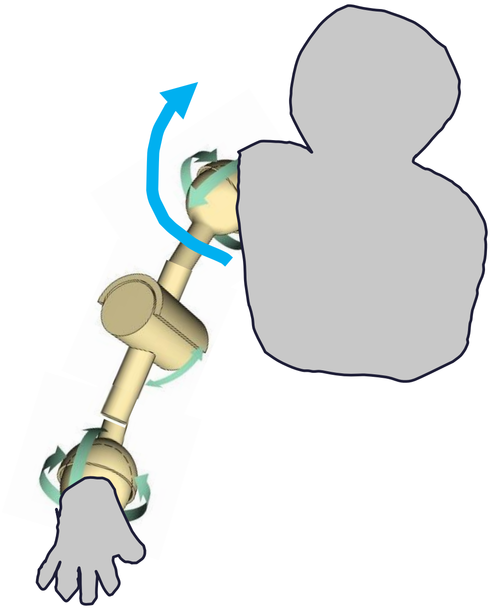
6 degrees of freedom
(no constraints)

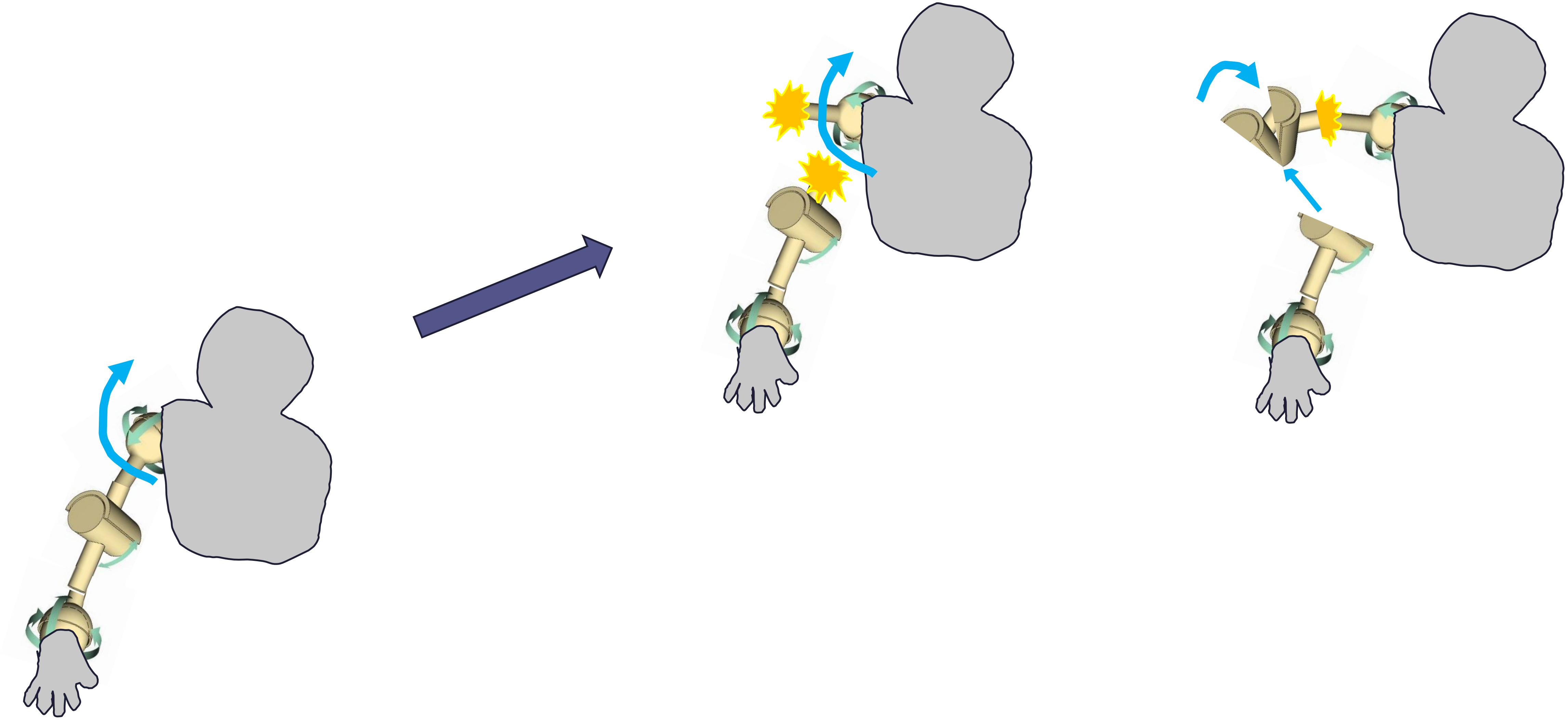
Model Transformations

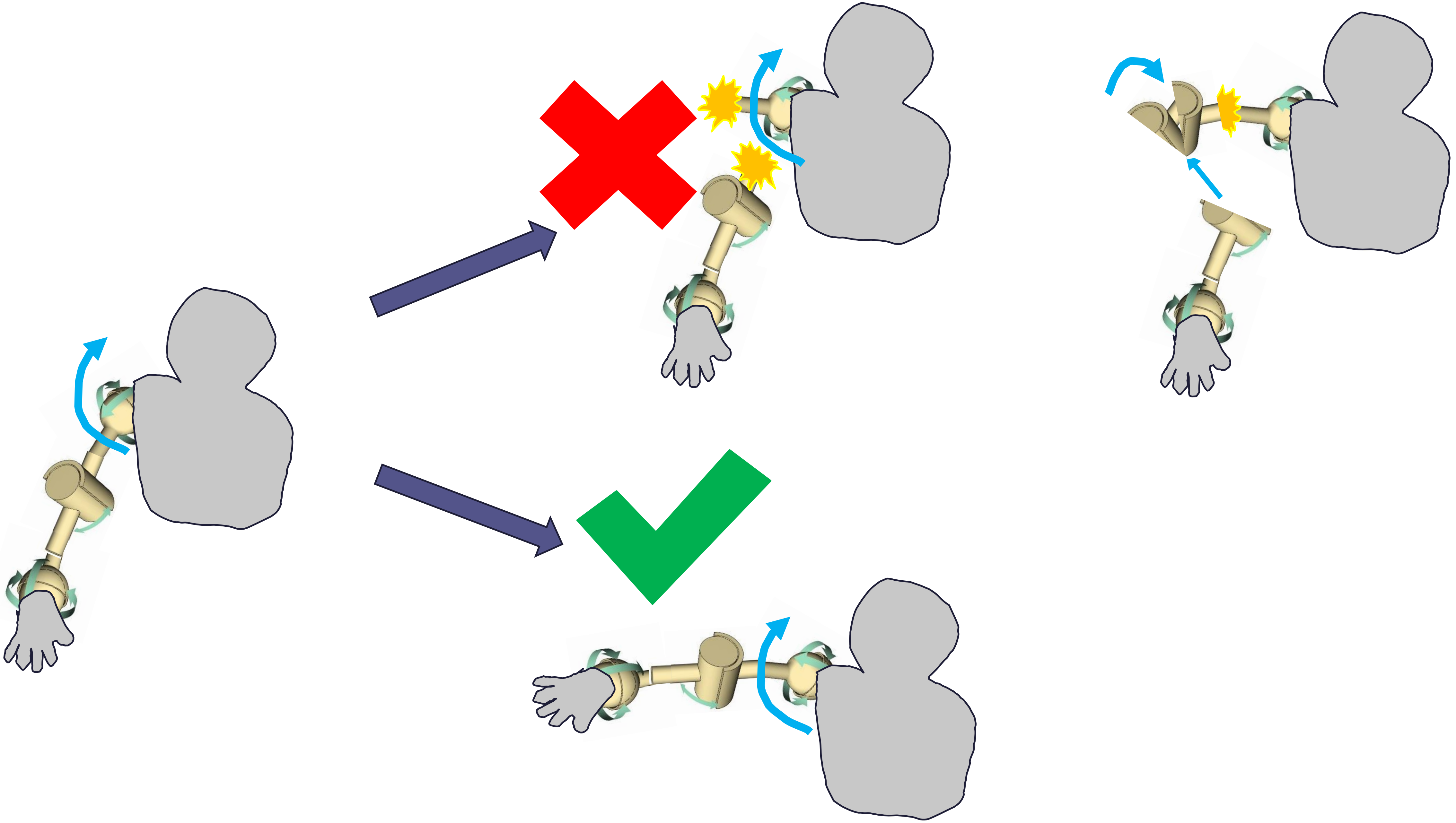
- thinking in “local frame view” is usually simpler.
- and mathematically easy to extends to a *hierarchical model*.

  → like for *jointed assemblies*, such as articulated figures (animals, robots etc.)
- each sub-component has its own *local frame*.
- changes made to the *parent frame* are propagated down to the *child frames* (thus all models in a branch are globally controlled by the parent).
- This simplifies the specification of *animation*.

note: frame = coordinate system



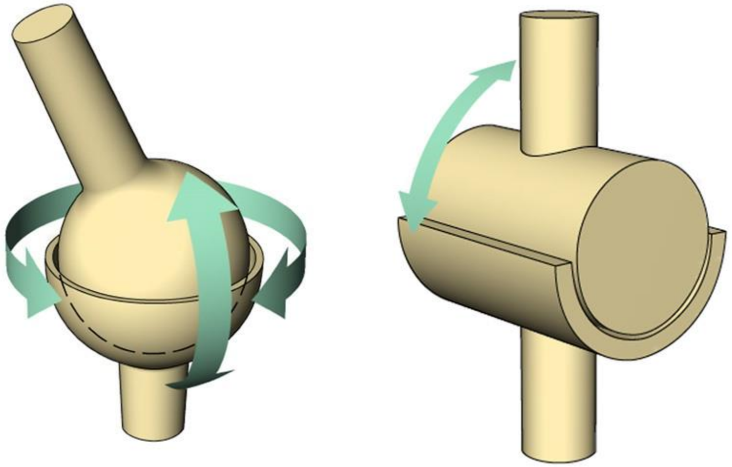


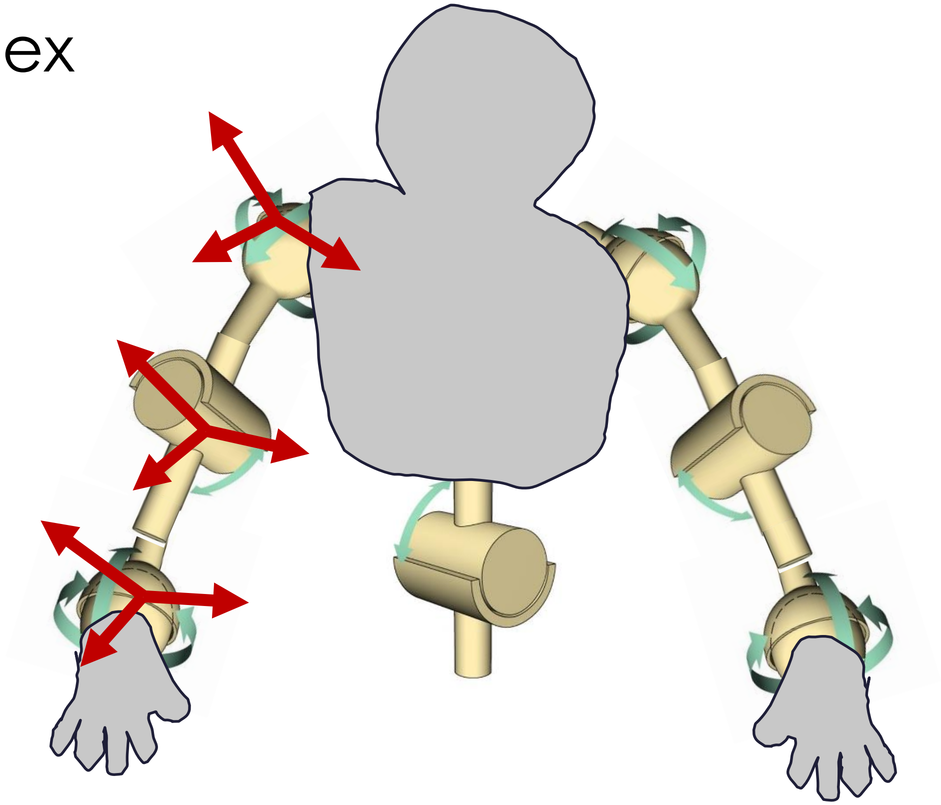


Hierarchical Transformations

This model are typically, used to model complex articulated assembly that already presents an implicit *hierarchy* (like the *human body*).

→ to do so we associate *local frames* with each *sub-objects* in the assembly.

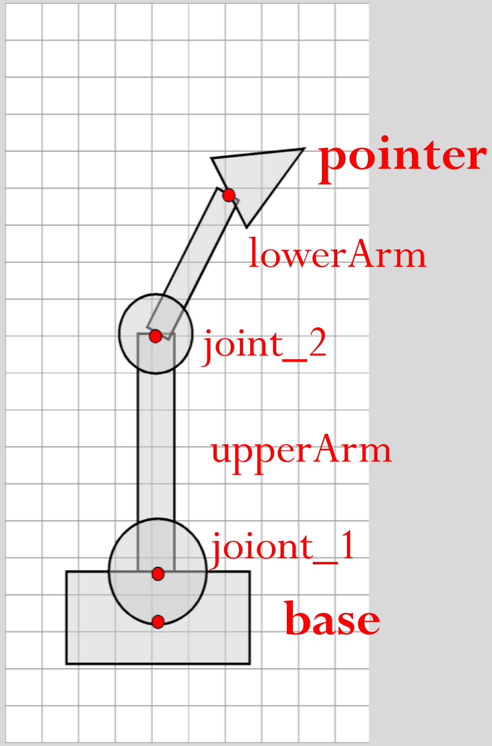




Hierarchical Transformations

how we construct a hierarchy?

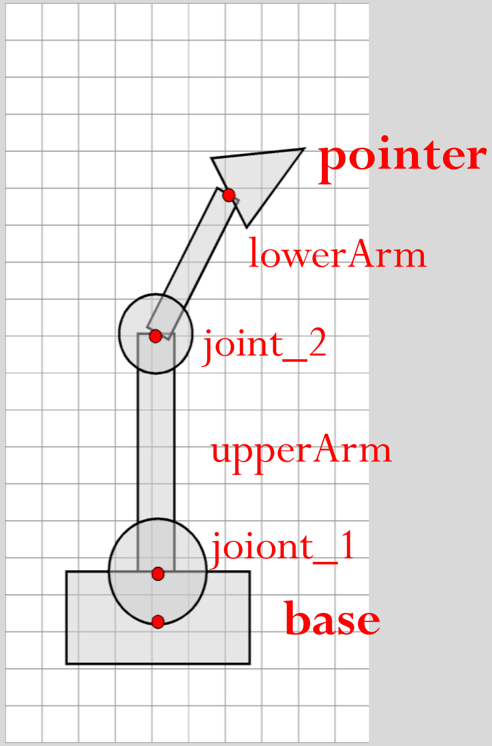
- *we relate parent-child frames* via a transformation (matrix).
  - this relation is described by a *tree*:
  - where each node has its own *local co-ordinate system*.

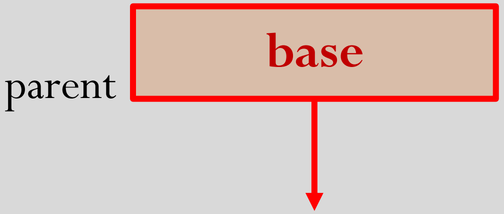


Hierarchical Transformations

how we construct a hierarchy?

- *we relate parent-child frames* via a transformation (matrix).
  - this relation is described by a *tree*:
  - where each node has its own *local co-ordinate system*.

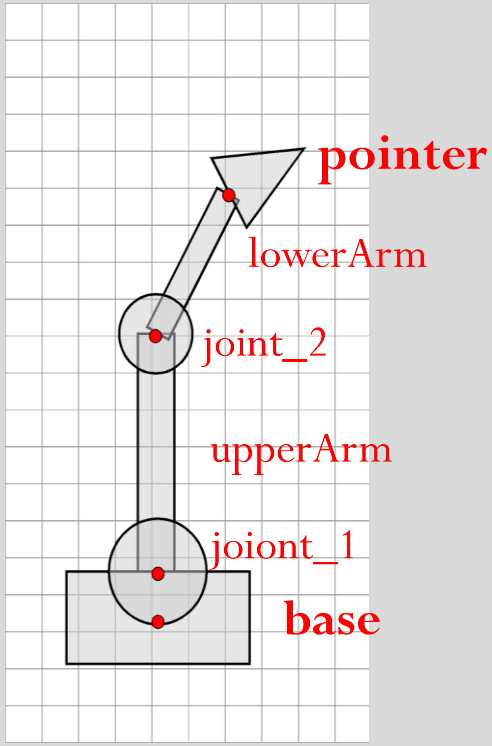


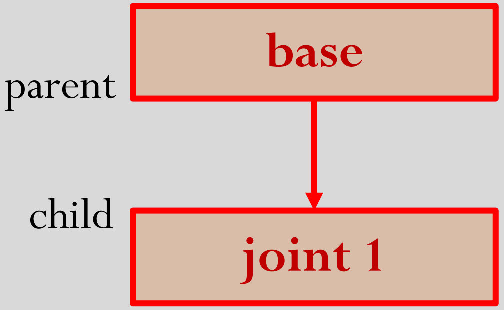


Hierarchical Transformations

how we construct a hierarchy?

- *we relate parent-child frames* via a transformation (matrix).
  - this relation is described by a *tree*:
  - where each node has its own *local co-ordinate system*.

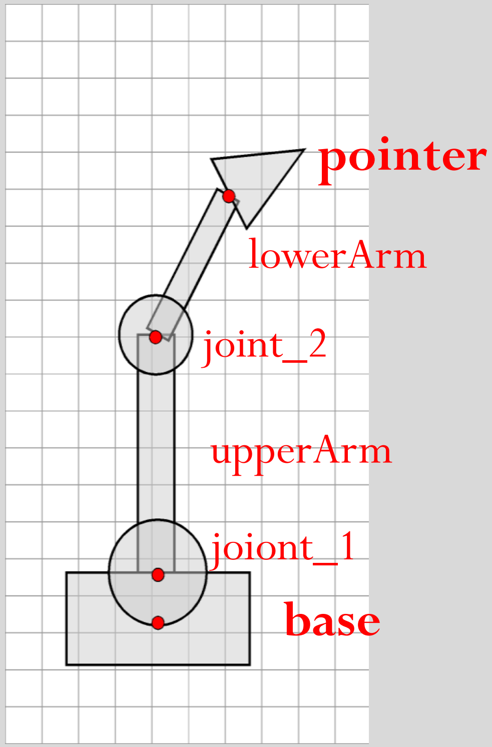


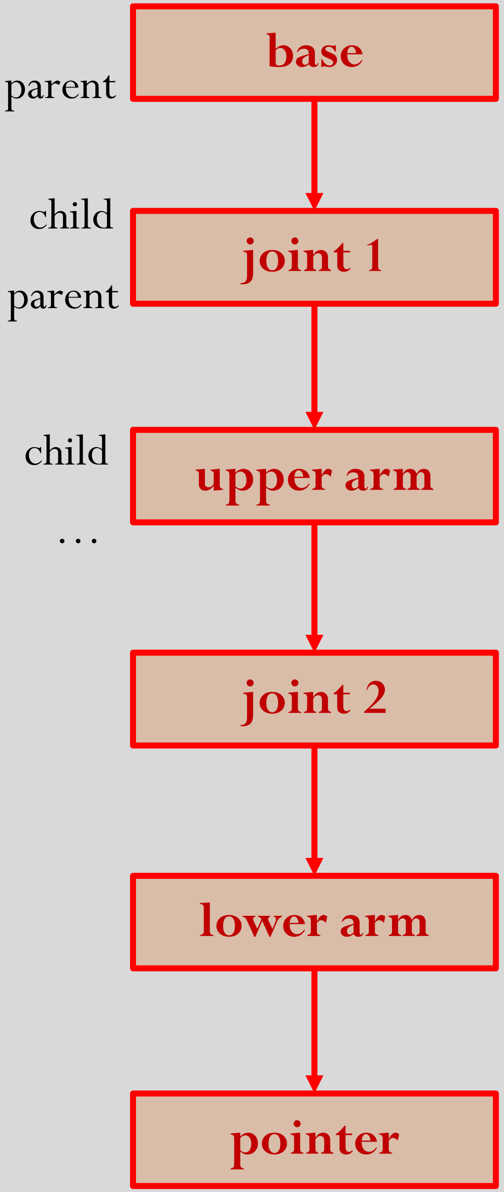


Hierarchical Transformations

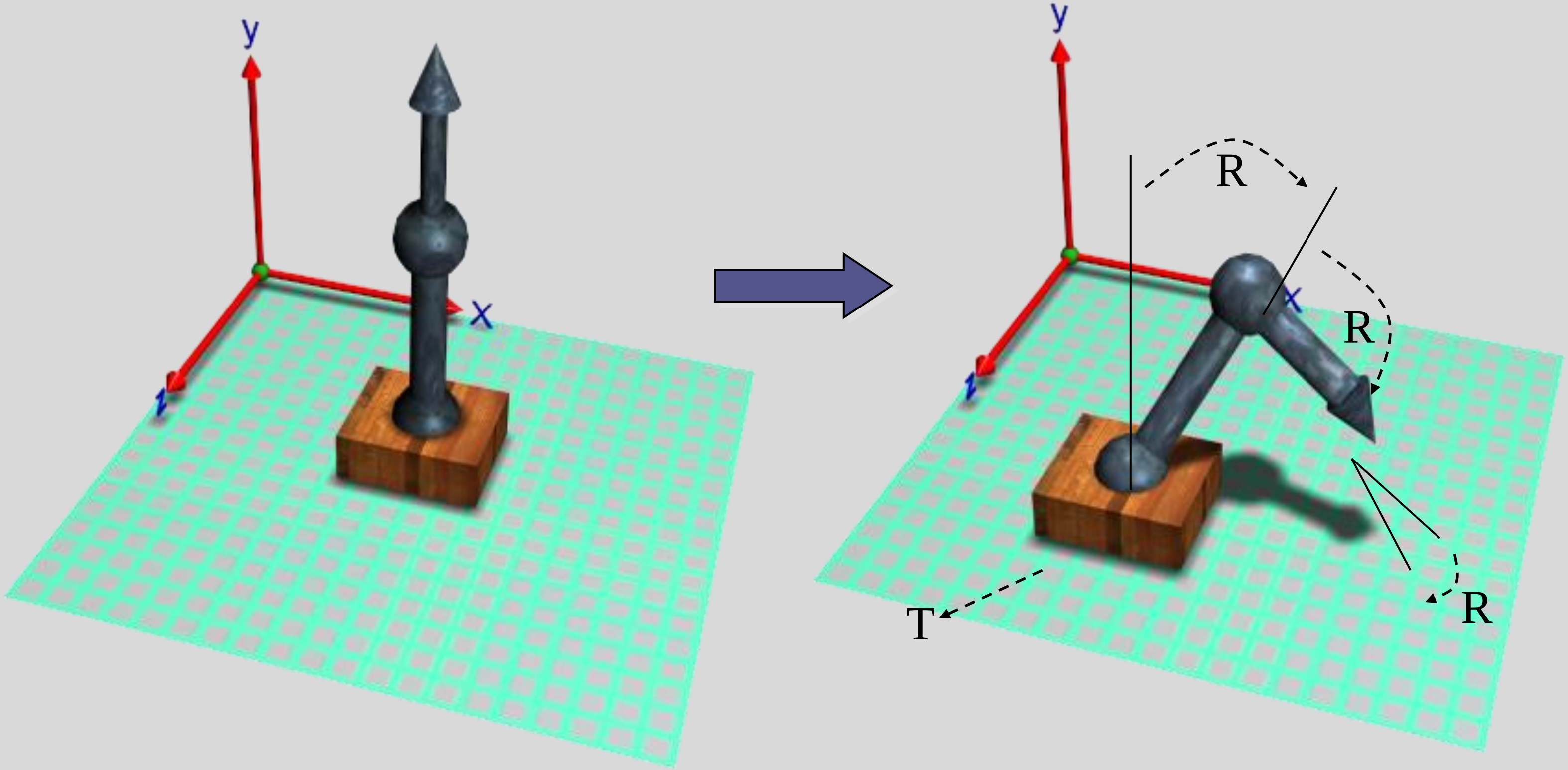
how we construct a hierarchy?

- *we relate parent-child frames* via a transformation (matrix).
  - this relation is described by a *tree*:
  - where each node has its own *local co-ordinate system*.

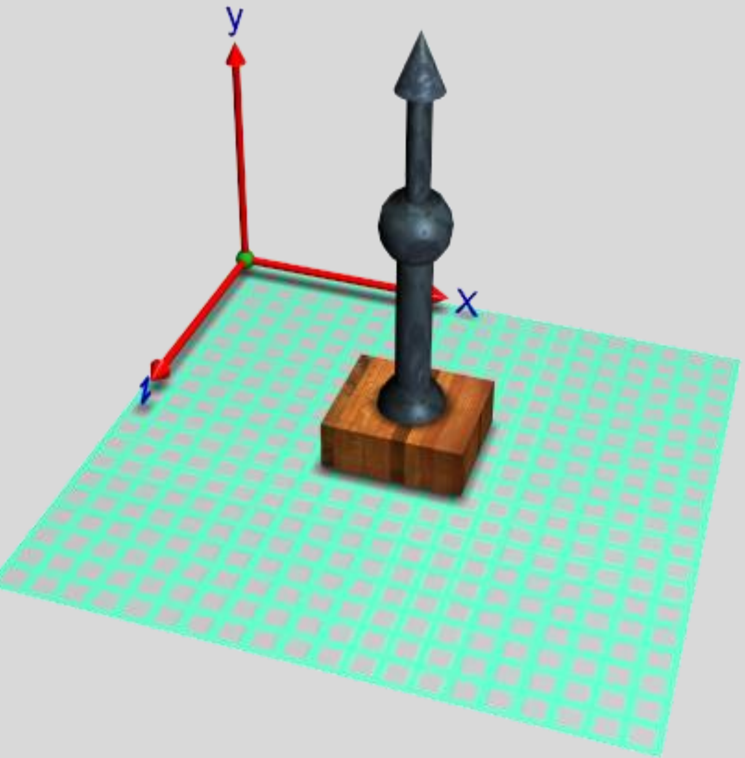


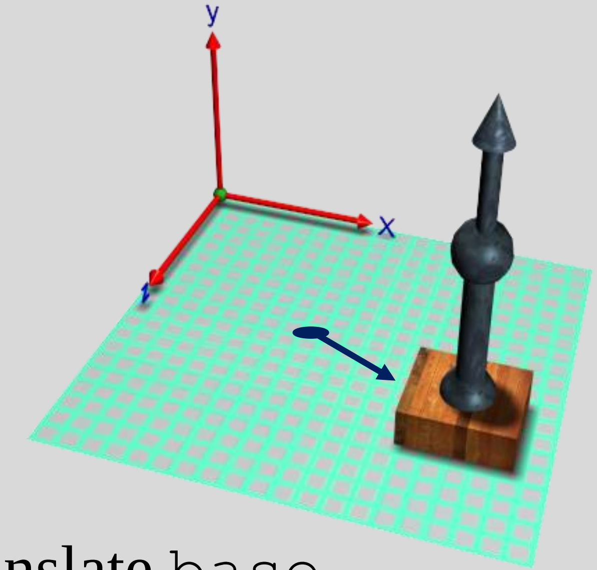


Hierarchical Transformations

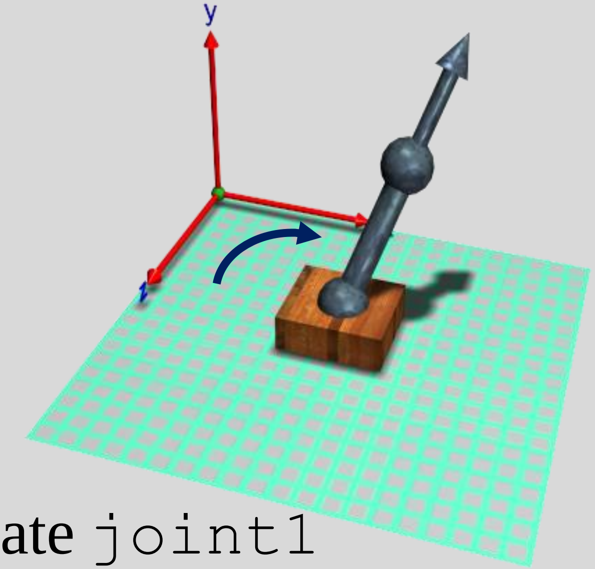


Hierarchical transformation allow independent control over sub-parts of an assembly

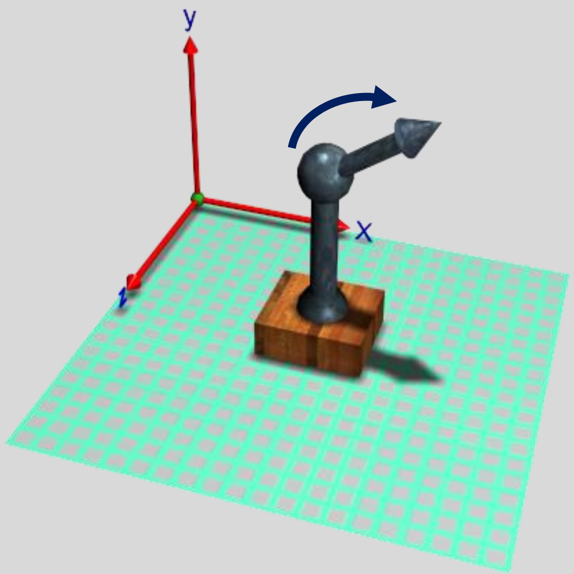




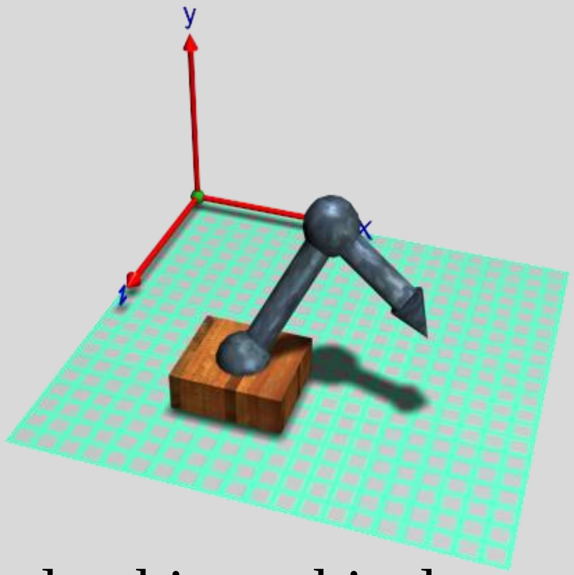
translate `base`



rotate `joint1`



rotate `joint2`

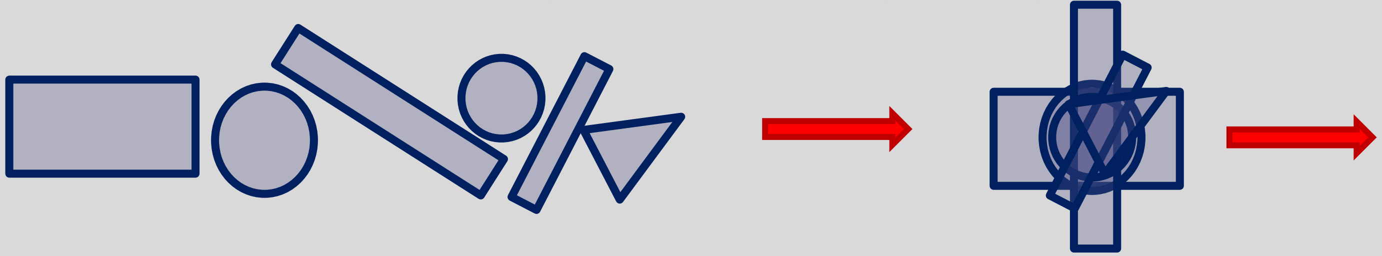


complex hierarchical transformation

OpenGL® Implementation

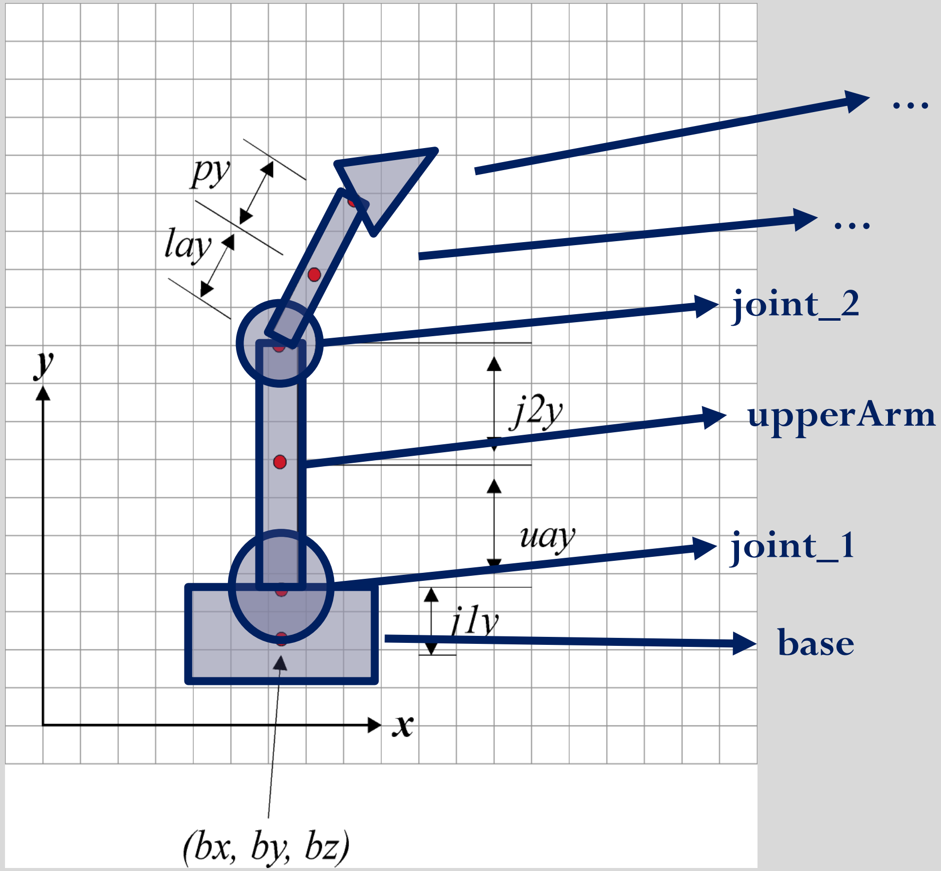
(**input**) what we define:

- a hierarchy(composed of asset of elements *base, joint_1, upperArm, joint_2, …*).
- location and orientation of each element of the hierarchy in their **local** frame (reference system).



(**output**) what we obtain:

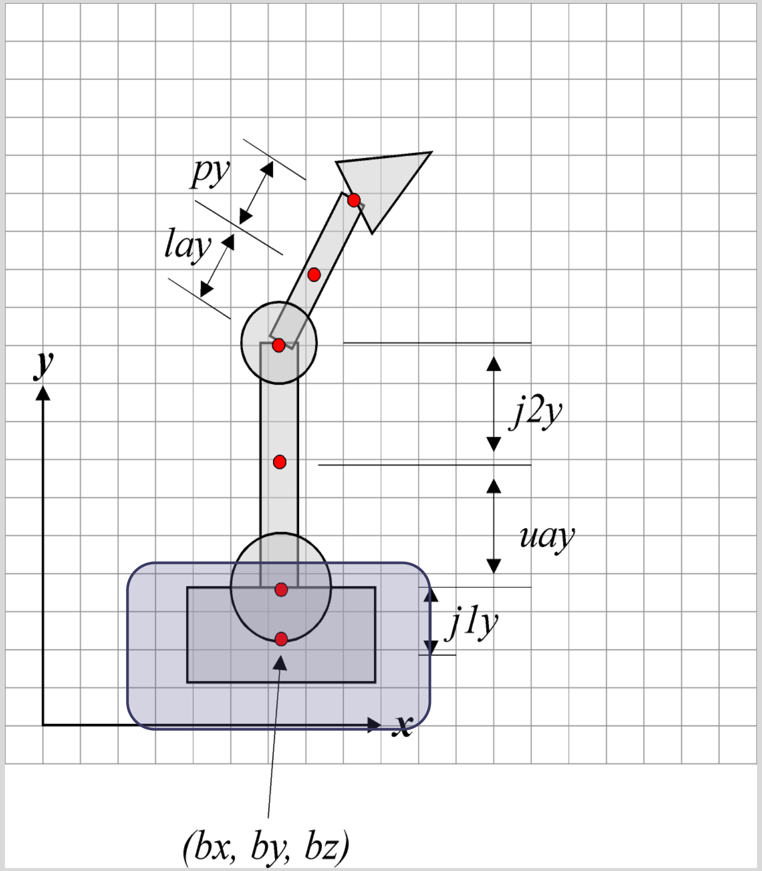
- render all the element of the final structure.
  → this means extract all the **global** coordinates (to send to the shaders).



OpenGL® Implementation

base (parent)

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```



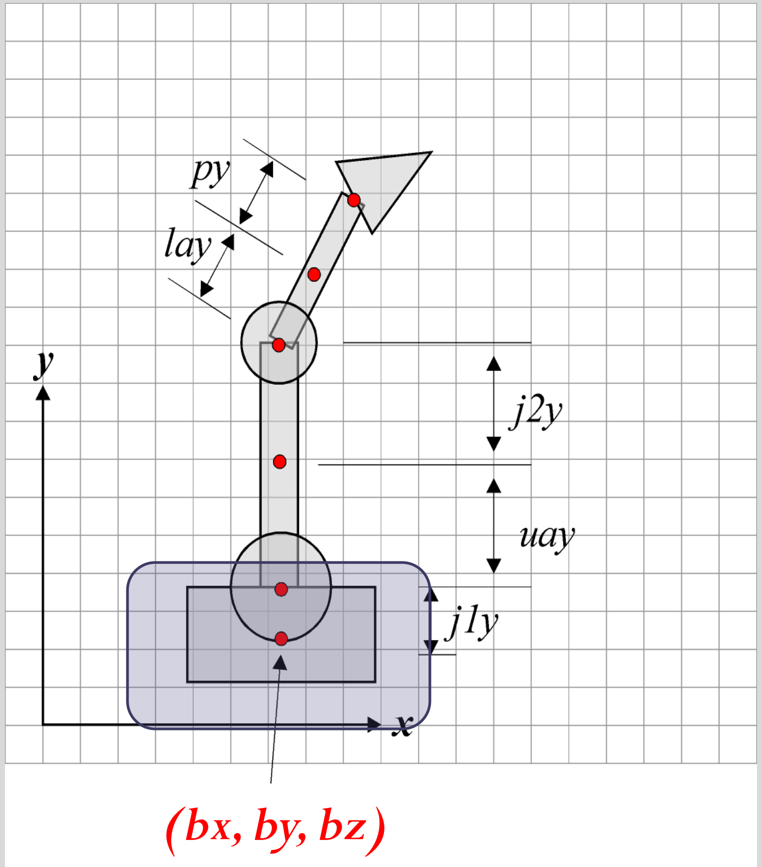
OpenGL® Implementation

base (parent)

define position and orientation in the world as transformation matrix

$\begin{bmatrix} \text{local1} \end{bmatrix}$

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```



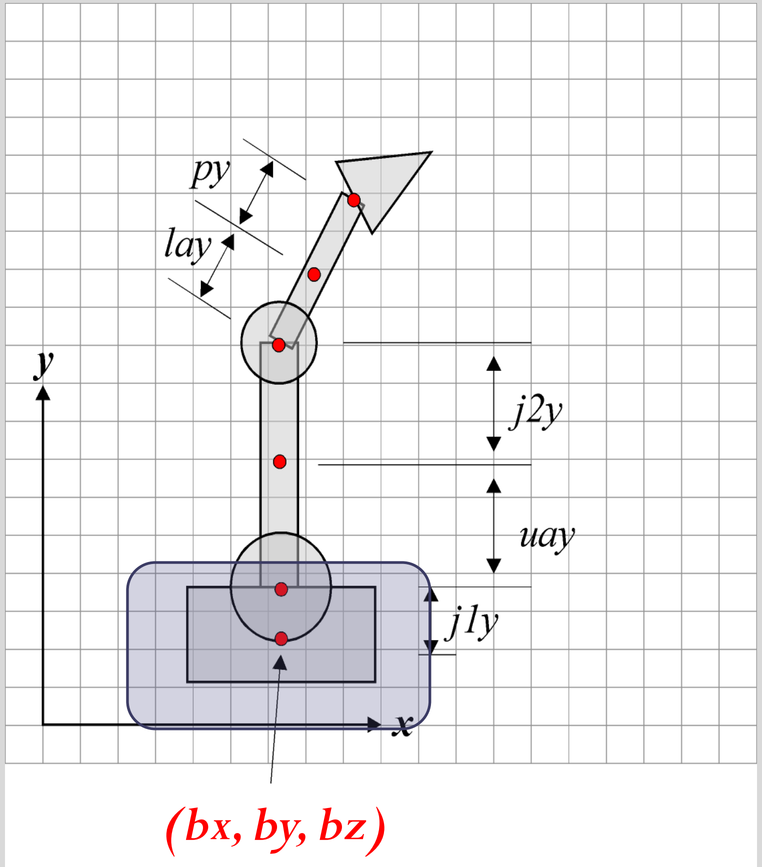
OpenGL® Implementation

base (parent)

in this case
global = local
(no parent)
→ so, we can
draw directly

$\left[\ \text{local1}\ \right]$

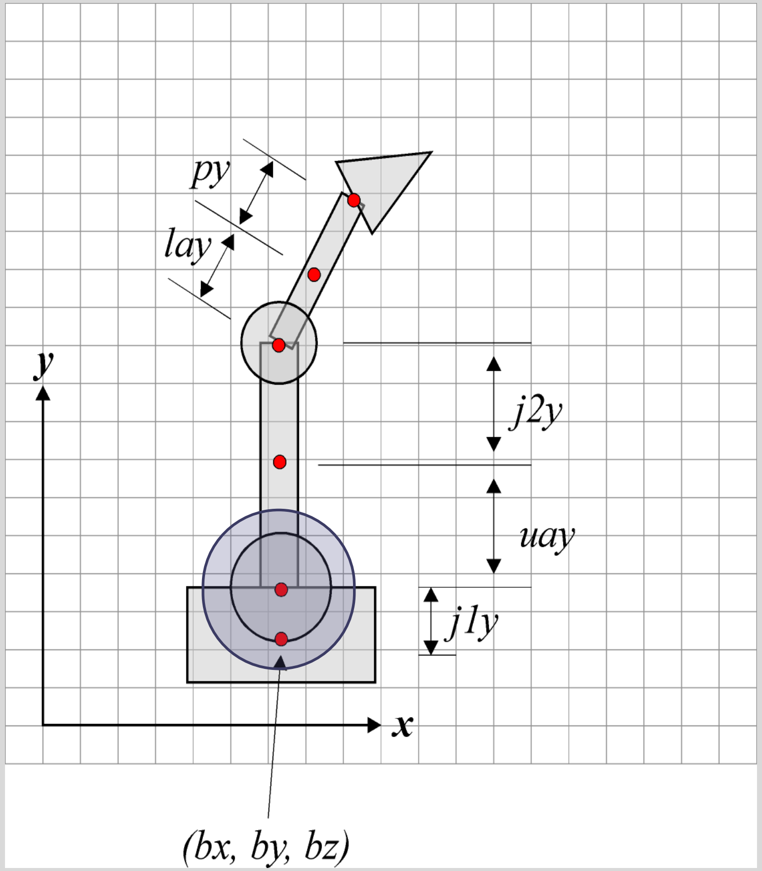
```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```



OpenGL® Implementation

joint 1
(child of base)

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```



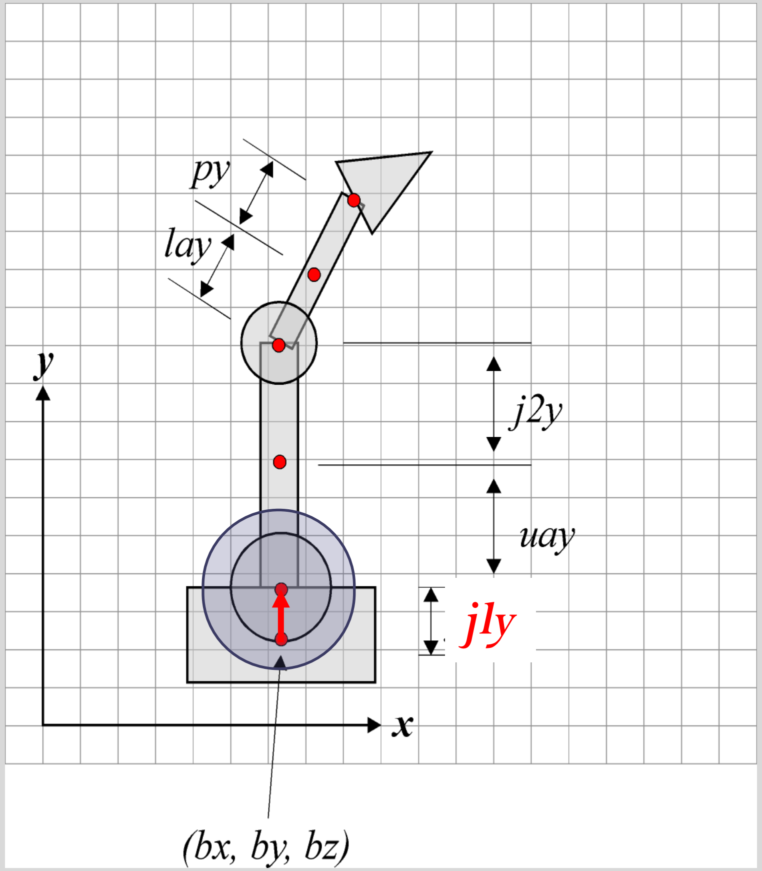
OpenGL® Implementation

joint 1
(child of base)

define position and orientation in the local frame.

[local2]

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```



OpenGL® Implementation

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();
```

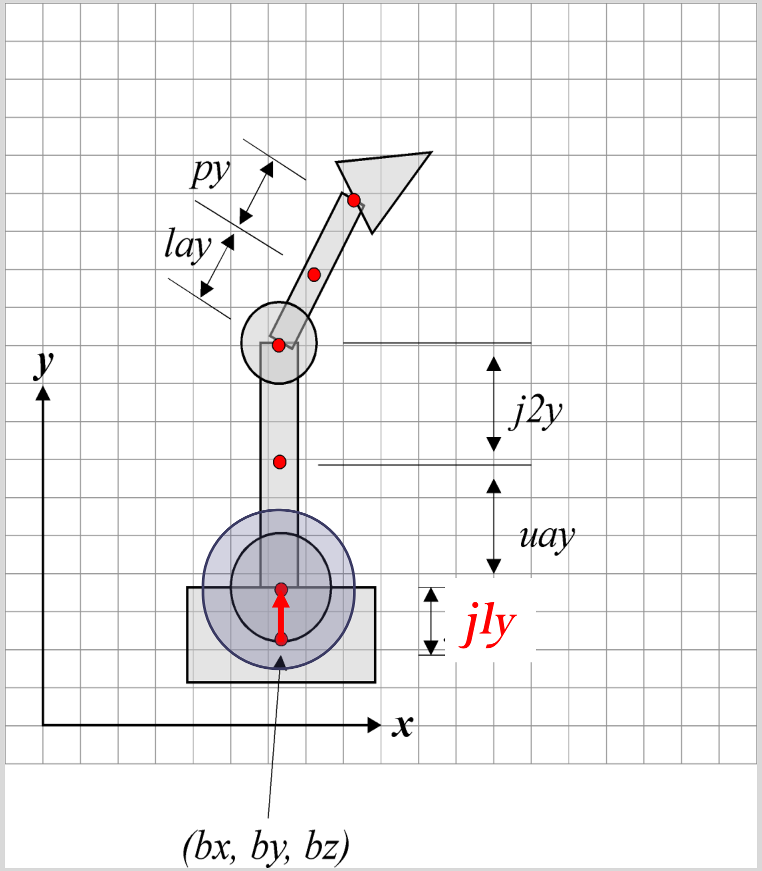
joint 1
(child of base)

```
local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();
```

we use parent
to transform
from local
→ to global

[local1] × [local2]

```
local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```

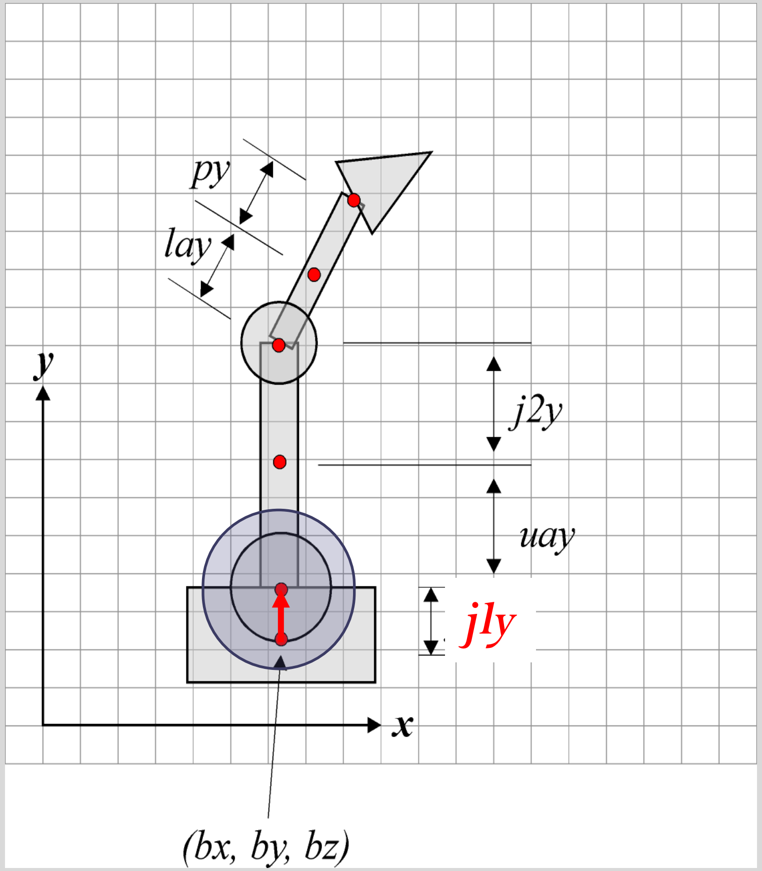


OpenGL® Implementation

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```

joint 1
(child of base)

we draw the
global
transform



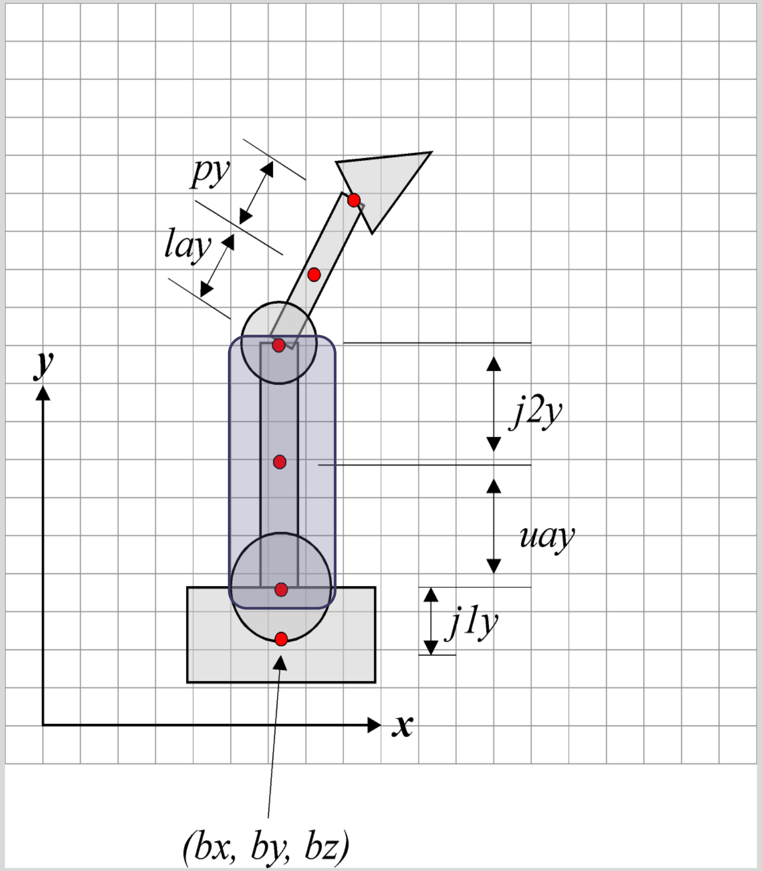
OpenGL® Implementation

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();
```

upperArm
(child of joint1)

```
local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();
```

etc.



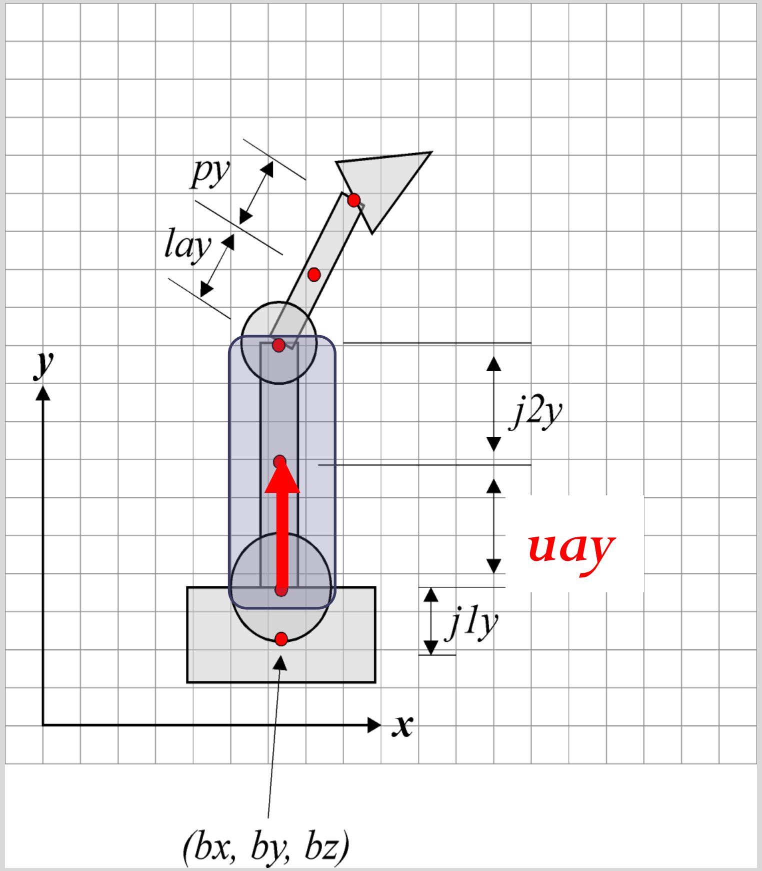
OpenGL® Implementation

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```

upperArm (child of joint1)

define position and orientation in the local frame.

$\left[\text{local3} \right]$

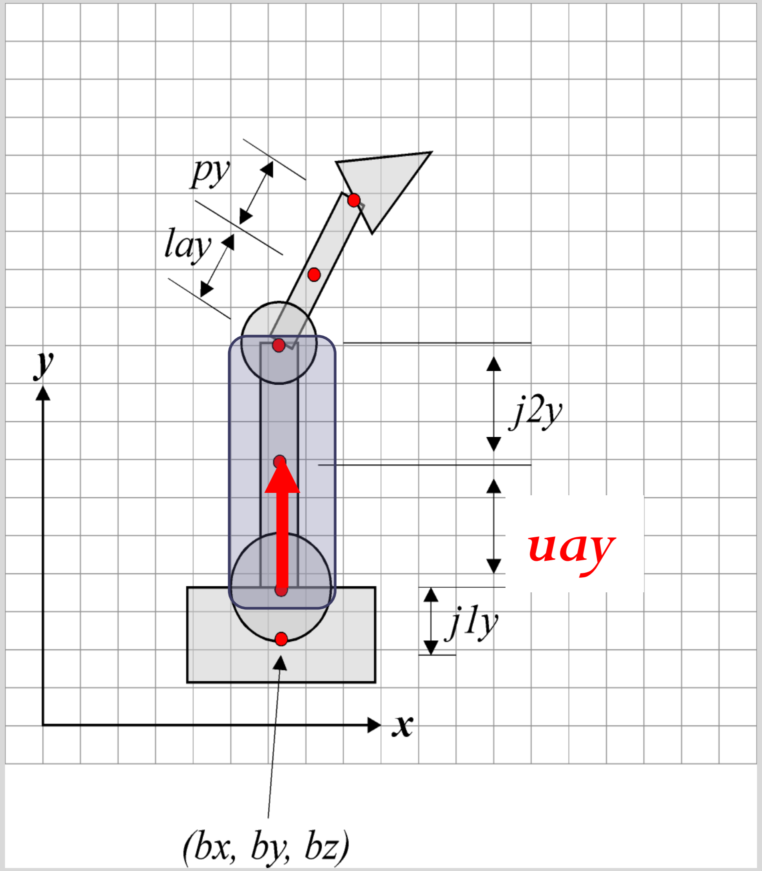


OpenGL® Implementation

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```

upperArm (child of joint1)

we use parents to transform from local → to global



NOTE: this is equivalent to `global3 = global2*local3;`

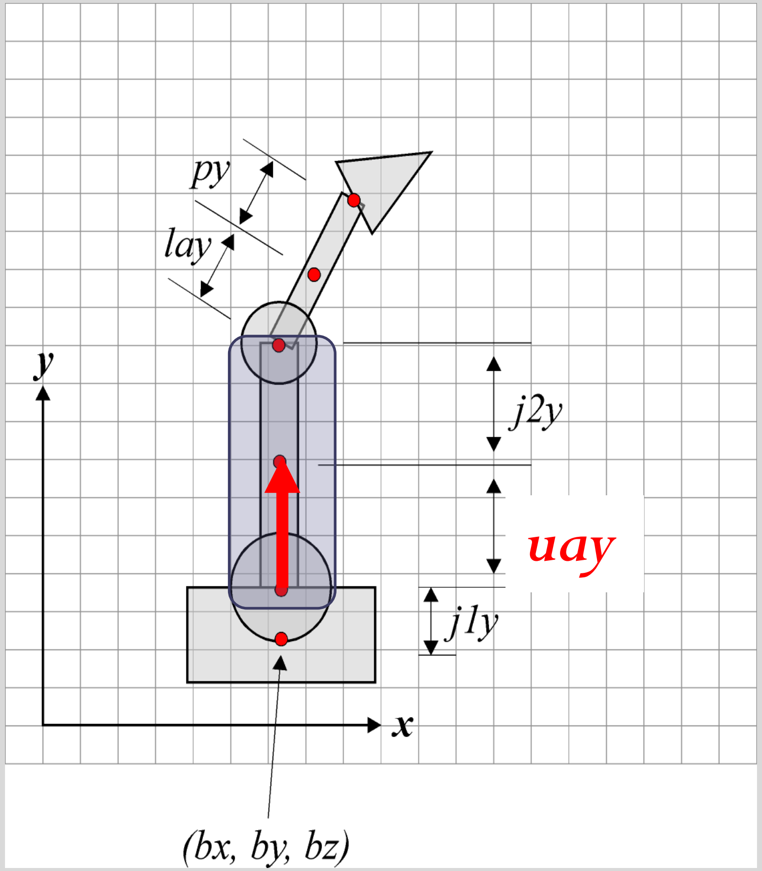
[local1] × [local2] × [local3]

OpenGL® Implementation

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```

upperArm (child of joint1)

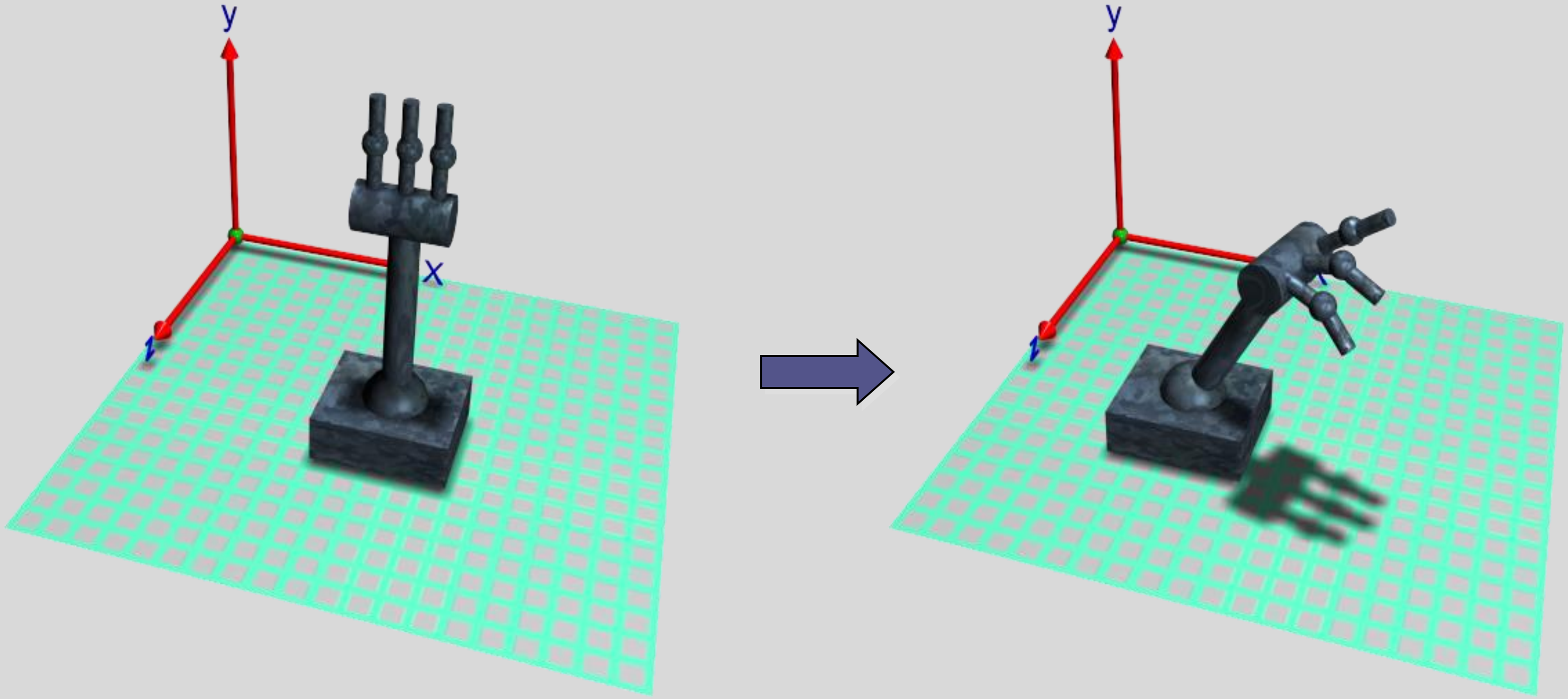
we draw the global transform



Hierarchical Transformations

- previous example had simple *one-to-one* parent-child linkages.
- in general there may be many *child frames* derived from a single parent frame.
- we need to remember the parent frame and return to it when creating new children.
  - →solution: to keep track of global transformation as we go.

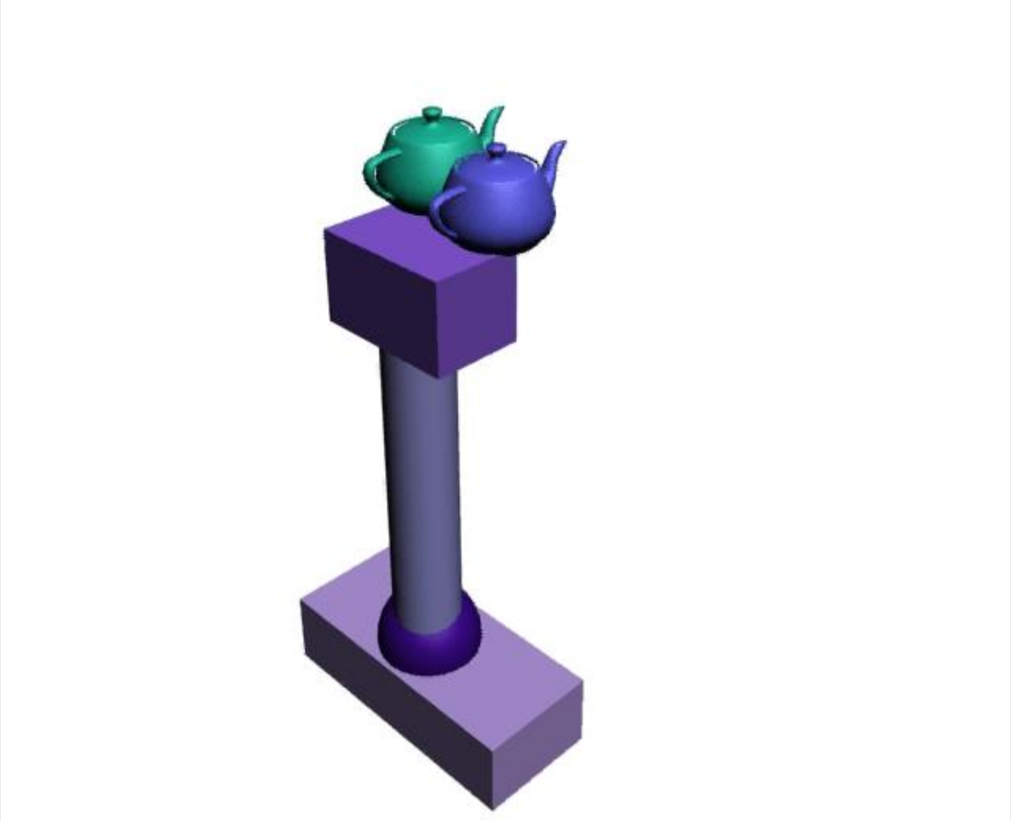
Hierarchical Transformations



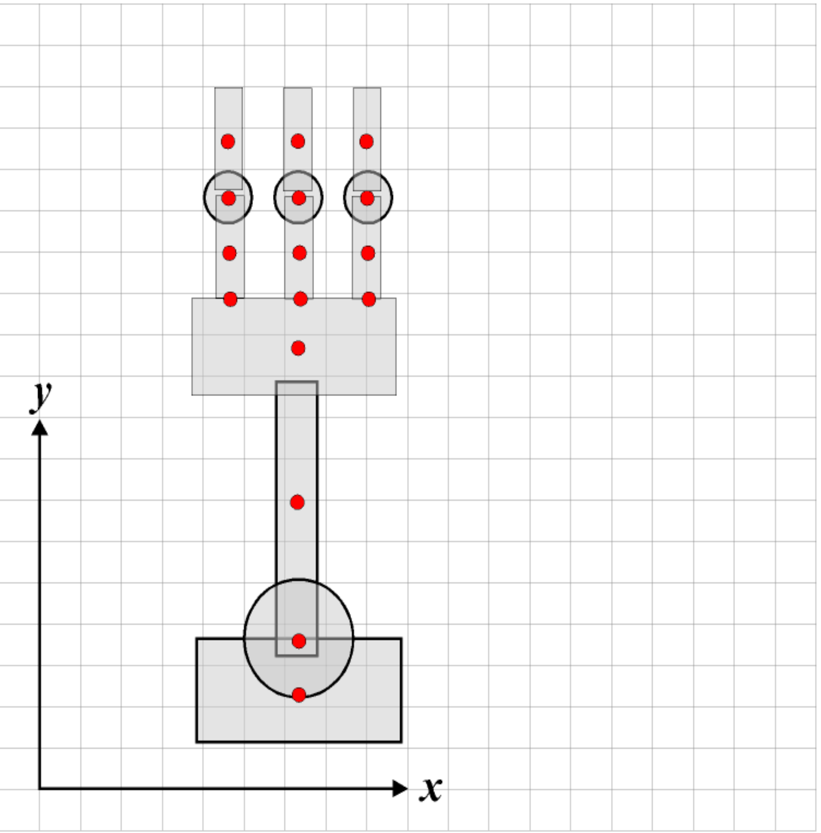
each finger is a child of the wrist (parent)

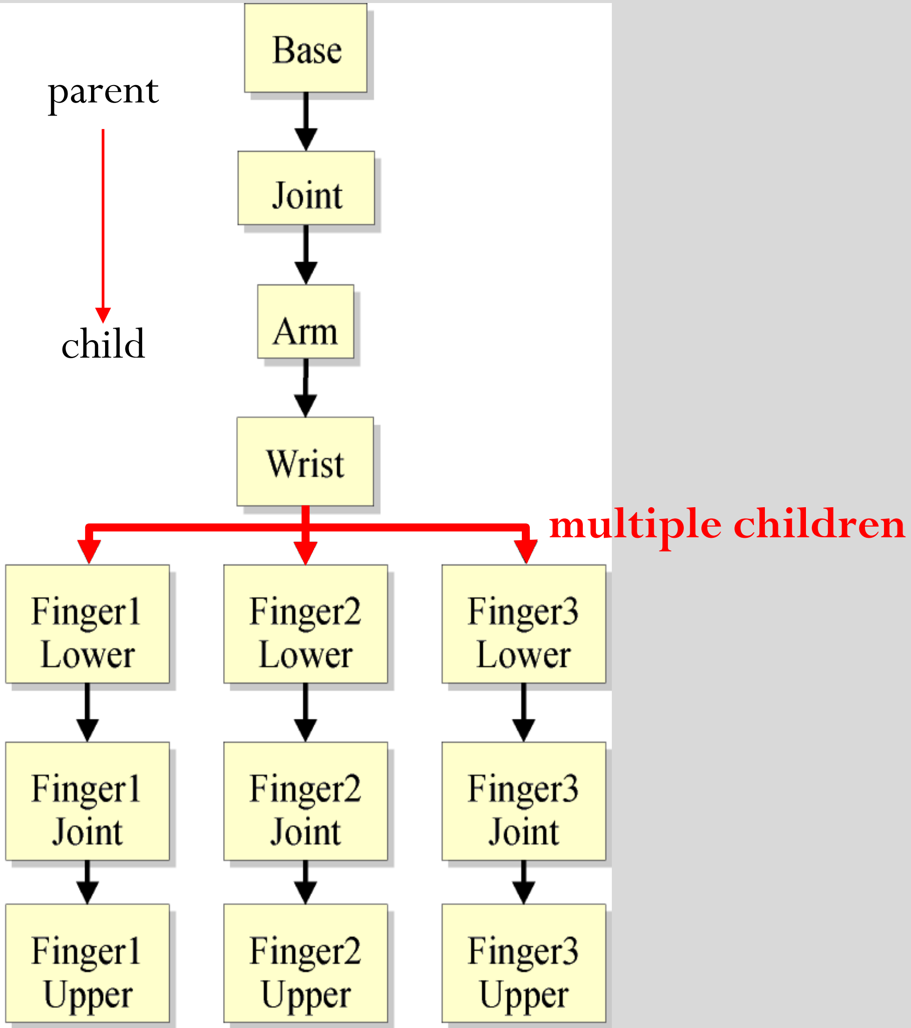
⇒ independent control over the orientation
of each fingers (the movement of the index should not affect the thumb)

Hierarchical Transformations



Hierarchical Transformations





parent

child

Base

Joint

Arm

Wrist

multiple children

Finger1 Lower

Finger2 Lower

Finger3 Lower

Finger1 Joint

Finger2 Joint

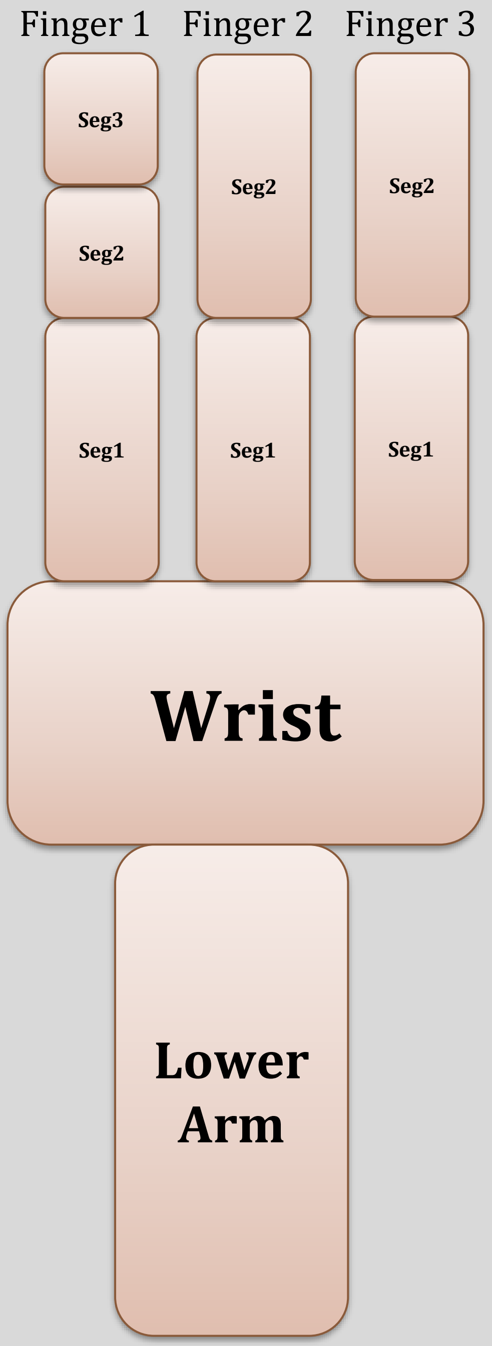
Finger3 Joint

Finger1 Upper

Finger2 Upper

Finger3 Upper

OpenGL® Implementation



(**input**) what we define:

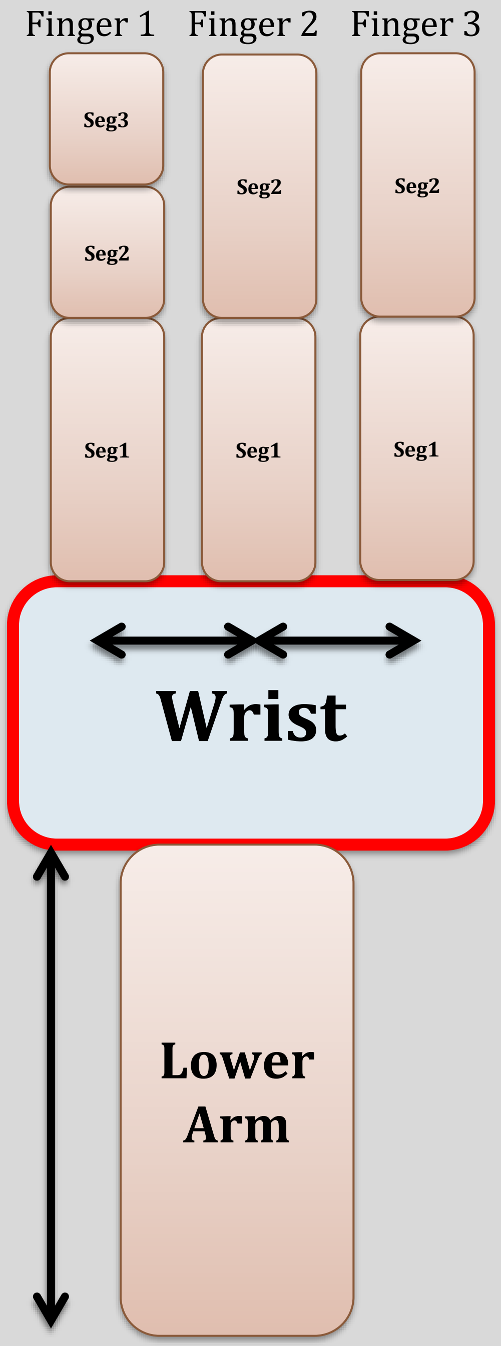
- a hierarchy.
- location and orientation of each element of the hierarchy in their **local** frame (reference system).

(**output**) what we obtain:

- render all the element of the final structure.
  → extract all the **global** coordinates.

like the previous example

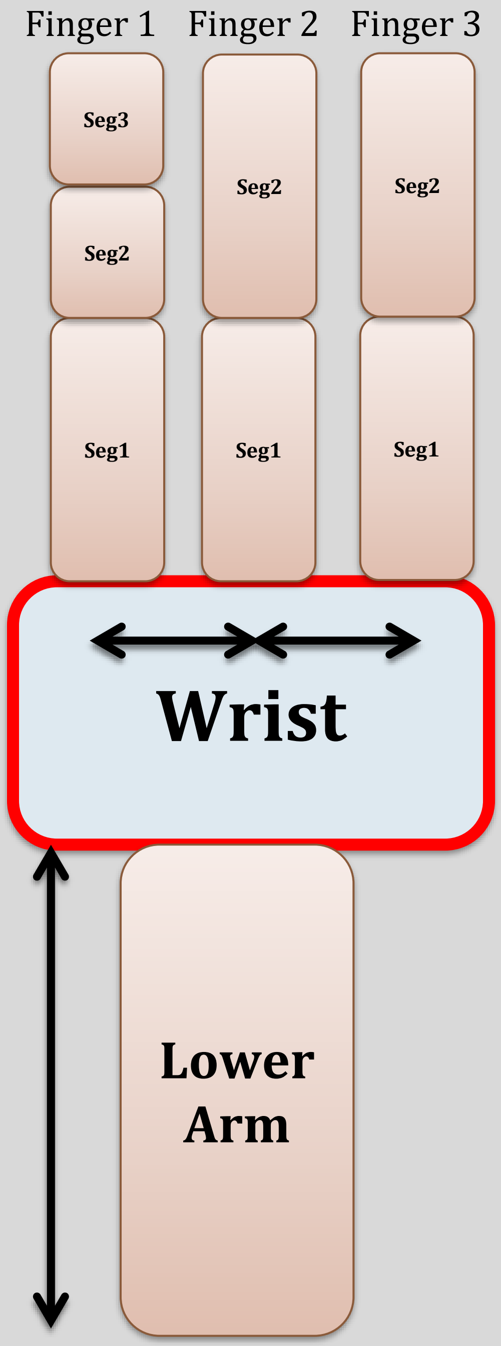
(new problem) how to manage multiple children?



```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();
```

$$\left[\text{global lowerarm}\right] \times \left[\text{local wrist}\right]$$

like the previous example

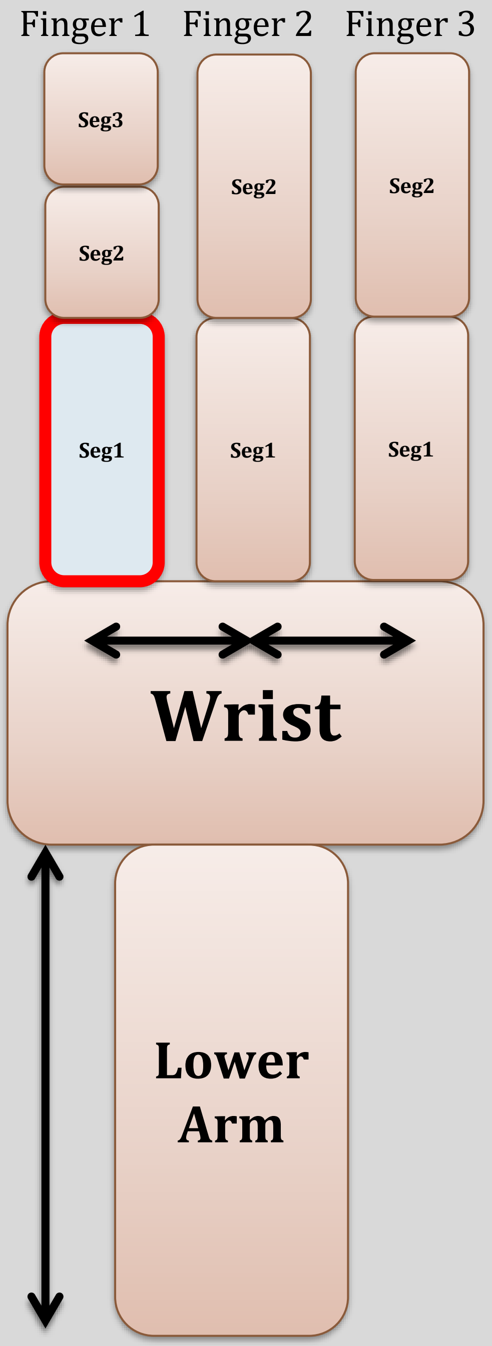


```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();
```

we draw the global transform

$$\begin{bmatrix}\text{global lowerarm}\end{bmatrix} \times \begin{bmatrix}\text{local wrist}\end{bmatrix}$$

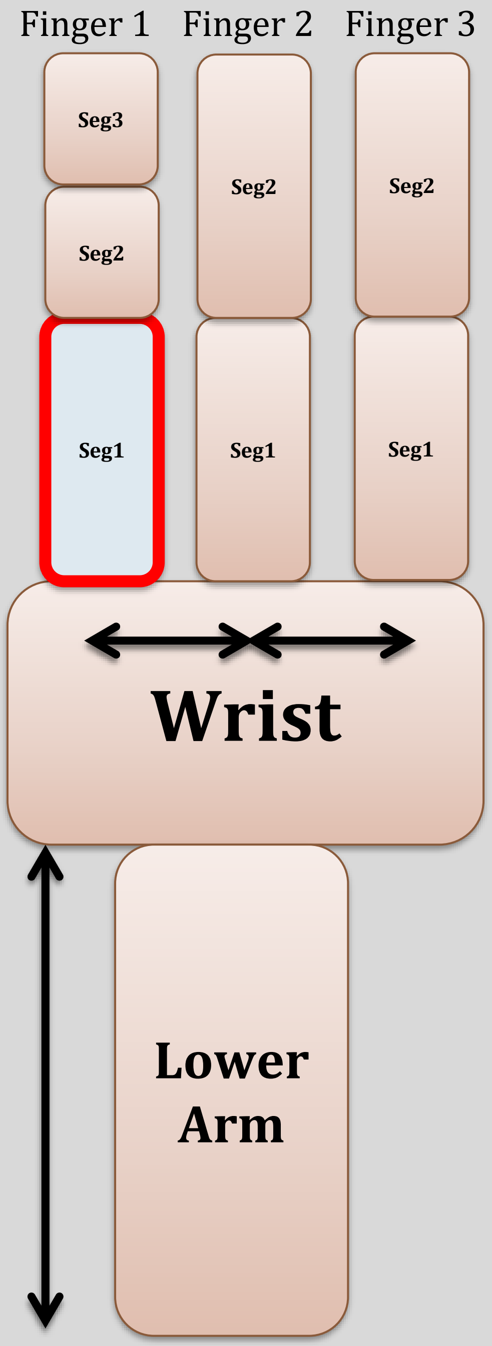
like the previous example



```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();

    finger1_segment1 – do local transformations
    global_finger1_1 = global_wrist*localfinger1_segment1;
    updateUniformVariables(model matrix = global_finger1_1);
    drawFinger1Segment1();
```

like the previous example

$$\begin{bmatrix}\text{global} \\ \text{wrist}\end{bmatrix} \times \begin{bmatrix}\text{local} \\ \text{finger1\_1}\end{bmatrix}$$

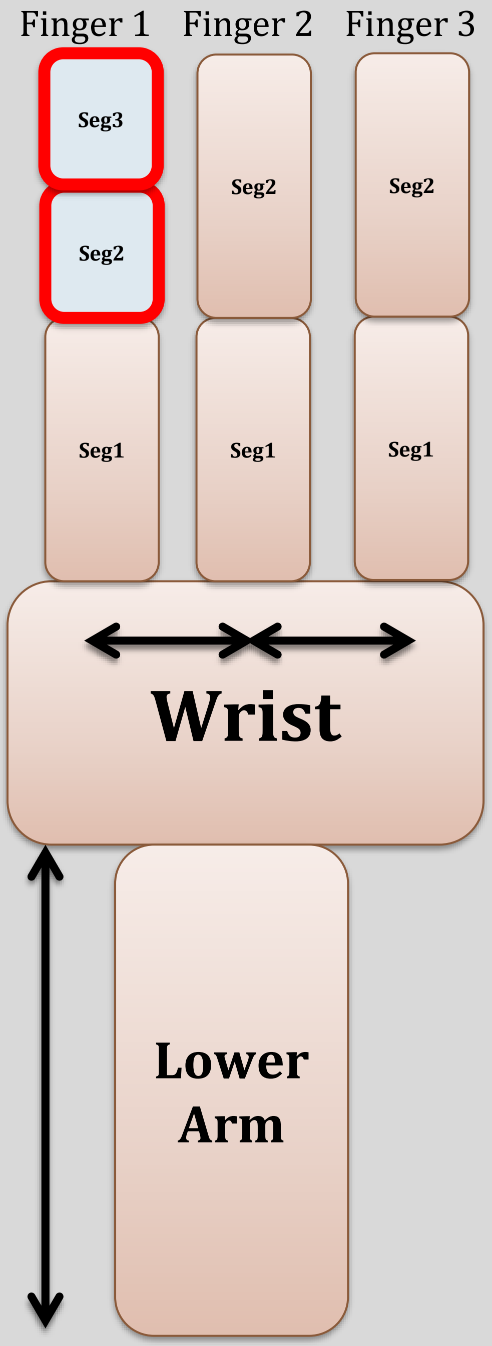


```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();

    finger1_segment1 – do local transformations
    global_finger1_1 = global_wrist*localfinger1_segment1;
    updateUniformVariables(model matrix = global_finger1_1);
    drawFinger1Segment1();
```

like the previous example

$\left[\text{global wrist}\right] \times \left[\text{local finger1\_1}\right]$

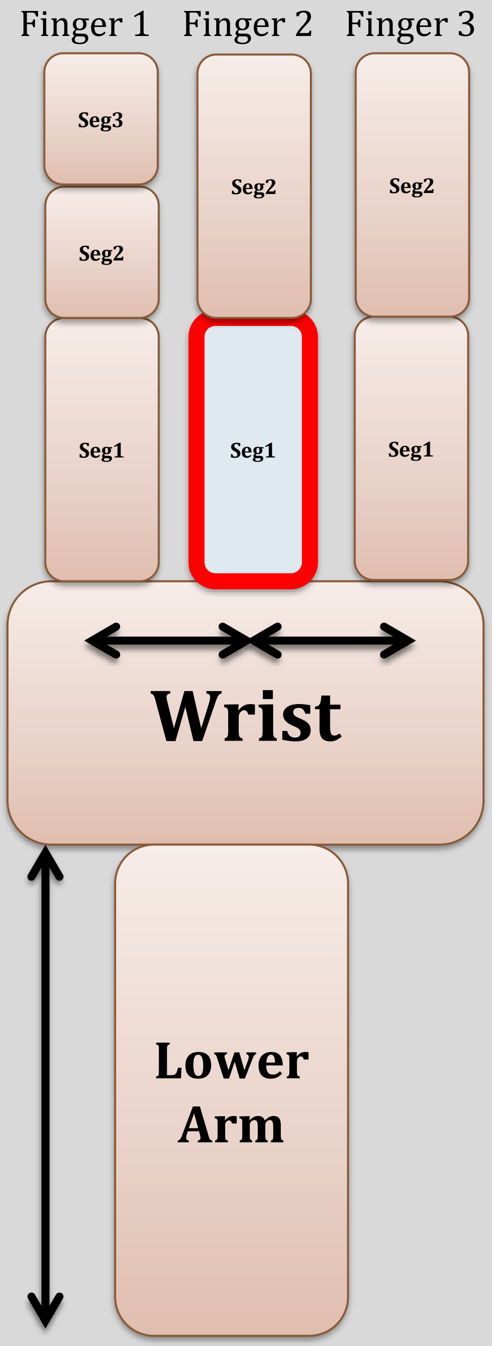
again we draw the global transform



```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();

    finger1_segment1 – do local transformations
    global_finger1_1 = global_wrist*localfinger1_segment1;
    updateUniformVariables(model matrix = global_finger1_1);
    drawFinger1Segment1();

        finger1_segment2 – do local transformations
        global_finger1_2 = global_finger1_1 *localfinger1_segment2;
        updateUniformVariables(model matrix = global_finger1_2);
        drawFinger1Segment2();

        finger1_segment3 – do local transformations
        global_finger1_3 = global_finger1_2 *localfinger1_segment3;
        updateUniformVariables(model matrix = global_finger1_3);
        drawFinger1Segment3();
        etc.
```

like the previous example

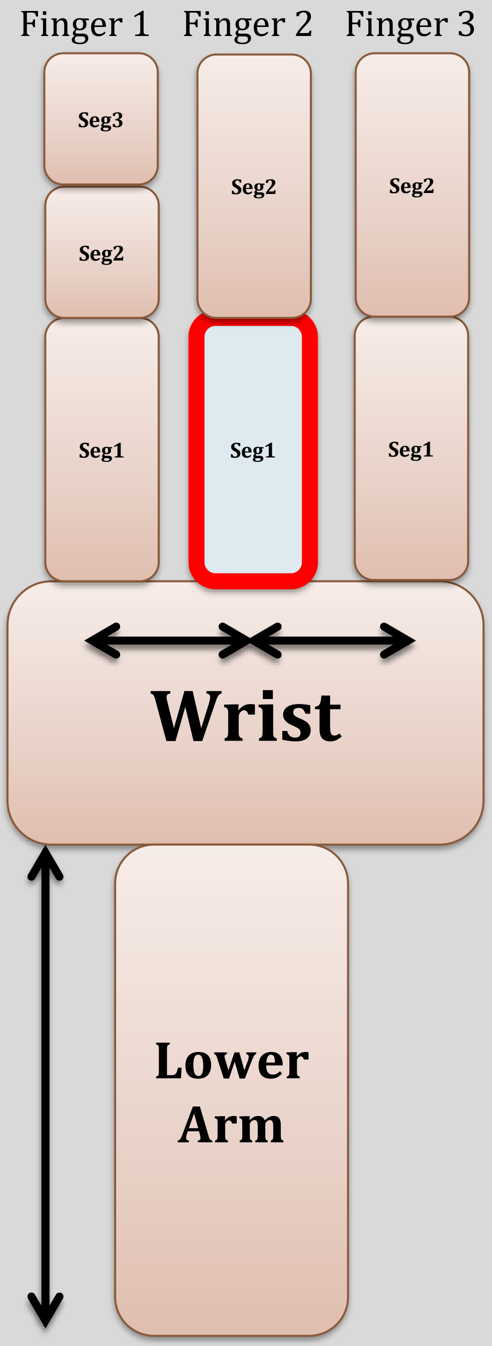
again we extract the global transform from the parent and we draw



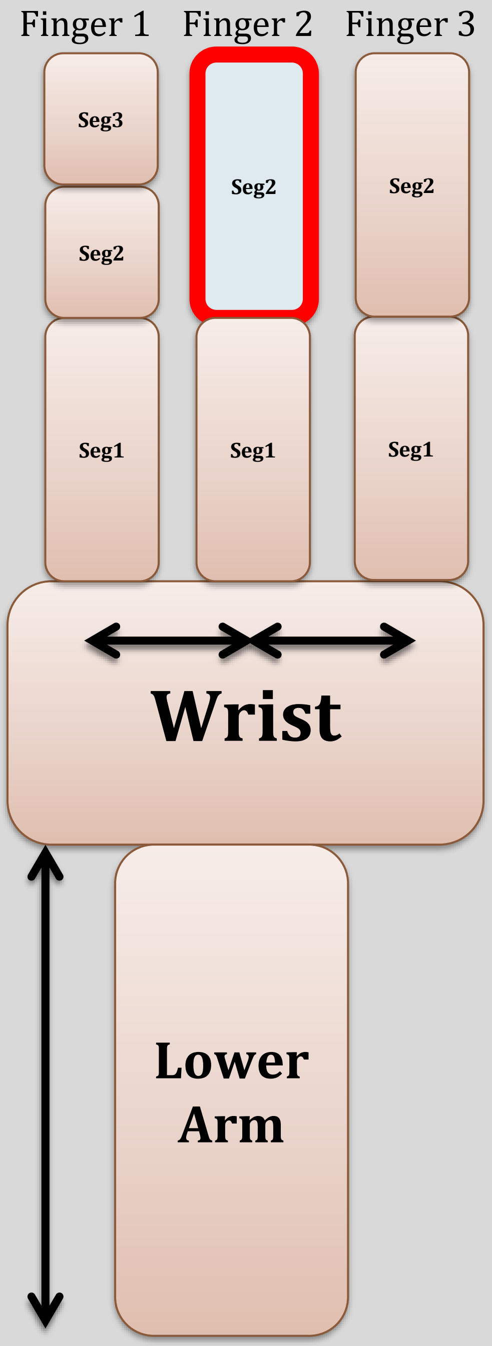
new part

```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();

    finger1_segment1 – do local transformations
    global_finger1_1 = global_wrist*localfinger1_segment1;
    updateUniformVariables(model matrix = global_finger1_1);
    drawFinger1Segment1();

        finger1_segment2 – do local transformations
        global_finger1_2 = global_finger1_1 *localfinger1_segment2;
        updateUniformVariables(model matrix = global_finger1_2);
        drawFinger1Segment2();

        finger1_segment3 – do local transformations
        global_finger1_3 = global_finger1_2 *localfinger1_segment3;
        updateUniformVariables(model matrix = global_finger1_3);
        drawFinger1Segment3();
        etc.

    finger2_segment1 – do local transformations
    global_finger2_1 = global_wrist*localfinger2_segment1;
    updateUniformVariables(model matrix = global_finger2_1);
    drawFinger2Segment1();

        finger2_segment2 – do local transformations
        global_finger2_2 = global_finger2_1 *localfinger2_segment2;
        updateUniformVariables(model matrix = global_finger2_2);
        drawFinger2Segment2();
```

new part

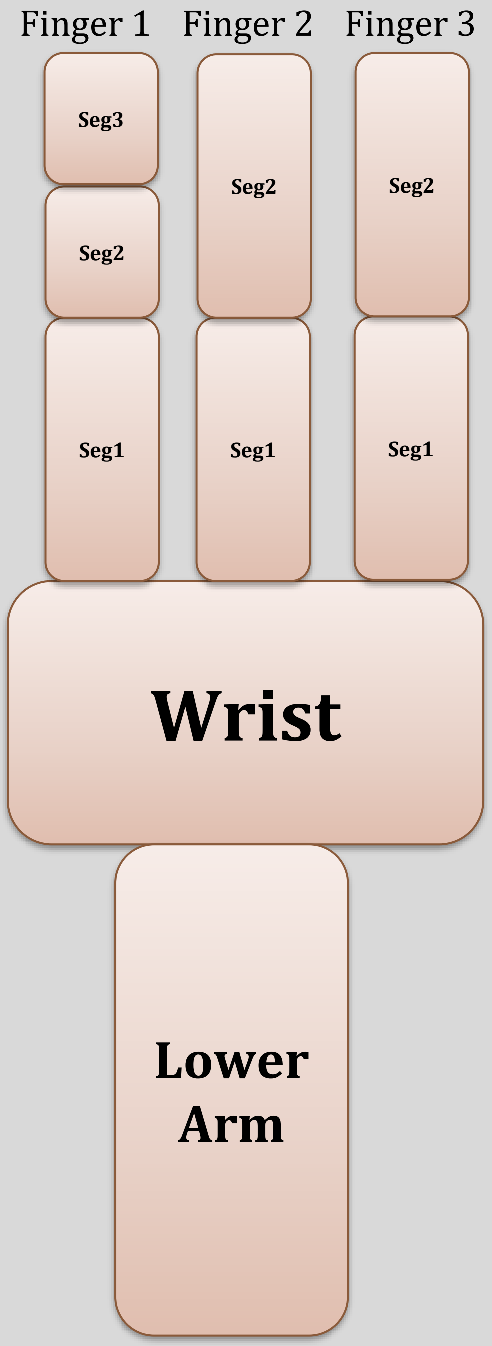


```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();

    finger1_segment1 – do local transformations
    global_finger1_1 = global_wrist*localfinger1_segment1;
    updateUniformVariables(model matrix = global_finger1_1);
    drawFinger1Segment1();

        finger1_segment2 – do local transformations
        global_finger1_2 = global_finger1_1 *localfinger1_segment2;
        updateUniformVariables(model matrix = global_finger1_2);
        drawFinger1Segment2();

        finger1_segment3 – do local transformations
        global_finger1_3 = global_finger1_2 *localfinger1_segment3;
        updateUniformVariables(model matrix = global_finger1_3);
        drawFinger1Segment3();
        etc.

    finger2_segment1 – do local transformations
    global_finger2_1 = global_wrist*localfinger2_segment1;
    updateUniformVariables(model matrix = global_finger2_1);
    drawFinger2Segment1();
```



```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();

    finger1_segment1 – do local transformations
    global_finger1_1 = global_wrist*localfinger1_segment1;
    updateUniformVariables(model matrix = global_finger1_1);
    drawFinger1Segment1();

        finger1_segment2 – do local transformations
        global_finger1_2 = global_finger1_1 *localfinger1_segment2;
        updateUniformVariables(model matrix = global_finger1_2);
        drawFinger1Segment2();

        finger1_segment3 – do local transformations
        global_finger1_3 = global_finger1_2 *localfinger1_segment3;
        updateUniformVariables(model matrix = global_finger1_3);
        drawFinger1Segment3();
        etc.

    finger2_segment1 – do local transformations
    global_finger2_1 = global_wrist*localfinger2_segment1;
    updateUniformVariables(model matrix = global_finger2_1);
    drawFinger2Segment1();

        finger2_segment2 – do local transformations
        global_finger2_2 = global_finger2_1 *localfinger2_segment2;
        updateUniformVariables(model matrix = global_finger2_2);
        drawFinger2Segment2();
```

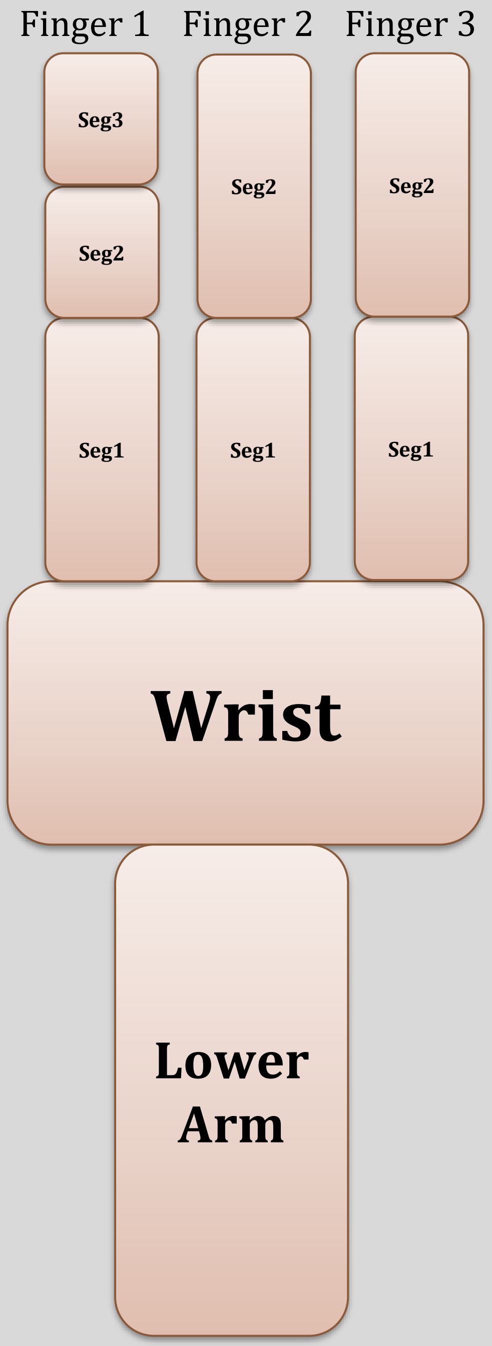
like the previous example



Variation example:

- instead of keep track of each global matrix in the hierarchy → use a memory stack.

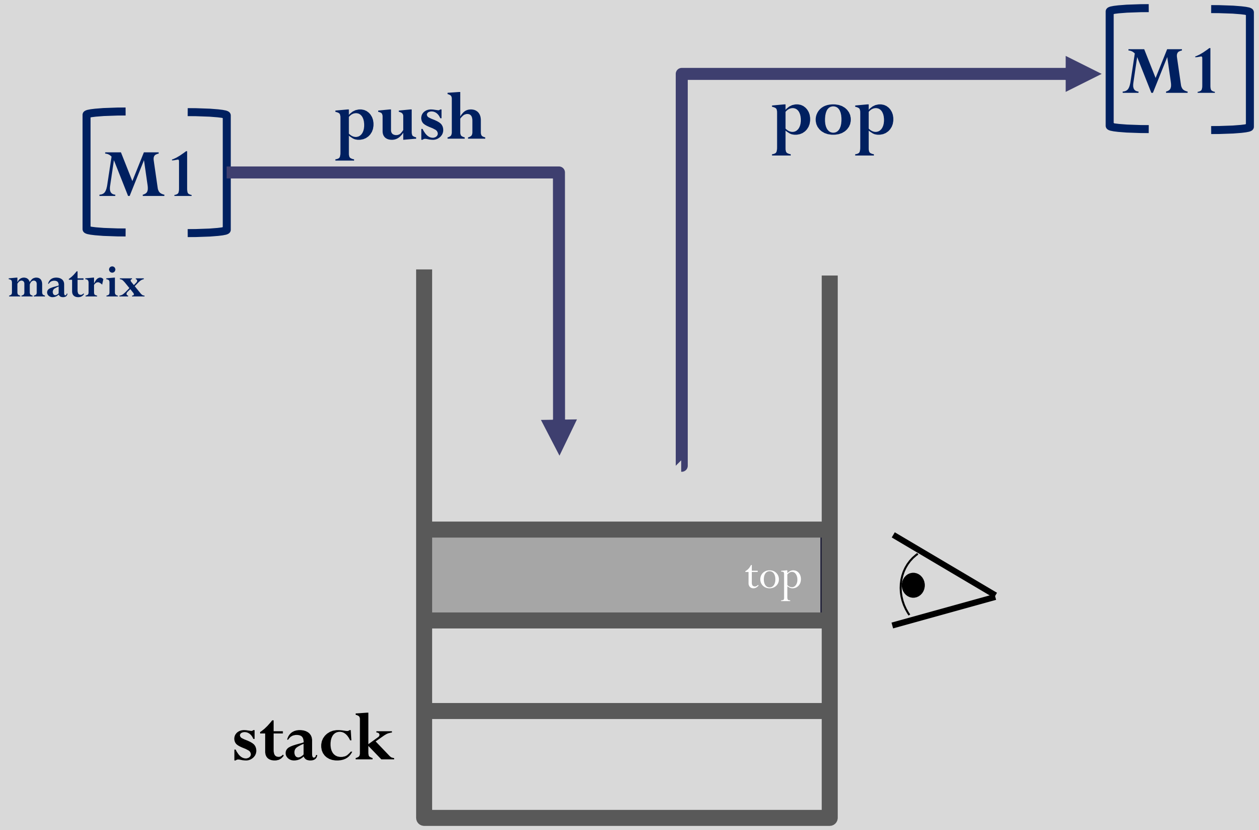
- why using a stack?
  - memory efficient.
  - avoid bugs in complex system.

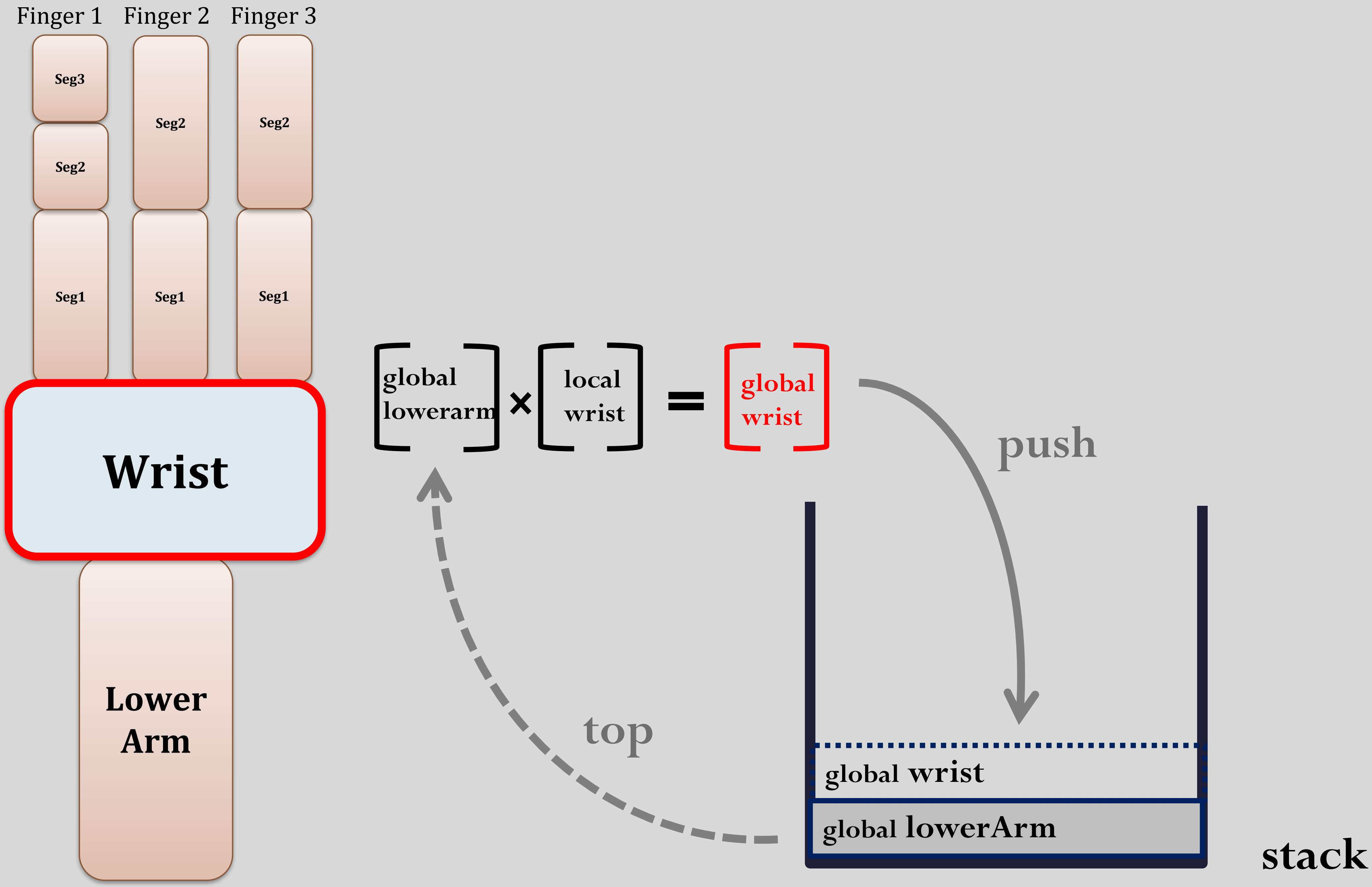


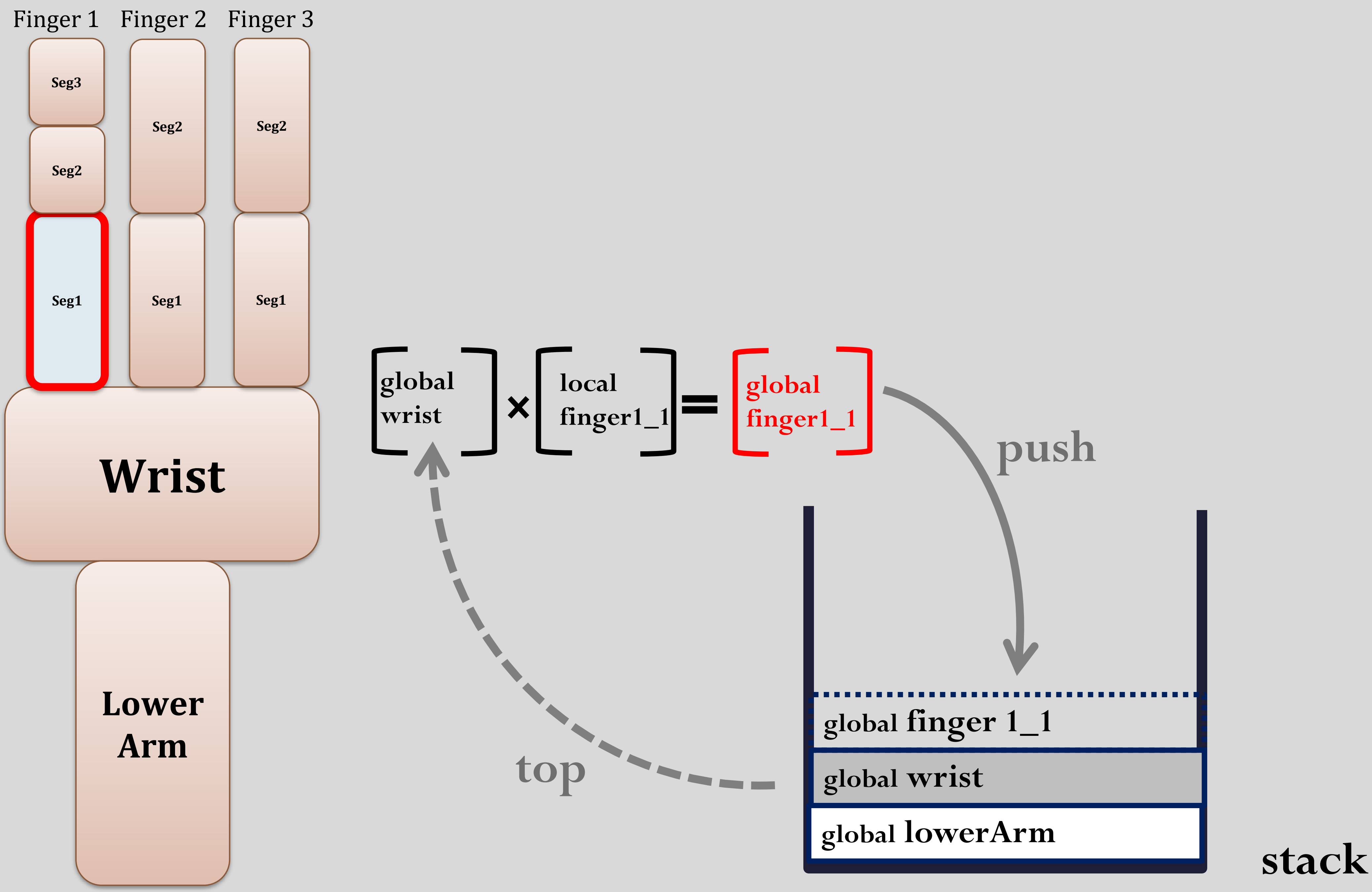
Variation example:

- instead of keep track of each global matrix in the hierarchy → use a memory stack.

- why using a stack?
  - memory efficient.
  - avoid bugs in complex system.




Finger 1
Finger 2
Finger 3
Seg3
Seg2
Seg2
Seg2
Seg1
Seg1
Seg1
Wrist
Lower Arm
global lowerarm
×
local wrist
=
global wrist
push
top
global wrist
global lowerArm
stack

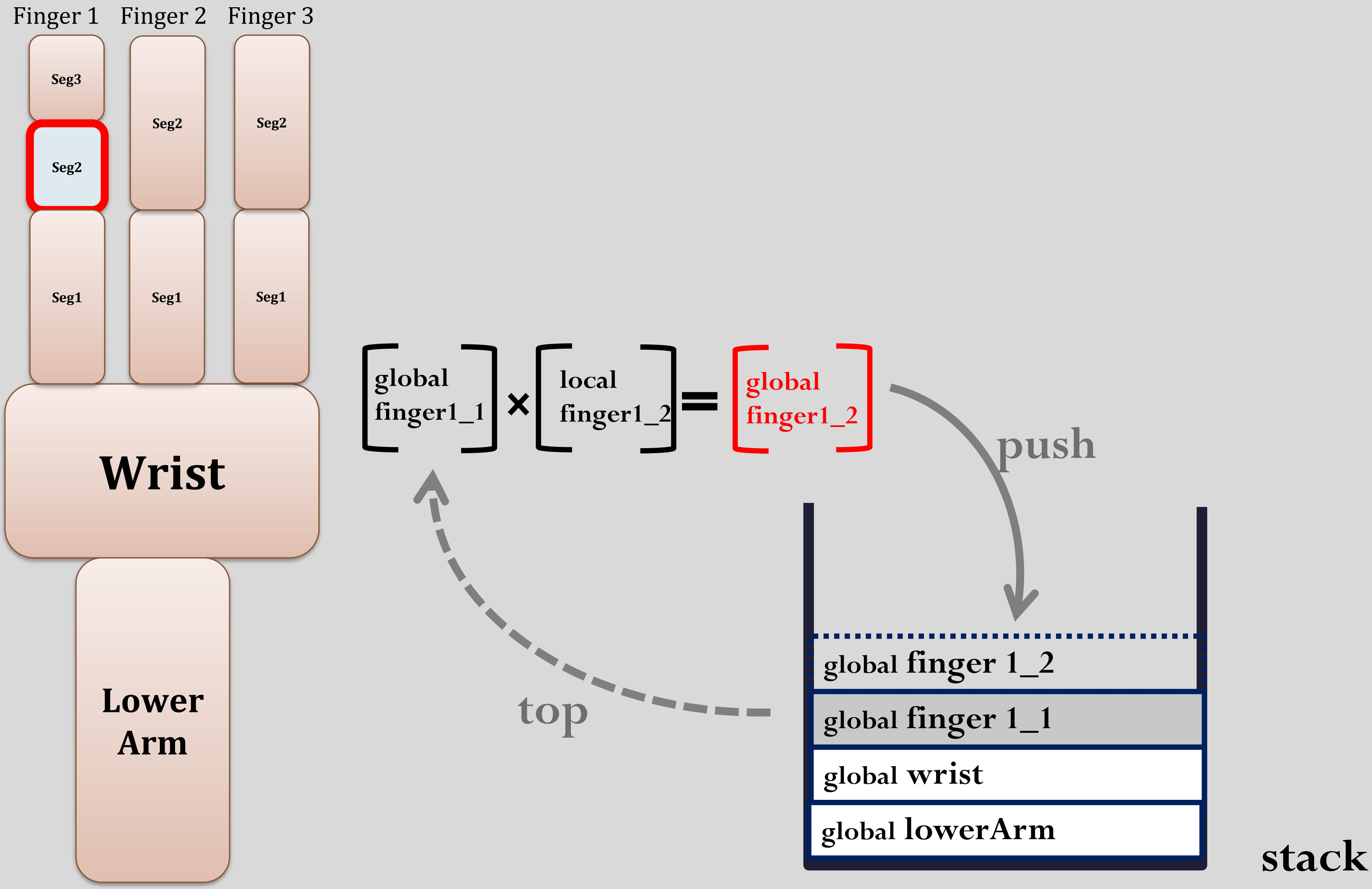


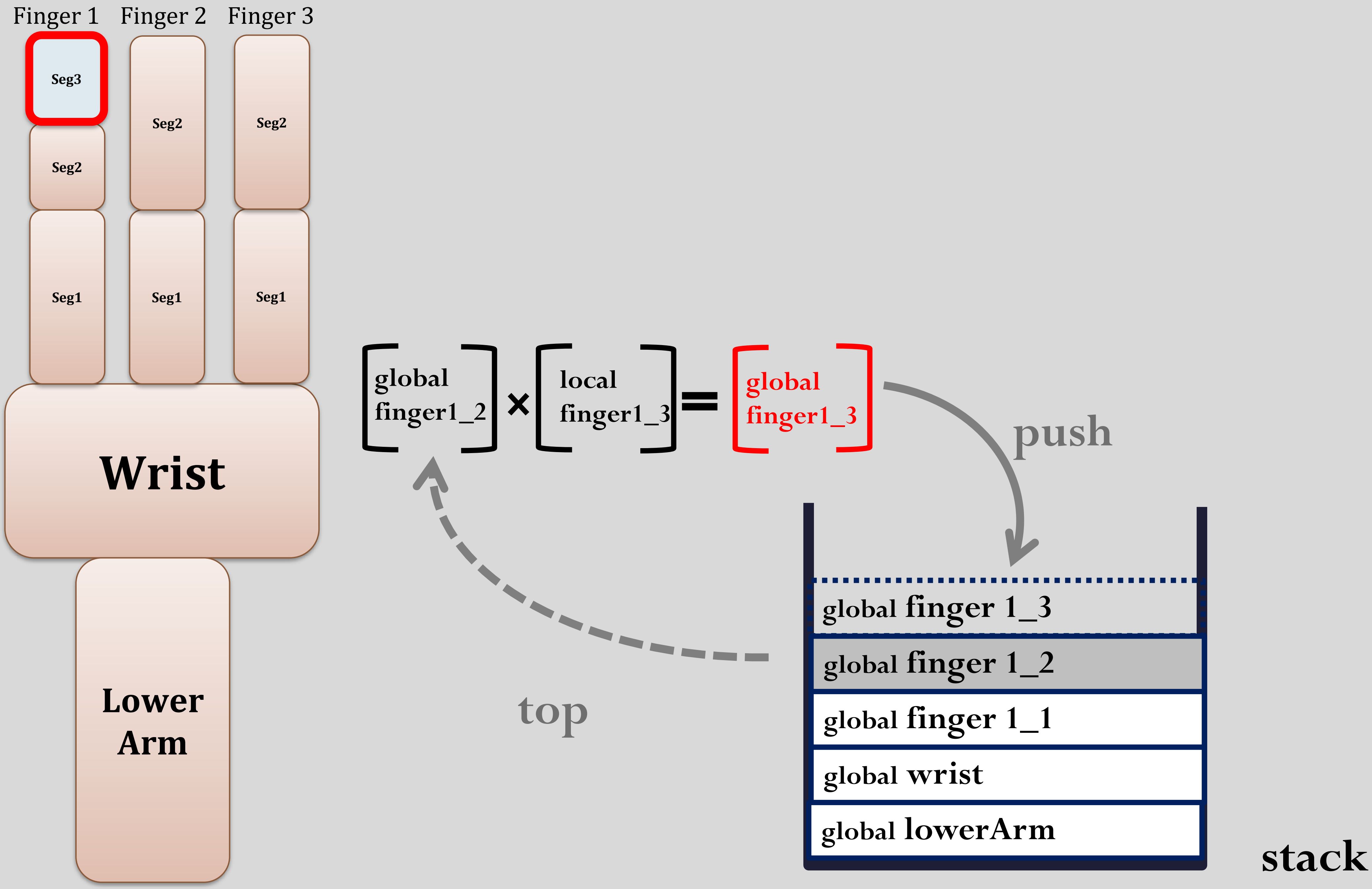
$$\begin{bmatrix}\text{global}\\ \text{wrist}\end{bmatrix} \times \begin{bmatrix}\text{local}\\ \text{finger1\_1}\end{bmatrix} = \begin{bmatrix}\text{global}\\ \text{finger1\_1}\end{bmatrix}$$

global finger 1_1

global wrist

global lowerArm


Finger 1
Finger 2
Finger 3
Seg3
Seg2
Seg2
Seg2
Seg1
Seg1
Seg1
Wrist
Lower Arm
global finger1_1
×
local finger1_2
=
global finger1_2
push
top
global finger 1_2
global finger 1_1
global wrist
global lowerArm
stack



Finger 1 Finger 2 Finger 3

Seg3

Seg2 Seg2 Seg2

Seg1 Seg1 Seg1

Wrist

Lower Arm

[global finger1_2] × [local finger1_3] = [global finger1_3]

push

top

global finger 1_3

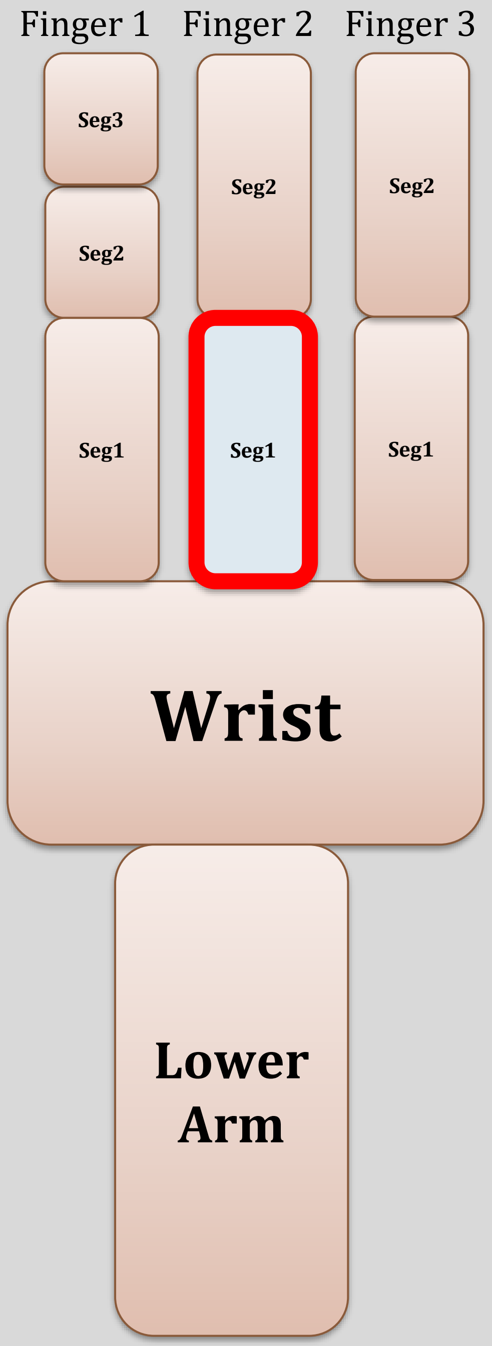
global finger 1_2

global finger 1_1

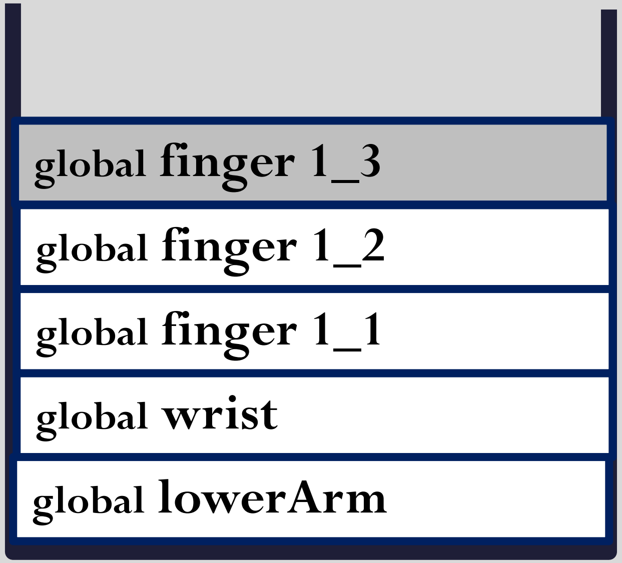
global wrist

global lowerArm

stack



$$\begin{bmatrix} \dots\dots \end{bmatrix} \times \begin{bmatrix} \text{local} \\ \text{finger2\_1} \end{bmatrix} = \begin{bmatrix} \text{global} \\ \text{finger2\_1} \end{bmatrix}$$



global finger 1_3

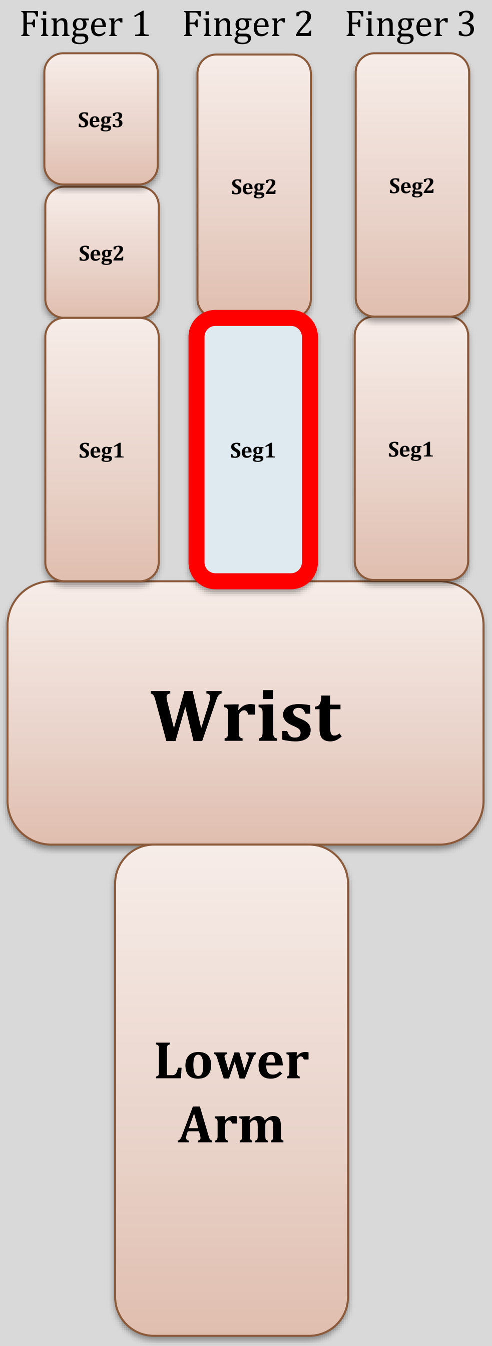
global finger 1_2

global finger 1_1

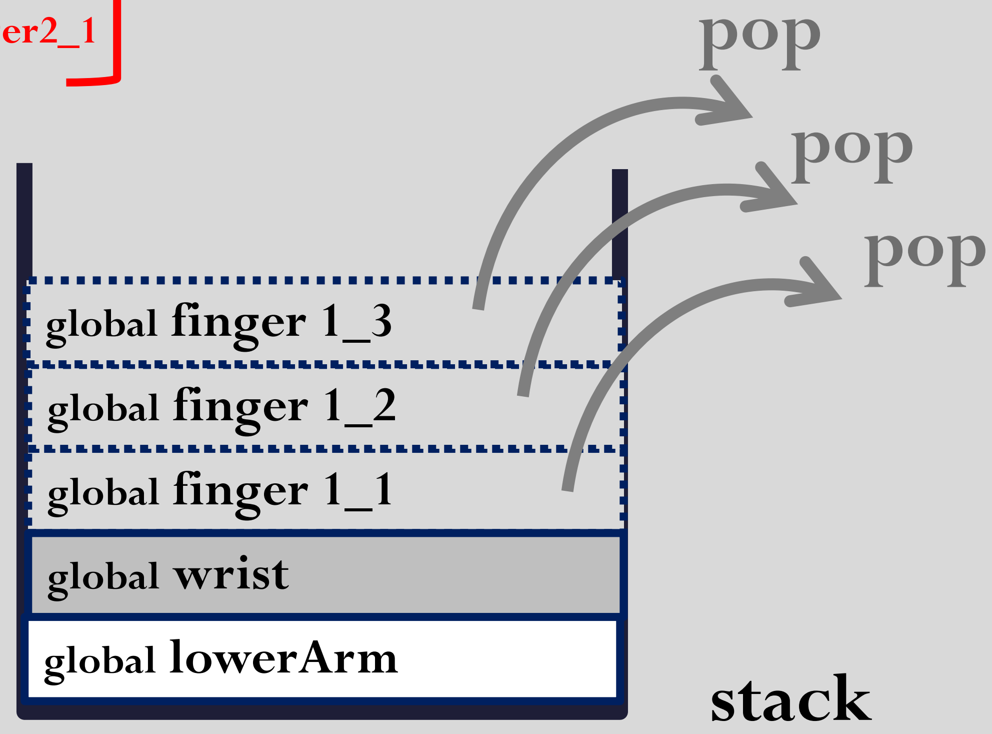
global wrist

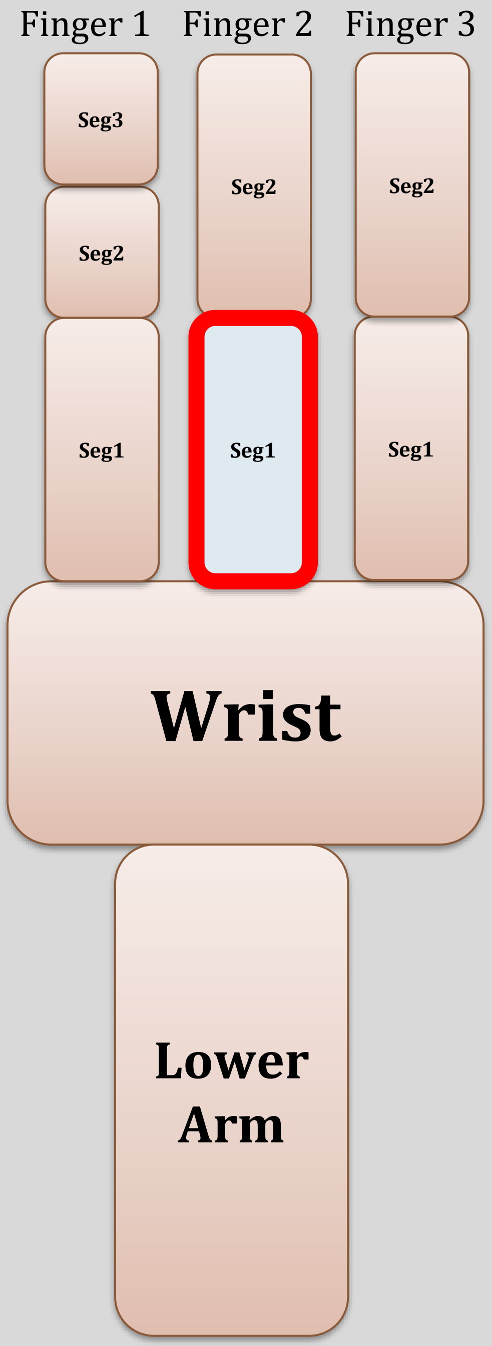
global lowerArm

stack

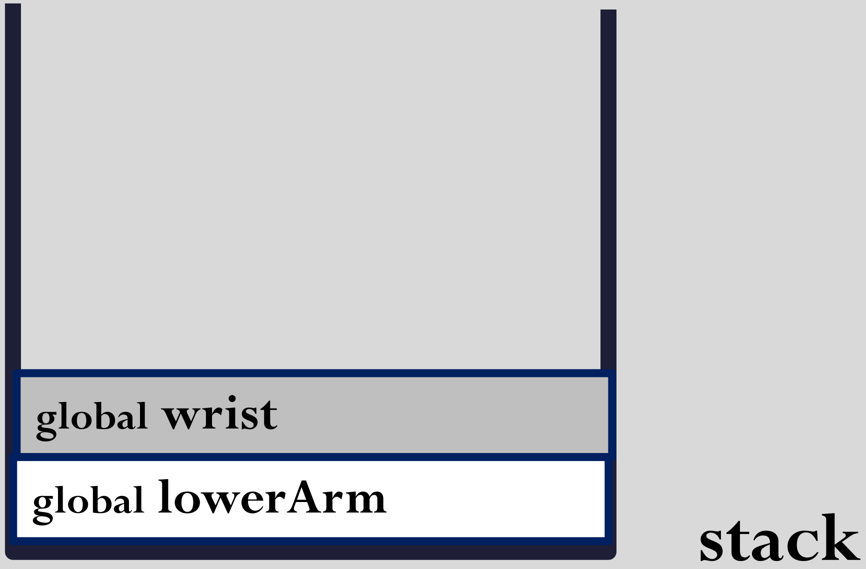


$$\begin{bmatrix} \text{....} \end{bmatrix} \times \begin{bmatrix} \text{local} \\ \text{finger2\_1} \end{bmatrix} = \begin{bmatrix} \text{global} \\ \text{finger2\_1} \end{bmatrix}$$





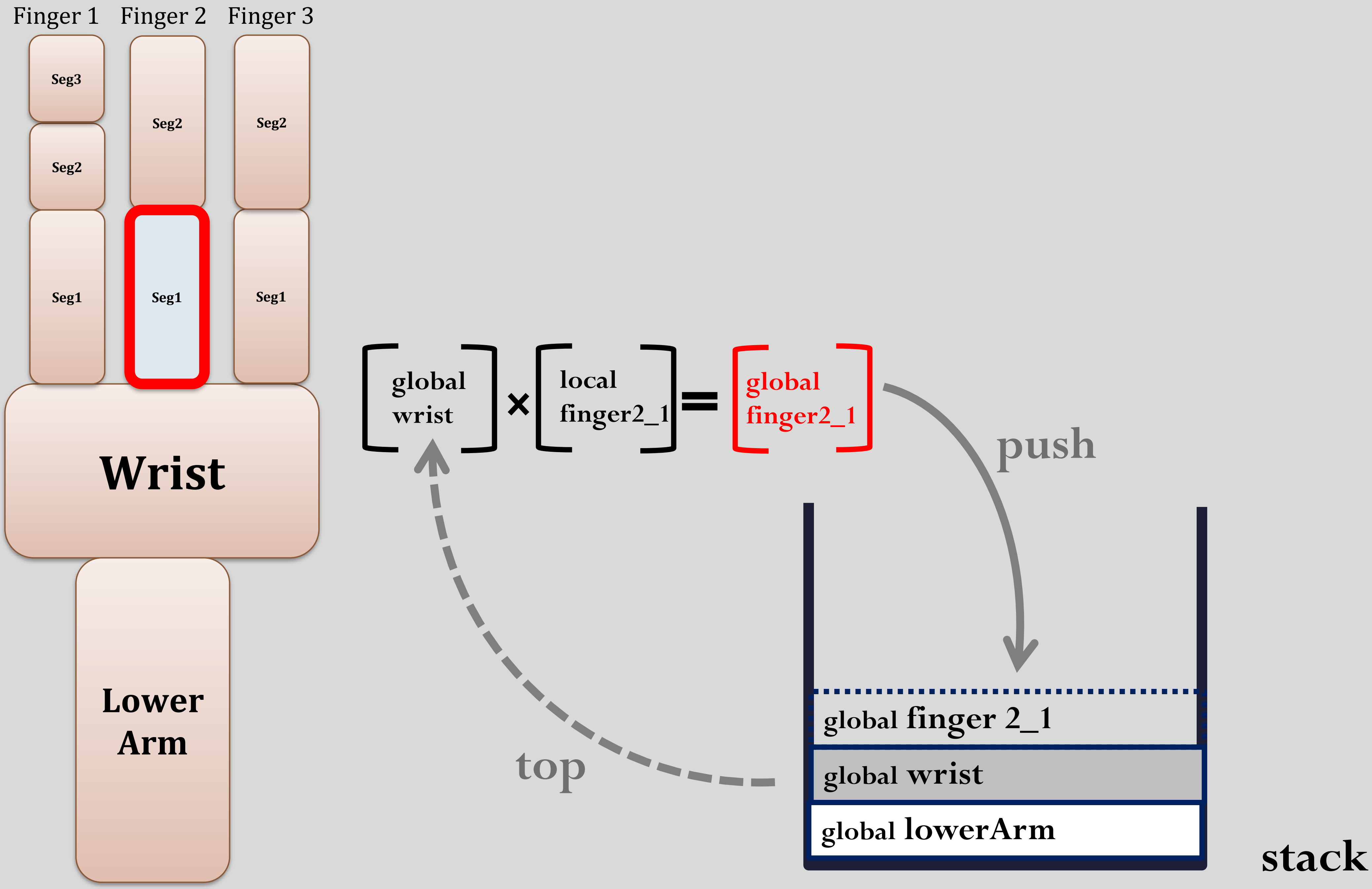
$$\begin{bmatrix} \text{....} \end{bmatrix} \times \begin{bmatrix} \text{local} \\ \text{finger2\_1} \end{bmatrix} = \begin{bmatrix} \text{global} \\ \text{finger2\_1} \end{bmatrix}$$



global wrist

global lowerArm

stack



Finger 1
Finger 2
Finger 3
Seg3
Seg2
Seg2
Seg2
Seg1
Seg1
Seg1
Wrist
Lower Arm
global wrist
×
local finger2_1
=
global finger2_1
push
top
global finger 2_1
global wrist
global lowerArm
stack

Summary

- Viewing
- Transformations
- Transformations in OpenGL
- Hierarchies
- Next - Animation!